Jere Honka

Graafisten käyttöliittymien testaus ja testausviitekehykset

Tietotekniikan kandidaatintutkielma

12. joulukuuta 2017

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Jere Honka

Yhteystiedot: jemihonk@student.jyu.fi

Ohjaaja: Sanna Juutinen

Työn nimi: Graafisten käyttöliittymien testaus ja testausviitekehykset

Title in English: Graphical user interface testing and testing frameworks

Työ: Kandidaatintutkielma

Sivumäärä: 20+0

Tiivistelmä: Testiautomaation toteuttaminen on tärkeä vaihe ohjelmistokehityksessä. Sen hankaluus ja ajanvienti vaihtelevat paljon riippuen testaustyökalujen- ja viitekehyksen valinnasta. Erityisen hankalaa on toimivan testiautomaation toteuttaminen graafiselle käyttöliittymälle. Tämän tutkielman tarkoituksena on vertailla eri testausviitekehysten toimivuutta graafisen käyttöliittymän testauksessa. Tavoitteena on selvittää, mitä merkittäviä eroja testausviitekehyksillä on, ja mitä hyötyä ne tarjoavat.

Avainsanat: testausviitekehys, GUI-testaus

Abstract: Implementing test automation is an important phase in software development. Its difficulty and the amount of time required depend greatly on the chosen test automation tools and framework. Implementing a functioning test automation is particularly difficult for graphical user interfaces. The purpose of this thesis is to compare the usefulness of different testing frameworks in the testing of a graphical user interface. The goal is to find out what important differences the testing frameworks have, and what advantages they offer.

Keywords: testing framework, GUI-testing

Sisältö

1 Johdanto

Ohjelmistokehitys on usein pitkä ja monivaiheinen prosessi, jonka jokainen vaihe voi aiheuttaa ongelmia. Virheiden tunnistamisen ja ohjelmiston laadun varmistamisen kannalta on tärkeää, että ohjelmisto on hyvin testattu. Ohjelmistotestaajilla ei kuitenkaan usein ole tarpeeksi aikaa testata kehitettävää ohjelmistoa täysin. Testauksen laiminlyönti johtaa usein ohjelmiston toimitusajan viivästymiseen, mikä puolestaan johtaa joko pidempiin työtunteihin testaajien kohdalla, tai resurssien lisäämiseen testaustiimille (Cervantes, 2009), (Wang, 2015).

Ennen graafisten käyttöliittymien yleistymistä sovellukset olivat pääasiassa komentoriviltä käytettäviä. Komentorivikäyttöliittymissä käyttäjät kirjoittavat komentoja terminaaliin, ja tämän jälkeen järjestelmä suorittaa komentojen mukaisia toimintoja. Komentorivipohjaisista sovelluksista ollaan kuitenkin ajan myötä paljolti siirrytty graafisiin käyttöliittymiin. Graafinen käyttöliittymä eli GUI (Graphical User Interface) koostuu komponenteista, joita ovat esimerkiksi ikkunat, ikonit ja menut. Käyttäjät ovat vuorovaikutuksessa käyttöliittymän komponenttien kanssa hiirellä klikkailemalla ja näppäimiä painelemalla. Koska GUI-ohjelmien kohdalla käyttäjän ei yleensä tarvitse muistaa komentoja ulkoa niin kuin komentoriviohjelmien kohdalla, pidetään GUI-ohjelmia usein käyttäjäystävällisempinä (Kanglin, Mengqi, 2006). Käyttäjäystävällisyys on merkittävä etu GUI-käyttöliittymissä, ja se onkin todennäköisesti vaikuttanut GUI-käyttöliittymien yleistymiseen.

Automaattisen testaamisen eli testiautomaation toteuttamisesta on tullut hankalampaa sovellusten kehittyessä monimutkaisemmiksi vuosien saatossa. Erityisesti siirtyminen komentorivikäyttöliittymistä graafisiin käyttöliittymiin on tuonut monia haasteita testiautomaation toteuttamiselle. Graafisten käyttöliittymien testaus vie paljon aikaa, ja toimivien testitapausten toteuttaminen on hankalaa, koska GUI-testauksessa pitää ottaa paljon enemmän huomioon kuin komentorivikäyttöliittymän testauksessa.

Kun toteutetaan testiautomaatiota suurille, monista tuhansista riveistä koostuvil-

le sovelluksille, tulee usein ongelmaksi testien hallinnointi, sekä testien tarpeeksi helppo luominen ja ylläpito. Monimutkaisuutta tähän tuo lisää graafisten käyttöliittymien testauksen kohdalla usein myös itse testitapausten monimutkaisuus. Apua löytyy testausviitekehyksistä, joiden tarkoituksena on tarjota ratkaisuja näihin haasteisiin.Testausviitekehykset ovat systeemejä, jotka tekevät testitapausten luomisesta, suorittamisesta ja ylläpitämisestä helppoa. Niistä on apua kaikenlaisen testiautomaation toteuttamisessa, mutta tässä tutkielmassa keskitytään testausviitekehysten vertailuun erityisesti GUI-testauksen toteuttamisen apuna. Miten viitekehykset eroavat toisistaan, ja mitä apua niistä on graafisten käyttöliittymien testauksessa?

Seuraavassa luvussa käydään läpi testaamiseen liittyvää terminologiaa, joka on hyvä olla hallussa ennen kuin GUI-testaukseen ja viitekehyksiin perehdytään syvemmin. Luvussa 3 käsitellään GUI-testaukseen liittyviä käsitteitä, sekä sille uniikkeja ominaisuuksia. Luvussa 4 vertaillaan testausviitekehyksiä ja pohditaan, mitä hyötyä ne tarjoavat graafisten käyttöliittymien testaukseen. Lopuksi yhteenvetona käydään läpi, mitä tutkielmassa selvisi, sekä pohditaan GUI-testauksen ja viitekehysten tulevaisuutta.

2 Testausterminologiaa

Testausta on monenlaista, ja ennen syvempää perehtymistä testausviitekehyksiin ja graafisten käyttöliittymien testaukseen käydään tässä läpi tutkielman ymmärtämisen kannalta tärkeitä termejä ja käsitteitä.

2.1 Regressiotestaus

Regressiotestaus on testausprosessi, jota sovelletaan sen jälkeen, kun testattavaa ohjelmistoa on muokattu. Regressiotestauksessa tarkoituksena on varmistaa, että muutokset testattavassa ohjelmistossa eivät aiheuta ohjelmointivirheitä tai muita ongelmia. Toisin sanoen ne ominaisuudet ohjelmistossa, joihin ei ole tehty muutoksia, toimivat ohjelmiston muokkauksen jälkeen samalla tavalla kuin ennen muokkausta. Regressiotestaus on merkittävä komponentti ohjelmistokehityksen ylläpitovaiheessa, jolloin esimerkiksi ohjelmiston suorituskykyä pyritään parantelemaan ja sen ohjelmointivirheitä korjailemaan (Wong, Horgan, London, Agrawal, 1997), (Leung, White, 1989).

Graafisten käyttöliittymien regressiotestauksessa on monia haasteita. Pienetkin muutokset käyttöliittymään, kuten nappien siirtely ja menujen uudelleenorganisointi voivat johtaa siihen, että testin toteutusta tarvitsee muokata (Memon, 2002). Tästä voidaan päätellä, että regressiotestauksen kannalta on hyödyllistä, jos testitapauksen toteutus ja testidata saadaan pidettyä erillään.

2.2 Manuaalitestaus

Manuaalitestaus on testausta, jonka testaajat toteuttavat ilman automaatiota. Tämäntyyppisessä testauksessa testaaja asettuu loppukäyttäjän rooliin, ja pyrkii mahdollisimman perusteellisesti käyttämään testattavan ohjelmiston ominaisuuksia ja varmistamaan, että testattavat ominaisuudet toimivat odotetulla tavalla. Tämän kaltaisessa testauksessa testitapaukset ovat luonnollisella kielellä kirjoitettuja askellet-

tuja ohjeita, joita testaaja pyrkii ohjelmistoa testatessaan noudattamaan (Kaur, Kumari, 2011).

Manuaalitestaus on paljon aikaa vievä prosessi, ja automaation puutteen takia testejä ei voi uudelleenkäyttää. Lisäksi koska testejä suorittaa ihminen eikä kone, niin jotkut viat saattavat jäädä huomaamatta inhimillisten virheiden vuoksi (Kaur, Kumari, 2011).

3 GUI-testaus

Tässä luvussa käsitellään GUI-testaukseen liittyviä käsitteitä, ja sille uniikkeja ominaisuuksia verrattuna esimerkiksi komentorivipohjaisen sovelluksen testaukseen. Kuvailen myös GUI-testauksen toteutuksessa hyödynnettäviä tekniikoita.

3.1 GUI-testauksen uniikkeja ominaisuuksia

Testiautomaatiossa käytetään hyödyksi testiskriptejä, eli komentosarjoja, jotka suoritetaan testattavassa järjestelmässä. Näillä skripteillä tarkistetaan, että testattava järjestelmä toimii oletetusti. Komentoriviltä ajettavien ohjelmien automaattinen testaus hoidetaan usein sellaisilla testiskripteillä, jotka sisältävät komentoriviltä ajettavia komentoja. Näiden komentojen suorittaminen ei riipu näytön nykyisestä tilasta, mikä on helpottava tekijä testin toteutuksen kannalta. GUI-komponenttien automaattinen testaus puolestaan on hankalampaa, sillä näytön tulee olla sopivassa tilassa testiskriptiä ajettaessa, ja testiskriptin tulee usein pystyä hiiren siirtelemiseen ja näppäimistön näppäinten painelemiseen oikeissa tilanteissa. Skriptin tulee myös pystyä lukemaan testattavan sovelluksen tilaa, jotta tätä tilaa voidaan verrata odotettuun tilaan (Kanglin, Mengqi, 2006). Tällä tavalla selvitetään, toimiiko käyttöliittymä odotetulla tavalla.

Monet GUI-testitapaukset jätetään automatisoimatta, mikä tarkoittaa, että ne toteutetaan vain manuaalitesteinä (Kanglin, Mengqi, 2006). Tämä johtunee siitä, että GUI-testien automatisointia pidetään kalliina operaationa, koska automaattisten GUI-testien ylläpito vaatii usein muita testejä enemmän vaivaa.

3.2 Nauhoita ja toista -testaus

Nauhoita ja toista -testaus on yleinen GUI-testauksen muoto. Tätä tekniikkaa hyödynnettäessä testaaja on vuorovaikutuksessa käyttöliittymän kanssa hiirtä ja näppäimistöä käyttäen. Testaajan tekemiset nauhoitetaan testiskriptiksi, ja tämä nau-

hoitus myöhemmin ajettuna suorittaa samat klikkailut ja näppäilyt käyttöliittymässä kuin testaaja, jota nauhoitettiin. Luotua nauhoitusta voi käyttää automatisoituna testinä käyttöliittymälle, joten kyseessä on manuaalisen testin automatisointi nauhoittamalla (Kanglin, Mengqi, 2006) (Wang, 2015).

Tämä lähestymistapa testaukseen sisältää monia ongelmia. Pienetkin muutokset testattavaan käyttöliittymään johtavat lähes aina nauhoitetun testiskriptin hajoamiseen, ja kun skriptiä joudutaan päivittämään, on usein hankala päätellä, mitä osia skriptistä on muutettava ja mitä ei (Wang, 2015). Nauhoita ja toista -testaustyökaluilla luotavat testiskriptit ovat siis vaikeasti ylläpidettäviä, mikä on merkittävä huono puoli.

3.3 Malliin perustuva testaus

Malliin perustuvassa testauksessa ideana on luoda malli, joka kuvaa testattavan graafisen käyttöliittymän toimintaa. Tälle mallille voidaan sen luomisen jälkeen generoida testitapauksia automaattisesti tapahtumasarjoina hyödyntämällä verkkojen läpikäyntialgoritmeja. Kun kaikki testitapaukset on generoitu, ne voidaan suorittaa käyttäen automatisoitua testien suoritustyökalua (Bae, Rothermel, Bae, 2014). Testitapausten automaattinen generointi vähentää testaajalta vaadittavaa työmäärää ja helpottaa ylläpitoa.

Malliin perustuvilla testauslähestymistavoilla on etuna se, että niillä pystytään generoimaan ytimekkäitä testitapauksia jättämällä mallista joitain alueita käymättä läpi tai käyttämällä jotain läpikäyntiehtoa. Huonona puolena on puolestaan se, että koska malli kuvaa testattavaa käyttöliittymää abstraktissa muodossa, eivät sille luodut testitapaukset välttämättä toimi riittävän hyvinä testeinä testattavan käyttöliittymän varsinaiselle toiminnalle (Bae, Rothermel, Bae, 2014).

3.4 Automaattisen GUI-testauksen kannattavuus

Automaattista GUI-testausta suunniteltaessa tulee aluksi määritellä, mitä osia käyttöliittymästä on kannattavaa testata automaattisesti. GUI-testaaminen on monimutkaista, ja GUI-testien suunnittelu vie usein huomattavasti enemmän aikaa kuin esimerkiksi komentorivipohjaisen ohjelman testien.

Automaattisen GUI-testitapauksen toteuttaminen on merkittävästi työläämpää kuin sisällöltään vastaavan manuaalitestitapauksen. Automaattitestiä luovalta testaajalta vaadittava taito- ja kokemustaso ovat myös korkeampia kuin manuaalitestiä luovalta testaajalta (Wang, 2015). Ajankäyttöön vaikuttaa edellä mainittujen tekijöiden lisäksi se, käytetäänkö testiskriptin toteutuksessa nauhoita ja toista -testaustyökalua vai kirjoitetaanko skripti ilman sellaista. Nauhoita ja toista -työkalun käyttö on usein näistä vaihtoehdoista helpompi (Marick, 1998). On hyvä miettiä, kuinka tärkeä testattava toiminnallisuus on käyttöliittymässä ennen automaattitestin toteuttamista, sillä jos testattavaa toiminnallisuutta käytetään vain harvoin, voi manuaalitesti olla parempi vaihtoehto ajankäytön kannalta (Marick, 1998).

Tiettyjen testitapausten suorittaminen ei ole mahdollista ihmiselle, kuten esimerkiksi testitapaukset, jotka vaativat useampien operaatioiden suorittamista samanaikaisesti. Myös testitapaukset, joissa samoja vaiheita toistetaan monia kertoja, sopivat paremmin automaattitestaukseen kuin manuaalitestaukseen (Wang, 2015). Monessa tapauksessa testiautomaation toteuttaminen on siis haastavuudestaan huolimatta kannattavaa ja tärkeää.

GUI-testauksen tapauksessa kaikkein yleisin syy testiautomaation toteuttamiselle on regressiotestaus. Kun käyttöliittymästä valmistuu uusi versio, varmistetaan regressiotesteillä, etteivät uuden version tuomat muutokset vaikuta ohjelmiston suorituskykyyn haitallisesti tai hajota jotain osaa käyttöliittymästä. Mitä useammin tiettyä testiskriptiä päästään hyödyntämään regressiotestauksessa, sitä hyödyllisempi kyseinen skripti on testiautomaatioprojektin kannalta (Wang, 2015).

4 Viitekehyksiä automaattisen GUI-testauksen toteuttamiselle

Testiautomaatiota toteutettaessa monimutkaiselle järjestelmälle, kuten kookkaan sovelluksen graafiselle käyttöliittymälle, tulee nopeasti tarvetta systeemille, joka tekee testitapausten luomisesta, suorittamisesta ja ylläpitämisestä helppoa. Tällaisia systeemejä kutsutaan testausviitekehyksiksi. Seuraavaksi käydään läpi testausviitekehyksen ideaa, kehitystä ja rakennetta, sekä kuvaillaan eri viitekehyksiä. Vertailen viitekehysten ominaisuuksia ja hyötyjä erityisesti graafisten käyttöliittymien testauksen näkökulmasta.

4.1 Testausviitekehysten historiaa

Testausviitekehykset ovat kehittyneet paljon vuosien aikana. Viitekehysten kehitysvaiheet voidaan jakaa kolmeen sukupolveen (Kit, 1999).

Ensimmäisen sukupolven viitekehykset ovat rakenteeltaan yksinkertaisia, ja niissä testidata on liitetty testiskripteihin. Lisäksi testiskriptien määrä usein vastaa testitapausten määrää. Kun näitä viitekehyksiä käytetään GUI-testauksessa, testiskriptejä usein generoidaan nauhoita ja toista -työkaluja hyödyntäen. Tämä johtaa testien ylläpidon haastavuuteen, sillä pienetkin muutokset testattavassa sovelluksessa vaativat testiskriptien uudelleentoteuttamista (Kit, 1999) (Laukkanen, 2006). Ensimmäisen sukupolven viitekehykset eivät hankalan ylläpidettävyytensä sovellu monimutkaisten graafisten käyttöliittymien testaamiseen.

Toisen sukupolven viitekehyksissä skriptit ovat modulaarisia ja joustavia. Skripteissä hoidetaan testin suorittamisen lisäksi myös testiympäristön alustus ja puhdistaminen. Testidata sisällytetään skripteihin kuten ensimmäisen sukupolven viitekehyksissä, ja testitapausten toteuttaminen sekä ylläpitäminen vaativat molemmat ohjelmointitaitoja, joita kaikilla testaajilla ei välttämättä ole. Tämän sukupolven testausviitekehysten on todettu soveltuvan melko hyvin GUI-testaukseen, vaikka on-

gelmia aiheutuukin siitä, että testien suunnittelua ja toteutusta ei tehokkaasti eroteta toisistaan (Kit, 1999), (Laukkanen, 2006). Testiskriptien modulaarisuus helpottaa ylläpitoa, mutta kun testattavaan ohjelmistoon tulee merkittävämpiä muutoksia, joudutaan testiskriptejä siitä huolimatta mahdollisesti muokkaamaan paljon.

Kolmannnen sukupolven viitekehyksillä on kaikki toisen sukupolven viitekehyksien hyvät puolet, ja paljon enemmän. Näissä viitekehyksissä testidata irrotetaan testiskripteistä, mistä saadaan se hyöty, että useampia erilaisia testitapauksia voidaan suorittaa muuttamalla vain testidataa ja jättäen testiskripti koskemattomaksi. Lisäksi testien suunnittelu ja toteutus saadaan erotettua, jolloin suunnittelua voi tehdä henkilö, jolla ei ole ohjelmointitaitoja ja toteutuksen henkilö, jolla ohjelmointitaitoja on (Laukkanen, 2006).

4.2 Modulaarinen ja kirjastoihin perustuva testaus

Yleinen testiautomaation työkalu on lineaariset testiskriptit. Lineaariset testiskriptit ovat skriptejä, jotka eivät käytä ulkoisia funktioita ja ovat suorassa vuorovaikutuksessa testattavan ohjelmiston kanssa. Näiden skriptien kirjoittaminen on usein nopeaa pienten toimintojen testaamiselle. Lineaarisissa testiskripteissä on kuitenkin se ongelma, että testattavien toiminnallisuuksien muuttuessa monimutkaisemmiksi myös testiskriptit monimutkaistuvat, ja skriptien ylläpitäminen sekä toteuttaminen hankaloituu (Laukkanen, 2006). Tähän ongelmaan tarjoaa apua modulaarinen ja testikirjastoihin perustuva testauslähestymistapa.

Hyödynnettäessä modulaarista testausviitekehystä pyritään kirjoittamaan pieniä, itsenäisiä testiskriptejä, jotka kuvaavat testattavan ohjelmiston osioita, moduuleja sekä funktioita. Näistä pienistä skripteistä rakennetaan hierarkkiseen tyyliin suurempia skriptejä, joilla kuvataan testitapauksia (Kelly, 2003). Toinen lähestymistapa, joka muistuttaa paljon modulaarista testausta, on kirjastoihin perustuva testaus. Tässä viitekehyksessä testattava ohjelmisto jaetaan proseduureihin ja funktioihin skriptien sijaan. Tämän jälkeen luodaan testikirjastoja, jotka kuvaavat testattavan ohjelmiston moduuleja, osioita ja funktioita. Kirjastoihin perustuva lähestymistapa

tarjoaa samat edut kuin modulaarinen lähestymistapa (Kelly, 2003).

Näiden testausviitekehysten hyötynä on se, että kun esimerkiksi ohjelmiston GUI:lle saadaan luotua tarpeeksi itsenäisiä, helppokäyttöisiä ja uudelleenkäytettäviä testiskriptejä, nopeutuu monimutkaisempien testitapausten toteuttaminen huomattavasti (Laukkanen, 2006). Myös ylläpitäminen helpottuu, sillä kun jokin testattavassa ohjelmistossa muuttuu, niin esimerkiksi kirjastoihin perustuvassa testauksessa vain yhden funktion muuttaminen testikirjastossa saattaa riittää ongelman korjaamiseen. Pahimmassakin tapauksessa tarvittavien muutosten tulisi kohdistua vain testikirjastoon (Laukkanen, 2006).

Testikirjastojen ja -moduulien haasteena on niiden luominen, mikä ei ole helppoa (Laukkanen, 2006). Luotujen skriptien dokumentoinnin, nimeämiskäytäntöjen sekä säilyttämisen hallinnointi vaatii paljon resursseja (Fewster, Graham, 1999). Testitapausten toteuttaminen ja ylläpito vaativat testaajalta ohjelmointitaitoja, sekä testiskriptien toiminnan ymmärrystä. Lisäksi kun testausviitekehys koostuu tässä tapauksessa vain joko testiskripteistä tai testikirjastoista, niin myös testidata sisällytetään skripteihin. Tämä johtaa siihen, että skriptejä täytyy päivittää silloinkin, kun vain dataan tarvitaan muutoksia. Koska testidata on osa testiskriptiä, kuuluu modulaarinen ja kirjastoihin perustuva testaus edellä esitellyssä sukupolvijaossa toisen sukupolven testausviitekehyksiin.

4.3 Aineisto-ohjattu testaus

Aineisto-ohjatussa testausviitekehyksessä ideana on testidatan ja testin toteutuksen pitäminen erillään toisistaan. Testidata sisältää testin sisääntuloarvot, sekä odotetut ulostuloarvot, joihin testin ulostuloa verrataan. Yleisiä tiedostomuotoja testidatan säilyttämiselle ovat esimerkiksi CSV- ja TSV -tiedostot, joita pystyy lukemaan monilla taulukkolaskentaohjelmilla. Dataa voidaan lukea myös monimutkaisemmista tietorakenteista, kuten tietokantatauluista (Kelly, 2003), (Laukkanen, 2006). Testiskriptit tulee toteuttaa siten, että skripti toimii odotetulla tavalla – ilman koodimuutoksia – vaikka testidatasettiä vaihdettaisiin (Pillai, 2012). Tämän testausviitekehyk-

sen on todettu vähentävän tarvittavien testiskriptien kokonaismäärää, ja se tarjoaa joustavuutta testaamiseen, sillä testitapausten lisääminen ja muuttaminen ei vaadi muutoksia testiskriptiin (Laukkanen, 2006).

Tämä testausviitekehys tarjoaa monia etuja testaukseen. Näistä suurin on se, että testitapausten luominen ja ajaminen on huomattavasti helpompaa verrattuna lähestymistapaan, jossa testidata sijaitsee testitapauksen toteuttavassa testiskriptissä. Testidatan pitäminen testin toteutuksen ulkopuolella mahdollistaa myös sen, että testitapausten muokkaaminen ei välttämättä vaadi lainkaan ohjelmointitaitoja (Laukkanen, 2006). Testitapauksien dataa voidaan luoda ennen automaattitestauksen toteuttamista, jolloin sitä voi käyttää esimerkiksi manuaalitestaukseen.

Aineisto-ohjattu testausviitekehys mainitaan monissa GUI-testiautomaatioprojekteissa (Wang, 2015). Monet edellä mainituista eduista ovatkin merkittäviä graafisten käyttöliittymien testauksessa. GUI-testiautomaatiossa joudutaan usein luomaan monimutkaisia testiskriptejä, joiden kunnollinen hahmottaminen vaatii perehtymistä, vaikka itse testattava GUI ei olisikaan monimutkainen. Testidatan pitäminen testiskriptien ulkopuolella on hyödyllistä, koska muutokset käyttöliittymään eivät välttämättä vaadi muutoksia testidataan, vaan ainoastaan testiskriptiin. Aineisto-ohjattu testausviitekehys kuuluu kolmannen sukupolven testausviitekehyksiin, koska testidataa ja testiskriptiä pyritään pitämään erillään.

4.4 Avainsanaohjattu testaus

Edellä mainitulla aineisto-ohjatulla testausviitekehyksellä on paljon lupaavia ominaisuuksia, mutta myös hankalia rajoituksia. Isoimpia rajoituksia on se, että monipuolisten testitapausten tekeminen on hankalaa, ja täysin uudenlaisten testien luominen vaatii ohjelmointityötä (Laukkanen, 2006). Näihin ongelmiin löytyy ratkaisuja avainsanaohjatusta testauksesta, jossa testidatan lisäksi testiskriptistä erotetaan myös dataa käsittelevä logiikka. Dataa käsitteleviä palikoita kutsutaan avainsanoiksi (Wang, 2015). Avainsanoihin kapseloidaan testattavan operaation toteutus. Avainsanaohjatussa testauksessa testattavan ohjelmiston toiminnallisuus dokumen-

toidaan testitiedostoon, ja tähän tiedostoon laitetaan edellä kuvailtuja avainsanoja, jotka toimivat askellettuina ohjeina testille. Avainsanaohjatut testitapaukset näyttävät usein hyvin samankaltaisilta, kuin manuaaliset testitapaukset (Kelly, 2003). Vaikka avainsanaohjattu testaus eroaakin monin tavoin aineisto-ohjatusta, niin perusidea, eli testidatan lukeminen erillisestä tiedostosta, pysyy samana (Laukkanen, 2006). Avainsanaohjattua testausta pidetäänkin loogisena laajennuksena aineisto-ohjatulle testaamiselle (Fewster, Graham, 1999).

Testitapausten vaiheiden kapseloiminen avainsanoiksi helpottaa testaajan työtä huomattavasti. Kun testattavalle sovellukselle on luotu kattava avainsanakirjasto, voi monimutkaisiakin testejä usein luoda henkilö, jolla ei ole kokemusta ohjelmoinnista (Wang, 2015). Tämä on rajoitetusti mahdollista aineisto-ohjatussakin viitekehyksessä, mutta avainsanaohjattu viitekehys laajentaa mahdollisuuksia huomattavasti. Avainsanat jakavat testiskriptin useisiin itsenäisiin osioihin, ja näitä osioita voidaan ylläpitää itsenäisesti ilman, että muiden avainsanojen toiminta muuttuu. Tämä ominaisuus parantaa uudelleenkäytettävyyttä sekä tekee ylläpidosta helpompaa (Wang, 2015).

Avainsanaohjattu testausviitekehys tarjoaa automaattiseen GUI-testaukseen hyödyt, joita saadaan aineisto-ohjatusta testauksesta, sekä paljon enemmän. Tarvittavien testiskriptien määrä riippuu testiautomaation skaalasta, mutta ei testitapausten määrästä, sillä testitapauksia voidaan lisätä käyttämällä eri avainsanoja uuden koodin sijasta (Wang, 2015). Avainsanaohjattu testaus myös vähentää testiskriptin ylläpidon hintaa sekä nopeuttaa testitapauksen toteuttamista (Kaner, Bach, Pettichord, 2001). Koska uusia testejä on mahdollista luoda käyttämällä pelkästään jo olemassa olevia avainsanoja, ei testaaminen välttämättä vaadi lainkaan ohjelmointitaitoja.

Avainsanaohjatussa testausviitekehyksessä on myös haasteita. Suurimpana haasteena voidaan pitää sen toteuttamista, joka vaatii paljon testaamiskokemusta sekä ohjelmointitaitoja (Fewster, Graham, 1999). Kattavan avainsanakirjaston luominen vaatii paljon aikaa, ja kirjaston on oltava korkealaatuinen, jotta viitekehyksestä saadaan kaikki hyöty irti (Wang, 2015).

Avainsanaohjattu testausviitekehys kuuluu kolmannen sukupolven testausviitekehyksiin, ja sitä voidaan pitää kehittyneempänä versiona aineisto-ohjatusta viitekehyksestä. Tämä viitekehys ei kuitenkaan välttämättä ole kaikissa tilanteissa parempi vaihtoehto kuin aineisto-ohjattu viitekehys edellä mainittujen haasteiden vuoksi. Voi olla tilanteita, joissa testiautomaation skaala ei ole riittävän suuri oikeuttamaan sitä työmäärää, mitä avainsanaohjatun testausviitekehyksen toteuttaminen vaatii.

5 Yhteenveto

Tässä tutkielmassa selvitettiin, miten testausviitekehykset eroavat toisistaan, ja mitä apua niistä on graafisten käyttöliittymien testauksessa. Tätä tarkoitusta varten aluksi keskityttiin GUI-testauksen tekniikoihin ja sen haasteisiin, ja sitten tutkittiin testausviitekehyksiä ja niiden tarjoamia ratkaisuja näihin haasteisiin.

Testausviitekehyksiä vertailtaessa oli nähtävissä selkeää kehitystä eri sukupolvien välillä. Pelkkiä lineaarisia testiskriptejä hyödyntävistä viitekehyksistä on edetty monipuoliseen, avainsanoja hyödyntävään viitekehykseen, jossa testitapausten luominen ei välttämättä vaadi lainkaan ohjelmointitaitoja. Tutkielmassa selvisi, että erityisesti testitapausten monimutkaisten vaiheiden abstrahointi tarjoaa huomattavia etuja testaajan työhön, sillä testien luominen ja ylläpito helpottuu paljon.

Verrattavista testausviitekehyksistä avainsanaohjattu lähestymistapa oli tutkimani perusteella paras viitekehys monimutkaisten graafisten käyttöliittymien testaamiseen, sillä se tarjoaa parhaiten ratkaisuja GUI-testauksen ongelmiin. GUI-testauksen tekniikoita ja haasteita tutkittaessa selvisi, että GUI-testitapauksia toteuttavat testiskriptit ovat usein monimutkaisia, ja hajoavat helposti pienistä muutoksista. Koska avainsanaohjatussa viitekehyksessä testin toteutuksesta saadaan erotettua testidata ja sitä käsittelevää logiikkaa, vähenee käyttöliittymään tulevien muutoksien takia tarvittavan refaktoroinnin määrä huomattavasti, sillä dataan ja avainsanoihin ei välttämättä tarvitse koskea testin toteutusta korjattaessa.

Avainsanaohjattu viitekehys ei kuitenkaan ole ylivoimainen verrattuna esimerkiksi aineisto-ohjattuun viitekehykseen, sillä kuten tutkielmassa selvisi, avainsanakirjaston luomisessa vaadittava työmäärä on suuri. Tästä voidaan päätellä, että yksinkertaisempien graafisten käyttöliittymien kohdalla helpommin toteutettava aineisto-ohjattu viitekehys saattaa olla parempi vaihtoehto. Aineisto-ohjattua viitekehystä yksinkertaisemmille viitekehyksille puolestaan ei vertailun perusteella ole paljoa käyttöä, sillä testidatan ja testin toteutuksen erillään pitämisestä saadut hyödyt ovat merkittäviä GUI-testauksessa.

GUI-testauksessa ja testausviitekehyksissä on yhä paljon tutkittavaa. GUI-testaus on monimutkainen aihe, sillä sen toteuttamiselle ei ole vielä muodostunut mitään yksinkertaista ja helppoa tapaa, ja se on kohdannut uusia haasteita siirryttäessä perinteisistä hiirellä ja näppäimistöllä manipuloitavista käyttöliittymistä kohti koskетusnäytöllä ja jopa silmillä sekä puheella ohjattavia käyttöliittymiä. Uusia, entistä parempia testausviitekehyksiä pyritään yhä kehittämään, ja niitä kehitettäessä luonnollisesti tutkitaan vanhojen viitekehysten onnistumisia sekä puutteita.

Tässä tutkielmassa vertailtiin viitekehyksiä teoreettisella tasolla. Tutkittavaa riittää myös viitekehysten toteuttamisessa. Esimerkiksi avainsanapohjaisen viitekehyksen toteuttavien työkalujen väliltä löytyy paljon eroja siinä, kuinka tarkkaan viitekehyksen ideaa noudatetaan. On myös monia työkaluja, joissa viitekehysten ominaisuuksia yhdistellään.

Kirjallisuutta

Bae, G., Rothermel, G. & Bae, D. 2014. *Comparing model-based and dynamic event-extraction based GUI testing techniques: An empirical study* Journal of Systems and Software, Volume 97, November 2014, Pages 15-46

Cervantes, A. 2009. *Exploring the use of a test automation framework*. Aerospace conference, 2009 IEEE ISSN: 1095-323X

Fewster, M. & Graham, D. 1999. *Software Test Automation*. Addison-Wesley, 1999. s. 517-535

Kaner, C., Bach, J. & Pettichord, B. 2001. *Lessons Learned in Software Testing: A Context-Driven Approach*. John Wiley & Sons, New York, 2001.

Kanglin, L. & Mengqi, W. 2006. *Effective GUI Testing Automation: Developing an Automated GUI Testing Tool*. s. 4-5 ISBN: 0782150675, 9780782150674

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

Kelly, M. 2003. *Choosing a test automation framework*. URL: https://www.ibm.com/developerworks/rational/library/591.html. Viitattu 13.11.2017.

Kit, E. 1999. *Integrated, effective test design and automation*. Software Development, pages 27–41.

Laukkanen, P. 2006. *Data-Driven and Keyword-Driven Test Automation Frameworks*. Master's Thesis, Helsinki University Of Technology

Leung, H. & White, L. 1989. *Insights into Regression Testing*. Proceedings. Conference on Software Maintenance 1989

Marick, B. 1998. *When Should a Test Be Automated?* Proc. 11th Int'l Software/Internet Quality Week

Memon, A. *GUI testing: Pitfalls and process* Computer, Volume 35, Issue 8, Aug 2002

Pillai, A. 2012. *Data Driven Framework in QTP : The Complete Guide – Part 1* URL: http://www.automationrepository.com/2012/05/qtp-data-driven-framework-design-with-examples/. Viitattu 6.11.2017

Wang, L. 2015. *GUI test automation for Qt application*. Final Thesis, Department of Computer and Information Science, Linköping University

Wong, W., Horgan, J., London, S., & Agrawal, H. 1997. *A study of effective regression testing in practice* Proceedings, The 8th IEEE International Symposium on Software Reliability Engineering