Heikki-Jussi Niemi

Graafisen käyttöliittymän automaattinen testaus

Tietotekniikan
kandidaatintutkielma
28. kesäkuuta 2012

Jyväskylän yliopisto

Tietotekniikan laitos

Jyväskylä

Tekijä: Heikki-Jussi Niemi
Yhteystiedot: hekku2@gmail.com
Työn nimi: Graafisen käyttöliittymän automaattinen testaus
Title in English: Automatic testing of graphical user interfaces
Työ: Tietotekniikan kandidaatintutkielma
Sivumäärä: 50
Tiivistelmä: Tässä kandidaatintutkielmassa tutkitaan graafisten käyttöliittymien automaattista testaamista. Pyritään selvittämään, mitä työkaluja tähän on tarjolla ja miten niiden käyttäminen eroaa toisistaan, sekä mitä elementtejä käyttöliittymästä näillä voidaan testata.
English abstract: Aim of this research is to find out what kind of tools there are for automating graphical user interface testing, specially in Java.
Avainsanat: graphical user interface, automatic testing, java, swing
Keywords: graafinen käyttöliittymä, automaattinen testaus, java, swing

Sanasto

GUI Graafinen käyttöliittymä

JFC Java Foundation Classes, kokoelma luokkia, jotka mahdollistavat esimerkiksi graafisten käyttöliittymien tekemisen Javalla.

TTD Test Driven Development, Testilähtöinen kehitys

Swing Javakirjasto graafisten käyttöliittymien toteuttamiseen

Lift The Library for InterFace Testing, työkalu graafisten käyttöliittymien automaattiseen testaukseen.

Fest Fixtures for Easy Software Testing, työkalu graafisten käyttöliittymien automaattiseen testaukseen.

CSS Cascading Style Sheets, yksi tapa määritellä WWW-sivuille tyyliohjeita.

URL Uniform Resource Locator, Osoite, joka määrittelee tietyn tiedon paikan.

Sisältö

1 Johdanto

Useimmille tietokoneen käyttäjille graafiset käyttöliittymät ovat arkipäivää ja koska käyttöliittymät ovat iso osa ohjelmakoodia, on näitä testattava ohjelman toimivuuden takaamiseksi [2, s. 1]. Esimerkiksi Microsoftin Paint-ohjelmassa [20] on useita kymmeniä käyttöliittymäelementtejä, puhumuttakaan monimutkaisemmista ohjelmista, joissa näitä voi olla jopa satoja. Käyttöliittymäelementtien määrästä voidaan päätellä, että näiden manuaaliseen testaamiseen kuluu aikaa. Jos manuaaliselle testaamiselle olisi varteenotettava, helppokäyttöinen vaihtoehto, millä saataisiin tehtyä automaattiset testit, saataisiin ohjelmasta mahdollisesti enemmän virheitä kiinni ja näin parempi ohjelma.

Tämän tutkimuksen tarkoituksena on selvittää, miten käyttöliittymän testaamista voidaan automatisoida Java-ympäristössä ja mitä työkaluja tähän on tarjolla. Tutkimuksessa keskitytään työkaluihin ja näiden eroihin, eli testikattavuuteen liittyvät ongelmat ja testauksen teoria sivuutetaan.

Varsinaisen vertailu suoritetaan tekemällä graafinen käyttöliittymä, joka sisältää yleisiä graafisten käyttöliittymien perusominaisuuksia, kuten nappuloita ja tekstikenttiä. Tälle käyttöliittymälle määrittellään testattavat asiat ja tehdään jokaisella testaustyökalulla testit, jotka testaavat määritellyt asiat. Tämän jälkeen vertaillaan syntyneitä testejä ja mitä näiden testien tekeminen vaati testaajalta.

Testien avulla pyritään löytämään helppokäyttöisin työkalu, jota koitetaan soveltaa Jyväskylän Yliopiston Tietotekniikan Laitoksen Ohjelmointi 1 ja 2-kursseilla käytössä olevaan ComTest-testityökaluun [18]. Jos soveltaminen onnistuu, voidaan graafisten käyttöliittymien testaamista mahdollisesti käyttää kurssilla.

Toisessa luvussa käsitellään automaattista testausta yleisesti, miksi siitä on hyötyä, mitä huonoja puolia siinä on ja mihin sitä voidaan käyttää. Kolmannessa luvussa esitellään graafisten käyttöliittymien automaattista testausta, miten sitä tehdään ja mitä ongelmia siinä on. Neljännessä luvussa esitellään testattavan käyttöliittymän luomiseen käytettyjä työkaluja sekä luotu käyttöliittymä ja sen toiminnallisuus. Viidennessä luvussa käsitellään testattavat työkalut, sekä niiden vertailu. Viimeisessä luvussa käsittellään tutkimuksen johtopäätökset.

Tutkimuksessa selvisi, että työkaluja on useita, ja ne voidaan jakaa ainakin kah-

teen tyyppiin, nauhoittaviin ja ohjelmoitaviin. Kaikki vertailtavat työkalut toimivat kuitenkin samankaltaisesti, ja isoja eroja ei esiintynyt, pieniä yksinkertaistuksia lukuunottamatta. Comtestiin sovellettua työkalua päästiin myös osittain testaamaan opetuskäytössä, mutta pienen otannan vuoksi tästä ei voida juuri vetää johtopäätöksiä.

2 Automaattinen testaus ja testilähtöinen kehitys

Tässä luvussa käsitellään automaattista testausta ja testilähtöistä kehitystä tutkimuksen kannalta olennaisin osin. Automaattinen testaus on menetelmä, jolla pyritään varmistamaan ohjelman toimivuus ja se, että ohjelmaa voidaan turvallisesti muokata, koska mahdolliset rikkoutuneet asiat saadaan selville heti muutosten jälkeen. Testeissä ohjelmalle kehitetään erilaisia syötteitä ja niitä vastaavia tuloksia ja nämä syötteet syötetään ohjelmalle. Sen jälkeen testiohjelma vertaa testattavan ohjelman antamia tuloksia testaajan antamiin odotettuihin tuloksiin. Jos tulokset eivät täsmää, on automaattinen testaus havainnut virheen testattavasta ohjelmasta. Tämä ei kuitenkaan vielä takaa, että ohjelmassa on virhe, vaan tämä voi myös tarkoittaa, että testit ovat virheellisesti laadittu.

Testauksen automaattisuuden taso vaihtelee kuitenkin paljon. Yksinkertaisimmillaan testit luodaan syötteineen itse ja ne ajetaan erikseen testaajan pyynnöstä, kun taas toiselle työkalulle määritellään tietyt lainalaisuudet mitä aliohjelman palautusarvon pitää toteuttaa ja sen jälkeen työkalu generoi itse suuren määrän satunnaisia testisyötteitä ja tarkastaa, toteutuuko aiemmin määritelty lainalaisuus kaikilla syötteillä [19]. Myös testien ajaminen voidaan automatisoida; testit voidaan esimerkiksi ajaa automaattisesti, kun lähdekoodit laitetaan versionhallintaan.

Automaattisen testauksen etuna on, että testit voidaan toistaa pienellä vaivalla. Näin voidaan tehtyjen muutosten jälkeen ajaa iso joukko valmiiksi tehtyjä testejä, jotka varmistavat, että ohjelman osat toimivat myös muutosten jälkeen. Manuaalisella testauksella testatut asiat jäisivät todennäköisesti vähäisemmiksi ja virheiden todennäköisyys kasvaisi. Automaattiset testit eivät kuitenkaan ole ongelmattomia.

Ongelmallista automaattisessa testauksessa on, että testejä on työlästä määritellä ja se, minkälaisia testien pitäisi olla, ei ole itsestäänselvää. Myöskin testien ajaminen voi viedä pitkään, erityisesti jos testataan asioita, joihinka liittyy ajastuksia tai odottealua. Esimerkiksi tällainen tilanne voisi tulla vastaan, jos pitäisi varmistaa, että

ohjelmisto palautuu virheestä, tai odottaa jotain tiettyä tapahtumaa vain tietyn ajan.

Ongelmallista on myös, miten testata esimerkiksi operaatioita, jotka vaativat tietokannan käyttöä. Näihin, ja muihin samankaltaisiin ongelmiin on kehitetty yhdeksi ratkaisuksi mock-objektien käyttö. Mock-objektit ovat olioita, jotka imitoivat jonkin toisen olion käytöstä, toteuttamatta kuitenkaan sisäistä toteutusta [5]. Esimerkiksi voitaisiin haluta testata kalenteriohjelmaa, joka hakee tapahtumatietokannasta tapahtumia, siivilöi pois menneet tapahtumat. Ongelmaksi tämän testaamisessa tulisi, että joutuisimme aina ennen testejä tyhjentämään tietokannan ja laittamaan sinne halutut arvot. Tämä olisi toki mahdollista, mutta helpompi ratkaisu on käyttää mock-toteutusta tietokannasta, jolle määritellään, että se palauttaa tietyllä kutsulla halutut oliot ja testin lopuksi varmistetaan, että simuloidulle tietokannalle olisi mennyt oikeat kutsut. Mock-toteutuksia voidaan käyttää myös testilähtöisessä kehityksessä, esimerkiksi tapauksissa, joissa simuloidaan olion tai järjestelmän osaa siihen asti, kun se toteutetaan oikeasti.

Automaattista testausta voidaan myös käyttää apuna ohjelmiston suunnittelussa [1]. Sen sijaan, että ensiksi tehtäisiin testattava koodi, tehdäänkin testit ennen koodia. Testien tekemisen jälkeen täydennetään ohjelmaa niin, että testit voidaan ajaa. Tämän jälkeen täydennetään ohjelmaa niin, että testit läpäistään. Kun testit on läpäisty, siistitään koodia ja ajetaan testit uudestaan, jotta varmistutaan siitä, ettei muutokset rikkoneet ohjelman toimivuutta. Kun ohjelma on refaktoroitu, tehdään lisää testejä, ja toistetaan sykliä. Näin myös testejä voidaan ajaa koko ohjelman kehityksen ajan ja varmistua, että ohjelma toimii määritellysti. Alla on pieni esimerkki testilähtöisestä kehityksestä. Tämä esimerkki ei kuitenkaan ota kantaa, onko määritellyt testit mitenkään kattavat.

Esimerkkinä voitaisiin määritellä funktio "onkoPaivamaara", joka tarkistaa, onko syötteenä annettu merkkijono päivämäärä, vai ei. Ohjelma palauttaa totuusarvon, jos syöte on päivämäärä. Tälle voitaisiin määritellä muutamia testitapauksia, esimerkiksi "11.2.1989", "1.1.2001"ja "11:34"ja näille syötteille oikeat paluuarvot. Tämän jälkeen kirjoitetaan ohjelmakoodi, joka läpäisee annetut testitapaukset. Kun tämä vaihe on tehty, siistitään koodia, ja kirjoitetaan uusia testejä, tässä tapauksessa esimerkiksi "1. tammikuuta 1978"ja "01.01.88"ja ajetaan testit. Jos testit läpäistään, kirjoitetaan uusia testejä ja toistetaan sykliä. Jos testejä ei läpäistä, muokataan koodia niin, että kaikki testit läpäistään. Tätä kiertoa toistetaan, kunnes ohjelman osa toimii tarpeeksi hyvin.

Testilähtöisen kehityksen etuna kehittäjällä on koko ajan varmuus siitä, että oh-

jelma toimii muutosten jälkeen, jos testit on tehty oikein. Ongelmaksi kuitenkin nousee ohjelmakoodiin tulevat muutokset, jotka rikkovat testit, vaikka ohjelma itsessään toimisi oikein. Näin myös testejä täytyy ylläpitää, mikä kuluttaa resursseja ja ei motivoi tekemään testejä.

3 Graafisten käyttöliittymien automaattinen testaus

Tässä luvussa käsitellään graafisten käyttöliittymien automaattista testausta, sen ongelmia ja työkalujen tarjoamia ratkaisuja näihin ongelmiin. Samoin kuin testauksessa yleensä, myös graafisten käyttöliittymien testaamisessa on ongelmana se, miten luodaan tarpeeksi kattavat testit [2, s. 12]. Tässä tutkimuksessa kuitenkin sivuutetaan testikattavuuden ongelmat, ja keskitytään testien tekniseen toteutukseen. Samoin kuin automaattisella testauksella yleensä, pyritään myös graafisten käyttöliittymien automaattisella testauksella parantamaan ohjelman luotettavuutta.

Yhtenä isona ongelmana graafisten käyttöliittymien testauksessa on jo luvussa 1 mainittu käytössä olevien toimintojen iso määrä. Tämä aiheuttaa sen, että kattava manuaalinen testaus olisi erittäin työlästä ja ei siis kovin käytännöllistä. Myöskin ohjelman tekijä käyttää ohjelmaa itse helposti niin kuin sen on ajateltu toimivan. Tämä lähestyminen ei kuitenkaan poista ongelmia, jotka aiheutuvat siitä, kun ohjelman väärin käyttäminen rikkoo oikeita toimintoja. Eikä tähän tarvita edes käyttäjää, joka yrittää käyttää ohjelmaa väärin, vaan tähän voi riittää käyttäjä, joka ei tiedä miten ohjelmaa käytetään. Ennalta määritelty automatisointikaan ei kuitenkaan ratkaise edellä mainittua ongelmaa, koska tapauksia on käytännössä usein niin paljon, että niiden määrittely on epäkäytännöllistä. Automatisoinnin muina ongelmina taas on, että kuka hoitaa syötteen, tarkistaa tulokset, avaa ikkunat ja sulkee ne.

Käyttäjän toimintojen automatisointi voitaisiin hoitaa yksinkertaisella ohjelmalla, joka nauhoittaisi käyttäjän toimenpiteet ja toistaisi ne tarvittaessa. Tässä lähestymistavassa ongelmallista on kuitenkin, että käyttöliittymään komponenttien paikkaa muuttaessa jouduttaisiin tekemään uudet testit, koska vanhat testit tekisivät toimintoja väärissä kohdissa. Jos nauhoittava työkalu taas ei nauhoittaisi absoluuttisia paikkoja, vaan tunnistaisi, että käyttäjä esimerkiksi painaa juuri jotain tiettyä nappia, siirrettävyys olisi mahdollista toteuttaa esimerkiksi ottamalla kuva komponentista ja vertaamalla sitä käyttöliittymään. Tämän ongelmana taas on, että myös komponenttien ulkonäkö voi vaihtua, esimerkiksi käytössä olevan teeman mukaan,

joten tämäkään lähestyminen ei ole kovin käytännöllinen. Jos käytössä oleva ohjelmointikieli on tiedossa, on joissain tapauksissa mahdollista saada viite juuri tiettyyn komponenttiin tietyllä tunnisteella, joten komponentin paikka ja ulkonäkö voisi vaihtua. Tämä tapa on käytössä nauhoittavissa työkaluissa, joita käsitellään myöhemmin.

Tuloksien tarkistamisessa on ongelmallista se, miten se tehdään. Tämäkin voitaisiin teoriassa toteuttaa vertaamalla kuvaa toivotusta tilanteesta tilanteeseen testien jälkeen. Ongelmaksi kuitenkin tulee, että tämäkään ei kestä ohjelman komponenttien paikkojen vaihtamista, tai komponenttien ulkonäön muuttumista. Ohjelmointikielen ollessa tiedossa, voimme joissain tapauksissa kysyä käyttöliittymältä esimerkiksi listan kaikista sen komponenteista [kts. koodiesimerkki 1], ja näin etsiä sieltä viite haluamaansa komponenttiin ja varmistaa sen oikea tila. Tätä käytetään myöhemmin käsitellyissä työkaluissa.

Koodiesimerkki 1 Esimerkki, miten johonkin komponenttiin voitaisiin hakea viite Javan GetComponents()-metodin avulla

```
Component[] komponentit = gui.getContentPane().getComponents();
RadioButton rbPunta;

// Haetaan testattavat komponentit ikkunasta asetetun
// nimen perusteella.
for (int i = 0; i < komponentit.length; i++) {
    if (komponentit[i].getName().equals("rbPunta"))
        rbPunta = (JRadioButton) komponentit[i];
}

//Varmistetaan, että rbPunta-radiobutton on valittu.
assertEquals(true,rbPunta.isSelected());
```

Graafisten käyttöliittymien automaattisessa testauksessa siis nauhoitetaan, ohjelmoidaan tai jollain muulla tavalla tehdään lista käyttäjää matkivia komentoja, jotka vievät ohjelman tietystä tilasta toiseen tilaan. Automatiikka toteuttaa nämä komennot ja sen jälkeen tarkistaa, onko testattavan ohjelman saavutettu tila se, jota odotettiin. Jos tila ei vastaa odotettua, on testityökalu havainnut virheen.

Tässä tutkimuksessa käsitellään kahta toisistaan eroavaa tapaa tehdä automaattisia testejä graafisille käyttöliittymille: käskyjen nauhoittamista ja käskyjen ohjel-

moimista. Käskyjen nauhoittamisessa testaaja tekee koneelle mallisuorituksen, joka nauhoitetaan. Tämän jälkeen testaaja määrittelee, missä tilassa ohjelman pitäisi olla suorituksen jälkeen. Käskyjen nauhoittamista ei voi käyttää alusta asti testilähtöisessä kehityksessä, koska käskyjen nauhoittaminen vaatii jonkin käyttöliittymän pohjalle. Työkalusta riippuen, tämän käyttöliittymän ei toki tarvitse olla toiminnoiltaan täydellinen, koska toivottu tulos voidaan osittain määritellä käsin. Erilaiset komponentit on kuitenkin luonnollisesti oltava olemassa, jotta niitä voidaan käsitellä.

Komentojen ohjelmoinnissa taas testaaja tuottaa testiohjelmiston käyttämällä kielellä ohjeet käyttäjää simuloivalle robotille, joka suorittaa nämä ohjeet. Tämän jälkeen testaaja ohjelmoi tarkastukset siitä, onko ohjelma oikeassa tilassa.

Oma osa-alueensa graafisissa käyttöliittymissä ovat web-käyttöliittymät. Ongelmia tuo esimerkiksi selainten eroavat tavat käsitellä Javascriptiä ja CSS-tyylityksiä. Vaikka ulkonäölliset seikat eivät välttämättä ole oleellisia sivun toiminnan kannalta, voi selainkohtainen ero Javascriptin tulkinnassa pahimmillaan rikkoa sivun toiminnan kokonaan, vaikka se toimisi toisella selaimella [kts. 3.1]. Näin testityökaluissa olisi hyvä huolehtia, että testit voidaan ajaa useammassa selaimessa. Helpottavana asiana testit eivät kuitenkaan välttämättä vaadi hiiren tai näppäimistön kontrollia, vaan testejä ajavaa konetta voidaan periaatteessa käyttää samalla muuhunkin. Esimerkiksi Selenium käyttää tässä apunaan selaimen tarjoamia työkaluja [34]. Tämä mahdollistaa periaatteessa myös testien rinnakkaistamisen, tosin tässä on huolehdittava, etteivät esimerkiksi tietokantaoperaatiot satu päällekkäin ja riko testejä. Myös erillisten tilojen välillä siirtyminen on parhaimmillaan yksinkertaista, ainakin niissä tapauksissa, joissa tietty sivu voidaan avata suoraan käyttäen URL-osoittetta.

Hidasteiksi web-käyttöliittymien testauksessa nousee juuri alustojen monipuolisuus ja mahdollisten sivulatausten ja muiden kutsujen vaihtelevat kestot. Joissain tapauksissa testityökalut mahdollistavat dynaamiset tauot, jotka esimerkiksi odottaa vain sivun lautauksen, eikä ennalta määriteltyä staattista aikaa.

Selain	Moottori
Internet Explorer 9	Chakra [29]
Safari	Nitro [32]
Mozilla Firefox	SpiderMonkey [31]
Chrome	V8 [30]

Kuva 3.1: Selainten käyttämät JavaScript-moottorit

Myös graafisten käyttöliittymien testilähtöistä kehitystä on tutkittu, mutta on-

gelmana on, että testejä on vaikea kirjoittaa, ellei käyttöliittymää ole olemassa. Testilähtöinen kehitys taas vaatii, että testit tehdään, ennen kuin toteutus on tehty [6, kpl. II]. Jonkinlaisia ratkaisuja esitettyihin ongelmiin on kuitenkin löydetty, kuten se, ettei toimintoja nauhoiteta, vaan ne ohjelmoidaan [7].

4 Työkalut

Tässä luvussa esitellään testattavaksi rakennettu ohjelma, sen tekemiseen käytetyt työkalut, sekä mitä testattavasta ohjelmasta aiotaan testata. Työkalujen vertailua varten tehtiin yksinkertainen ohjelma, josta testataan tietyt asiat, jonka jälkeen testityökalujen tuottamia testejä vertaillaan. Näin joudutaan kuitenkin jättämään testilähtöisen kehityksen näkökulman pienelle huomiolle, koska testattavaa ohjelmaa ei olisi mielekästä rakentaa uudestaan jokaiselle työkalulle, eikä vertailu olisi toimiva. Testilähtöinen kehitys pyritään kuitenkin käsittelemään mahdollisuuksien mukaan, ainakin sen perusteella, onko se mahdollista kullakin työkalulla.

Testattavaksi rakennettu ohjelma rakennettiin Java Foundation Classes-kokoelmasta löytyvällä Swing-kirjastolla. Java Foundation Classes on kokoelma Java-kirjastoja, jotka mahdollistavat graafisten käyttöliittymien tekemisen Javalla. Nämä kirjastot tulevat Java platformin mukana, joten komponentteja ei tarvitse erikseen paketoida levitettävän ohjelman mukaan, toisin kuin kolmansien osapuolten GUI-kirjastoissa [9] [10]. Testattava ikkuna tehtiin Swingillä, koska se on käytössä Jyväskylän Yliopiston Tietotekniikan Laitoksen Ohjelmointi 2-kurssilla [11].

Swing-ohjelmia luodessa ja testatessa on huomioitava myös säieturvallisuus, koska Swing käyttää säikeitä taatakseen, ettei ohjelman käyttöliittymä jäädy [33]. Testatessa tämän huomiointi on tärkeää, koska pahimmassa tapauksessa testin tulos tarkistetaan, ennen kuin itse toiminnallisuuden aiheuttavaa ohjelmakoodia on ajettu. Tässä tapauksessa testi palauttaisi väärän tuloksen, vaikka ohjelma toimisi oikein. Testatuissa työkaluissa säieturvallisuus on kuitenkin huomioitu, joten testien tekijän ei tarvitse tähän juurikaan kiinnittää huomiota. Erikseen pitää tietysti huomioida tapaukset, joissa käytetään muita säikeitä suorittamaan raskaampia operaatioita.

4.1 Testattava ohjelma

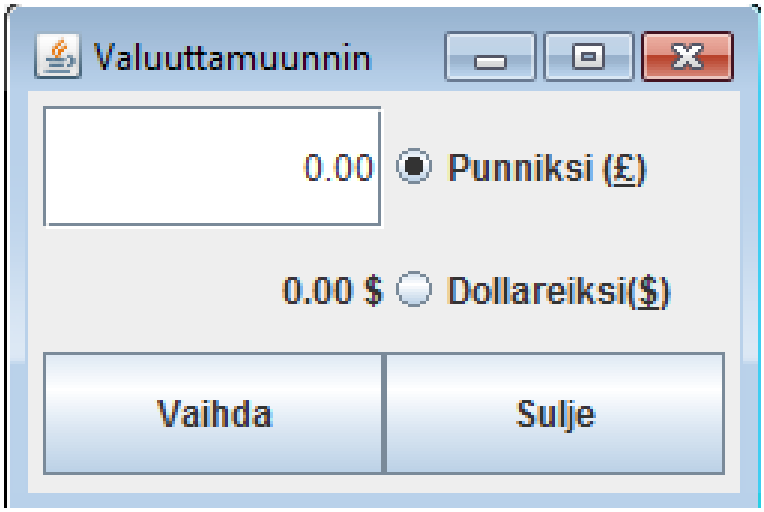
Testattavaksi on rakennettu yksinkertainen valuuttamuunnin 5.1, jonka tarkoituksena on, että käyttäjä voi syöttää halutun rahasumman, ja painaa vaihda-painiketta, jolloin ohjelma kertoo, kuinka paljon syötetty summa on vieraalla valuutalla. Vaihdettavan valuutan voi valita kahdesta samassa ButtonGroupissa olevasta JRadioButtonista. Kun käyttäjä painaa rastia tai Sulje-nappia, ilmestyy About-ikkuna, jossa on nappi joka sulkee ohjelman lopullisesti.

Syötteen ollessa väärin, jos esimerkiksi rahamäärää ei voida parsia, kentän taustaväri muuttuu punaiseksi. Jos käyttäjä kuitenkin painaa vaihda, tulee rahamäärän tilalle teksti: "Virheellinen syöte".

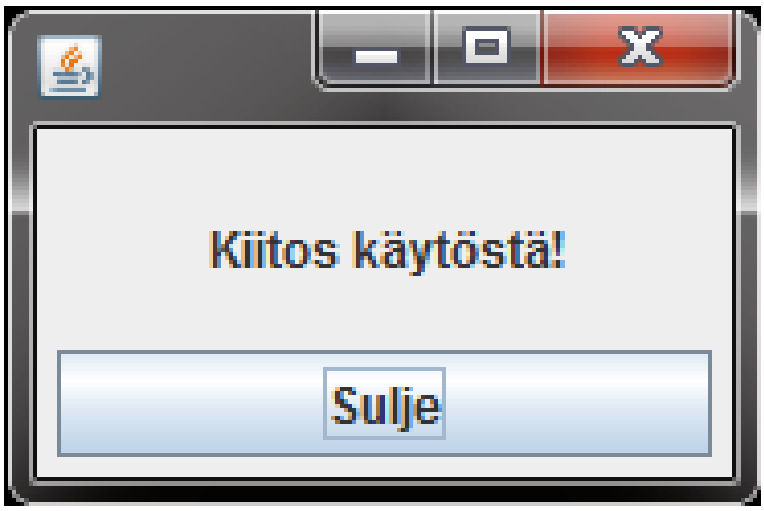
Testattavat asiat:

1. Syöte oikein ja punta valittu
2. Syöte oikein ja dollari valittu
3. Syöte väärin
4. Syöte väärin ja sen jälkeen oikein (palaako valkoinen väri tekstikenttään)
5. Sulje-nappi avaa About-ikkunan
6. Rasti avaa About-ikkunan

Näiden tarkoitus ei ole kuitenkaan tuottaa kattavia testejä, vaan selvittää, miten näiden asioiden testaaminen eroaa eri työkalujen kesken.



Kuva 4.1: Kuva testattavasta valuuttamuuntimen pääikkunasta.

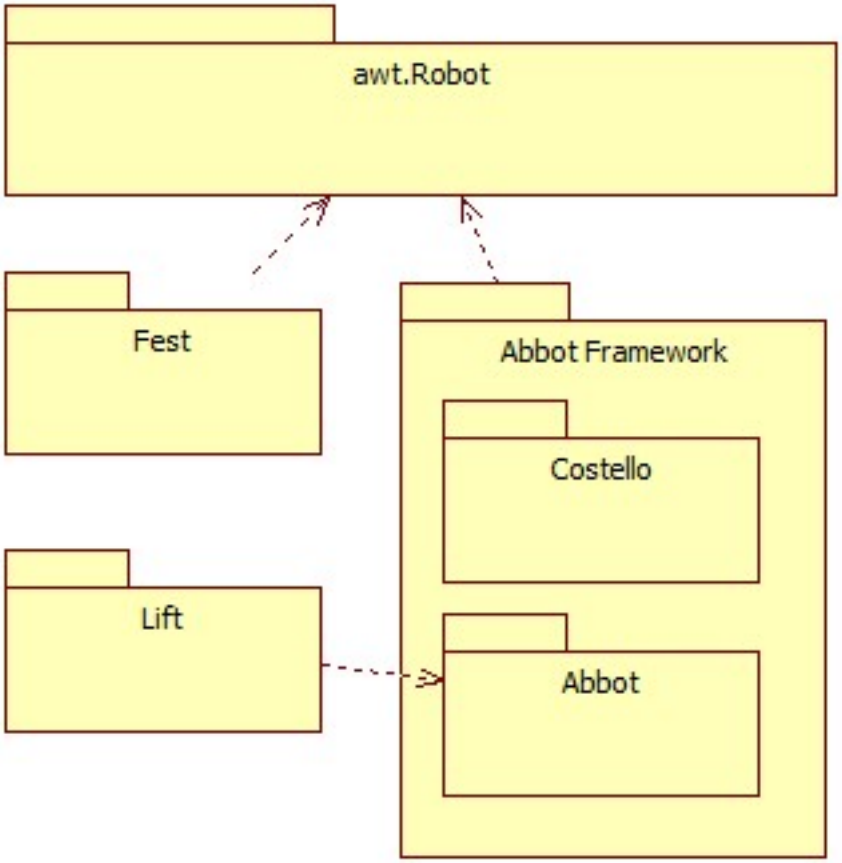


Kuva 4.2: Kuva testattavasta valuuttamuuntimen About-ikkunasta.

5 Automaattinen GUItestaus Javassa

Tässä osiossa käsitellään löydetyt työkalut ja niiden toiminta. Työkaluja etsittiin alan julkaisuista, sekä internetistä yleisesti. Työkaluja löydettiin useita, mutta käsiteltäväksi otettiin neljä: Javan Robot, ja sitä käyttävät Abbot (A Better Bot) Framework, Fest eli Fixtures for Easy Software Testing ja Lift, eli The Library for InterFace Testing.

Työkalujen vertailu käsitellään myöhemmin.



Kuva 5.1: Testattavien työkalujen keskinäisiä riippuvuuksia selventävä kaavio.

5.1 Robot (Java/Oracle)

Robot on Javan työkalu käyttäjän syötettä vaativien ohjelmien testaamisen automatisointiin. [8]. Robotille annetaan koodissa ohjeet, miten sen pitää toimia, ja näin se matkii käyttäjän syötettä. Käyttäjän syötettä pystytään matkimaan hiiren ja näppäimistön osalta, sekä myös taukojen määrittäminen on mahdollista. Myös pikselien vertaaminen ja kuvan ottaminen ohjelmasta olisi mahdollista, mutta niitä ei kuitenkaan käytetä tässä esimerkissä.

Robot-testiä varten tehtiin kaksi testausta helpottavaa aliohjelmaa, click, joka painaa hiiren vasemmalle näppäimellä parametrinä annettua komponenttia, sekä find, joka etsii rekursiivisesti komponentin toisen komponentin sisältä name-attribuutin perusteella, ja palauttaa viitteen tähän [kts. C]. Myöskin tekstin syöttämiselle komponenttiin olisi voinut tehdä oman aliohjelmansa, mutta tätä ei kuitenkaan lähdetty toteuttamaan.

Tekstin valitsemisessa tuli myös ongelmia, koska Robotissa oli ohjelmointivirhe [16]. Tämä pystyttiin kuitenkin kiertämään laittamalla numlock pois päältä.

5.2 Abbot

Abbot Framework on Javan graafisten käyttöliittymien testaamista helpottava kokoelma työkaluja [12]. Ensimmäisenä työkaluna on Robot ("A Better 'Bot"), jolla toistetaan käyttäjän syöte. Abbotin Robot käyttää pohjalla Javan Robottia [13], laajentaen tämän toimintaa helppokäyttöisemmäksi. Robotille voidaan esimerkiksi antaa suoraan viite painettavaan Component-olioon [kts. koodiesimerkki 2].

Koodiesimerkki 2 Esimerkki Abbotin komponenttien etsimisestä.

```
JButtonTester s = new JButtonTester();
s.click((JButton) getFinder().find(new NameMatcher("vaihda")));
```

Toisena työkaluna on ComponentLocation-luokka, joka helpottaa komponenttien käsittelyä, joilla on sisäinen hierarkia. Näin voidaan tehdä yksi metodi kullekkin toiminnolle, joka ottaa vastaan ComponentLocation-olion, eikä tarvitse tehdä erikseen metodeita erillisille tavoille ilmoittaa sijainteja. Tämä voisi olla ongelma esimerkiksi taulukon tapauksessa, jossa voitaisiin joutua tekemään useampia metodeja esimerkiksi rivin, sarakkeen tai solun klikkaamiselle. Tätä ei kuitenkaan hyö-

dynnetä laisinkaan tässä esimerkissä.

Kolmantena työkaluna on ComponentReferences-luokka, jonka kautta voidaan viitata johonkin komponenttiin sille luodun yksilöllisen ID-arvon kautta. Näin skriptissä voidaan viitata johonkin komponenttiin, vaikka sitä ei ole vielä luotu [12] [14].

Edellä mainitut skriptit ovat neljäs työkalu. Skriptit sisältävät tarvittavat ohjeet testien suorittamiseksi. Skriptien etuna on se, ettei niitä tarvitse erikseen kääntää, ja niitä on näin helpompi muokata ja ylläpitää [15].

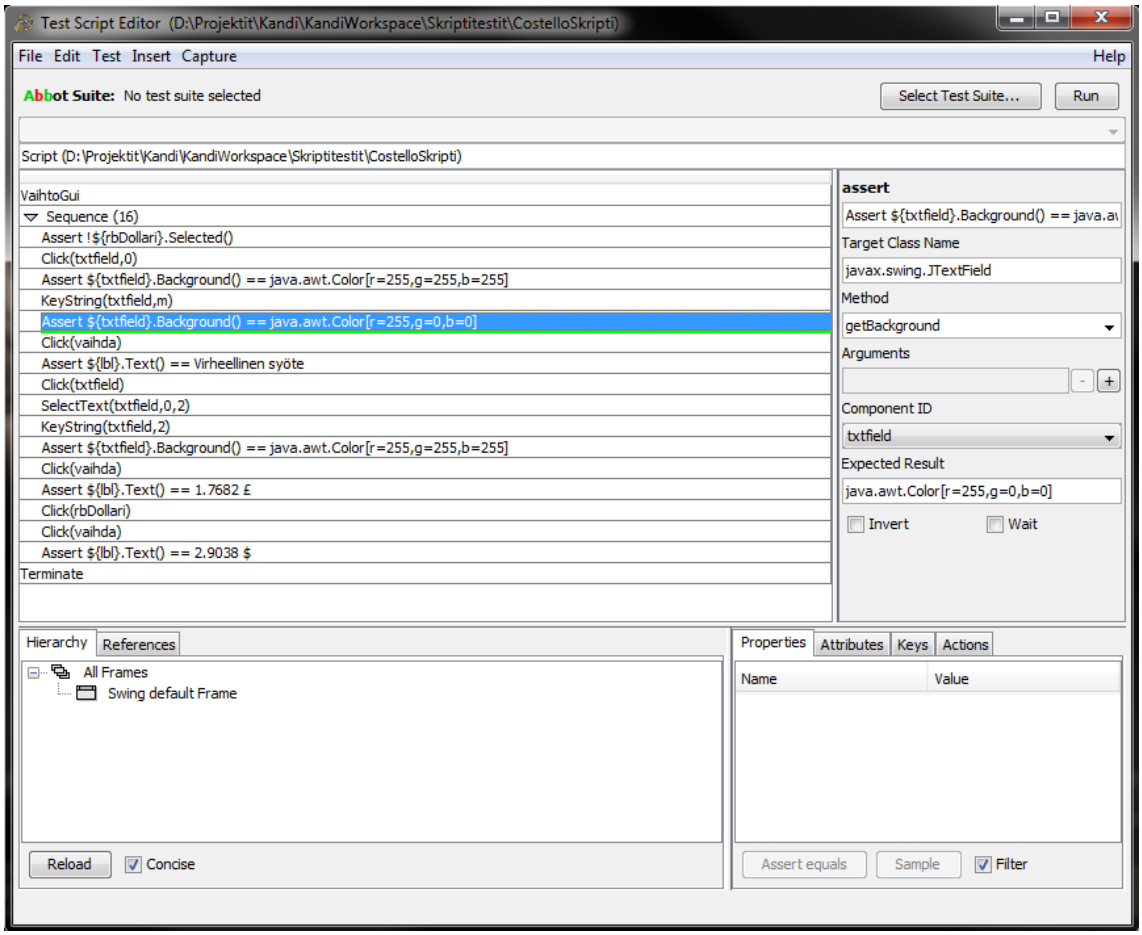
Viidentenä työkaluna on ComponentTestFixture, joka helpottaa testien kirjoittamista suoraan Java-koodiin. Tämä työkalu huolehtii esimerkiksi ikkunoiden sulkemisen testin päätyttyä [12].

Kuudentena työkaluna on skriptieditori Costello. Costellolla voidaan nauhoittaa käyttäjän toimenpiteet ja muokata skriptejä sekä ajaa niitä [12]. Costellolla tehdyt testit käsitellään vertailussa erikseen, koska niitä tehdessä ei kirjoiteta ohjelmakoodia. Ohessa [Kuva 5.2] on kuva valmiista testistä, jossa tarkastetaan tehdyn ohjelman toiminnallisuus. Vasemmalla puolella ikkunaa on testin vaiheet ja oikealla puolella voi tarkastella testivaiheen yksityiskohtia. Ohjelman ollessa käynnissä alhaalta nähdään ohjelman hierarkinen rakenne ja viitteet, joita käytetään testeissä. Tämän ikkunan oikealla puolella näkyy valitun ikkunan ominaisuudet, joista näitä voidaan valita testiin vertailtavaksi. Vertailuja ja muita toimenpiteitä voidaan tehdä myös yläosan menupalkin Insert-valikon kautta. Nauhoituksessa ohjelma käynnistetään Costellon kautta, ja sen jälkeen suoritetaan halutut toimenpiteet ja tarvittaessa nauhotus tauotetaan, jotta voidaan lisätä vertailut.

5.3 Fest

Fest, eli Fixtures for Easy Software Testing on kokoelma moduuleja, joiden tarkoitus on helpottaa ohjelmiston testaamista [4]. Tässä tutkimuksessa käsittellään kuitenkin vain Swing-ohjelmien testaamista koskevat asiat. Fest-testit voidaan luoda perimällä valmiiksi tehty luokka tai toteuttaa itse muutama aliohjelmat, jotka ajetaan ennen testejä ja testien jälkeen. Käyttäjän syötteiden simuloimiseen Fest käyttää Robot-luokkaa, joka on rakennettu Javan Robot-toteutuksen päälle.

Fest tarjoaa myös työkaluja vertailujen helpottamiseen. Näitä ei kuitenkaan käytetty kuin kerran, koska pääasiassa komponenttien etsimisen ehdot hoitivat suuren osan tarkistuksista, kuten koodiesimerkissä 3, jossa tarkistetaan, että komponentin name on "txtfield"ja että se sisältää tekstin "0.00". Tämän jälkeen valitaan kentän si-



Kuva 5.2: Kuva Costellon käyttöliittymästä.

sältö ja syötetään teksti "M00.00".

Koodiesimerkki 3 Esimerki Festin komponenttien etsimisestä.

```
window.textBox("txtfield").requireText("0.00").selectAll()
      .enterText("M00.00");
```

Esimerkkitestit tehtiin perimällä valmiiksi tehty luokka, koska tämä todennäköisesti vastaa enemmän tilannetta, jossa koitetaan tarjota mahdollisimman yksinkertainen testiympäristö opetuskäyttöön. Fest-testeihin joudutaan kuitenkin toteuttamaan itse setUp-metodi, joka luo testattavan ohjelman säieturvallisesti. setUp-metodilla ja valmiin luokan käytöllä varmistetaan, että testeissä käytetään testattavaa ohjelmaa säieturvallisesti [21]. Näin on tarkoitus varmistaa, että testit toimivat aina samoin [22].

Muita festin tarjoamia työkaluja oli esimerkiksi Fluent Reflection-moduuli, jonka tarkoituksena on helpottaa Javan reflektion käyttöä. Festin kehittäjien mukaan [27] Festin kehityksessä on tullut vastaan tilanteita, joissa ei ole tarpeeksi tietoa testattavan ohjelman alustasta, jotta voitaisiin simuloida käyttäjän syötettä Swing-komponentille. Tähän kehittäjät käyttivät reflektiota ja kehittivät yksinkertaisemman rajapinnan reflektiolle. Tutkielman testeissä tälle moduulille ei kuitenkaan noussut tarvetta.

Fest sisältää myös EasyMock Template-moduulin, jonka tarkoituksena on poistaa simuloitujen objektien käytön aiheuttamaa koodin toistumista. Tätä ei kuitenkaan enää kehitetä, vaan kehittäjät ovat siirtyneet käyttämään Mockitoa mock-objektien tekemiseen [28]. Tehdyissä testeissä ei kuitenkaan noussut tarvetta mock-toteutuksien käyttöön, koska tehty ohjelma oli hyvin yksinkertainen. Jos ohjelmaa olisi laajennettu esimerkiksi hakemaan valuuttoja palvelimelta, pitäisi testeissä varautua kurssien muutoksille. Tässä tapauksessa mock-toteutus tulisi tarpeelliseksi.

5.4 Lift

Lift, eli The Library for InterFace Testing on kirjasto Swingillä tehtyjen graafiseen käyttöliittymien testaamiseen. Kirjaston tarkoituksena on helpottaa graafisten käyttöliittymien testaamista, ja se on suunniteltu erityisesti uusia ohjelmoijia varten [3]. Liftissä GUI-testit tehdään perimällä GUITestCase. Kirjasto käyttää testaamiseen

Abbot-työkaluja, mutta pyrkii yksinkertaistamaan niitä käyttäjälle. Suurimpana erona on, että käyttäjän ei tarvitse monimutkaisemmilla kriteereillä komponenttien etsimiseen tehdä itse Matcher-oliota, vaan käyttäjän voi yhdistellä Filter-olioita "minikielen"avulla.

Esimerkiksi jos haluttaisiin hakea käytössä oleva JRadioButton, jonka nimi on "rbPunta", voitaisiin tällainen Filter-olio luoda kohdassa 4 esitetyllä tavalla.

Koodiesimerkki 4 Lift-filter esimerkki

```
getComponent(JRadioButton.class, where.enabledIs(true)
        .and.nameIs("rbPunta"));
```

Komponenttien käyttämistä yksinkertaistetaan Abbotin vastaavasta piilottamalla haettavat ComponentTesterit ja tarjoamalla yksinkertaisempi toteutus tästä, jossa riittää esimerkiksi pelkkä click-metodin kutsuminen, jollekka annetaan parametrinä komponentti [3, Figure 9].

5.5 Soveltaminen comtestiin

Comtest on aiemmin mainitulla Ohjelmointi 2-kurssilla käytössä oleva testityökalu, joka luo yksikkötestit Java-ohjelmakoodin kommentteihin tehdyistä testeistä. Näin saadaan samalla aliohjelmille sekä testit, että käyttöesimerkit [18][kts. myös koodiesimerkki 5].

Käsiteltävistä työkaluista Comtestin kanssa käytettäväksi valittiin Lift, koska se vaikutti yksinkertaisimmalta ja se on ollut opetuskäytössä [3]. Comtestiä varten Liftiin lisättiin textMatches-metodi, jotta komponenttejä voidaan myös etsiä osittain vastaavan tekstin kanssa. Tällä pyrittiin välttämään työlääksi koetun name-ominaisuuden käyttöä [kts. koodiesimerkki 6].

Muuten Liftin käyttö säilyi ennallaan, ainoana erona, että Liftin ominaisuuksia käytetään itse luotavan GUITester-olion kautta. Tämän ansiosta GUITestCase-luokkaa ei tarvitse periä.

Tämä oli myös testikäytössä kurssilla vapaaehtoisten tehtävien muodossa. Kurssilla sai merkitä lisäpisteen, jos tehtävänä tehtyyn graafiseen käyttöliittymään teki myös testit suoraan Lift-kirjastolla tai Comtestin avulla. Gui-testejä kurssilla teki kuitenkin vain kaksi opiskelijaa, joten suurta otantaa ei saatu. On mahdollista, että itse GUI-testaus tai sen opettelu koettiin niin työlääksi, ettei sitä nähty yhden

Koodiesimerkki 5 Esimerkki Comtest-testeistä.

```
/**
 * Palauttaa taulukosta pienimmän alkion paikan.
 * Jos on useampi pienin, palautetaan ensimmäinen.
 * Jos alkioita ei ole, palauttaa -1.
 * @param taulukko
 * @return Taulukon pienimmän alkion indeksi
 * @example
 * <pre name="test">
 * int taulukko1[] = {1,2,3,2,4};
 * int taulukko3[] = {1,2,3,2,0};
 * int taulukko2[] = new int[0];
 * int taulukko4[] = {2,1,1,1,5};
 * pienimmanPaikka(taulukko1) === 0;
 * pienimmanPaikka(taulukko3) === 4;
 * pienimmanPaikka(taulukko2) === -1;
 * pienimmanPaikka(taulukko4) === 1;
 * </pre>
 */
public static int pienimmanPaikka(int[] taulukko) {
    // Tähän metodin toteutus
}
```

Koodiesimerkki 6 Liftin soveltaminen Comtestiin

```
 *   comtest.GUITester g = new comtest.GUITester();
 *   SwingAanestys frame = new SwingAanestys();
 *   g.showWindow(frame);
 *   JButton buttonAanesta =
            g.getComponent(JButton.class,
                g.where.textMatches("Äänes.*"));
```

pisteen arvoiseksi. Näin pienellä otannalla ei voida kuitenkaan testeistä suurempia johtopäätöksiä tehdä.

5.6 Vertailu

Tässä osioissa vertaillaan tehtyjä testejä. Vertailussa käydään läpi miten Abbot, Fest, Lift ja Costello eroavat Robotista ja mitenkä Liftissä on koitettu tehdä asioita yksinkertaisemmiksi, kuin Abbotissa.

Työkalujen käyttöönotossa löytyi jonkinlaisia eroja. Käyttöönotossa helpoin on Javan Robot, koska se tulee Javan mukana, ja erillistä asentamista ei tarvita. Käytännössä tehokkaaseen käyttöön joutuu työkalua kuitenkin laajentamaan niin paljon, että helpolla käyttöönotolla ei juuri voiteta mitään oleellista. Abbot, Fest ja Lift eivät juuri eroa käyttöönotossa toisistaan. Kaikissa otetaan ulkoinen kirjasto mukaan projektiin, ja peritään testiluokka, johonka tulevat testit tehdään. Comtestissä joudutaan kirjaston käyttöönotoksi tekemään kehitysympäristöön hieman muutoksia, että testit saadaan generoitua helposti. Comtestiä pystytään käyttämään myös ilman kehitysympäristöön laittettavaa pluginia, mutta tällöin ajettavat testit joudutaan generoimaan erikseen. Costellon käyttöönotossa riitti ohjelman käynnistäminen ja tarvittavien Classpathien asettaminen työkaluun, jotta työkalu osaa käynnistää oikean ohjelman testattavaksi. Tämä vaihe voi kuitenkin aiheuttaa vaikeuksia, koska Classpathin ja käynnistettävän luokan nimen joutuu kirjoittamaan käsin.

Kun verrataan pelkällä Javan Robotilla tehtyjä testejä Robotin päälle rakennettuilla testityökaluilla tehtyihin testeihin, huomataan, että testien tekeminen on erittäin työlästä. Esimerkiksi näppäimistön syötteiden luominen ei ole esimerkiksi luodussa testissä kovin kätevää [kts. C]. Tähän kuitenkin voitaisiin luoda asiaa helpottava aliohjelma, kuten Abbotissa, Festissä ja Liftissä on tehty. Esimerkissä tekstin syöttöä tehtiin kuitenkin niin vähän, että tätä ei nähty tarpeelliseksi. Robotilla jouduimme myös huolehtimaan JTextFieldin tyhjentämisestä ja tässä kohti tuli vastaan shift-näppäimen käyttöön liittyvä ohjelmointivirhe [16]. Costellolla nauhoittaessa kirjoittamiseen ei tarvinnut kiinnittää erityistä huomioita ja ainakin tässä tapauksesa testejä tehdessä tehdyt näppäimistöpainalukset tapahtuivat myös testin suorituksen aikana.

Hiirtä vaativissa toimenpiteissä ei tehdyssä esimerkissä Abbotissa ja Liftissä ollut juuri eroa. Kummassakin annettiin aliohjelmalle komponentti, jota halutaan napauttaa. Robot-testeissä tälle tehtiin yksinkertainen aliohjelma. Festissä tähän oli

hieman erilainen lähestymistapa, jossa toimenpiteet tehtiin ContainerFixtureiden kautta. Costellolla nauhoittaessa hiirtä käytetään kuin normaalistikkin ohjelmaa käyttäessä. Kaikki hiiren toiminnot eivät kuitenkaan tässä tapauksessa tallennu skriptiin, ellei näin erikseen päätetä. Tehdyissä testeissä tälle ei kuitenkaan ollut tarvetta, joten hiiren liikkeet jätettiin huomiotta, ja tallennettiin pelkät toiminnot.

Komponenttien etsimisessä työkalujen välillä löytyi isompia eroja. Komponenttien etsimisellä tarkoitetaan sitä, mitenkä työkalulla saadaan viite tiettyyn komponenttiin, jotta sille voidaan tehdä toimenpiteitä. Yhteistä työkaluilla oli, että kaikki tuottivat virheen, jos komponenttia ei löytynyt. Abbotin ratkaisu komponenttien hakemiseen on Matcher-luokka, joka tarkastaa, vastaako komponentti tiettyjä ehtoja. Yksinkertaisiin hakuihin (luokka, nimi...) löytyi valmiit luokat, mutta monimutkaisempiin hakuihin voi toteutuksen joutua tekemään itse.

Fest tarjoaa etsimiseen kaksi mahdollisuutta, ComponentFinder-luokan, mikä muistuttaa Abbotin ratkaisua, ja ContainerFixturen. ContainerFixture on abstrakti luokka, joka tarjoaa viitteet testattavan komponentin GUI-komponentteihin. Tätä kautta voidaan myös metodien kautta esittää vaatimuksia komponenteille. Alla esimerkki, jossa etsitään JRadioButton, jonka nimi on "rbPunta"ja se on valittu.

Koodiesimerkki 7 Festin komponenttien etsiminen

```
window.radioButton("rbPunta").requireSelected();
```

Liftin ratkaisu komponenttien etsimiseen muistuttaa Festin ContainerFixture-ratkaisua. Yksinkertainen etsintä voidaan tehdä GUI-komponentin luokalla ja Name-ominaisuudella, mutta monimutkaisempiin hakuihin pitää käyttää Filter-luokkaa, jolla kuvataan etsittävän komponentin ominaisuuksia. Luoduissa testeissä ei kuitenkaan noussut tarvetta tehdä monimutkaisempia hakuja. Alla esimerkki Filter-luokan käytöstä.

Koodiesimerkki 8 Esimerkki Filter-luokan käytöstä.

```
JButton button = getComponent(JButton.class,
    where.nameIs("sulje"));
```

Costellossa komponenttien etsiminen tietyillä arvoilla, esimerkiksi vaikka nimellä, ei ole niiden operoinnin kannalta tarpeellista, koska testit nauhotetaan ja käyttäjä

voi näin tehdä haluamansa toimenpiteen esillä oleville komponenteille. Jos vertailuja halutaan tehdä, löytyvät testattavan ohjelman sillä hetkellä esillä olevat komponentit ikkunan alalaidan listasta. Testin vaiheiden arvoja voidaan muuttaa ikkunan oikean laidan kentistä. [Kuva 5.2].

Pelkästään koodirievien määrää vertaillessa päätyivät kaikki testiohjelmat hyvin samankaltaisiin lukuihin. Abbotilla kirjoitettujen testialiohjelmien pituus oli yhteensä 43 riviä, Liftillä 34, Festillä 37 ja Comtest + Lift-yhdistelmällä 37. Koska tämän suurempia eroja ei syntynyt, ja kirjoitettuja ohjelmia voitaisiin myös halutessa tiivistää, ei rivien määrissä ole testityökalujen välillä merkittäviä eroja. Javan Robottestejä varten kirjoitettiin 96 riviä.

Testien ylläpidettävyydestä ei keretty saada paljoa tietoa, koska testiohjelman kehitysaika oli hyvin, eikä sitä juuri päivitetty. Ainoat ongelmat tulivat Abbotin ja Liftin kanssa, mutta nämä johtuivat siitä, että ensimmäiset testit tehtiin Javan versiolla 6 ja tämän jälkeen Java päivittyi versioon 7, mikä rikkoi osan testeistä. Tätä ei voida kuitenkaan pitää kovin yleisenä tapahtumana, joten tältä osin testityökalut olivat tasaveroisia. Kaikkien käytettyjen työkalujen testit kestäisivät ikkunan sisällön rakenteellisen muuttamisen, jos komponentit pysyvät näkyvissä, sillä niihin viitataan nimi-ominaisuuden avulla.

6 Johtopäätökset

Tutkimuksessa tutkittiin, minkälaisia työkaluja Javan graafisten käyttöliittymien testaamisen automatisointiin löytyy ja miten nämä eroavat toisistaan. Automatisointiin löytyi työkaluja, ja niistä löytyi eroja, mutta työkalut olivat kuitenkin verrattain samankaltaisia. Suurimpana erona oli, että toisissa toiminnot nauhoitettiin ja toisissa ne ohjelmoitiin. Tutkimuksessa jätettiin kuitenkin myös tarkemmin käsittelemättä useita työkaluja, jotta voitiin perehtyä tarkemmin käsiteltyjen eroihin. Käsittelemättä jäivät esimerkiksi WindowTester [23], jfcUnit [24], Jacareto [25] sekä UISpec4J [26].

Ainakin yksinkertaisten sovellusten testaamiseen Lift ja Fest olivat mukavimmat käyttää, koska komponenttien haku on näissä yksinkertaisempaa. Monimutkaisemmissa sovelluksissa, joissa komponentteja luodaan dynaamisesti, voi monimutkaisemmille hakuehdoille tulla käyttöä, mutta Lift ja Fest kuitenkin mahdollistivat myös nämä. Abbotin Komponenttien etsimisessä monimutkaisuutta tuo myös eksplisiittinen tyyppien pakottaminen, miltä vältyttiin Festiä ja Liftiä käytettäessä.

Testit eivät aiheuttaneet juurikaan muutoksia testattavaan ohjelmaan. Ainoana muutoksena oli, että testattaville komponenteille asetettiin name-ominaisuus komponenttien hakemisen helpottamiseksi. Tämä koettiin myös hieman työlääksi, koska name-ominaisuutta ei käytetty ohjelmasa muuten ja tämän luominen olisi myös mahdollista automatisoida kehittämisympäristön toimesta.

Yhtenä tutkimuksen motiiveista oli selvittää, pystytäänkö jotain näistä työkaluista käyttämään yhdessä ComTestin kanssa GUI-testauksen lisäämiseksi Ohjelmointi 2-kurssille. Vaikka selvitettiin, että tämä on mahdollista, ei tässä tutkimuksessa kuitenkaan pystytä antamaan vastausta siihen, onko tämä liian haastavaa tai työlästä kurssin opiskelijoille.

A VaihtoGui.java

```
package gui;

import java.awt.*;
import java.awt.event.*;
import java.util.Hashtable;

import javax.swing.*;
import javax.swing.border.*;
import javax.swing.event.*;

/**
 * Rahanvaihto-ohjelma
 *
 * Ohjelman tarkoituksena selvittää, paljonko annettu eurosumma
 * on dollareina ja puntina.
 *
 * @author Hekku2
 * @version 1.0
 */
public class VaihtoGui extends JFrame {

    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private final JButton sulje = new JButton("Sulje");
    private final JButton vaihda = new JButton("Vaihda");
    private final JTextField txtfield = new JTextField();
    private final JLabel lbl = new JLabel("0.00 $");
    private final JRadioButton rbPunta = new JRadioButton(
            "Punniksi (\u00A3)");
    private final JRadioButton rbDollari = new JRadioButton(
            "Dollareiksi($)");
    private final ButtonGroup valuuttavalinnat = new ButtonGroup();

    private double summa = 0;
    private Hashtable<Character, Double> kurssi =
            new Hashtable<Character, Double>();
```

```
/**
 * Käynnistää ohjelman.
 *
 * @param args
 *            Ei käytössä.
 */
public static void main(String[] args) {
    EventQueue.invokeLater(new Runnable() {
        @Override
        public void run() {
            try {
                VaihtoGui frame = new VaihtoGui();
                frame.setVisible(true);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    });
}

/**
 * Luo ikkunan.
 */
public VaihtoGui() {
    kurssi.put('$', 1.4519);
    kurssi.put('£', 0.8841);

    addWindowListener(new WindowAdapter() {
        @Override
        public void windowClosing(WindowEvent arg0) {
            avaaAbout();
            dispose();
        }
    });
    setTitle("Valuuttamuunnin");
    setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
    setBounds(100, 100, 254, 172);
    contentPane = new JPanel();
    contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
    setContentPane(contentPane);
    contentPane.setLayout(new GridLayout(0, 2, 0, 0));
```

```
txtfield.getDocument().addDocumentListener(
        new DocumentListener() {
            @Override
            public void changedUpdate(DocumentEvent e) {
                textChanged();
            }

            @Override
            public void removeUpdate(DocumentEvent e) {
                textChanged();
            }

            @Override
            public void insertUpdate(DocumentEvent e) {
                textChanged();
            }
        });
txtfield.setName("txtfield");
contentPane.add(txtfield);
txtfield.setAlignmentY(Component.TOP_ALIGNMENT);
txtfield.setBorder(UIManager.getBorder("TextField.border"));
txtfield.setBackground(Color.WHITE);
txtfield.setSelectedTextColor(Color.WHITE);

txtfield.setHorizontalAlignment(SwingConstants.TRAILING);
txtfield.setText("0.00");
txtfield.setColumns(10);
valuuttavalinnat.add(rbPunta);
rbPunta.setName("rbPunta");
contentPane.add(rbPunta);
rbPunta.setSelected(true);
rbPunta.setMnemonic('£');
lbl.setName("lbl");
contentPane.add(lbl);
lbl.setAlignmentX(Component.CENTER_ALIGNMENT);
lbl.setBorder(null);
lbl.setHorizontalAlignment(SwingConstants.TRAILING);
lbl.setMaximumSize(new Dimension(999999, 9999999));
valuuttavalinnat.add(rbDollari);
rbDollari.setName("rbDollari");
contentPane.add(rbDollari);
rbDollari.setMnemonic('$');
```

```
        vaihda.setName("vaihda");
        contentPane.add(vaihda);
        vaihda.setAlignmentX(Component.CENTER_ALIGNMENT);
        vaihda.setMargin(new Insets(2, 2, 2, 2));
        sulje.setName("sulje");

        sulje.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                dispose();
                avaaAbout();

            }
        });
        contentPane.add(sulje);
        vaihda.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                vaihda();
            }
        });
    }

    /**
     * Vaihtaa annetun rahamäärän valituksi rahamääräksi
     */
    private void vaihda() {
        char valuutta = (char) (valuuttavalinnat.getSelection()
            .getMnemonic());
        double kerroin = kurssi.get(valuutta);
        if (txtfield.getBackground().equals(Color.red)) {
            this.lbl.setText("Virheellinen syöte");
            return;
        }
        this.lbl.setText((summa * kerroin) + " " + valuutta);
    }

    /**
     * Tekstin vaihtuessa tarkistaa onko sisältö oikein
     */
    private void textChanged() {
        try {
```

```
            summa = Double.parseDouble(txtfield.getText());
            txtfield.setBackground(Color.white);
        } catch (Exception e) {
            txtfield.setBackground(Color.red);
        }
    }

    /**
     * Luo ja näyttää About-ikkunan
     */
    private void avaaAbout() {
        About a = new About();
        a.setVisible(true);
    }
}
```

B About.Java

```
package gui;

import java.awt.BorderLayout;
import java.awt.event.*;

import javax.swing.*;
import javax.swing.border.*;

/**
 * About-ikkuna
 *
 * Tarkoituksena "kiitos käytöstä"-teksti.
 *
 * @author Hekku2
 * @version 1.0
 */
public class About extends JFrame {

    private static final long serialVersionUID = 2191922410688036393L;
    private JPanel contentPane;
    private final JButton ok = new JButton("Sulje");
    private final JLabel kiitos = new JLabel(
```

```
            "Kiitos k\u00E4yt\u00F6st\u00E4!");

    /**
     * Luo ikkunan.
     */
    public About() {
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setBounds(100, 100, 186, 123);
        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        contentPane.setLayout(new BorderLayout(0, 0));
        setContentPane(contentPane);
        ok.setName("ok");
        ok.addActionListener(new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent arg0) {
                dispose();
            }
        });
        contentPane.add(ok, BorderLayout.SOUTH);
        kiitos.setName("kiitos");
        kiitos.setHorizontalAlignment(SwingConstants.CENTER);
        contentPane.add(kiitos, BorderLayout.CENTER);
    }
}
```

C Robot-testit

```
package tests;

import static org.junit.Assert.assertEquals;
import gui.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
import org.junit.Test;

/**
 * Javan Robotilla tehdyt testit
 *
```

```
 * @author Hekku2
 * @version 1.0
 */
public class RobotTest {

    /**
     * Jostain syystä numlockin ollessa pohjassa testi ei toimi,
     * koska shifti ei pysy pohjassa
     *
     * http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=4908075
     * http://bugs.sun.com/bugdatabase/view_bug.do?bug_id=6463168
     *
     * @throws AWTException
     *             Virhe robotin luomisessa
     */
    @Test
    public void testInit() throws AWTException {
        VaihtoGui gui = new VaihtoGui();

        // VaihtoGui gui = new VaihtoGui();
        gui.setVisible(true);

        // Etsitään komponentit
        Component[] komponentit = gui.getContentPane()
            .getComponents();

        JRadioButton rbPunta = null;
        JRadioButton rbDollari = null;
        JButton vaihda = null;
        JTextField vaihdettava = null;
        JLabel tulos = null;

        // Haetaan testattavat komponentit ikkunasta
        // asetetun nimen perusteella.
        for (int i = 0; i < komponentit.length; i++) {
            if (komponentit[i].getName().equals("rbPunta"))
                rbPunta = (JRadioButton) komponentit[i];
            if (komponentit[i].getName().equals("rbDollari"))
                rbDollari = (JRadioButton) komponentit[i];
            if (komponentit[i].getName().equals("vaihda"))
                vaihda = (JButton) komponentit[i];
            if (komponentit[i].getName().equals("txtfield"))
```

```
            vaihdettava = (JTextField) komponentit[i];
        if (komponentit[i].getName().equals("lbl"))
            tulos = (JLabel) komponentit[i];
    }

    Robot tester = new Robot();
    tester.setAutoDelay(100);

    assertEquals("0.00", vaihdettava.getText());
    assertEquals(true, rbPunta.isSelected());
    assertEquals(false, rbDollari.isSelected());

    click(vaihdettava, tester);

    tester.keyPress(KeyEvent.VK_M);
    assertEquals(Color.red, vaihdettava.getBackground());

    click(vaihda, tester);
    assertEquals("Virheellinen syöte", tulos.getText());

    // Tyhjennetään tekstikenttä
    click(vaihdettava, tester);
    tester.keyPress(KeyEvent.VK_SHIFT);
    tester.keyPress(KeyEvent.VK_END);
    tester.keyPress(KeyEvent.VK_DELETE);
    tester.keyRelease(KeyEvent.VK_DELETE);
    tester.keyRelease(KeyEvent.VK_END);
    tester.keyRelease(KeyEvent.VK_SHIFT);

    // Kirjoitetaan 2.00
    tester.keyPress(KeyEvent.VK_2);
    tester.keyRelease(KeyEvent.VK_2);
    tester.keyPress(KeyEvent.VK_PERIOD);
    tester.keyRelease(KeyEvent.VK_PERIOD);
    tester.keyPress(KeyEvent.VK_0);
    tester.keyRelease(KeyEvent.VK_0);
    tester.keyPress(KeyEvent.VK_0);
    tester.keyRelease(KeyEvent.VK_0);
    assertEquals(Color.white, vaihdettava.getBackground());

    // Katsotaan vaihtuuko labelin teksti oikein
    click(vaihda, tester);
```

```
    assertEquals("1.7682 £", tulos.getText());

    click(rbDollari, tester);
    click(vaihda, tester);
    assertEquals("2.9038 $", tulos.getText());

    gui.dispose();
}

/**
 * Testaa sulkuoperaation
 *
 * @throws AWTException
 *             Virhe robotin luomisessa.
 */
@Test
public void testClose() throws AWTException {
    VaihtoGui gui = new VaihtoGui();
    gui.setVisible(true);

    // Etsitään komponentit
    Component[] komponentit = gui.getContentPane().getComponents();
    JButton sulje = null;

    // Haetaan testattavat komponentit ikkunasta
    // asetetun nimen perusteella.
    for (int i = 0; i < komponentit.length; i++) {
        if (komponentit[i].getName().equals("sulje"))
            sulje = (JButton) komponentit[i];
    }
    Robot tester = new Robot();
    tester.setAutoDelay(200);

    click(sulje, tester);

    JLabel lbl = null;
    JButton ok = null;

    Window[] s = About.getOwnerlessWindows();
    Component c = s[2].getComponents()[0];

    ok = (JButton) find(c, "ok");
```

```
    lbl = (JLabel) find(c, "kiitos");

    assertEquals("Kiitos k\u00E4yt\u00F6st\u00E4!", lbl.getText());
    click(ok, tester);
    s[2].dispose();
    gui.dispose();
}

/**
 * Painaa hiiren vasemmalla näppäimellä komponenttia.
 *
 * @param component
 *            Komponentti, jota klikataan
 * @param tester
 *            Robotti, joka klikkaa
 */
public static void click(Component component, Robot tester) {
    tester.mouseMove((int) component.getLocationOnScreen().getX(),
            (int) component.getLocationOnScreen().getY());
    tester.mousePress(InputEvent.BUTTON1_MASK);
    tester.mouseRelease(InputEvent.BUTTON1_MASK);
}

/**
 * Etsii rekursiivisesti componenteistä ja sen lapsista annetun
 * komponentti.
 *
 * Jos löytyy useampi komponentti, palautetaan ensimmäinen
 * löydetty.
 *
 * @param comp
 *            komponentti, mistä etsitään nimellä
 * @param name
 *            nimi, jolla etsitään
 * @return null, jos ei löydy, muuten etsitty komponentti.
 */
public static Component find(Component comp, String name) {
    if (comp.getName() != null && comp.getName().equals(name))
        return comp;
    if (comp instanceof Container) {
        Component[] components = ((Container) comp).getComponents();
        for (int i = 0; i < components.length; i++) {
```

```
                Component c = find(components[i], name);
                if (c != null)
                    return c;
            }
        }
        return null;
    }
}
```

D Abbot-testit

```
package tests;

import gui.VaihtoGui;
import java.awt.Color;
import javax.swing.*;
import junit.extensions.abbot.ComponentTestFixture;
import org.junit.Test;
import abbot.finder.*;
import abbot.finder.matchers.*;
import abbot.tester.*;

/**
 * Abbot-testit
 *
 * @author Hekku2
 * @version 1.0
 */
public class AbbotTest extends ComponentTestFixture {

    /**
     * Yleistestit
     *
     * @throws MultipleComponentsFoundException
     *             Jos tulee useampia osumia
     * @throws ComponentNotFoundException
     *             Jos ei löydy komponenttia
     */
```

```
@Test
public void test() throws ComponentNotFoundException,
        MultipleComponentsFoundException {

    VaihtoGui gui = new VaihtoGui();
    showWindow(gui);

    JTextFieldTester txtTester = new JTextFieldTester();
    JTextField txtfield = (JTextField) getFinder().find(
            new ClassMatcher(JTextField.class));
    assertEquals("0.00", txtfield.getText());

    // Tässä ei voida käyttää ClassMatcheria,
    // koska tulisi useampia osumia.
    JRadioButton punta = (JRadioButton) getFinder().find(
            new NameMatcher("rbPunta"));
    JRadioButton dollari = (JRadioButton) getFinder().find(
            new NameMatcher("rbDollari"));
    assertEquals(true, punta.isSelected());
    assertEquals(false, dollari.isSelected());

    JButton vaihda = (JButton) getFinder().find(
             new NameMatcher("vaihda"));
    JLabel lbl = (JLabel) getFinder().find(
            new ClassMatcher(JLabel.class));

    txtTester.actionEnterText(txtfield, "M");
    assertEquals(Color.red, txtfield.getBackground());

    txtTester.click(vaihda);
    assertEquals("Virheellinen syöte", lbl.getText());

    txtTester.actionEnterText(txtfield, "2.00");
    assertEquals(Color.white, txtfield.getBackground());

    txtTester.click(vaihda);
    assertEquals("1.7682 £", lbl.getText());
    txtTester.click(dollari);
    txtTester.click(vaihda);
    assertEquals("2.9038 $", lbl.getText());
}
```

```
    /**
     * Sulkemisen testaus
     *
     * @throws MultipleComponentsFoundException
     *             Jos tulee useampia osumia
     * @throws ComponentNotFoundException
     *             Jos ei löydy komponenttia
     */
    @Test
    public void test2() throws ComponentNotFoundException,
            MultipleComponentsFoundException {
        VaihtoGui gui = new VaihtoGui();
        showWindow(gui);

        ComponentTester tester = new ComponentTester();

        JButton close = (JButton) getFinder().find(
             new NameMatcher("sulje"));
        tester.click(close);

        JLabel lbl = (JLabel) getFinder().find(
             new ClassMatcher(JLabel.class));
        assertEquals("Kiitos k\u00E4yt\u00F6st\u00E4!", lbl.getText());

        JButton ok = (JButton) getFinder()
                .find(new ClassMatcher(JButton.class));
        tester.click(ok);
    }
}
```

E Costello-testit

Costello-editorilla tehdyt testit. Testeihin lisätty rivinvaihdot.

```
<?xml version="1.0" encoding="UTF-8"?>
<AWTTestScript>
  <component class="javax.swing.JLayeredPane" id="JLayeredPane Instance"
        index="1" parent="JRootPane Instance" window="Valuuttamuunnin" />
  <component class="javax.swing.JPanel" id="JPanel Instance" index="0"
        parent="JLayeredPane Instance" window="Valuuttamuunnin" />
```

```
<component class="javax.swing.JRootPane" id="JRootPane Instance"
      index="0" parent="Valuuttamuunnin" />
<component class="gui.VaihtoGui" id="Valuuttamuunnin" root="true"
      title="Valuuttamuunnin" />
<component class="javax.swing.JLabel" id="lbl" index="2" name="lbl"
      parent="JPanel Instance" text="Virheellinen syöte"
      window="Valuuttamuunnin" />
<component class="javax.swing.JRadioButton" id="rbDollari" index="3"
      name="rbDollari" parent="JPanel Instance" text="Dollareiksi($)"
      window="Valuuttamuunnin" />
<component class="javax.swing.JRadioButton" id="rbPunta" index="1"
      name="rbPunta" parent="JPanel Instance" text="Punniksi (£)"
      window="Valuuttamuunnin" />
<component class="javax.swing.JTextField" id="txtfield" index="0"
      name="txtfield" parent="JPanel Instance"
      window="Valuuttamuunnin" />
<component class="javax.swing.JButton" id="vaihda" index="4"
      name="vaihda" parent="JPanel Instance" text="Vaihda"
      window="Valuuttamuunnin" />
<launch args="[]" class="gui.VaihtoGui"
      classpath="D:\Projektit\Kandi\KandiWorkspace\Kandi\bin\"
      desc="VaihtoGui" method="main" />
<sequence>
  <assert component="rbDollari" invert="true" method="isSelected" />
  <action args="txtfield,0" class="javax.swing.text.JTextComponent"
      method="actionClick" />
  <assert component="txtfield" method="getBackground"
      value="java.awt.Color[r=255,g=255,b=255]" />
  <action args="txtfield,m" method="actionKeyString" />
  <assert component="txtfield" method="getBackground"
      value="java.awt.Color[r=255,g=0,b=0]" />
  <action args="vaihda" class="javax.swing.AbstractButton"
      method="actionClick" />
  <assert component="lbl" method="getText"
      value="Virheellinen syöte" />
  <action args="txtfield,m0.00" class="javax.swing.text.JTextComponent"
      desc="Click(txtfield)" method="actionClick" />
  <action args="txtfield,0,2" class="javax.swing.text.JTextComponent"
      method="actionSelectText" />
  <action args="txtfield,2" method="actionKeyString" />
  <assert component="txtfield" method="getBackground"
      value="java.awt.Color[r=255,g=255,b=255]" />
```

```
    <action args="vaihda" class="javax.swing.AbstractButton"
        method="actionClick" />
    <assert component="lbl" method="getText" value="1.7682 £" />
    <action args="rbDollari" class="javax.swing.AbstractButton"
        method="actionClick" />
    <action args="vaihda" class="javax.swing.AbstractButton"
        method="actionClick" />
    <assert component="lbl" method="getText" value="2.9038 $" />
  </sequence>
  <terminate />
</AWTTestScript>

<?xml version="1.0" encoding="UTF-8"?>
<AWTTestScript>
  <component class="gui.About" id="About Instance" root="true"
        title="" />
  <component class="javax.swing.JLayeredPane"
        id="JLayeredPane Instance"
        index="1" parent="JRootPane Instance"
        window="Valuuttamuunnin" />
  <component class="javax.swing.JPanel" id="JPanel Instance"
        index="0"
        parent="JLayeredPane Instance" window="Valuuttamuunnin" />
  <component class="javax.swing.JRootPane" id="JRootPane Instance"
        index="0"
        parent="Valuuttamuunnin" />
  <component class="gui.VaihtoGui" id="Valuuttamuunnin" root="true"
        title="Valuuttamuunnin" />
  <component class="javax.swing.JLabel" id="kiitos" index="1"
        name="kiitos"
        parent="JPanel Instance" text="Kiitos käytöstä!"
        window="About Instance" />
  <component class="javax.swing.JButton" id="ok" index="0"
        name="ok"
        parent="JPanel Instance" text="Sulje"
        window="About Instance" />
  <component class="javax.swing.SwingUtilities$SharedOwnerFrame"
        id="shared frame" root="true" title="" />
  <component class="javax.swing.JButton" id="sulje" index="5"
        name="sulje"
        parent="JPanel Instance" text="Sulje"
        window="Valuuttamuunnin" />
```

```
  <launch args="[]" class="gui.VaihtoGui"
        classpath="D:\Projektit\Kandi\KandiWorkspace\Kandi\bin\"
        desc="VaihtoGui" method="main" />
  <sequence>
    <action args="sulje" class="javax.swing.AbstractButton"
        method="actionClick" />
    <assert component="kiitos" method="getText"
        value="Kiitos käytöstä!" />
    <assert component="About Instance" method="isShowing" />
    <action args="ok" class="javax.swing.AbstractButton"
        method="actionClick" />
  </sequence>
  <terminate />
</AWTTestScript>
```

F Lift-testit

```
package tests;

import gui.VaihtoGui;
import java.awt.*;
import javax.swing.*;

/**
 * VaihtoGui-luokalle tehdyt testit liftillä
 * @author Hekku2
 * @version 1.0
 */
public class LiftTest extends student.GUITestCase {

    /**
     * Yleistestit
     */
    public void testInit(){
        VaihtoGui gui = new VaihtoGui();
        showWindow(gui);

        JTextField txtfield = getComponent(JTextField.class);
        assertEquals("0.00", txtfield.getText());
        JRadioButton punta = getComponent(JRadioButton.class,
```

```
        "rbPunta");
    JRadioButton dollari = getComponent(JRadioButton.class,
        "rbDollari");
    assertEquals(false, dollari.isSelected());
    assertEquals(true, punta.isSelected());

    JButton vaihda = getComponent(JButton.class, "vaihda");
    JLabel lbl = getComponent(JLabel.class);


    enterText(txtfield, "M");
    assertEquals(true, txtfield.getBackground().equals(Color.red));
    click(vaihda);
    assertEquals("Virheellinen syöte", lbl.getText());

    enterText(txtfield, "2.00");
    assertEquals(true, txtfield.getBackground().equals(Color.white));

    click(vaihda);
    assertEquals("1.7682 £", lbl.getText());
    click(dollari);
    click(vaihda);
    assertEquals("2.9038 $", lbl.getText());
}

/**
 * Testaa sulkuoperaation
 */
public void testClose(){
    VaihtoGui gui = new VaihtoGui();
    showWindow(gui);

    JButton close = getComponent(JButton.class, "sulje");
    click(close);

    JLabel lbl = getComponent(JLabel.class);
    assertEquals("Kiitos k\u00E4yt\u00F6st\u00E4!", lbl.getText());
    JButton ok = getComponent(JButton.class);
    click(ok);

    assertEquals(1, getAllComponentsMatching(Component.class)
        .size());
```

```
    }
}
```

G Fest-testit

```
package tests;

import java.awt.Color;
import javax.swing.*;
import gui.VaihtoGui;
import org.fest.swing.core.BasicComponentFinder;
import org.fest.swing.core.ComponentFinder;
import org.fest.swing.edt.GuiActionRunner;
import org.fest.swing.edt.GuiQuery;
import org.fest.swing.fixture.FrameFixture;
import org.fest.swing.junit.testcase.FestSwingJUnitTestCase;
import org.junit.*;

import static org.fest.assertions.Assertions.assertThat;

/**
 * Fest-testit
 * @author Hekku2
 * @version 1.0
 */
public class FestTestit extends FestSwingJUnitTestCase {
    private FrameFixture window;

    @Before
    public void onSetUp() {
        VaihtoGui frame = GuiActionRunner.execute(
            new GuiQuery<VaihtoGui>() {
               protected VaihtoGui executeInEDT() {
                  return new VaihtoGui();
               }
            });
        window = new FrameFixture(robot(), frame);
        window.show();
      }
```

```
/**
 * Yleistesti
 */
@Test
public void testYleistesti() {
    window.textBox("txtfield").requireText("0.00").selectAll()
        .enterText("M00.00");
    window.radioButton("rbPunta").requireSelected();
    window.radioButton("rbDollari").requireNotSelected();
    window.textBox("txtfield").background()
        .requireEqualTo(Color.red);

    window.button("vaihda").click();
    window.label().requireText("Virheellinen syöte");
    window.textBox("txtfield").requireText("M00.00").selectAll()
        .enterText("2.00");
    window.textBox("txtfield").background()
        .requireEqualTo(Color.white);

    window.button("vaihda").click();
    window.label().requireText("1.7682 £");
    window.radioButton("rbDollari").click();
    window.button("vaihda").click();
    window.label().requireText("2.9038 $");
    window.button("sulje").click();
}

/**
 * Sulkemisen testaus
 */
@Test
public void testAukeaminen() {
    ComponentFinder finder =
         BasicComponentFinder.finderWithNewAwtHierarchy();
    window.button("sulje").click();
    window.requireNotVisible();
    JLabel teksti = finder.findByType(JLabel.class);
    assertThat(teksti.getText()).isSameAs(
          "Kiitos k\u00E4yt\u00F6st\u00E4!");
    JButton nappula = finder.findByType(JButton.class);
    this.robot().click(nappula);
```

```
    }
}
```

H Comtest

Tähän laitettu pelkät kommentit, joista testit generoidaan.

```
    /**
     * Luo ikkunan.
     * @example
     * <pre name="test">
     * #import javax.swing.*;
     * #import java.awt.Color;
     * comtest.GUITester g = new comtest.GUITester();
     * g.showWindow(new VaihtoGui());
     *
     * JTextField txtfield = g.getComponent(JTextField.class);
     * txtfield.getText() === "0.00"
     * JRadioButton punta = g.getComponent(JRadioButton.class,
         "rbPunta");
     * JRadioButton dollari = g.getComponent(JRadioButton.class,
         "rbDollari");
     *
     * dollari.isSelected() === false
     * punta.isSelected() === true
     *
     * JButton vaihda = g.getComponent(JButton.class, "vaihda");
     * JLabel lbl = g.getComponent(JLabel.class);
     *
     * g.enterText(txtfield, "M");
     * txtfield.getBackground() === Color.red
     * g.click(vaihda);
     * lbl.getText() === "Virheellinen syöte"
     *
     * g.enterText(txtfield, "2.00");
     * txtfield.getBackground() === Color.white
     *
     * g.click(vaihda);
     * lbl.getText( ) === "1.7682 £"
```

```
 * g.click(dollari);
 * g.click(vaihda);
 * lbl.getText() === "2.9038 $"
 * </pre>
 */

/**
 * Luo ja näyttää About-ikkunan
 * @example
 * <pre name="test">
 * #import javax.swing.*;
 * #import java.awt.Color;
 * #import java.awt.Component;
 * comtest.GUITester g = new comtest.GUITester();
 * g.showWindow(new VaihtoGui());
 *
 * JButton close = g.getComponent(JButton.class, "sulje");
 * g.click(close);
 *
 * JLabel lbl = g.getComponent(JLabel.class);
 * lbl.getText() === "Kiitos k\u00E4yt\u00F6st\u00E4!"
 * JButton ok = g.getComponent(JButton.class);
 * g.click(ok);
 *
 * g.getAllComponentsMatching(Component.class).size() === 1
 * </pre>
 */
```

Lähteet

[1] Kent Beck, *Test-driven development: by example*, Addison-Wesley Professional, 2003

[2] Atif M. Memon, *A Comprehensive framework for testing graphical user interfaces*, University of Pittsburgh, 2001.

[3] Jason Snyder, Stephen H. Edwards, Manuel A. Pérez-Quiñones *LIFT: Taking GUI Unit Testing to New Heights*, SIGCSE 2011

[4] Alex Ruiz, Yvonne Wang Price *GUI Testing Made Easy*
Practice and Research Techniques, 2008. TAIC PART '08. Testing: Academic & Industrial Conference

[5] Taeksu Kim, Chanjin Park ja Chisu Wu *Mock Object Models for Test Driven Development* Software Engineering Research, Management and Applications, 2006. Fourth International Conference on Software Engineering Research, Management and Applications

[6] Theodore D. Hellmann, Ali Hosseini-Khayat, Frank Maurer *Supporting Test-Driven Development of Graphical User Interfaces Using Agile Interaction Design* Software Testing, Verification, and Validation Workshops (ICSTW), 2010 Third International Conference on Software Testing, Verification, and Validation Workshops

[7] Alex Ruiz and Yvonne Wang Price *Test-Driven GUI Development with TestNG and Abbot* IEEE Software 24, Issue 3, 51-57

[8] Javan Robot-luokan dokumentaatio
`http://download.oracle.com/javase/1,5.0/docs/api/java/awt/Robot.html`,
katsotty 5.22.2011

[9] JFC Usein kysytyt kysymykset
`http://java.sun.com/products/jfc/reference/faqs/index.html`,
katsottu 5.11.2011

[10] What is Swing?
`http://download.oracle.com/javase/tutorial/ui/overview/intro.html`,
katsottu 5.11.2011

[11] Ohjelmointi 2-kurssin kotisivu
`http://users.jyu.fi/~vesal/kurssit/ohjelmointi2011/`,
katsottu 5.11.2011

[12] Abbot API
`http://abbot.sourceforge.net/doc/api/`

overview-summary.html,
katsottu 5.11.2011

[13] Abbot lähdekoodi
http://sourceforge.net/projects/abbot/files/abbot/1.2/,
katsottu 5.11.2011

[14] ComponentReference-dokumentaatio
http://abbot.sourceforge.net/doc/api/abbot/script/
ComponentReference.html,
katsottu 5.11.2011

[15] Script-paketin dokumentaatio
http://abbot.sourceforge.net/doc/api/abbot/script/
package-summary.html,
katsottu 5.11.2011

[16] Shift-bugin bugiraportti
http://bugs.sun.com/bugdatabase/
view_bug.do?bug_id=4908075,
katsottu 28.11.2011

[17] GUITestCasen api-dokumentaatio
http://courses.cs.vt.edu/ cs1114/api/student/
GUITestCase.html,
katsottu 29.11.2011

[18] Vesa Lappalainen, Jonne Itkonen, Ville Isomöttönen, Sami Kollanus ComTest: A Tool to Impart TDD and Unit Testing to Introductory Level Programming, in proceedings of the fifteenth annual conference on Innovation and technology in computer science education, 2010
http://dl.acm.org/citation.cfm?doid=1822090.1822110,
katsottu 16.12.2011

[19] QuickCheck-työkalun kotisivu
http://www.cse.chalmers.se/ rjmh/QuickCheck/

[20] Windowsien mukana tuleva Paint-ohjelma
http://windows.microsoft.com/fi-FI/windows7/

products/features/paint,
katsottu 1.3.2012

[21] Testing that access to Swing components is done in the EDT
http://docs.codehaus.org/display/FEST/
Testing+that+access+to+Swing
+components+is+done+in+the+EDT,
katsottu 5.3.2012

[22] Fest and Swing's Event Dispatch Thread (EDT)
http://docs.codehaus.org/pages/viewpage.action?
pageId=117899725,
katsottu 5.3.2012

[23] WindowTester Pro User Guide
http://code.google.com/intl/fi-FI/javadevtools/
wintester/html/index.html,
katsottu 5.3.2012

[24] Introduction to jfcUnit
http://jfcunit.sourceforge.net/,
katsottu 5.3.2012

[25] Jacareto Homepage
http://sourceforge.net/projects/jacareto/
, katsottu 5.3.2012

[26] UISpec4J: Java/Swing GUI testing made simple!
http://www.uispec4j.org/,
katsottu 5.3.2012

[27] Fest: Reflection Module
http://docs.codehaus.org/display/FEST/Reflection+Module,
katsottu 18.6.2012

[28] Fixtures for Easy Software Testing
http://code.google.com/p/fest/,
katsottu 18.6.2012

[29] IE9 Uudistukset
http://blogs.msdn.com/b/ie/archive/2010/03/16/html5-hardware-accelerated-first-ie9-platform-preview-available-for-developers.aspx,
katsottu 21.6.2012

[30] V8-projektisivu
http://code.google.com/p/v8/, katsottu 21.6.2012

[31] SpiderMonkey-projektisivu
https://developer.mozilla.org/en/SpiderMonkey,
katsottu 21.6.2012

[32] Safari features
http://www.apple.com/fi/safari/features.html,
katsottu 21.6.2012

[33] Lesson: Concurrency in Swing
http://docs.oracle.com/javase/tutorial/uiswing/concurrency/,
katsottu 22.6.2012

[34] Selenium Documentation
http://seleniumhq.org/docs/03_webdriver.html,
katsottu 23.6.2012