Jere Honka

Robot Frameworkiä hyödyntävän testiautomaatiototeutuksen parantaminen MFC-pohjaisen sovelluksen graafiselle käyttöliittymälle

Tietotekniikan pro gradu -tutkielma

14. lokakuuta 2019

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Jere Honka

Yhteystiedot: jemihon@outlook.com

Ohjaajat: Antti-Juhani Kaijanaho ja Jukka-Pekka Santanen

Työn nimi: Robot Frameworkiä hyödyntävän testiautomaatiototeutuksen parantaminen MFC-pohjaisen sovelluksen graafiselle käyttöliittymälle

Title in English: Improving a test automation implementation utilizing Robot Framework for the graphical user interface of an MFC-based application

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 60+0

Tiivistelmä: Graafisten käyttöliittymien automaattinen testaaminen on hankalaa. Erityisen hankalaa se on silloin, kun testaustyökalut ovat ominaisuuksiltaan puutteellisia sekä käytössä olevat testiskriptien laadinnan käytänteet johtavat vaikeaselkoisiin ja heikosti laajennettaviin testiskripteihin. Tutkielmassa tutkitaan MFC:llä toteutetun ohjelmiston graafisen käyttöliittymän testiautomaatiototeutusta. Toteutuksen ongelmia sekä onnistumisia kartoitetaan, ja ongelmille kehitetään ratkaisuja. Ratkaisujen vaikutusta arvioidaan ohjelmiston kehittäneen organisaation testaajien kanssa haastattelun ja kokeen perusteella.

Avainsanat: Ohjelmistotestaus, automaattinen testaus, testiskripti, testauksen laatutekijät, GUI-testaus, MFC, Robot Framework, toimintatutkimus

Abstract: Automatic testing of graphical user interfaces is difficult. It is especially difficult when the tools used for testing lack essential features, and the policies applied to test script design result in unclear and unexpandable scripts. In the thesis the test automation implementation of a certain MFC-based application's GUI is examined. The problems and successes of the implementation are identified, and solutions are developed for the problems. The effect of the solutions is evaluated with the testers of the organisation which developed the application based on an interview and an experiment.

Keywords: Software testing, automatic testing, test script, quality attributes of testing, GUI-testing, MFC, Robot Framework, action research

Kuviot

Kuvio 1. Kirjautumisnäkymä. 28
Kuvio 2. Kirjautumisen epäonnistuessa avautuva näkymä. 29

Taulukot

Taulukko 1. Testiskriptin luomisen helppous. 45
Taulukko 2. Testiskriptin luettavuus. 45
Taulukko 3. Testiskriptin luomisen nopeus. 45
Taulukko 4. Testausviitekehyksen parannuksien hyödyllisyys. 45

Sisältö

1 JOHDANTO 1

2 TUTKIELMAN TAVOITTEET JA TUTKIMUSMENETELMÄ 3
2.1 Tutkimuksen kohde 3
2.2 Tutkimuksen tavoitteet ja suoritustapa 3
2.3 Tutkimusmenetelmän kuvaus 3
2.4 Tutkimusmenetelmän hyödyntäminen tutkielmassa 4

3 OHJELMISTOTESTAUS 6
3.1 Ohjelmistotestauksen määritelmä 6
3.2 Testaus ja ohjelmiston laatuominaisuudet 6
3.3 Manuaalinen testaus 8
3.4 Automaattinen testaus 9
3.5 Testauksen tehokkuus 9
3.6 Testauksen kattavuus 10
3.7 Testauksen tasot 10
3.8 Testaustyypit 12
3.9 Testausviitekehykset 13
3.10 Testausviitekehysten tyypit 13
3.11 Robot Framework 14

4 GRAAFISTEN KÄYTTÖLIITTYMIEN TESTAUS 17
4.1 GUI-testauksen ominaisuuksia 17
4.2 Automaattisen GUI-testauksen tekniikoita 18
4.3 Robot Framework ja GUI-testaus 19

5 MICROSOFT FOUNDATION CLASS LIBRARIES ELI MFC 21
5.1 MFC:n historiaa ja ominaisuuksia 21
5.2 MFC ja GUI-testaus Robot Frameworkilla 22

6 TESTIAUTOMAATIOTOTEUTUKSEN PARANNUKSET 27
6.1 Nykyisen toteutuksen ongelmien kartoitus 27
6.2 Esimerkki automatisoidusta kirjautumisen testitapauksesta 28
6.3 GUI-elementtien identifioijien nimeäminen ja ryhmittely 30
6.4 Abstrahoinnin parantaminen avainsanojen avulla 32
6.5 Testattavan sovelluksen testausrajapinnan laajentaminen 34

7 KOKEEN SUORITTAMINEN KOEHENKILÖILLÄ 37
7.1 Kokeen toteutusyksityiskohtia 37
7.2 Koehenkilöiden haastattelu 39

8 HAASTATTELUIDEN TULOKSET 40
8.1 Ensimmäinen koehenkilö 40
8.2 Toinen koehenkilö 41

8.3 Kolmas koehenkilö 43
8.4 Haastatteluiden tulosten pohdinta 44
8.5 Laatutekijöiden toteutuminen 47

9 YHTEENVETO 48

LÄHTEET 50

1 Johdanto

Ohjelmistotestaus on toimintaa, jonka tavoitteena on varmistaa, että ohjelmistokehityksessä tehdään oikeaa tuotetta ja se tehdään oikein. Ohjelmistotestaus koostuu testitapauksista, jotka kuvaavat, miten testaus suoritetaan. Testiskriptillä puolestaan tarkoitetaan toimintosarjaa syötteiden ja toimenpiteiden toistamiseen (Hambling, Morgan, Samaroo, Thompson & Williams, 2015).

Testitapaukset voidaan jakaa automaattisiin ja manuaalisiin. Automaattiset testitapaukset ovat usein enemmän tavoiteltuja kuin manuaaliset, sillä automaattisten testitapauksien suorittaminen vaatii vähemmän aikaa ja työtä. Automaattiset testitapaukset ovat erityisen olennaisia regressiotestauksessa, jossa pyritään varmistamaan, että muutokset testattavaan ohjelmistoon eivät riko ohjelmiston toiminnallisuutta. Aina testitapauksien automatisointi ei kuitenkaan ole toteuttamisen arvoista, sillä se voi olla kohtuuttoman työlästä.

Työlästä on erityisesti sovellusten graafisten käyttöliittymien testaus. Käyttöliittymätestaus on muuttunut entistä hankalammaksi vuosien saatossa, kun sovellukset ovat kehittyneet monimutkaisemmiksi. Komentorivipohjaisista käyttöliittymistä graafisiin käyttöliittymiin siirtyminen on tuonut monia uusia haasteita automaattiselle testaukselle. Graafisen käyttöliittymän (*graphical user interface*, GUI) testitapausten laatiminen vaatii paljon aikaa ja vaivaa. Lisäksi usein merkittävänä haasteena laadittavien testiskriptien kannalta on graafisten käyttöliittymien epävakaisuus (Marick, 1998), minkä seurauksena testiskripti saattaa kaatua tai jäädä jumiin kesken ajon.

Yleisimmät ohjelmistokehityksessä käytetyt kirjastot ja kehitystyökalut ovat vuosien saatossa vaihtuneet useaan otteeseen. 1990-luvulta 2000-luvun alkuun yksi yleisimpiä Windows-alustalle kehitettäessä käytettyjä kirjastoja oli Microsoft Foundation Class Library (MFC), jolla monet viime vuosikymmenen lopullakin laajassa käytössä olleet Windows-sovellukset kehitettiin (Neumann, Günther & Zenker, 2009). MFC on graafisten työpöytäsovellusten kehittämiseen suunniteltu kirjastojen joukko, jonka Microsoft toi markkinoille vuonna 1992. MFC:n suosio on selkeästi vähentynyt 2000-luvun alun jälkeen Microsoftin vaihtaessa fokusta .NET -viitekehyksen suuntaan. Tutkielman kirjoitushetkellä on kuitenkin jäljellä usei-

ta MFC-pohjaisia sovelluksia, jotka tarvitsevat ylläpitoa.

MFC:n suosion hiipumisen jälkeen testiautomaatiotyökalut kuitenkin jatkoivat kehitystään, ja uusia on tullut saataville. Robot Framework on yksi tehokas ja helposti laajennettava geneerinen avainsanapohjainen testausviitekehys, joka on kasvattanut suosiotaan erityisesti 2010-luvulla.

Tutkielman tavoitteena on selvittää, kuinka Hansen Technologies Oy:n MFC:llä toteuttaman ohjelmiston GUI:n testiautomaatiota voisi parantaa. Testausviitekehyksenä Hansenilla on käytetty Robot Frameworkiä, jonka käytön ongelmia ja onnistumisia yrityksen MFC-pohjaisen sovelluksen GUI-testiautomaatiossa tutkielmassa kartoitetaan. Olemassa olevan testiautomaatiototeutuksen kartoittamisen ja lähdekirjallisuuteen perehtymisen perusteella tutkielmassa kehitetään parannuksia yrityksen testiautomaatiossa käytettyihin toteutusratkaisuihin ja rajapintoihin. Parannuksien toimivuutta testataan käyttämällä organisaatiossa työskenteleviä testaajia koehenkilöinä, joille laaditaan parannuksien toimivuutta testaava koe. Koehenkilöiden kanssa suoritetaan myös haastattelu, jossa arvioidaan parannuksien toimivuutta.

Tutkielman aihe on ajankohtainen, sillä MFC-pohjaisia sovelluksia on yhä paljon ylläpidettävänä, mutta niiden graafisten käyttöliittymien testausta Robot Frameworkin kaltaisilla moderneilla testausviitekehyksillä ei ole paljoa tutkittu. Tutkielman tuloksista on siis hyötyä tutkittavan ohjelmiston testiautomaation kehitykselle ja myös muita testiautomaatiototeutuksia kehitettäessä.

2 Tutkielman tavoitteet ja tutkimusmenetelmä

Luvussa määritellään tutkielman tavoitteet ja tutkimusmenetelmä. Lisäksi luvussa kuvataan tutkimusmenetelmää ja sen soveltamista tutkimuksessa.

2.1 Tutkimuksen kohde

Testauksen kohteena oleva ohjelmisto on suunnattu mm. energiamarkkinoihin liittyvän mittausdatan tallentamiseen ja analysointiin. Mittausdataa pystyy siirtämään järjestelmästä ulos ja järjestelmään sisään standardoiduilla formaateilla kuten XML:llä. Sovelluksen käyttäjiä ovat asiakasorganisaatioissa henkilöt, jotka vastaavat mittausdatan käsittelystä. Käyttäjärooleja ovat järjestelmänvalvoja ja tavallinen käyttäjä. Testauksen kohteena on ohjelmiston graafinen käyttöliittymä.

2.2 Tutkimuksen tavoitteet ja suoritustapa

Tutkielman päämääränä on kehittää parannuksia Hansen Technologies Oy:n MFC:llä toteuttaman ohjelmiston GUI:n testiautomaatioon. Testausviitekehyksenä Hansenilla on käytetty Robot Frameworkiä, jonka käytön **ongelmia ja onnistumisia** yrityksen MFC-pohjaisen sovelluksen GUI-testiautomaatiossa tutkielmassa **kartoitetaan**. Kartoituksen perusteella **kehitetään parannuksia** yrityksen testiautomaatiossa käytettyihin toteutusratkaisuihin ja rajapintoihin kehittämällä testausviitekehystä eteenpäin sekä **esittelemällä suosituksia**. **Parannuksien toimivuutta arvioidaan** hyödyntämällä organisaatiossa työskenteleviä testaajia koehenkilöinä. Valituille koehenkilöille annetaan ensiksi suoritettavaksi parannuksien toimivuutta mittaava koe, ja tämän jälkeen heitä haastatellaan parannuksiin liittyen.

2.3 Tutkimusmenetelmän kuvaus

Tutkimusmenetelmänä toimii toimintatutkimus. Toimintatutkimuksessa pyritään kehittämään kohteena olevaa organisaatiota vaikuttamalla sen toimintatapoihin. Kyseinen tutkimusmenetelmä sopii tutkielmaan, sillä tutkielmassa kehitetään kohteena olevaa organisaatiota (Hansen

Technologies Oy:tä) parantamalla sen testiautomaation toteutusratkaisuja.

Toimintatutkimuksessa tavoitteena on ensin kartoittaa tarvittavat muutokset, sitten toteuttaa muutoksia tutkimuksen kohteeseen, ja tämän jälkeen arvioida saavutettuja muutoksia. Tutkija osallistuu tutkittavan kohteen toimintaan tutkijan tai konsultin roolissa muutosagenttina. On olennaista, että tutkimuksen kohteena olevan organisaation kanssa toimitaan kiinteässä yhteistyössä. Tutkimus tapahtuu yhdessä tai useammassa syklissä, jotka koostuvat viidestä vaiheesta: diagnosointi, suunnittelu, toteutus, arviointi ja oppiminen (Järvinen & Järvinen, 2004). Vaiheet on kuvattu Susmanin ja Everedin artikkelissa (Susman & Evered, 1978) seuraavasti:

1. **Diagnosointivaiheessa** pyritään tunnistamaan ja määrittämään ongelma.
2. **Suunnitteluvaiheessa** tarkastellaan eri lähestymistapoja ongelman ratkaisemiseksi.
3. **Toteutusvaiheessa** valitaan lähestymistapa ja toteutetaan sen mukaiset toimenpiteet.
4. **Arviointivaiheessa** tutkitaan toteutettujen toimenpiteiden vaikutuksia.
5. **Oppimisvaiheessa** tunnistetaan löydökset.

Oppimisvaiheen jälkeen yksi sykli on suoritettu. Syklin suorittamisen jälkeen voidaan suorittaa uusi sykli, jossa palataan oppimisvaiheesta diagnosointivaiheeseen ja sovelletaan tunnistettuja löydöksiä.

2.4 Tutkimusmenetelmän hyödyntäminen tutkielmassa

Tutkielmassa suoritetaan yksi toimintatutkimuksen sykli seuraavasti:

1. Kartoitetaan organisaation testiautomaation ongelmat.
2. Kehitetään parannuksia testiautomaatiototeutukseen ongelmien ratkaisemiseksi.
3. Toteutetaan parannukset ja testataan niiden toimivuutta suorittamalla koe, johon osallistuu kolme koehenkilöä.
4. Arvioidaan parannuksien toimivuutta haastattelemalla edellisen vaiheen koehenkilöitä.
5. Pohditaan, mitä hyötyä tutkimuksesta oli.

Tutkielmassa toteutettujen suositusten ja viitekehyksen parannuksien toimivuutta arvioidaan kokeella, jossa koehenkilöille annetaan toteutettavaksi graafisen käyttöliittymän testitapaus

ensin vanhoja käytänteitä hyödyntäen, ja sitten uusia parannettuja käytänteitä hyödyntäen. Kokeen perusteella arvioidaan, kuinka parannukset vaikuttavat testiskriptin luomisen helppouteen, luettavuuteen ja nopeuteen. Kokeen jälkeen koehenkilöitä haastatellaan, ja haastattelun avulla selvitetään, mitä mieltä he ovat parannuksista. Koehenkilöinä toimii kolme testiautomaation parissa työskentelevää henkilöä Hansen Technologies Oy:stä. Materiaalina tutkimukselle toimii lähdekirjallisuuden lisäksi tutkittavan ohjelmiston testiautomaatiototeutukseen liittyvät testiskriptit ja ohjelmakoodi.

3 Ohjelmistotestaus

Luvussa kuvataan tutkielman kannalta keskeisiä ohjelmistotestauksen käsitteitä.

3.1 Ohjelmistotestauksen määritelmä

Kirjassa *IEEE Computing Society's Guide to Software Engineering Body of Knowledge* (IEEE Computer Society, 2014) ohjelmistotestaus määritellään aktiviteetiksi, joka suoritetaan ohjelmiston laadun arvioimiseksi sekä myös sen parantamiseksi identifioimalla vikoja ja ongelmia. Tämän määritelmän mukaan ohjelmistotestaus voidaan jakaa ohjelmiston laadun arvioimiseen ja sen parantamiseen.

Ohjelmistotestaus voidaan nähdä myös sidosryhmien näkökulmasta. Ohjelmistotestaus on tutkivaa toimintaa, jonka tavoitteena on tarjota sidosryhmille informaatiota testattavan ohjelmiston laadusta (Meyer & Bertrand, 2008). Sidosryhmät voivat tätä informaatiota hyödyntäen tehdä päätöksen siitä, onko testattava ohjelmisto valmis julkistettavaksi.

3.2 Testaus ja ohjelmiston laatuominaisuudet

Ohjelmiston laadun määrittää se, kuinka hyvin ohjelmisto toteuttaa sille asetetut vaatimukset, sekä kuinka hyvin ohjelmisto täyttää käyttäjiensä tarpeet ja odotukset (IEEE Computer Society, 1990). Laadun tasoa arvioidaan ohjelmistotestauksella, joka koostuu validoinnista ja verifioinnista. Validoinnilla varmistetaan, että testauksen kohde sopii sen kohderyhmien käyttötarkoituksiin kohdealueella. Verifioinnilla pyritään varmistamaan, että sovellus täyttää sille asetetut vaatimukset ja suunnitellut toteutusratkaisut.

Ohjelmiston laatuominaisuuksien avulla määritellään testattavan ohjelmiston laatu eri näkökulmista. Ohjelmiston tärkeimmät laatuominaisuudet ISO/IEC 25010 -standardin (ISO/IEC, 2011) mukaan ovat toiminnallisuus, käytettävyys, luotettavuus, turvallisuus, tehokkuus, ylläpidettävyys, siirrettävyys ja yhteensopivuus. Ne on kuvattu Stefan Wagnerin kirjassa (Wagner, 2013) seuraavasti:

- **Toiminnallisuudella** tarkoitetaan sitä, että ohjelmisto tarjoaa toiminnallisten vaatimusten ja odotusten mukaisen toiminnallisuuden sen käyttäjälle.
- **Käytettävyys** kuvaa, kuinka tyydyttävästi käyttäjä pystyy käyttämään ohjelmistoa.
- **Luotettavuus** kuvaa sitä, kuinka usein ohjelmiston odotetaan tarjoavan palveluitaan. Luotettavuuteen kuuluu virheidenkäsittely kuvaten sitä, kuinka hyvin ohjelmisto kykenee toipumaan virheistä.
- **Tietoturvallisuus** kuvaa, kuinka hyvin ohjelmisto on suojattu tietoturvahyökkäyksiä vastaan.
- **Tehokkuus** kuvaa, kuinka tehokkaasti ohjelmisto käyttää sille varattuja resursseja, sekä kuinka nopeasti ohjelmiston käyttäjät saavat vastetta ohjelmistolta.
- **Ylläpidettävyys** kuvaa ohjelmiston muutettavuutta ja laajennettavuutta.
- **Siirrettävyys** kuvaa, kuinka vahvasti ohjelmisto on sidottu alustaansa, ja kuinka paljon muokkaamista ohjelmisto vaatii alustan muuttuessa.
- **Yhteensopivuus** kuvaa, kuinka helposti ohjelmisto on yhdistettävissä muihin ohjelmistoihin ja laitteistoihin.

Yllä kuvatuille laatuominaisuuksille on lisäksi määritelty ISO/IEC 25010 -standardissa (ISO/IEC, 2011) useita alilaatuominaisuuksia. Näistä alilaatuominaisuuksista tutkielman aiheen kannalta olennaisia ovat vikasietoisuus, modulaarisuus ja testattavuus. Ne on kuvattu ISO/IEC 25010 -standardissa seuraavasti:

- **Vikasietoisuus** on luotettavuuden alilaatuominaisuus. Se kuvaa tason, jolla ohjelmisto toimii tarkoitetulla tavalla pois lukien ympäristön viat.
- **Modulaarisuus** on ylläpidettävyyden alilaatuominaisuus. Se kuvaa tasoa, jolla ohjelmisto on jaettu erillisiin komponentteihin. Modulaarisessa järjestelmässä yhden komponentin muutos ei vaikuta merkittävästi muiden komponenttien toimintaan.
- **Testattavuus** on ylläpidettävyyden alilaatuominaisuus. Se kuvaa, kuinka helppoa järjestelmän testaukselle on määritellä kriteerejä, ja kuinka helppoa näiden kriteereiden toteutumista on testata.

Testattavuus sisältää seuraavat alilaatuominaisuudet (Santanen, 2018a):

- **Hallittavuus** kuvaa, kuinka helppoa testitapauksia on suorittaa, ja kuinka helppoa tes-

tattavalle järjestelmälle on määrittää syötteitä ja sisäisiä tiloja.

- **Näkyvyys** kuvaa, kuinka helppoa testitapauksien vasteita ja sisäisiä lopputiloja on tulkita.
- **Vakaus** kuvaa, kuinka hyvin järjestelmä tukee laajojen muutoksien tekemistä.
- **Luotettavuus** kuvaa tasoa, jolla testitapausten toistaminen samasta lähtötilasta samoilla syötteillä johtaa samoihin vasteisiin ja samaan lopputilaan.
- **Eristettävyys** kuvaa, kuinka hyvin kukin moduuli on testattavissa erillään muista.
- **Erillisyys** kuvaa, kuinka hyvin joka moduulilla on oma rajattu ja hyvin määritelty toiminnallisuus.
- **Ymmärrettävyys** kuvaa, kuinka hyvin testattava moduuli on dokumentoitu ja kuinka helposti se on tulkittavissa oikein lähdekoodin mukaan.
- **Automatisoitavuus** kuvaa, kuinka hyvin testattava moduuli on automatisoitavissa.
- **Yhdenmukaisuus** kuvaa, kuinka hyvin samanlaiset osat käyttäytyvät samalla tavalla, ja kuinka hyvin testitapauksia ja niiden osia voi uudelleenkäyttää.

3.3 Manuaalinen testaus

Manuaalitestauksessa testitapaus suoritetaan ilman automaatiota ihmisen toimesta. Manuaalisessa testauksessa ohjelmiston testaaja pyrkii käyttämään testattavan ohjelmiston ominaisuuksia riittävän kattavasti ja tehokkaasti sekä varmistamaan, että testattavat ominaisuudet toimivat odotetusti.

Manuaalisen testauksen testitapaukset ovat askellettuja ohjeita, joita testaaja pyrkii ohjelmistoa testatessaan seuraamaan (Kaur & Kumari, 2011). Manuaalitestaus on paljon aikaa vievä prosessi, sillä testitapauksen suorittaminen vaatii joka kerta ihmisen automaation puutteen takia. On havaittu, että manuaalisia testitapauksia suoritetaan harvemmin kuin automaattisia (Maximilien & Williams, 2003). Lisäksi koska testitapausten suorittaja on ihminen koneen sijaan, niin jotkut testattavan ohjelmiston viat saattavat inhimillisistä virheistä johtuen jäädä huomaamatta (Kaur & Kumari, 2011).

3.4 Automaattinen testaus

Automaattisessa testauksessa testitapauksia suoritetaan automaattisesti hyödyntämällä siihen tarkoitettuja työkaluja. Työkaluja käytetään myös testitapausten hallintaan ja niiden suorituksen tuloksien vertailuun odotettuihin tuloksiin (Kolawa & Huizinga, 2007). Automaattista testausta pidetään yleensä parempana vaihtoehtona manuaalitestaukselle, koska automaatiota käytettäessä olemassa olevia testitapauksia pystytään ajamaan useammin ja nopeammin. Kun automaattinen testitapaus on kerran laadittu, sen suorittaminen vaatii vain muutaman minuutin verran manuaalista työtä. Testitapauksien suorittaminen mahdollisimman usein johtaa testattavan järjestelmän luotettavuuden kasvuun (Fewster & Graham, 1999).

Automaatio mahdollistaa sellaisten testien suorittamisen, joiden suorittaminen on vaikeaa tai mahdotonta manuaalisesti (Fewster & Graham, 1999). Tällaisia testitapauksia ovat esimerkiksi sellaiset, joissa kysytään syöte useammalta sadalta käyttäjältä.

Testiautomaatioon sisältyy hyötyjen lisäksi myös ongelmia. Tällaisia ongelmia ovat esimerkiksi mahdollinen testiautomaatiotyökalujen yhteensopimattomuus organisaation muiden työkalujen kanssa, sekä hyvän testausprosessin puuttuminen organisaatiossa. Myös testaajien puutteellinen koulutus työkalujen käyttöön voi muodostua ongelmaksi (Rice, 2003).

3.5 Testauksen tehokkuus

Testauksen tehokkuudella kuvataan testauksesta saatuja tuloksia suhteessa sen tavoitteisiin ja käytettävissä oleviin resursseihin. Testauksen tehokkuuden määritelmä voidaan jakaa ulkoiseen ja sisäiseen tehokkuuteen.

Ulkoisella tehokkuudella tarkoitetaan saavutettujen tulosten suhdetta niille asetettuihin päämääriin (IEEE Computer Society, 2010). Mitä korkeampi on ulkoinen tehokkuus, sitä lähempänä tulokset ovat tavoiteltua tulosta. Ulkoista tehokkuutta arvioidaan testauksessa esimerkiksi virheitä havainneiden testien osuudella kaikista testeistä (Santanen, 2018a).

Sisäinen tehokkuus kuvaa kulutettujen resurssien suhdetta arvioituihin resursseihin. Mitä korkeampi on sisäinen tehokkuus, sitä tehokkaammin valittuja keinoja käytetään suorituskyvyn tai hyötysuhteen osalta. Testauksessa sisäistä tehokkuutta arvioidaan esimerkiksi

kulutettujen resurssien suhteesta arvioituihin resursseihin (Santanen, 2018a). Tutkielmassa keskitytään erityisesti sisäisen tehokkuuden parantamiseen.

3.6 Testauksen kattavuus

Testauksen kattavuudella mitataan sitä, kuinka suuri osa ohjelmistosta käydään läpi testitapauksia suoritettaessa. Testauksen kattavuusmittoja ovat esimerkiksi seuraavat:

- **Testikattavuus** kuvaa sen, kuinka suuri osa testattavan ohjelmiston vaatimuksista tulee testattua suorittamalla testitapaukset (IEEE Computer Society, 2010).
- **Koodikattavuus** kuvaa, kuinka suuri osuus lähdekoodista testitapauksilla suoritetaan (Santanen, 2018b). Koodikattavuuteen sisältyy mm. lausekattavuus, reunaehtokattavuus, ehtolausekattavuus sekä polkukattavuus (Myers, 2004)
- **Syötealuekattavuus** kuvaa, kuinka suuri osuus syötealueista otetaan huomioon testitapauksissa (Santanen, 2018b).

3.7 Testauksen tasot

Ohjelmistotestaus voidaan jakaa useisiin tasoihin. Tasot voidaan erotella toisistaan testauksen kohteen tai tavoitteen mukaan (IEEE Computer Society, 2014). Testauksen tasoja katsotaan yleensä olevan kolme tai neljä riippuen määritelmästä. Yhden määritelmän mukaan testauksen tasoihin kuuluvat yksikkötestaus, integraatiotestaus ja järjestelmätestaus (IEEE Computer Society, 2014) ja (Dooley, 2011). Joissain määritelmissä testauksen tasoihin lasketaan edellisten lisäksi hyväksymistestaus (Craig & Jaskiel, 2002).

Yksikkötestauksessa testataan yksiköitä, jotka ovat ohjelmakoodin pienimpiä toiminnallisia osia (Kolawa & Huizinga, 2007). Automatisoitu yksikkötesti suorittaa testauksen kohteen lähdekoodin osan ja tarkistaa, että kohde toimii sille asetettujen oletusten mukaisesti. Jos testattavan lähdekoodin osan tuottama tulos ei vastaa oletuksia, niin yksikkötesti on epäonnistunut (Osherove, 2009). Yksikkötestauksen hyöty on se, että se mahdollistaa yksittäisten moduulien testaamisen erikseen, ja virheiden löytämisen mahdollisimman ajoissa (Kolawa & Huizinga, 2007).

Integraatiotestauksessa tavoitteena on selvittää, toimivatko erillään suunnitellut ja toteutetut ohjelmistoyksiköt oikein, kun ne yhdistetään toisiinsa (Fowler, 2018) ja (IEEE Computer Society, 2010). Tällä testauksen tasolla pyritään löytämään virheitä rajapinnoista ja niiden käytöstä (McGregor & Sykes, 2001). Integraatiotestaus koostuu inkrementaalisesta ohjelmien ja moduulien linkittämisestä ja testaamisesta (IEEE Computer Society, 2010).

Integraatiotestaus on usein kallein ja eniten aikaa vaativa testauksen taso. Monissa ohjelmistokehitysprojekteissa integraatiotestaus vie 50–70% testaukseen allokoiduista resursseista (Tsai, Bai, Paul, Shao & Agarwal, 2001). Integraatiotestauksen voi jakaa neljään lähestymistapaan: kokoavaan testaukseen, jäsentävään testaukseen, voileipätestaukseen sekä kertarysäystestaukseen.

Kokoavassa testauksessa testaaminen aloitetaan testaamalla ohjelmiston rakenteen pienimpiä osia, ja yhdistämällä näitä suuremmiksi kokonaisuuksiksi. Tätä jatketaan, kunnes lopuksi testauksen kohteena on koko järjestelmä (Pressman & Maxim, 2014).

Jäsentävässä testauksessa ohjelmiston moduuleja integroidaan toisiinsa aloittaen ohjelmiston hierarkian huipulla olevasta kutsuvasta osajärjestelmästä ja edeten ohjelmiston moduuleihin, jotka tarjoavat perustason ominaisuuksia (Pressman & Maxim, 2014).

Voileipätestauksessa yhdistetään kokoavaa ja jäsentävää lähestymistapaa. Tässä lähestymistavassa ohjelmiston moduuleja integroidaan yhtä aikaa jäsentävästi ja kokoavasti (Rajkumar 2019).

Kertarysäystestauksessa kaikki ohjelmiston moduulit pyritään integroimaan keskenään yhdellä kertaa (Rajkumar 2019).

Järjestelmätestauksessa testataan kokonaista ja integroitua ohjelmistoa. Tavoitteena on varmistaa, että testauksen kohteena oleva ohjelmisto täyttää sille asetetut vaatimukset (IEEE Computer Society, 2010). Järjestelmätestauksessa testattavaa ohjelmistoa saatetaan tarkastella käyttäjän näkökulmasta. Testauksella voidaan myös pyrkiä löytämään ongelmia ohjelmiston kriittisistä toiminnallisuuksista, jotka loppukäyttäjältä jäisivät huomaamatta (Black, 2002).

Hyväksymistestauksessa tavoitteena on selvittää, täyttääkö testattava ohjelmisto sille asete-

tut hyväksymiskriteerit. Hyväksymistestauksen suorittaa ohjelmiston tilaaja tai muu hyväksymistestaukseen valtuutettu taho (IEEE Computer Society 2010). Hyväksymistestauksen kohteena ovat yleensä tyypilliset käyttötilanteet poikkeustilanteiden sijaan (Black, 2002).

3.8 Testaustyypit

Testaustyypit rajaavat niitä toimenpiteitä, joilla arvioidaan testauksen kohteen laatuominaisuuksia (Santanen, 2018b). Alaluvussa käsitellään tutkielman kannalta olennaisia testaustyyppejä ja sitä, miten testaustyypit liittyvät tutkielman aiheeseen.

Toiminnallisella testauksella viitataan testaukseen, jossa testauksen kohteena olevan ohjelmiston tietoja ja toimintoja testataan erikseen tai yhdessä ohjelmiston käsittelemän datan kanssa (Santanen, 2018b). Toiminnallisessa testauksessa tavoitteena on testata, että sovelluksen toiminnot käyttäytyvät niille asetettujen vaatimusten tai suunnitelmien mukaisesti (Everett & McLeod, 2007). Ohjelmiston toiminnallinen testaus koostuu usein testitapauksista, jotka koostuvat lähtötilasta, suoritettavista askelista, testisyötteistä sekä odotetuista tuloksista (IEEE Computer Society, 2010). Toiminnallinen testaus on usein niin sanottua mustalaatikkotestausta, jossa testauksen kohdetta testataan tutustumatta sen toteutusratkaisuihin tai lähdekoodiin (Kaner, Falk & Nguyen, 1999). Tutkielman aiheena oleva graafisten käyttöliittymien testaus Robot Framework -testiskripteillä on toiminnallista testausta.

Ei-toiminnallisessa testauksessa keskitytään testauksen kohteen ei-toiminnallisten laatuominaisuuksien testaamiseen. Ei-toiminnalliset laatuominaisuudet kuvaavat, miten testauksen kohde hahmotetaan sen kohdeympäristössä. Tällaisia ominaisuuksia ovat esimerkiksi luvussa 3.2 kuvatuista laatuominaisuuksista käytettävyys, luotettavuus ja tehokkuus.

Regressiotestausta suoritetaan, kun testauksen kohteena olevaa ohjelmistoa on muokattu. Regressiotestitapauksien avulla varmistetaan, että ohjelmistoon tehdyt muutokset eivät ole rikkoneet aikaisemmin toimivaksi todettua toiminnallisuutta (Leung & White, 1989). Regressiotestaus voidaan määritellä myös vaatimusten näkökulmasta testauksena, jolla varmistetaan testattavan ohjelmiston noudattavan muutosten jälkeen yhä sille asetettuja vaatimuksia (IEEE Computer Society 2014). Tutkielman aiheena olevaa graafisten käyttöliittymien automatisoitavaa testausta voidaan pitää pääosin regressiotestauksena, sillä testiskrip-

tien pääasiallinen tavoite on varmistaa, etteivät ohjelmistoon tehtävät muutokset riko käyttöliittymää.

Savutestauksella tarkoitetaan testausta, joka keskittyy testattavan ohjelmiston tärkeimpien ominaisuuksien testaamiseen. Savutestaus tunnetaan myös nimellä aloitustestaus. Savutestauksen tavoitteena on varmistaa, että testattavan ohjelmiston perusominaisuudet toimivat. Savutestauksen suorituksen jälkeen voidaan siirtyä testaamaan monimutkaisempia ominaisuuksia (Dustin, Rashka & Paul, 1999). Tutkielman aiheena olevaan graafisten käyttöliittymien automaattiseen testaukseen liittyen esimerkiksi yksinkertainen kirjautumisdialogin toimivuutta testaava testiskripti toimii savutestinä testattavalle ohjelmistolle.

3.9 Testausviitekehykset

Testiautomaatioviitekehykset ovat järjestelmiä, jotka määrittelevät testattavan ohjelmiston testiautomaation säännöt. Testiautomaatioviitekehyksen tavoitteena on tarjota perusta testiautomaatiolle ja helpottaa automaattisten testiskriptien toteutusta. Testiautomaatioviitekehyksiä kutsutaan tutkielmassa myös testausviitekehyksiksi.

Testiautomaatioviitekehyksen tulee sisältää ainakin seuraavat ominaisuudet (Laukkanen, 2006):

- Testiautomaatioviitekehyksen tulee mahdollistaa testiskriptien automaattinen suorittaminen. Lisäksi sen tulee raportoida testiskriptien tulokset, käsitellä virheet sekä tarjota työkalut testiskriptien tulosten analysointiin.
- Testiautomaatioviitekehyksen tulee olla helppokäyttöinen testaajille. Sen tulee tarjota hyvät työkalut testiskriptien laatimiselle, muokkaamiselle ja ajamiselle.
- Testiautomaatioviitekehyksen tulee tarjota hyvät työkalut testidatan, viitekehyksen ohjelmakoodin sekä testiskriptien ylläpitämiseen.

3.10 Testausviitekehysten tyypit

Testiautomaatioviitekehyksiä on monia eri tyyppejä. Kehittyneimpiä ovat aineisto-ohjatut ja avainsanapohjaiset viitekehykset, sillä ne sisältävät kaikki luvussa 3.9 kuvatut testiautomaatioviitekehyksen tärkeät ominaisuudet (Laukkanen, 2006).

Aineisto-ohjatun viitekehyksen pääpiirre on testidatan ja testiskriptin toteutuksen pitäminen erillään toisistaan. Testidata koostuu testiskriptin syötteistä ja vasteista, sekä se usein säilytetään esimerkiksi CSV-tiedostoissa tai tietokantatauluissa (Kelly, 2003) ja (Laukkanen, 2006). Testiskriptien luominen on nopeampaa ja helpompaa, kun testidata pidetään testiskriptin ulkopuolella. Tällöin testiskriptiä ei tarvitse muokata, kun testidata muuttuu.

Avainsanaohjattu viitekehys tarjoaa ratkaisuja aineisto-ohjatun viitekehyksen ongelmista mm. siihen, että monipuolisten testiskriptien laatiminen on hankalaa, ja uudenlaisten testiskriptien laatiminen vaatii lähes poikkeuksetta ohjelmointityötä (Laukkanen, 2006). Avainsanaohjatussa viitekehyksessä testiskripteistä erotetaan testidatan lisäksi myös testidataa käsittelevä logiikka. Testiskriptit koostuvat avainsanoista, jotka abstrahoivat testiaskelien operaatioiden toteutukset. Samoja avainsanoja voi käyttää useissa erilaisissa testiskripteissä, ja kattava avainsanojen kokoelma eli **avainsanakirjasto** mahdollistaa sen, että testiskriptejä voivat luoda myös henkilöt, joilla ei ole ohjelmointikokemusta (Wang, 2015).

3.11 Robot Framework

Robot Framework on geneerinen avoimen lähdekoodin testiautomaatioviitekehys, joka hyödyntää avainsanaohjattua lähestymistapaa. Se tarjoaa testiskriptien luomiseen kattavan avainsanakirjaston, jota voi laajentaa omilla Python- tai Java-ohjelmointikielillä luoduilla testikirjastoilla (Bisht, 2013). Robot Framework tarjoaa testiskriptien luontia auttavien avainsanakirjastojen lisäksi työkaluja testiskriptien tulosten raportointiin ja analysointiin. Robot Frameworkin suunnitteluperiaatteet ja toteutus perustuvat Pekka Klärckin pro gradu -tutkielmaan (Laukkanen, 2006), joka julkaistiin vuonna 2006.

Seuraavaksi esitetään yksinkertainen esimerkki tyypillisestä viitekehyksellä laaditusta testiskriptistä, joka testaa hypoteettisen sovelluksen kirjautumisen toimivuutta:

```
*** Settings ***
Resource          LoginKeywords.robot

*** Variables ***
${USERNAME} test-user
${PASSWORD} test-pass

*** Test Cases ***
Login is Successful
   Open Application
   Verify that Login View Is Open
   Input Username ${USERNAME}
   Input Password ${PASSWORD}
   Send Login Credentials
   Verify That User Is Logged In
   [Teardown] Close Application

*** Keywords ***
#Keywords used by the test could be added here
```

Esimerkissä otetaan aluksi asetusosiossa käyttöön resurssitiedosto `LoginKeywords.robot`, joka sisältää kirjautumistestin kannalta olennaiset avainsanat. Tämän jälkeen testitapausosiossa kuvataan aluksi testitapauksen nimi, ja sen jälkeen sisennettyinä avainsanat, joista testitapaus koostuu.

Robot Framework-testitiedosto koostuu neljästä osiosta. Nämä osiot ovat asetusten määrittely, muuttujien määrittely, testitapausten määrittely ja avainsanojen määrittely:

1. Testiskripti alkaa avainsanasta `*** Settings ***`, jonka jälkeen kuvataan ne kirjastot, resurssitiedostot ja muuttujatiedostot, joita testiskripti tarvitsee toimiakseen.
2. Tämän jälkeen on mahdollista määritellä testiskriptin käyttämät muuttujat avainsanan `*** Variables ***` alle. Testiskriptin ei tarvitse sisältää tätä osiota, jos muuttujia ei käytetä tai ne on määritelty erillisessä tiedostossa.
3. Seuraavaksi määritellään testitapaukset avainsanan `*** Test Cases ***` alle. Testitapauksia voi olla yksi tai useampia yhdessä testitiedostossa.

4. Lopuksi on mahdollista määritellä avainsanaosio `*** Keywords ***` -avainsanalla. Osioon määritellään testiskriptien hyödyntämiä avainsanoja. Yllä olevassa esimerkissä ei tarvitsisi olla avainsanaosiota, koska avainsanat on määritelty erillisessä `LoginKeywords.robot` -nimisessä tiedostossa.

4 Graafisten käyttöliittymien testaus

Graafiset käyttöliittymät ovat yleistyneet merkittävästi viimeisen 30 vuoden aikana. Vuonna 2005 julkaistussa artikkelissa (Memon, Nagarajan & Xie, 2005) todetaan, että 45–60% keskimääräisen sovelluksen lähdekoodista liittyy sen graafiseen käyttöliittymään. Luvussa käsitellään graafisten käyttöliittymien testaukseen liittyviä käsitteitä ja siihen liittyviä piirteitä verrattuna esimerkiksi komentorivipohjaisten käyttöliittymien testaukseen.

4.1 GUI-testauksen ominaisuuksia

Graafiset käyttöliittymät tarjoavat monia uusia haasteita testaamiselle ja erityisesti automaattiselle testaamiselle verrattuna komentorivipohjaisiin käyttöliittymiin. Komentorivipohjaisten sovellusten testitapausten automatisoimista on perinteisesti pidetty helpompana (Kanglin & Mengqi, 2006). Tyypillisessä komentorivipohjaisen sovelluksen testitapauksessa sovelluksen käyttöliittymää käytetään syöttämällä sovellukselle komentorivikomentoja, joiden suoritus ei riipu esimerkiksi näytön tilasta.

Tietokoneen näyttö on keskeisessä roolissa informaation esittämisessä graafisissa käyttöliittymissä. Graafisten käyttöliittymien testauksessa **käyttöliittymän tilalla** on usein suuri vaikutus, sillä testitapauksen suoritukseen vaikuttaa se, mitkä käyttöliittymän elementit ovat käyttöliittymässä aktiivisina suoritushetkellä. GUI-testitapauksissa on usein askelia, joissa painetaan painikkeita, siirretään hiirtä ja ollaan vuorovaikutuksessa muiden näytöllä esillä olevien elementtien kanssa (Kanglin & Mengqi, 2006). Testitapauksen automatisoinnin kannalta on edellä mainitun perusteella tärkeää, että automatisoiduista testiskripteistä löytyy tarvittavat mekanismit käyttöliittymän tilan hallintaan.

Yksi tyypillinen ominaisuus graafisilla käyttöliittymillä on se, että ne muuttuvat usein sovelluksen kehityksen aikana. Muutokset käyttöliittymään johtavat siihen, että myös olemassa olevia testitapauksia pitää muuttaa. Leottan ym. artikkelissa (Leotta, Clerissi, Ricca & Tonella, 2013) on todettu, että pienikin muutos GUI:ssa voi vaatia merkittäviä muutoksia jo toteutettuun testitapaukseen. Jos muutoksia tulee usein, voi testitapauksen jatkuva korjailu käydä työlääksi.

Graafisten käyttöliittymien automaattisesta testauksesta tekee hankalaa se, että käyttöliittymän elementtien kanssa on vaikea olla vuorovaikutuksessa. Useiden GUI-testaustyökalujen ongelmana on se, että ne eivät tunnista kaikkia GUI-elementtejä (Pettichord, 2002). Usein testattavalle graafiselle käyttöliittymälle ei myöskään ole olemassa standardia testausrajapintaa. Tämä johtaa "yritys ja erehdys" -tyyppiseen työnkulkuun, jossa testausviitekehykseen pitää usein tehdä muutoksia uusia testiskriptejä luotaessa (Pettichord, 2002).

4.2 Automaattisen GUI-testauksen tekniikoita

Graafisten käyttöliittymien automaattiseen testaukseen on olemassa useita lähestymistapoja. Yleisin lähestymistapa on jättää automaattinen GUI-testaus kokonaan tekemättä. Toisessa lähestymistavassa GUI-testauksessa käytetään testausympäristöä, jossa kutsutaan testattavan sovelluksen metodeja ja funktioita kuin niitä olisi kutsuttu GUI:sta. Kolmannessa lähestymistavassa käytetään olemassa olevia testaustyökaluja GUI-testauksen suorittamiseen (Memon, Nagarajan & Xie, 2005).

Yleinen tekniikka GUI-testiautomaatiossa on testiskriptin luominen nauhoitustyökalun avulla. **Nauhoitustekniikkaa** hyödynnettäessä testaaja on vuorovaikutuksessa GUI:n kanssa hiirtä ja näppäimistöä käyttäen. Työkalu nauhoittaa testaajan suorittamat toimenpiteet testiskriptiksi. Kun tämän testiskriptin myöhemmin ajaa, se suorittaa testaajalta nauhoitetut toimenpiteet GUI:ssa (Kanglin & Mengqi, 2006) (Wang, 2015).

Nauhoitustekniikan hyvä puoli on se, että sitä hyödyntämällä on helppo toteuttaa uusia testiskriptejä. Lisäksi nauhoitustyökalujen käyttäminen ei vaadi ohjelmointitaitoja, joten testiskriptejä voi toteuttaa myös henkilöt, joilla ei ole ohjelmointikokemusta (Leotta, Clerissi, Ricca & Tonella, 2013). Nauhoita ja toista -testaukseen liittyy kuitenkin monia ongelmia. Jos testattavaa käyttöliittymää muutetaan, pitää lähes aina muuttaa myös nauhoitettua testiskriptiä. Testiskriptiä muutettaessa on vaikea päätellä, mitä osia pitää muuttaa ja mitä ei (Wang, 2015). Muutokset käyttöliittymään saattavat tehdä nauhoitetun testiskriptin täysin käyttökelvottomaksi, jolloin nauhoitus pitää suorittaa uudestaan muutetulle käyttöliittymälle (Leotta, Clerissi, Ricca & Tonella, 2013).

Ohjelmoitavassa GUI-testauksessa käyttöliittymää testaava testiskripti luodaan manuaalisesti. Se koostuu joukosta komentoja, jotka suoritetaan testiskriptiä ajettaessa. Näin luodut testiskriptit ovat usein joustavampia kuin nauhoitustekniikkaa hyödyntäen luodut testit, sillä niitä on helpompi muokata ja korjata tarvittaessa. Ohjelmoidut testiskriptit on myös mahdollista parametrisoida, jolloin yhdellä testiskriptillä on mahdollista testata useita eri käyttöliittymän toimintoja parametreista riippuen. Ohjelmoitujen testiskriptien huono puoli on se, että niiden laatiminen vaatii korkeampaa teknistä osaamista, kuten ohjelmointitaitoja. Lisäksi testiskriptien laatimiseen vaadittava työmäärä on huomattavasti suurempi, kuin esimerkiksi nauhoitustekniikkaa hyödynnettäessä (Leotta, Clerissi, Ricca & Tonella, 2013). Työkaluja ohjelmoitavien testiskriptien luomiseen graafisille käyttöliittymille ovat esimerkiksi Selenium ja AutoTester. Seleniumia käytetään WWW-käyttöliittymien testaamiseen ja AutoTesteriä työpöytäsovellusten testaamiseen.

Malliin perustuvassa testauksessa testattavasta käyttöliittymästä luodaan malli, joka kuvaa sen toimintaa. Kun malli on luotu, sille generoidaan automaattisesti testitapauksia verkkojen läpikäyntialgoritmeja hyödyntäen. Testitapausten generoinnin jälkeen testit voidaan suorittaa niille tarkoitetulla suoritustyökalulla. Koska generoitu malli kuvaa testattavaa käyttöliittymää vain abstraktisti, voivat sille generoidut testitapaukset testata puutteellisesti käyttöliittymää, johon malli perustuu (Bae, Rothermel & Bae, 2014).

4.3 Robot Framework ja GUI-testaus

Robot Framework tarjoaa Windows-työpöytäsovellusten GUI-testaukseen `AutoItLibrary` -nimisen kirjaston AutoIt-testaustyökalulle. AutoIt tarjoaa funktioita ja metodeja automaattiseen näppäimistön näppäinten painamiseen ja hiiren siirtämiseen. Se myös mahdollistaa GUI-elementtien suoran manipuloinnin automaattisesti testiskriptissä. GUI-elementtejä ovat esimerkiksi käyttöliittymän painikkeet, listarakenteet ja valikot. AutoIt-työkalulla on mahdollista tehdä Windows API -kutsuja, mikä mahdollistaa MFC-pohjaisten sovellusten testaamisen (Robot Framework, 2019) ja (AutoIt Consulting Ltd, 2019).

Robot Frameworkilla ei ole tarjolla muita virallisia työkaluja Windows-työpöytäsovellusten GUI-testaukseen. Muille alustoille toteutettuja GUI-kirjastoja ovat esimerkiksi SwingLibra-

ry ja Selenium2Library. SwingLibrary tarjoaa testaustyökaluja Javan Swing-kirjastolla tehdylle GUI:lle, ja Selenium2Library puolestaan on suunnattu WWW-sovellusten testaamiseen (Robot Framework, 2019).

5 Microsoft Foundation Class Libraries eli MFC

MFC on sovellusviitekehys työpöytäsovellusten ohjelmointiin Microsoft Windows -käyttöjärjestelmälle. Sen tavoitteena on tehdä monimutkaisten työpöytäsovellusten toteuttamisesta mahdollisimman helppoa. Se tarjoaa kääreluokkia Windowsin Win32- ja COM-API:n päälle, sekä myös globaaleja funktioita, muuttujia ja makroja (Microsoft, 2016a). Vaikka MFC:n tavoitteena on ollut työpöytäsovellusten toteuttamisen helpottuminen, joiltain osin MFC jopa vaikeuttaa sovellusten toteuttamista verrattuna Windows API:n käyttämiseen suoraan (Petzold, 1999)

Luvussa käydään läpi MFC:n historiaa ja esitellään sen perusominaisuudet. Lisäksi kuvataan GUI-testiautomaation kannalta mielenkiintoisia ominaisuuksia, joita MFC:llä on.

5.1 MFC:n historiaa ja ominaisuuksia

MFC:n ensimmäinen versio julkaistiin vuonna 1992 Microsoftin C/C++ -kääntäjän mukana. MFC on tarjonnut ensimmäisestä versiostaan lähtien kevyen oliosuuntautuneen rajapinnan Windows API:n päälle. Rajapinnan ansiosta MFC:llä toteutetussa työpöytäsovelluksessa on harvoin tarvetta suorille Windows API -kutsuille (Prosise, 2003). Suorien Windows API -kutsujen sijaan MFC:tä käyttävät sovellukset kutsuvat MFC-luokkien jäsenmetodeja.

Microsoft toi .NET-viitekehyksen markkinoille vuonna 2002. Julkistuksen yhteydessä päätettiin, ettei MFC:tä enää ylläpidettäisi. Monet sovelluskehittäjät olivat kuitenkin kehittäneet koodimäärältään suuria MFC:tä hyödyntäviä sovelluksia ja olivat riippuvaisia päivityksistä MFC:hen voidakseen jatkaa sovellustensa tukemista moderneille Windows-versioille. Vuonna 2008 Microsoft julkisti aiemmasta päätöksestään poiketen suuren päivityksen MFC:hen ensimmäistä kertaa kymmeneen vuoteen (Neumann, Günther & Zenker, 2009). Uusia versioita on tullut markkinoille tämän jälkeen vuosittain.

MFC sisältää monia eri **GUI-elementtejä**, joista graafisen käyttöliittymän voi muodostaa. Näitä ovat esimerkiksi erilaiset painikkeet, menuvalikot, listarakenteet ja tekstikentät. MFC:n `combobox`-elementti yhdistää muokattavan kentän ja listakentän ominaisuudet. `Tabview`-

elementti puolestaan vastaa moderneista internet-selaimista tuttua välilehti-rakennetta. `Static`-elementti on yksinkertainen tekstiä sisältävä elementti.

MFC:llä toteutettu graafinen käyttöliittymä käsittelee käyttäjän sille suorittamat toimenpiteet omalla viestintärakenteellaan. Viesti syntyy jokaisesta käyttäjän toimenpiteestä, kuten esimerkiksi painikkeen painamisesta ja listanäkymän alkion valitsemisesta. Jokaiselle viestille on sovelluksessa määritelty käsittelijä, joka suorittaa käyttäjän toimenpidettä vastaavan toiminnon sovelluksessa. Viestien käsittelijät määritellään MFC:n viestien hakurakenteessa nimeltä `Message Map` (Microsoft, 2016b). Eri GUI-elementit identifioidaan niille asetetuilla ID-numeroilla.

Yllä kuvattua viestintärakennetta hyödynnetään testattaessa MFC:llä luotua graafista käyttöliittymää. Käyttäjän toimenpiteistä lähetettäviä viestejä vastaavia viestejä voidaan testiautomaatiossa lähettää esimerkiksi putkirajapinnan kautta. Putkirajapinnan avaaminen tapahtuu kutsumalla Windows APIn `CreateNamedPipe()`-metodia. Metodi ottaa ensimmäisenä parametrinaan putken nimen. Putken luomisen jälkeen asiakasprosessi voi luoda lukemis- tai kirjoitusyhteyden putkeen kutsumalla `CreateFile()`-metodia, antaen tälle metodille aiemmin määritellyn putken nimen (Microsoft, 2018). Testiautomaatiossa testattava sovellus ja testiautomaatiotyökalu voivat molemmat avata putkirajapinnan samannimiseen putkeen, ja tämän jälkeen ne pystyvät kommunikoimaan viestejä lähettämällä ja lukemalla putkirajapinnan kautta.

MFC-pohjaisten graafisten käyttöliittymien testaukseen on saatavilla useita työkaluja. Avoimen lähdekoodin vaihtoehto on esimerkiksi Python-kirjasto Pywinauto, joka tarjoaa rajapinnan eri GUI-toimintoja vastaavien viestien lähettämiseen MFC:lle Windows API:n kautta. Tätä kirjastoa voi hyödyntää esimerkiksi käytettäessä Robot Frameworkiä testaukseen. Maksullisia testiautomaatiotyökaluja ovat esimerkiksi AutoTester ja Maveryx.

5.2 MFC ja GUI-testaus Robot Frameworkilla

Robot Framework tarjoaa virallisesti vain AutoItLibrary-kirjaston kautta tuen Windows-työpöytäsovellusten testaamiseen. AutoItLibrary-kirjasto tarjoaa tuen AutoIt-testaustyökalu-

lle, jolla MFC-pohjaisen sovelluksen GUI-testiautomaatio on mahdollista toteuttaa. AutoIt -työkalua ei kuitenkaan ole käytetty tutkielman kohteena olevassa organisaatiossa. Organisaation testiautomaatiototeutus on sen sijaan rakennettu käyttämään suoraan MFC:n putkirajapintaa, joka kuvattiin luvussa 5.1.

Testattavaan ohjelmiston putkirajapinta avataan `CreateNamedPipe()` -metodikutsulla, ja tämän jälkeen putkeen tulevia viestejä voi alkaa kuuntelemaan `CreateFile()` -metodikutsulla. `WriteFile()` -kutsulla putkeen voi kirjoittaa, ja `ReadFile()` -kutsulla siitä voi puolestaan lukea. Alla on esitetty C++ -ohjelmointikielellä laadittu koodiesimerkki putken luonnista ja viestien lukemisesta putkesta:

```
// windows.h sisältää Windows API:n metodien esittelyt.
#include <windows.h>
#include <iostream>

// Datapuskurin koko.
constexpr int BUFFERSIZE = 1024;

int main()
{
  // Puskuri, johon luetaan dataa.
  CHAR dataBuffer[BUFFERSIZE];
  // Muuttuja, johon tallennetaan luettujen tavujen määrä.
  DWORD numOfBytes;
  // Putken nimi.
  LPCTSTR pipeName = L"\\\\.\\pipe\\examplePipe";

  // Putken luonti.
  HANDLE pipe = CreateNamedPipe(pipeName, // Putken nimi.
      PIPE_ACCESS_DUPLEX, // Luku- ja kirjoitusoikeudet.
      PIPE_TYPE_MESSAGE | // Putken tyyppi (viestintäputki).
      PIPE_READMODE_MESSAGE | // Lukemistyyppi (viesti).
      PIPE_WAIT, // Lukemiskutsu odottaa viestien saapumista.
      PIPE_UNLIMITED_INSTANCES, // Instanssien maksimimäärä.
      BUFFERSIZE, // Lähetyspuskurin koko.
      BUFFERSIZE, // Vastaanottopuskurin koko.
```

```
        5000, // Viestien odotusaika millisekunteina.
        NULL); // Turvallisuusattribuutit (esimerkissä ei käytetä).

    while (true)
    {
      // Odotetaan, kunnes putkeen yhdistetään.
      BOOL isConnected = ConnectNamedPipe(pipe, NULL);

      // Jos asiakasprosessin yhdistäminen onnistui, luetaan viesti.
      if (isConnected)
      {
        // Viestin lukeminen.
        BOOL isSuccess = ReadFile(pipe, // Putken kahva.
          dataBuffer, // Puskuri, johon dataa vastaanotetaan.
          BUFFERSIZE, // Puskurin koko.
          &numOfBytes, // Luettujen tavujen määrä.
          NULL); // Ei päällekkäistä IO:ta.

        // Jos lukeminen epäonnistui tai vastaanotettiin tyhjä viesti,
        // lopetetaan kuuntelu.
        if (!isSuccess || numOfBytes == 0)
          break;

        // Lisätään merkkijonon loppuun nolla, joka päättää merkkijonon.
        dataBuffer[numOfBytes] = '\0';
        // Kirjoitetaan viesti konsoliin.
        std::cout << dataBuffer << std::endl;

        // Katkaistaan asiakasprosessin yhteys putkeen.
        DisconnectNamedPipe(pipe);
      }
      else
        // Jos yhdistäminen epäonnistui, suljetaan putki.
        CloseHandle(pipe);
    }
}
```

Edeltävässä esimerkissä luodaan `\\.\pipe\examplePipe`-niminen putki `CreateNamedPipe()` -metodikutsulla, ja tämän jälkeen toistorakenteen sisällä otetaan vastaan asiakasprosessien putkeen lähettämiä viestejä. `ConnectNamedPipe()` -metodikutsu odottaa, että asiakasprosessi on yhdistänyt putkeen, ja tämän jälkeen `ReadFile()` -metodikutsulla luetaan asiakasprosessin lähettämä viesti.

Robot Frameworkin GUI-testiskriptit luovat yhteyden putkirajapintaan käyttämällä pywin32 -kirjastoa, joka tarjoaa edellä kuvatut Windows API-metodit Python-skriptin käytettäväksi. Alla kuvattu Python-metodi luo yhteyden putkirajapintaan ja kirjoittaa siihen parametrinaan saamansa viestin:

```
import win32file

def write_to_pipe(message):
  pipeName = r'\\.\pipe\examplePipe' #Putken nimi.
  handle = win32file.CreateFile(
  pipeName, #Putken Nimi.
  win32file.GENERIC_WRITE, #Asetetaan kirjoitusoikeudet.
  0, #Kielletään putken jakaminen yhteyden ollessa voimassa.
  None, #Turvallisuusattribuutit (None tarkoittaa, että niitä ei ole).
  win32file.OPEN_EXISTING, #Avataan olemassa oleva putki.
  0, #Ei lisä-attribuutteja.
  0) #Ei mallitiedostoa.

  #Kirjoitetaan viesti putkeen.
  win32file.WriteFile(handle, bytes(message, "ISO-8859-1"))
  #Suljetaan yhteys putkeen.
  win32file.CloseHandle(handle)
```

Testiskripti pystyy yllä esitetyn metodin avulla lähettämään viestejä putkeen nimeltä `\\.\pipe\examplePipe`, josta testattava sovellus lukee viestit. Testattava sovellus parsii viestin, ja sovelluksen käyttöliittymä suorittaa sitten viestin mukaisen toiminnon GUI:ssa. Viesti, joka asettaa tekstin `hello world` tekstikenttään jonka ID on `1106`, voisi olla esimerkiksi seuraavanlainen: `SETTEXT,1106,"hello world"`.

Tutkielman testauksen kohteena olevassa sovelluksessa on myös sisäänrakennettu toiminto GUI-elementtien ID-numeroiden selvittämiseen. Tietyllä komentoriviparametrilla käynnistettynä sovellus näyttää hiiren alla olevan GUI-elementin ID:n käyttöliittymänäkymän vasemmassa alareunassa.

6 Testiautomaatiototeutuksen parannukset

Organisaation testiautomaatiototeutukseen on tutkielmassa kehitetty useita käytettävyyteen, luettavuuteen ja toimivuuteen liittyviä parannuksia. Luvussa kuvataan organisaation testiautomaatiototeutuksessa kartoituksen kautta esiin tulleita ongelmia ja niiden perusteella tehtyjä muutoksia, sekä perustellaan muutokset lähdekirjallisuutta ja aiempien lukujen teoriaa hyödyntäen.

6.1 Nykyisen toteutuksen ongelmien kartoitus

Organisaation testiautomaatiototeutuksen ongelmia kartoitettiin tutkimalla, kuinka hyvin valitut luvussa 3.2 kuvatut eri laatuominaisuudet toteutuvat testiautomaatiossa. Kartoituksen kohteena olivat olemassa olevat testiskriptit ja testausviitekehyksen lähdekoodi. Lisäksi organisaation testiautomaation parissa työskentelevien henkilöiden kanssa keskusteltiin, mitä ongelmia testiautomaatiototeutuksessa oli havaittu, ja miten toteutusta voisi parantaa.

Laatutekijöitä, joiden toteutumista testiskripteissä ja testausviitekehyksessä pohdittiin, olivat ymmärrettävyys, vikasietoisuus, luotettavuus, automatisoitavuus ja yhdenmukaisuus. Nämä laatutekijät valittiin, koska ne koettiin tutkielman tavoitteiden kannalta tärkeimmiksi arvioitaviksi laatutekijöiksi.

Kartoituksessa ilmeni **testiskriptien ymmärrettävyyteen** liittyviä ongelmia. Monien testiskriptien kohdalla oli parannettavaa luettavuudessa, joka on ymmärrettävyyden osa. **Luettavuus** kärsi liian vähäisestä abstrahoinnista, mikä johti siihen, että testiskriptejä oli helppo tulkita väärin. Monia testiskriptien yhteenkuuluvia askelia käytettiin suoraan testiskripteissä, eikä niitä abstrahoitu esimerkiksi Robot Frameworkin avainsanoiksi. Muuttujiin abstrahoinnin sijaan monia GUI-elementtien ID-numeroita käytettiin suoraan avainsanojen parametreissa, mikä vaikeutti testiskriptien tulkitsemista.

Testiskriptien yhdenmukaisuudessa oli myös parannettavaa. Kartoituksessa tuli vastaan testiskriptejä, joissa sama toiminnallisuus oli toteutettu useaan kertaan. Usein samankaltaiset toteutukset myös käyttäytyivät hieman eri tavoin. Yhdenmukaisuus paranisi, jos sama

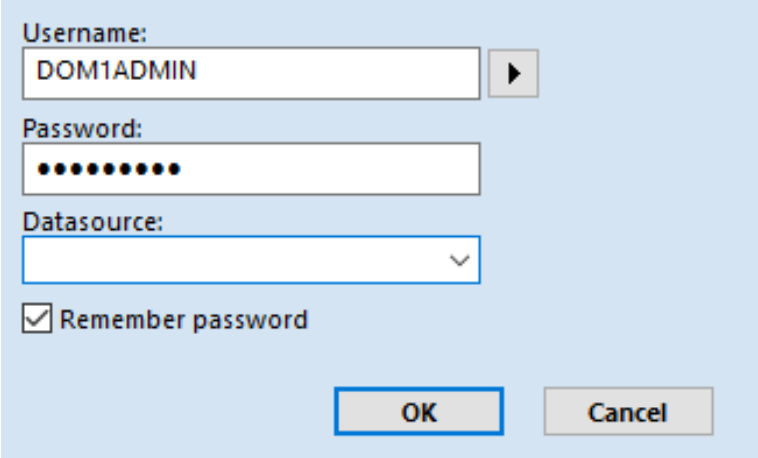
toiminnallisuus toteutettaisiin vain yhdessä paikassa, jolloin tätä kerran toteutettua toiminnallisuutta voisi sitten käyttää useassa testitapauksessa.

Myös testattavan sovelluksen GUI:n testiskriptien **automatisointiin** liittyviä ongelmia tuli esille kartoituksessa. Monien **GUI-elementtien tuki** puuttui kokonaan testausviitekehyksestä tai oli vain osittain toteutettu. GUI-testejä, joissa oltiin vuorovaikutuksessa näiden GUI-elementtien kanssa, **ei ollut mahdollista automatisoida lainkaan**. Myös joiltain testattavan sovelluksen osilta (kuten ns. “ohjausnäkymiltä”) puuttui tuki kokonaan testausrajapinnasta. Lisäksi testausviitekehyksessä ei ollut kunnollista tukea **GUI-elementtien tilan ja sisällön selvittämiselle** testiskriptin ajon aikana.

Kartoituksessa tuli esille myös onnistumisia testiskripteissä ja testausviitekehyksessä. Testiskriptit ja niiden käyttämät testausviitekehyksen ominaisuudet olivat **vikasietoisia**. Testiskriptit kaatuivat vain hyvin harvoin ja silloinkin poikkeuksellisissa olosuhteissa. Testiskriptien **luotettavuus** todettiin myös hyväksi, sillä testiskriptien mennessä läpi myös niiden testaamat ominaisuudet toimivat oikein. Koska näiden laatutekijöiden koettiin toteutuvan, ei niitä lähdetty kehittämään tutkielman puitteissa.

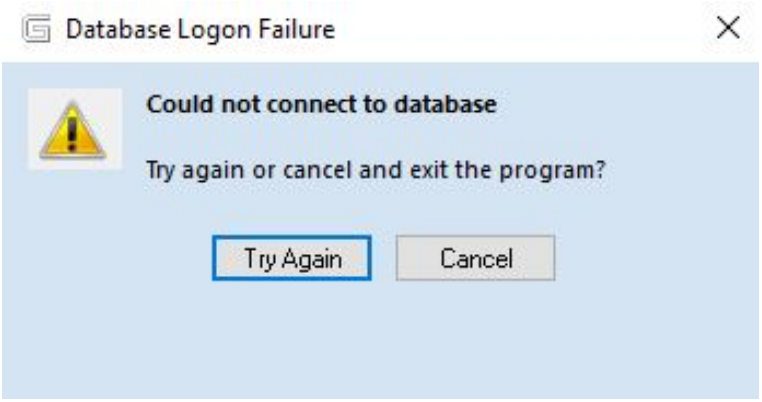
6.2 Esimerkki automatisoidusta kirjautumisen testitapauksesta

Testattavan sovelluksen kirjautumisnäkymä esitetään kuviossa 1.



Kuvio 1: Kirjautumisnäkymä.

Tavanomaisimmassa käyttötilanteessa sovelluksen käyttäjä syöttää kirjautumisnäkymän kenttiin käyttäjänimen, salasanan ja kohdetietokannan, sekä tämän jälkeen painaa *OK*-painiketta. Jos käyttäjätunnus on oikein, kirjautumisnäkymä sulkeutuu ja käyttäjä pääsee järjestelmään sisään. Jos käyttäjätunnus tai salasana on väärä, aukeaa kuvion 2 näkymä.



Kuvio 2: Kirjautumisen epäonnistuessa avautuva näkymä.

Virheilmoitusnäkymässä käyttäjä voi palata kirjautumisnäkymään ja yrittää uudelleen tai sulkea sovelluksen.

Edellä kuvatun toiminnallisuuden testaamiseksi on luotu seuraavanlainen testiskripti Robot Frameworkilla:

```
Verify that login dialog works as expected
  Open Browser
  Set Control Text 1009 HKI-D076/EDMS2
  Set Control Text 1012 wrong_user
  Set Control Text 1021 wrong_password
  Send Command  1
  Send Command  10
  Set Control Text 1009 HKI-D076/EDMS2
  Set Control Text 1012 WEBSERVICEUSER
  Set Control Text 1021 wrong_password
  Send Command  1
  Send Command  10
  Set Control Text 1009 HKI-D076/EDMS2
  Set Control Text 1012 WEBSERVICEUSER
  Set Control Text 1021 WEBSERVICEUSER_PW
```

```
Send Command  1
Set Treeview Focus /Time Series
Close Browser
```

Testitapauksessa avataan aluksi sovellus `Open Browser` -avainsanalla, joka avaa testattavan sovelluksen. Lisäksi avainsana asettaa testattavan sovelluksen tilaan, jossa se kykenee ottamaan vastaan testiskriptin lähettämiä viestejä. Kun sovellus on käynnistynyt, se avaa kuviossa 1 kuvatun kirjautumisnäkymän. Kirjautumisnäkymässä jokaisella GUI-elementillä on oma yksilöivä tunniste eli ID-numero: tietokantakentällä `1009`, käyttäjätunnuskentällä `1012` ja salasanakentällä `1021`. *OK*-painikkeen ID on `1`, ja kirjautumisen epäonnistuessa avautuvassa virheilmoitusnäkymässä (ks. kuvio 2) *Try Again* -painikkeen ID on `10`. Painikkeiden painaminen tapahtuu `Send Command` -avainsanalla, ja syötekenttiin tekstin syöttäminen tapahtuu `Set Control Text` -avainsanalla.

Testitapauksessa tarkistetaan ensin, että kirjautuminen epäonnistuu väärällä käyttäjätunnuksella ja salasanalla. Tämän jälkeen tarkistetaan, että oikealla käyttäjätunnuksella ja väärällä salasanalla kirjautuminen epäonnistuu. Seuraavaksi tarkistetaan, että kirjautuminen onnistuu oikealla käyttäjätunnuksella ja salasanalla. Tämä tapahtuu syöttämällä oikean käyttäjätunnuksen ja salasanan kirjautumisnäkymään, painamalla sen jälkeen *OK*-painiketta, ja lopuksi kokeilemalla `Time Series` -kansion avaamista sovelluksessa. Kansion avaaminen ei onnistu, jos kirjautuminen epäonnistuu. Viimeisenä askeleena testattava sovellus suljetaan `Close Browser` -avainsanalla, jonka jälkeen testitapauksen suoritus päättyy.

Testitapauksen luettavuus kärsii alhaisesta abstraktiotasosta. GUI-elementtien ID:t eivät kerro mitään kuvaamistaan elementeistä, sekä lisäksi avainsanojen `Set Control Text` ja `Send Command` nimet kuvaavat huonosti käyttötarkoituksiaan.

6.3 GUI-elementtien identifioijien nimeäminen ja ryhmittely

Luvussa 6.2 kuvatussa esimerkissä yksi ongelma on se, että GUI-elementtien ID-numeroiden suora käyttäminen testitapauksessa vaikeuttaa luettavuutta, sillä ne eivät kerro mitään kuvaamistaan elementeistä. Skripteissä usein käytettävien vakioiden käyttäminen hyvin nimettyjen muuttujien kautta helpottaa luettavuutta (Klarck, 2016). Luettavuuden parantaminen paran-

taa myös ymmärrettävyyttä, joka on yksi ongelmien kartoituksessa (ks. luku 6.1) mainituista laatuominaisuuksista, jonka toteutumisessa on parannettavaa.

Ensimmäisessä parannusehdotuksessa sovelletaan muuttujien käyttöä luvussa 6.2 kuvattuun testiskriptiin siten, että skriptissä käytetyt ID:t nimetään niiden identifioimien GUI-elementtien mukaan.

```
*** Variables ***
${DATABASE_FIELD} 1009
${USERNAME_FIELD} 1012
${PASSWORD_FIELD} 1021
${OK_BUTTON} 1
${TRY_AGAIN_BUTTON} 10

*** Test Cases ***
Verify that login dialog works as expected
  Open Browser
  Set Control Text ${DATABASE_FIELD} HKI-D076/EDMS2
  Set Control Text ${USERNAME_FIELD} wrong_user
  Set Control Text ${PASSWORD_FIELD} wrong_password
  Send Command  ${OK_BUTTON}
  Send Command  ${TRY_AGAIN_BUTTON}
  Set Control Text ${DATABASE_FIELD} HKI-D076/EDMS2
  Set Control Text ${USERNAME_FIELD} WEBSERVICEUSER
  Set Control Text ${PASSWORD_FIELD} wrong_password
  Send Command  ${OK_BUTTON}
  Send Command  ${TRY_AGAIN_BUTTON}
  Set Control Text ${DATABASE_FIELD} HKI-D076/EDMS2
  Set Control Text ${USERNAME_FIELD} WEBSERVICEUSER
  Set Control Text ${PASSWORD_FIELD} WEBSERVICEUSER_PW
  Send Command  ${OK_BUTTON}
  Set Treeview Focus /Time Series
  Close Browser
```

Tämän muutoksen jälkeen on selvempää, minkä GUI-elementtien kanssa testiskriptissä ollaan vuorovaikutuksessa.

6.4 Abstrahoinnin parantaminen avainsanojen avulla

Yhdenmukaisuus on yksi keskeinen testattavuuden tekijä (ks. luku 3.2). Luvussa 6.1 todettiin, että organisaation testiautomaation yhdenmukaisuuden toteutumisessa oli ongelmia. Tässä luvussa kuvataan, miten yhdenmukaisuutta voisi parantaa.

Robot Frameworkiä käytettäessä on uudelleenkäytettävyyden kannalta hyödyllistä abstrahoida yhteenkuuluvia toiminnallisuuden osia saman ja hyvin nimetyn avainsanan alle (Stanislav & Hocenski, 2011). Kun abstrahoinnissa yhteenkuuluvat toiminnallisuudet erotellaan omiksi kokonaisuuksikseen, toiston määrä vähenee ja saman toiminnallisuuden uudelleenkäyttö helpottuu, mikä edistää testiskriptien yhdenmukaisuutta. Myös testiautomaatiototeutuksen modulaarisuus paranee. Robot Framework -testiskriptien suhteen on myös todettu lähteessä (Klarck, 2016), että abstraktiotaso kannattaa valita siten, että Robot Frameworkin testitapausrakenteessa ei ole toteutukseen liittyviä yksityiskohtia. Sen sijaan tästä rakenteesta kutsuttavien avainsanojen nimistä pitäisi voida päätellä kaiken tarvittavan testitapauksen ymmärtämisen kannalta. Näin toimittaessa myös testiskriptien ymmärrettävyys paranee.

Luvuissa 6.2 ja 6.3 kuvatusta kirjautumisskriptistä saadaan eroteltua useampia yhteenkuuluvia kokonaisuuksia erillisiin avainsanoihin. Kirjautumiskenttien syöttämisen sekä *Try Again* -painikkeen kautta takaisin kirjautumisnäkymään siirtymisen saa eroteltua seuraavaan kahteen abstrahoituun avainsanaan:

```
*** Keywords ***
Set Login information and press OK
   [Arguments] ${username} ${password} ${database}
   Set Control Text ${DATABASE_FIELD} ${username}
   Set Control Text ${USERNAME_FIELD} ${password}
   Set Control Text ${PASSWORD_FIELD} ${database}
   Send Command  ${OK_BUTTON}

Press Try Again Button
   Send Command  ${TRY_AGAIN_BUTTON}
```

Näiden kahden avainsanan avulla voidaan luoda kolme uutta abstrahoivaa avainsanaa:

```
Try to log in with the wrong username and password
  Set Login information and press OK
  ... wrong_user wrong_password HKI-D076/EDMS2
  Press Try Again Button

Try to log in with the correct username and wrong password
  Set Login information and press OK
  ... WEBSERVICEUSER wrong_password HKI-D076/EDMS2
  Press Try Again Button

Try to log in with the correct username and password
  Set Login information and press OK
  ... WEBSERVICEUSER WEBSERVICEUSER_PW HKI-D076/EDMS2
  Press Try Again Button
  Set Treeview Focus /Time Series
```

Yllä olevassa esimerkissä kolme pistettä tarkoittaa, että edellinen rivi jatkuu. Esimerkissä ei enää toteuteta kirjautumisnäkymän kenttientäyttämiseen vaadittavaa logiikkaa kolmeen kertaan, vaan ainoastaan kerran. Avainsanaa `Set Login information and press OK` voi nyt uudelleenkäyttää missä tahansa kirjautumisen sisältävässä testitapauksessa. Tämä mahdollistaa yhdenmukaisempien testitapausten laatimisen. Luvussa käsitelty kirjautumista testaava testitapaus voidaan nyt muodostaa edellä kuvattuja kolmea avainsanaa käyttäen seuraavasti:

```
Verify that login dialog works as expected
  Open Browser
  Try to log in with the wrong username and password
  Try to log in with the correct username and wrong password
  Try to log in with the correct username and password
  Close Browser
```

Testitapausrakenteessa käytetään nyt korkeamman abstraktiotason avainsanoja, jonka tavoitteena on testiskriptin ymmärrettävyyden paraneminen. Luvussa 8 selvitetään kokeella se, kuinka hyvin tämä tavoite on saavutettu.

6.5 Testattavan sovelluksen testausrajapinnan laajentaminen

Luvussa kuvataan testattavan sovelluksen testausrajapintaan tekemiäni korjauksia ja parannuksia, joiden tavoitteena on helpottaa testiskriptien toteuttamista. Muutokset pyrkivät parantamaan testitapausten automatisoitavuutta. Automatisoitavuus on yksi kartoituksessa (ks. luku 6.1) mainituista laatuominaisuuksista, jonka toteutumisessa oli ongelmia. Luvussa käydään läpi testausrajapinnassa esiintyneet ongelmat, ja niihin kehitetyt ratkaisut.

Joidenkin testiskriptien toteuttamisen esteenä oli se, että niissä oli tarve olla vuorovaikutuksessa GUI-elementtien kanssa, joilta puuttui tuki testausrajapinnasta. Ratkaisuna näille elementeille lisättiin tutkielmassa tuki testausrajapintaan. Näihin elementteihin kuuluivat MFC:n `combobox`- ja `tabview`-elementit. `Combobox` -elementin aktiivisen alkion valitsemiselle toteutettiin `Select Combobox Item` -avainsana, ja dokumenttinäkymän sivulehden vaihtamiselle toteutettiin `Select Tab` -avainsana.

Avainsanoja `Select Combobox Item` ja `Select Tab` käytetään esimerkiksi seuraavasti:

```
Login to Browser
  Open User Group ${GROUP_ID}
  Select Tab ${PROPERTIES_TAB}
  Select Combobox Item ${EXAMPLE_COMBOBOX} ${FIRST_ITEM}
```

Testiskripteissä oli myös tarve saada tietoa testattavan sovelluksen GUI-elementtien tilasta ja sisällöstä. Testattava sovellus lähettää kuittausviestin jokaiselle testiskriptin lähettämälle viestille ollessaan valmis vastaanottamaan seuraavan viestin. Kuittausviestin kautta on mahdollista lähettää GUI-elementtien tilaan ja sisältöön liittyvää tietoa. Tätä ominaisuutta hyödyntäen testausrajapintaan lisättiin `get_latest_message()` -metodi, joka mahdollistaa viimeisimmän sovelluksen lähettämän kuittausviestin sisällön lukemisen testiskriptissä. Tätä hyödyntäen toteutettiin avainsanat `Request Listcontrol Data`, `Get Window Title` ja `Read Notification Bar`. Nämä avainsanat kuvataan seuraavaksi.

`Request Listcontrol Data` -avainsana palauttaa parametrien määrittelemän listanäkymän valitun alkion sisällön. Avainsanaa `Request Listcontrol Data` hyödyntävä abstrahoiva avainsana `Get Listcontrol Cell Contents` voitaisiin toteuttaa esi-

merkiksi seuraavasti:

```
Get Listcontrol Cell Contents
  [Arguments] ${listcontrol_id} ${row} ${column}
  Request Listcontrol Data ${listcontrol_id} ${row} ${column}
  ${message} = Get Latest Message
  [Return] ${message}
```

Yleinen tarve testiskripteissä oli varmistaa, mikä testattavan sovelluksen käyttöliittymän ikkuna on testiskriptin suorituksen tietyllä hetkellä aktiivinen. Tietyissä tilanteissa testiskriptin piti odottaa, että uusi ikkuna avautuu ennen kuin sen tuli jatkaa suoritusta. Tähän ongelmaan ratkaisuksi testausrajapintaan lisättiin tuki ikkunan otsikon lukemiselle `Get Window Title` -avainsanalla, joka palauttaa kutsuhetkellä aktiivisen ikkunan otsikon.

Avainsanaa `Get Window Title` käytetään esimerkiksi seuraavasti:

```
Verify that User Group Properties window is open
  Get Window Title
  ${message} = Get Latest message
  Should Be Equal As Strings ${message} User group properties
```

Tuki lisättiin myös MFC:n ilmoituspalkkielementin sisällön lukemiselle `Read Notification Bar` -avainsanalla. Tämän taustalla oli tarve lukea ilmoituspalkkiin ilmestyvien virheviestien sisältö.

Avainsanaa `Read Notification Bar` käytetään esimerkiksi seuraavanlaisesti:

```
Verify that the Object Creation Failed error notification is showing
  Read Notification Bar
  ${message} = Get Latest message
  Should Be Equal As Strings ${message} Object creation failed
```

Lisäksi osa melkein valmiista, mutta kesken jääneistä testausrajapinnan ominaisuuksista (ks. luku 6.1) tehtiin loppuun. Näihin kuuluu `Listcontrol Click` -avainsana, joka mahdollistaa listanäkymän alkion klikkaamisen tai tuplaklikkaamisen.

Avainsanaa `Listcontrol Click` käytetään esimerkiksi seuraavasti:

```
Listcontrol Click ${MAIN_LIST} row=1 column=1
```

Testattava sovellus tukee ns. ohjausnäkymien luomista. Ohjausnäkymät ovat sovelluksen käyttäjän luomia yksinkertaisia käyttöliittymiä sovelluksen sisällä. Niiden avulla sovelluksen käyttäjä saa helposti luotua näkymän, jossa näkyy kaikki hänen kannaltaan mielenkiintoinen informaatio sovelluksen käsittelemästä datasta. Ohjausnäkymien huono puoli oli aiemmin se, että niiden automaattinen testaaminen ei ollut mahdollista, sillä ohjausnäkymien sisältämiin GUI-elementteihin ei päässyt käsiksi testausrajapinnan kautta. Ratkaisuna ohjausnäkymien GUI-elementeille lisättiin tuki testausrajapintaan.

7 Kokeen suorittaminen koehenkilöillä

Testiautomaatiototeutukseen tehtyjen luvuissa 6.3 ja 6.4 kuvattujen parannusten toimivuutta testattiin kokeella, jossa koehenkilöt loivat luvuissa 6.2, 6.3 ja 6.4 kuvattua testitapausta `Verify that login dialog works as expected` vastaavan testiskriptin kirjautumistoiminnallisuudelle. Kokeessa verrattiin, kuinka nopeasti toimivan testiskriptin saa valmiiksi ilman parannuksia, sekä kuinka nopeasti parannusten kanssa. Koehenkilöitä oli kolme, joista kahdella oli kokemusta GUI-testiautomaatiosta Robot Frameworkillä yrityksen sisällä ja yhdellä ei. Kukaan koehenkilöistä ei ollut ollut tekemisissä kirjautumisnäkymään liittyvän testauksen kanssa. Koe toteutettiin siten, että satunnaisesti kaksi henkilöä toteutti kirjautumistestin ensin ilman parannuksia ja sitten parannusten kanssa, kun taas yksi toteutti testin ensin parannusten kanssa ja sitten ilman. Järjestyksen vaihtelun tavoitteena oli varmistaa, etteivät kokeen tulokset riippuneet suoritusjärjestyksestä.

Kokeen jälkeen koehenkilöille esiteltiin myös luvussa 6.5 kuvatut parannukset testausrajapintaan. Lopuksi koehenkilöitä haastateltiin kysyen alaluvun 7.2 kysymykset. Kysymysten kysymisjärjestys vaihteli koehenkilöiden välillä, ja joidenkin kysymysten kohdalla kysyttiin tarkentavia lisäkysymyksiä.

7.1 Kokeen toteutusyksityiskohtia

Koe jakautui kolmeen osaan:

1. Yhdessä ei hyödynnetty parannuksia lainkaan.
2. Toisessa hyödynnettiin GUI ID -numeroiden laittamista muuttujiin (ks. luku 6.3).
3. Kolmannessa abstrahoitiin avainsanoihin (ks. luku 6.4) osa skriptissä tarvittavista askelkokonaisuuksista.

Kaksi koehenkilöä toteutti skriptit ensin ilman parannuksia, sitten ID-numeroiden muuttujia hyödyntäen ja lopuksi abstrahoivia avainsanoja hyödyntäen. Yksi koehenkilö toteutti skriptit ensin abstrahoivia avainsanoja hyödyntäen, sitten ID-numeroiden muuttujia hyödyntäen ja lopuksi ilman parannuksia.

Kun kokeessa ei hyödynnetty parannuksia, oli koehenkilöiden käytössä testattavan sovelluksen sisältämä työkalu, jolla eri GUI-elementtien ID:t pystyy selvittämään testattavaa sovellusta käyttämällä. Työkalu kertoo hiiren kursorin alla olevan GUI-elementin ID:n sovelluksen käyttöliittymän vasemmassa alareunassa. Lisäksi avainsanat `Open Browser` ja `Close Browser` olivat käytössä sovelluksen avaamista ja sulkemista varten. Kokeessa toteutettu testiskripti testaa, että kirjautuminen oikealla käyttäjätunnuksella ja salasanalla onnistuu (ks. luku 6.2).

Kun kokeessa hyödynnettiin GUI-elementtien identifioijien nimeämistä ja ryhmittelyä (ks. luku 6.3), oli koehenkilöillä käytössä seuraavat GUI-elementtien ID:t sisältävät muuttujat:

```
${DATABASE_FIELD} 1009
${USERNAME_FIELD} 1012
${PASSWORD_FIELD} 1021
${OK_BUTTON} 1
${TRY_AGAIN_BUTTON} 10
${CANCEL_BUTTON} 2
```

Kokeessa toteutettu testiskripti testaa, että kirjautuminen oikealla käyttäjätunnuksella ja väärällä salasanalla epäonnistuu, sekä tietojen syöttämisen jälkeen käyttöliittymä palaa takaisin kirjautumisruutuun.

Kun kokeessa hyödynnettiin abstrahointia avainsanojen avulla (ks. luku 6.4), oli koehenkilöiden käytettävissä edellä kuvattujen työkalujen lisäksi luvussa 6.4 kuvatut avainsanat `Set Login information and press OK` ja `Press Try Again Button`. Kokeessa toteutettu testiskripti testaa, että kirjautuminen väärällä käyttäjätunnuksella ja väärällä salasanalla epäonnistuu, sekä tietojen syöttämisen jälkeen käyttöliittymä palaa takaisin kirjautumisruutuun.

Koetta suoritettaessa mitattiin kokeen suorittamiseen mennyt aika. Koehenkilöt suorittivat kokeen omilla työpisteillään käyttäen omia työkoneitaan. Tutkielman tekijä ei ollut fyysisesti samassa tilassa koehenkilöiden kanssa kokeen suoritusaikana. Koehenkilöt olivat yhteydessä tutkielman tekijään Skype-palaverissa, jossa he myös jakoivat työkoneidensa näytön tutkielman tekijän kanssa kokeen ajaksi. Tarvittaessa tutkielman tekijä auttoi koehenkilöä,

jos kokeen suorittamisessa esiintyi merkittäviä ongelmia. Jos koehenkilö tarvitsi apua kokeen suorittamiseen, tämä dokumentoitiin kokeen tuloksiin.

7.2 Koehenkilöiden haastattelu

Kukin koehenkilö haastateltiin kysyen seuraavia avoimia kysymyksiä parannusehdotuksista:

1. Kuinka paljon kokemusta sinulla oli aiemmin GUI-testiautomaatiosta ja minkälaista kokemusta?
2. Millä tavalla testiskriptin toteuttaminen oli helpompaa parannuksia hyödynnettäessä?
3. Oliko testiskriptin toteuttaminen jollain tavalla hankalampaa parannuksia hyödynnettäessä?
4. Miten GUI ID -numeroiden nimeäminen ja muuttujiin ryhmittely vaikutti luettavuuteen?
5. Mitkä testiautomaatioviitekehyksen parannuksista ovat hyödyllisimpiä ja miksi?
6. Mitkä testiautomaatioviitekehyksen parannuksista ovat vähiten hyödyllisiä ja miksi?
7. Mihin suuntaan GUI-testiautomaatiota kannattaisi vielä kehittää? Millä tavalla?

Näiden kysymysten lisäksi koehenkilöiltä pyydettiin mahdollisia yleisiä kommentteja kokeeseen liittyen.

Ensimmäisen kysymyksen tavoitteena oli kartoittaa koehenkilöiden taustoja testiautomaatioon liittyen. Toisella ja kolmannella kysymyksellä selvitettiin, helpottivatko parannukset koehenkilöiden mielestä testiskriptien toteuttamista. Neljännellä kysymyksellä arvioitiin, kuinka testiskriptien ymmärrettävyys parani GUI ID -numeroiden nimeämisen kautta. Viidennellä ja kuudennella kysymyksellä selvitettiin, miten testattavan sovelluksen automatisoitavuus parani viitekehyksen parannuksien kautta. Viimeisellä kysymyksellä oli tavoitteena saada koehenkilöiltä näkemyksiä organisaation GUI-testiautomaation tulevaisuuden kehityskohteista, ja myös siitä, mitä tai miten tutkielman aihealueesta voisi tehdä jatkotutkimusta.

8 Haastatteluiden tulokset

Luvussa kuvataan koehenkilöiden kanssa suoritetun kokeen ja haastattelun tulokset.

8.1 Ensimmäinen koehenkilö

Ensimmäinen koehenkilö suoritti kokeen ensin ilman parannuksia ja sitten parannusten kanssa. Koehenkilöllä oli aikaisempaa kokemusta testattavan sovelluksen GUI-testaamisesta ja Robot Frameworkistä. Hän suoriutui jokaisesta kokeen osuudesta ongelmitta. Ensimmäisessä kokeen osuudessa meni noin kuusi minuuttia, toisessa kolme minuuttia ja kolmannessa kaksi minuuttia.

Kokeen toisessa osuudessa tavoitteena oli, että testattavan sovelluksen käyttöliittymää ei tarvitsisi avata ollenkaan. Koehenkilö kuitenkin avasi sovelluksen testiskriptiä luodessaan tarkistaakseen, että hän ymmärsi testattavan toiminnallisuuden oikein. Kokeen jälkeen koehenkilö totesi, että vaikka kokeen toisen osuuden testiskriptin pystyi luomaan avaamatta testattavaa sovellusta, niin yleensä GUI-testiskriptiä luotaessa testattavan sovelluksen käyttöliittymä tulee kuitenkin avattua, jotta voidaan varmistua siitä, että testauksen kohde on ymmärretty oikein.

Kokeen kolmas osuus tuli suoritettua nopeasti ja vaivattomasti. Tässä osuudessa koehenkilö ei avannut testattavaa sovellusta. Hän totesi, ettei se ollut välttämätöntä olettaen, että saatavilla oleviin avainsanoihin pystyi luottamaan.

Haastattelussa koehenkilö totesi, että testiskriptien toteuttamisen nopeuden näkökulmasta tärkeimpiä parannuksista olivat luvussa 6.4 kuvatut avainsanat, joihin abstrahoitiin yhteenkuuluvia testiskriptien toiminnallisuuksia. Koehenkilön näkemystä tukee myös se, että kokeessa luoduista testiskripteistä avainsanoja hyödyntävän skriptin toteuttaminen vei vähiten aikaa. Luvussa 6.3 kuvattu GUI-elementtien identifioijien nimeäminen oli myös hyödyllistä testiskriptien toteuttamisen kannalta. Sen hyöty ilmeni koehenkilön mukaan pääasiassa luettavuuden paranemisessa, eikä niinkään toteuttamisen nopeuden kasvussa.

Kysyttäessä parannuksien huonoista puolista koehenkilö mainitsi, että jos tulevaisuudes-

sa pyritään parantamaan testausviitekehyksen abstraktiotasoa ja luomaan lisää avainsanoja erilaisille toiminnoille, niin testiskriptien luomisesta tulee huomattavasti helpompaa. Tämä helppous ei ole välttämättä kaikin puolin hyvä asia, vaan se voi johtaa siihen, että tulevaisuudessa testiskriptien toteuttajat tietävät vähemmän siitä, miten sovelluksen ja testiskriptin välinen kommunikointi toimii matalalla tasolla. Matalan tason avainsanat kuten `Send Command` ja `Set Control Text` voivat siis koehenkilön mukaan vaikuttaa hyvin hankalilta käyttää, jos on tottunut korkeampaan abstraktiotasoon.

Koehenkilö koki kaikki parannukset testiautomaatioviitekehykseen hyödyllisiksi eri tarkoituksiin. Tärkeää oli luvussa 6.5 kuvattu tuen lisääminen GUI-elementeille, joilla sitä ei aikaisemmin ollut. Tämä mahdollistaa uudenlaisten testien tekemisen, mikä koehenkilön mukaan parantaa testikattavuutta (ks. luku 3.6). Koehenkilö huomautti olevan kuitenkin vielä monia GUI-elementtejä, joilta tuki puuttuu. Esimerkiksi MFC:n `Static`-elementtien (ks. luku 5.1) lukeminen ei ole testiskripteissä nykyisellä toteutuksella mahdollista, vaikka siitä olisi hyötyä. `get_latest_window()` -metodi vaikutti hyödylliseltä. Koehenkilön mukaan sitä voisi käyttää esimerkiksi ponnahdusikkunoiden sisältämien viestien lukemiseen testiskriptissä.

Vähemmän hyödylliseksi koehenkilö koki ilmoituspalkkielementin lukemisen. Vaikka sekin on ominaisuutena hyödyllinen, koehenkilö koki, että saatu hyöty on vähäisempi kuin muiden parannusten kohdalla. Ilmoituspalkkiin ilmestyvät virheviestit ovat nimittäin usein luettavissa myös esimerkiksi sovelluksen käyttämästä lokitiedostosta. Ohjausnäkymien tukemisen lisäämiselle koehenkilöllä ei ollut kommentoitavaa, sillä hän ei ole testannut ohjausnäkymiä.

Kokonaisuutena koehenkilö oli tyytyväinen tehtyihin parannuksiin. Hän koki, että ne mahdollistavat uudenlaisten testiskriptien luomisen, ja että testiskriptien luominen on tulevaisuudessa nopeampaa.

8.2 Toinen koehenkilö

Toinen koehenkilö suoritti kokeen samassa järjestyksessä kuin ensimmäinen, ts. ensin ilman parannuksia ja sitten parannusten kanssa. Koehenkilöllä oli kokemusta MFC-pohjaisen so-

velluksen GUI-testauksesta, mutta ei GUI-testien tekemisestä Robot Frameworkilla. Robot Framework oli kuitenkin hänelle tuttu, sillä hän oli laatinut GUI-testaukseen liittymätöntä testiautomaatiota Robot Frameworkiä käyttäen. Hän oli myös laatinut GUI-testiskriptejä AutoTester-nimisellä työkalulla yli kaksi vuotta tutkimuksen ajankohtaa aiemmin.

Koehenkilö suoriutui kokeen osuuksista melko ongelmitta. Ensimmäisessä kokeen osuudessa meni noin kahdeksan minuuttia, toisessa kaksi minuuttia ja kolmannessa minuutti.

Ensimmäisessä osuudessa aikaa meni noin kolme minuuttia siihen, että koehenkilö sai ID-numeroiden hakemiseen tarkoitetun työkalun käyttöön testattavassa sovelluksessa. Koska koehenkilö ei ollut tehnyt GUI-testiautomaatiota pitkään aikaan, niin ymmärrettävästi aikaa meni tähän vaiheeseen enemmän kuin muilla koehenkilöillä. Kun koehenkilö oli saanut GUI-elementtien ID:t haettua sovelluksesta, testin toteuttaminen eteni pitkälti yhtä sujuvasti kuin ensimmäisen koehenkilön kohdalla.

Toisessa osuudessa meni aikaa selkeästi vähemmän. Tähän syyksi voi päätellä koehenkilön hyvän Robot Frameworkin osaamisen tason. Testiskriptin laatiminen ei vaatinut GUI-elementtien ID-numeroiden hakemista sovelluksen käyttöliittymästä, joten testiskriptin luomiseen riitti Robot Frameworkin tunteminen. Kokeen toisen osuuden jälkeen koehenkilö totesi muuttujien käyttämisen parantavan testiskriptin luettavuutta.

Koehenkilö suoritti kokeen kolmannen osuuden yhtä nopeasti ja vaivattomasti kuin ensimmäinen koehenkilö. Koehenkilö totesi avainsanojen nopeuttavan testin toteutusta merkittävästi.

Haastattelussa koehenkilö totesi, että GUI-elementtien ID-numeroiden hakeminen sovelluksesta vaikutti merkittävästi ensimmäisen kokeen testiskriptin toteutukseen tarvittavaan aikaan. Toisessa kokeessa nimetyt muuttujat nopeuttivat testiskriptin luontia ja paransivat luettavuutta koehenkilön mukaan. Kolmannen kokeen abstrahointimuutokset olivat koehenkilön mielestä merkittävimpiä. Ne nopeuttivat testiskriptin luontia huomattavasti, ja luotu testiskripti oli myös luettavampi kuin kaksi edellistä. Koehenkilö ei kokenut, että millään parannuksista olisi ollut huonoja puolia.

ID-numeroiden nimeäminen paransi koehenkilön mukaan luettavuutta, koska ID:t eivät sel-

laisenaan sisällä mitään tietoa elementeistä, joita ne kuvaavat.

Testiautomaatioviitekehyksen parannuksia läpikäytäessä koehenkilö totesi merkittävimmäksi parannukseksi tuen ohjausnäkymien testaukselle. Koehenkilö on työskennellyt ohjausnäkymien parissa ja koki, että niiden testiautomaatiossa on parannettavaa. Testiautomaatiotuen lisääminen ohjausnäkymille siis mahdollistaa GUI-testien kattavuuden merkittävän kasvattamisen. Tuen lisääminen muille GUI-elementeille oli myös koehenkilön mukaan hyödyllistä, mutta niistä saatava hyöty riippuu siitä, kuinka yleisiä nämä GUI-elementit ovat. Myös ilmoituspalkkielementin lukemisen mahdollistava parannus oli koehenkilön mielestä hyödyllinen virheviestien lukemiseen ja viestien perusteella erilaisten virhetilanteiden tulkitsemiseen testiskripteissä. Koehenkilöllä ei ollut kommentoitavaa `get_latest_window()` -parannukseen.

Haastattelun lopuksi koehenkilö totesi, että abstraktiotasoa parantavat avainsanat ja ohjausnäkymätuki olivat erityisesti hyödyllisiä parannuksia. Avainsanat helpottavat testiskriptien toteuttamista ja luettavuutta. Ohjausnäkymätuki mahdollistaa uudenlaisten GUI-testien tekemisen, mikä mahdollistaa GUI-testiautomaation kattavuuden parantamisen.

8.3 Kolmas koehenkilö

Kolmas koehenkilö suoritti kokeen päinvastaisessa järjestyksessä kuin kaksi edellistä. Koehenkilöllä oli aikaisempaa kokemusta GUI-testiautomaatiosta Robot Frameworkillä. Ensimmäisessä kokeen osuudessa meni kaksi minuuttia, toisessa neljä minuuttia ja kolmannessa viisi minuuttia.

Koehenkilö suoriutui suoritusjärjestyksen päinvastaisuudesta huolimatta parannuksia hyödyntävistä osuuksista nopeammin kuin parannuksia hyödyntämättömistä osuuksista. Tästä voidaan päätellä, että parannukset todella nopeuttavat testiskriptien laatimista, eikä suoritusjärjestys vaikuttanut merkittävästi kokeen tuloksiin.

Haastattelussa koehenkilö totesi, että parannuksien vaikutus testiskriptien luomisen nopeuteen näkyi selvästi kokeen suorituksessa, kun hän vertasi ensimmäiseen testiin vaadittua vaivaa viimeisen testin vaatimaan vaivaan. Erityisesti valmiit avainsanat tekivät testin toteutta-

misesta helppoa.

Parannuksien huonoista puolista kysyttäessä koehenkilö mainitsi yhden negatiivisen puolen liittyen testaajiin, jotka oppivat tekemään GUI-testiskriptejä vain käyttämällä valmiita avainsanoja. Nämä testaajat eivät välttämättä ymmärrä niin hyvin matalammalla tasolla sitä, miten sovelluksen ja testiskriptin välinen kommunikointi toimii. Myös mahdollisten testiskriptien luonnissa esiintyvien ongelmien selvittäminen voi olla hankalampaa, jos ymmärrys testausviitekehyksen toiminnasta on heikko. Koehenkilön mukaan olisi siis tärkeää, että testaajien tavoitettavissa olisi aina sellainen henkilö, joka ymmärtää testausviitekehyksen toiminnan matalammalla tasolla ja osaa täten auttaa tarvittaessa.

Parannuksista hyödyllisimpiä olivat koehenkilön mukaan tuen lisääminen uusille GUI -elementeille ja `get_latest_message()` -metodi. Metodin hyöty näkyy erityisesti siinä, että se mahdollistaa lisätarkistusten tekemisen `get_window_title()` -metodin kanssa. Vähemmän hyödyllinen ominaisuus oli tuen lisääminen ilmoituspalkin lukemiselle, sillä tuen puuttuminen ei estänyt uusien testiskriptien tekemistä, kuten esimerkiksi `combobox`-elementin tuen puuttuminen esti. Koehenkilö kuitenkin totesi, että kaikki parannukset olivat hyödyllisiä omalla tavallaan.

Lopuksi kysyttiin, mihin suuntaan GUI-testautomaatiota kannattaisi vielä kehittää. Koehenkilön mukaan uusien avainsanojen luominen yleisesti tarvittaville toiminnallisuuksille on tärkeää, ja näitä avainsanoja tulisi ryhmitellä järkevästi kirjastoihin.

8.4 Haastatteluiden tulosten pohdinta

Luvussa analysoidaan koehenkilöiden haastatteluiden tuloksia. Aluksi käydään läpi, mitä keskeisiä aiheita käsiteltiin jokaisessa haastattelussa, ja mitä näkemyksiä jokaisella koehenkilöllä oli aiheisiin liittyen. Tämän jälkeen käydään läpi ne asiat, jotka tulivat esille vain osassa haastatteluista.

Yhteisiä todettuja vaikutuksia olivat testiskriptin luomisen helppous, testiskriptin luettavuus, testiskriptin luomisen nopeus ja testausviitekehyksen parannuksien hyödyllisyys. Taulukoissa 1-4 käydään tiivistetysti läpi koehenkilöiden näkemykset aiheista.

Taulukko 1: Testiskriptin luomisen helppous.

Koehenkilö 1	Abstrahoinnin parantaminen ja avainsanojen luominen erilaisille toiminnoille helpottaa testiskriptien luomista, mikä ei kuitenkaan ole välttämättä kaikin puolin hyvä asia.
Koehenkilö 2	Avainsanat helpottavat testiskriptien toteuttamista.
Koehenkilö 3	Valmiit avainsanat helpottavat GUI-testiskriptien luomista, mutta testaamisen tekeminen liian helpoksi voi pitkässä juoksussa heikentää sovelluksen ja testiskriptien toiminnan ymmärrystä matalammalla tasolla.

Taulukko 2: Testiskriptin luettavuus.

Koehenkilö 1	GUI-identifioijien nimeäminen paransi testiskriptien luettavuutta.
Koehenkilö 2	Muuttujien käyttö paransi luettavuutta.
Koehenkilö 3	Luettavuus parani muuttujien käytön myötä.

Taulukko 3: Testiskriptin luomisen nopeus.

Koehenkilö 1	Avainsanat nopeuttivat testiskriptien luomista.
Koehenkilö 2	Avainsanat ja GUI-elementtien ID-numeroiden abstrahointi muuttujiin nopeutti testiskriptien luomista.
Koehenkilö 3	Parannukset nopeuttivat testiskriptien luomista. Erityisen tärkeitä olivat avainsanat.

Taulukko 4: Testausviitekehyksen parannuksien hyödyllisyys.

Koehenkilö 1	Kaikki parannukset olivat hyödyllisiä eri tarkoituksiin. Erityisen tärkeää oli tuen lisääminen GUI-elementeille, joilta se puuttui.
Koehenkilö 2	Merkittävin parannus oli tuen lisääminen ohjausnäkymien testaukselle. Myös ilmoituspalkkielementin lukemisen mahdollistava parannus oli tärkeä.
Koehenkilö 3	Tuen lisääminen uusille GUI-elementeille oli hyödyllisintä.

Testiskriptin luomisen helppouden suhteen jokainen koehenkilö oli sitä mieltä, että erityisesti avainsanojen käytön tuoma abstrahointi helpotti testiskriptien luomista. Koehenkilöiden huomioiden perusteella kattavaa avainsanakirjastoa voidaan pitää testiskriptien luomisen helppoutta edistävänä tekijänä. Helppouteen liittyen mainittiin kuitenkin myös, että jos testiskriptien luonti tehdään liian helpoksi, voi testausviitekehyksen syvällisempi ymmärtäminen heikentyä. Tämän liian helppouden tuoman ongelman ehkäisemiseksi esitetään testausviitekehyksen toimintojen hyvää dokumentointia. Lisäksi testaajien tavoiteltavissa tulee olla aina joku henkilö, jolta voi tarvittaessa kysyä neuvoa viitekehyksen toimintoihin liittyen.

Testiskriptien luettavuutta edisti kaikkien koehenkilöiden mukaan GUI-elementtien identifioijien nimeäminen, sekä hyvin nimettyjen avainsanojen käyttäminen. Näitä parannuksia voidaan tutkimuksen perusteella pitää luettavuutta edistävinä ominaisuuksina.

Testiskriptien luomisen nopeuteen eniten vaikuttava parannus oli kaikkien koehenkilöiden mukaan hyvän avainsanakirjaston saatavuus. Kattavaa avainsanakirjastoa voidaan siis pitää testiskriptien luomisen nopeutta edistävänä tekijänä.

Testausviitekehyksen parannuksista keskusteltaessa jokainen koehenkilö piti eri parannuksia tärkeinä. Esimerkiksi ilmoituspalkin lukeminen oli toisen koehenkilön mielestä merkittävä parannus, mutta muiden mielestä ei. Kaikki koehenkilöt kuitenkin pitivät GUI-elementtien tuen parantamista tärkeänä, josta voi päätellä sen olevan merkittävimpiä parannuksia.

GUI-testauksen tulevaisuuden kehityskohteista kysyttäessä koehenkilöillä ei ollut kovin vahvoja mielipiteitä asiasta. Kolmas koehenkilö mainitsi kuitenkin, että yleiskäyttöisten avainsanojen luominen on hyödyllistä GUI-testiautomaatiossa ja testiautomaatiossa yleisesti. Lisäksi viitekehyksen parannuksista yleisesti suosituin oli tuen lisääminen uusille GUI-elementeille. Tuki puuttuu vielä joiltain GUI-elementeiltä, joten tuen lisääminen näille elementeille voisi olla hyvä kehityskohde haastattelun perusteella.

8.5 Laatutekijöiden toteutuminen

Luvussa 6.1 tuli esille, että kolmen laatutekijän toteutumisessa oli parantamisen varaa. Nämä laatutekijät olivat ymmärrettävyys, automatisoitavuus ja yhdenmukaisuus. Luvussa pohditaan, kuinka parannukset edistivät näiden laatutekijöiden toteutumista haastatteluiden perusteella.

Koehenkilöt kokivat, että **testiskriptien luettavuus** ja samalla ymmärrettävyys parani abstrahoinnin ja GUI ID -numeroiden nimeämisen myötä. Koehenkilöt kuitenkin arvioivat luettavuutta itse toteuttamiensa testiskriptien kautta eivätkä tutustuneet toistensa testiskripteihin, mikä laskee näkemysten luotettavuutta.

Koehenkilöiden mukaan myös **testiskriptien automatisoitavuus** parani parannusten myötä. Näkemykset siitä, mitkä parannukset testausviitekehykseen olivat hyödyllisimpiä, vaihtelivat eri koehenkilöiden välillä. Jokainen koehenkilö oli kuitenkin sitä mieltä, että ainakin osa parannuksista oli erittäin hyödyllisiä.

Testiskriptien yhdenmukaisuuden paranemiseen liittyviä kysymyksiä ei kysytty. Abstrahoivat avainsanat ja GUI ID -numeroiden nimeäminen kuitenkin johtivat siihen, että koehenkilöiden kokeissa laatimat testiskriptit olivat rakenteiltaan hyvin samankaltaisia. Tästä voidaan päätellä, että yhdenmukaisuuden toteutuminen parani.

9 Yhteenveto

Tutkielman tavoitteena oli selvittää, miten Hansen Technologies Oy:n MFC:llä toteuttaman ohjelmiston GUI:n testiautomaatiota voisi parantaa. Tutkimus suoritettiin toimintatutkimuksena, jossa ensin kartoitettiin testiautomaation ongelmia ja sen jälkeen esitettiin kartoituksen tulosten perusteella parannusehdotuksia. Parannusehdotuksien toimivuutta arvioitiin kokeella, jossa koehenkilöinä toimi kolme organisaation testaajaa. Lopuksi haastattelun avulla pohdittiin parannuksien toimivuutta koehenkilöiden kanssa.

Kartoituksessa arvioitiin testausviitekehyksen käytön ongelmia ja onnistumisia viiden laatuominaisuuden toteutumisen kautta. Kartoituksen perusteella todettiin, että testausviitekehyksen vikasietoisuus ja luotettavuus ovat hyvällä mallilla, mutta ymmärrettävyydessä, automatisoitavuudessa ja yhdenmukaisuudessa oli parannettavaa. Kartoitusvaiheessa löydettyihin testiautomaation ongelmiin onnistuttiin löytämään ratkaisuja testiautomaatioviitekehyksen toteutuksen ja testiskriptien laatimisen käytänteiden parannuksien muodossa. Esitetyt parannukset olivat testiskriptien GUI-elementtien identifioijien nimeäminen ja ryhmittely, abstrahoinnin parantaminen Robot Frameworkin avainsanojen avulla ja testattavan sovelluksen testausrajapinnan laajentaminen.

Parannuksien kehittämisen jälkeen kolmelle koehenkilölle, jotka olivat organisaation testaajia, suoritettiin parannuksien laatua arvioiva koe ja haastattelu. GUI ID -numeroiden ja abstrahoivien avainsanojen todettiin kokeen tuloksien perusteella nopeuttavan ja helpottavan testiskriptien laatimista ja lukemista. Koehenkilöt kokivat, että testiskriptien luominen oli parannuksia hyödyntäen helpompaa ja nopeampaa kuin ilman parannuksia, sekä testiskriptien luettavuus parani. Lisäksi testausviitekehyksen toteutuksen parannukset koettiin hyödyllisiksi. Lopuksi todettiin koehenkilöiden kokemuksiin perustuen, että kehitettävät kolme laatuominaisuutta toteutuivat nyt paremmin tutkielmassa tehtyjen muutosten ansiosta. Koska kartoitus suoritettiin mielestäni kattavasti ja sen perusteella esitetyt parannukset todettiin laatuominaisuuksia edistäviksi, koen, että organisaation testiautomaation laatu parani tutkielman ansiosta ja tutkielma onnistui tavoitteessaan.

Tutkielman tulokset ovat voimassa vain tietyin rajauksin. Tutkielmassa otetaan kantaa vain

MFC-pohjaisten sovellusten GUI-testaukseen, ja GUI-testauksen toteuttamiseen Robot Frameworkillä.

Tutkielman havaintojen luotettavuutta arvioitaessa on otettava huomioon, että koehenkilöitä oli vain kolme. Vaikka koehenkilöiden pieni määrä laskee luotettavuutta, voidaan kokeen tuloksia kuitenkin pitää suuntaa-antavina. Testiskriptien luettavuutta arvioitaessa on myös huomioitava, että koehenkilöt arvioivat luettavuutta itse luomiensa testiskriptien perusteella, eivätkä he tutustuneet toistensa testiskripteihin. Tämä laskee luettavuudesta tehtyjen havaintojen luotettavuutta.

Organisaation testiautomaatiototeutuksessa on vielä kehityskohteita jäljellä. Joiltain testattavan sovelluksen ominaisuuksilta puuttuu vielä tuki testausrajapinnasta, ja avainsanakirjastossa on täydennettävää. Tutkielman aiheesta on myös mahdollista tehdä jatkotutkimusta. Tutkielman tuloksia voisi kokeilla yleistää muihin MFC-pohjaisten sovellusten testiautomaatiototeutuksiin, ja tutkielmassa esiteltyjä parannusehdotuksia voisi arvioida laajemmalla koehenkilöjoukolla.

Tutkielman kirjoittaminen oli minulle opettavainen kokemus. Opin paljon uutta ohjelmistotestauksesta, ja erityisesti työpöytäsovellusten graafisten käyttöliittymien testauksesta. Myös MFC-pohjaisten työpöytäsovellusten toteutusratkaisujen yksityiskohdat tulivat tutummiksi. Ymmärrykseni tietotekniikasta on tutkielman kirjoittamisen seurauksena laajempaa ja syvällisempää.

Lähteet

AutoIt Consulting Ltd 2019, `https://www.autoitscript.com/site/autoit/`, viitattu 8.8.2019.

Bae, G., Rothermel, G. & Bae, D. 2014, *Comparing Model-Based and Dynamic Event-extraction Based GUI Testing Techniques: An Empirical Study*, s. 1–2, Journal of Systems and Software Volume 97, November 2014, Pages 15–46.

Bisht, S. 2013, *Robot Framework Test Automation*, s. 10 ja 14, PACKT, ISBN:1783283033.

Black, R. 2002, *Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing*, s. 7–8, Wiley Publishing, Inc., ISBN:0471223980.

Craig, R. & Jaskiel, S. 2002, *Systematic Software Testing*, s. 95, Artech House Publishers, ISBN:1580535089.

Dooley, J. 2011, *Software Development and Professional Practice*, s. 193, Apress, ISBN:1430238011.

Dustin E., Rashka J. & Paul, J. 1999, *Automated Software Testing: Introduction, Management, and Performance*, s. 43-44, Addison-Wesley Professional, ISBN:0201432870.

Everett G. & McLeod, R. 2007, *Software Testing: Testing Across the Entire Software Development Life Cycle*, s. 99, Wiley, ISBN:9780470146354.

Fewster, M. & Graham, D. 1999, *Software test automation: effective use of test execution tools*, s. 26–27, Addison-Wesley Professional, ISBN:9780201331400.

Fowler, M. 2018, *IntegrationTest*, `https://martinfowler.com/bliki/IntegrationTest.html`, viitattu 4.6.2019.

Hambling B., Morgan P., Samaroo A., Thompson G. & Williams P. 2015, *Software Testing*, s. 80, BCS Learning & Development Limited, ISBN:1780172990.

IEEE Computer Society 1990, *IEEE Standard Glossary of Software Engineering Terminolo-*

gy, s. 60, Institute of Electrical & Electronics Enginee, ISBN:9781559370677.

IEEE Computer Society 2010, *Systems and software engineering - Vocabulary*, s. 11, 126, 367–368 ja 374–376, ISO/IEC/IEEE International Standard, ISO/IEC/IEEE 24765:2010(E).

IEEE Computer Society 2014, *Guide to the Software Engineering Body of Knowledge*, s. 4-1 ja 4-6, IEEE Computer Society Press, ISBN:0769551661.

ISO/IEC 2011, *Systems and Software Engineering – Systems and Software Quality Requirements and Evaluation (SQuaRE) – System and Software Quality Models*, s. 4 ja 13–14, BSI Standards Publication, ISBN:9780580702235.

Järvinen, P. & Järvinen, A. 2004, *Tutkimustyön metodeista*, s. 128–129, Opinpajan Kirja, ISBN:9529923325.

Kaner, C., Falk, J. & Nguyen H. 1999, *Testing Computer Software*, s. 43, TAB Books Inc., ISBN:0471358460.

Kanglin, L. & Mengqi, W. 2005, *Effective GUI Testing Automation: Developing an Automated GUI Testing Tool*, s. 4–5, Sybex, ISBN:9780782143515.

Kaur, M. & Kumari, R. 2011, *Comparative Study of Automated Testing Tools: TestComplete and QuickTest Pro*, s. 1, International Journal of Computer Applications (0975 — 8887) Volume 24 No.1.

Kelly, M. 2003, *Choosing a test automation framework*, s. 8–9, `https://www.ibm.com/developerworks/rational/library/591.html`, viitattu 25.7.2019.

Klarck, P. 2016, *How to write good test cases using Robot Framework*, `https://github.com/robotframework/HowToWriteGoodTestCases/blob/master/HowToWriteGoodTestCases.rst`, viitattu 18.4.2019.

Kolawa, A & Huizinga, D 2007, *Automated Defect Prevention: Best Practices in Software Management*, s. 74, Wiley-IEEE Computer Society Press, ISBN:0470042125.

Laukkanen, P. 2006, *Data-Driven and Keyword-Driven Test Automation Frameworks*, s. 2, 15, 23, 27, Helsinki University of Technology.

Leotta, M., Clerissi, D., Ricca, F. & Tonella, P. 2013, *Capture-Replay vs. Programmable Web Testing: An Empirical Assessment during Test Case Evolution*, s. 2–3, 2013 20th Working Conference on Reverse Engineering (WCRE), ISBN:9781479929313.

Leung, H. & White, L. 1989, *Insights into Regression Testing*, s.1, Proceedings. Conference on Software Maintenance - 1989, ISBN:0818619651.

Marick, B. 1998, *When Should a Test Be Automated?*, s. 8, Testing Foundations.

Maximilien, M. & Williams, L. 2003, *Assessing Test-Driven Development at IBM*, s. 5, 25th International Conference on Software Engineering, 2003 Proceedings, ISBN:076951877X.

McGregor, J. & Sykes, D. 2001, *A Practical Guide to Testing Object-oriented Software*, s. 98, Addison-Wesley Professional, ISBN:9780201325645.

Memon, A., Nagarajan, A. & Qing, X. 2005, *Automating regression testing for evolving GUI software*, s. 2–3, Journal of Software Maintenance and Evolution: Research and Practice - 2003 International Conference on Software Maintenance: The Architectural Evolution of Systems, ISSN:1532060X.

Meyer, B. 2008, *Seven Principles of Software Testing*, s.1, Computer Volume 41 Issue 8, DOI:10.1109/mc.2008.306.

Microsoft 2016a, *MFC Desktop Applications*, `https://docs.microsoft.com/en-us/cpp/mfc/mfc-desktop-applications?view=vs-2019`, viitattu 4.7.2019.

Microsoft 2016b, *Message Maps (MFC)*, `https://docs.microsoft.com/en-us/cpp/mfc/reference/message-maps-mfc?view=vs-2019`, viitattu 11.7.2019.

Microsoft 2018, *Named Pipes*, `https://docs.microsoft.com/en-us/windows/win32/ipc/named-pipes`, viitattu 11.7.2019.

Myers, G. 2004, *The Art of Software Testing, Second Edition*, s. 35, John Wiley & Sons, ISBN:9780471469124.

Neumann, R. & Günther S. & Zenker, N. 2009, *Reengineering deprecated component fra-*

meworks: A case study of the Microsoft Foundation Classes, s. 1–2, Wirtschaftsinformatik 2009, ISBN:9783834911346.

Osherove, R. 2009, *The Art of Unit Testing*, s. 4, Manning, ISBN:1933988274.

Pettichord, B. 2002 *Design for Testability*, s. 13, Pacific Northwest Software Quality Conference.

Petzold, C. 1999, *Programming Windows*, s. 11, Microsoft Press, ISBN:9781572319950.

Pressman, R. & Maxin, B. 2014, *Software Engineering: A Practitioner's Approach*, s. 489–490, McGraw Hill, ISBN:9780073655789.

Prosise, J. 2003, *Programming Windows with MFC*, s. 12, Microsoft Press, ISBN:9781572316959.

Rajkumar 2019, *Integration Testing – Big Bang, Top Down, Bottom Up & Hybrid Integration*, `https://www.softwaretestingmaterial.com/integration-testing/` viitattu 4.6.2019.

Rice, R. 2003, *Surviving the Top Ten Challenges of Software Test Automation*, s. 4–7, Rice Consulting Services Inc.

Robot Framework 2019, `https://robotframework.org/`, viitattu 8.8.2019.

Santanen, J. 2018a, *Ohjelmiston ja testauksen laatu*, s. 11, `https://moodle.jyu.fi/pluginfile.php/281099/mod_resource/content/1/OhjelmistonTestauksenLaatu2s.pdf`, viitattu 17.7.2019.

Santanen, J. 2018b, *Ohjelmistotestauksen tasot ja tyyppejä*, s. 3, 22, `https://moodle.jyu.fi/pluginfile.php/265849/mod_resource/content/2/OhjelmistotestauksenTasot2s.pdf`, viitattu 13.8.2019.

Stanislav, S.& Hocenski, Z. 2011, *Usage of Robot Framework in Automation of Functional Test Regression*, s. 1–2, ICSEA 2011: The Sixth International Conference on Software Engineering Advances.

Susman, G. & Evered, R. 1978, *An Assessment of the Scientific Merits of Action Research* s. 8, Administrative Science Quarterly, Dec 78, Vol. 23, Issue 4, ISSN:00018392.

Tsai, W., Bai, X., Paul, R., Shao, W. & Agarwal, V. 2001, *End-To-End Integration Testing Design*, s. 1, 25th Annual International Computer Software and Applications Conference. COMPSAC 2001, ISBN:0769513727.

Wagner, S. 2013, *Software Product Quality Control*, s. 10, Springer-Verlag Berlin Heidelberg, ISBN:9783642385704.

Wang, L. 2015, *GUI test automation for Qt application*, s. 9–12, Linköping University.