Samuli Kärkkäinen

Automaattisen GUI-testauksen perusteet

Tietotekniikan

kandidaatintutkielma

19. joulukuuta 2012

Jyväskylän yliopisto

Tietotekniikan laitos

Tekijä: Samuli Kärkkäinen

Yhteystiedot: samuli.p.p.karkkainen@jyu.fi

Ohjaajat: Anneli Heimbürger ja Sanna Mönkölä

Työn nimi: Automaattisen GUI-testauksen perusteet

Title in English: Basics of automated GUI testing

Työ: Kandidaatin tutkielma

Suuntautumisvaihtoehto: Ohjelmisto- ja tietoliikennetekniikka

Sivumäärä: 32+0

Tiivistelmä: Testaus on tärkeä osa ohjelmistokehitystä. Sovellusten toiminnallisuuteen ja laadunvalvontaan keskittyviä testaajia kuormitetaan valtavasti ja heidän olisikin syytä pystyä keskittymään oikeanlaisiin asioihin. Tämän tutkielman tarkoituksena on esitellä käyttöliittymän päällä toimivaa automaattista testausta, joka on yksi ratkaisu helpottamaan muun muassa regressiotestausta. Tutkielmassa tuodaan esille automaattisen testauksen hyötyjä ja sen asettamia haasteita sovelluksille sekä itse automatisointityökaluille. Tutkielmassa myös esitellään yhden nauhoita ja toista –sovelluksen, TestCompleten, ominaisuuksia. Tutkielmassa pyritään tuomaan esille TestCompleten etuja ja sen ratkaisuja automatisoinnin ongelmiin.

Avainsanat: Testaus, automaattinen testaus, regressiotestaus, käyttöliittymä, nauhoita ja toista –työkalut, aineisto-ohjattu testaus, TestComplete.

Abstract: Testing is an important part of software development. Testers who test software's usability and quality have a lot of work to do and they should be able to focus on the right things in their projects. This bachelor's thesis introduces one concept, automated GUI testing which could be one solution to make testing, for example regression testing, easier to execute. This thesis points out the pros of automated testing and explains the difficulties of automation. It also tries to explain the requirements for the capture and replay –

tools and finally introduces one of those tools, TestComplete by SmartBear Software, focusing on its advantages and solutions for the problems of automated testing.

Keywords: Testing, automated testing, regression testing, Graphical User Interface, capture and replay tools, data-driven testing, TestComplete.

Sisältö

1 Johdanto

Nykyään iso osa sovelluksista on todella laajoja ja kattavia, joten testaaminen vie paljon aikaa. Testaajia on kuitenkin rajoitetusti. Olisikin hyvä, jos testaajat saisivat keskittyä testaamaan olennaisia osia testattavasta sovelluksesta, eivätkä he joutuisi käyttämään kallista aikaansa siihen, että he jokaisen muutoksen jälkeen testaisivat asioita, jotka todennäköisesti eivät ole muuttuneet.

Graafinen käyttöliittymä (GUI, Graphical User Interface) on käsitteenä tuttu kaikille. Esimerkiksi The Linux Information Project (2004) määrittelee sen sellaiseksi tietokone-ihminen käyttöliittymäksi, jossa käytetään ikkunoita, ikoneita ja menuja navigointiin, joka tapahtuu hiirellä ja rajoitetuilla näppäinkomennoilla. Tässä tutkielmassa esitellään juuri käyttöliittymän päällä toimivaan automaattista GUI-testausta. Tällaisella automaattisella testauksella tarkoitetaan käyttäjän liikkeitä ja toimia mallintavaa testaustyökalun suorittamaa testausta, jonka yhteydessä tarkastetaan niin testattavan ohjelman toiminnallisuutta kuin syötteisiin reagointiakin. Tutkielma esittelee automaatisen testauksen mahdollisuuksia, joita ei hyödynnetä vielä riittävästi. Tietysti automaattiseen testaukseen liittyy myös haasteita ja tietynlaisia vaatimuksia, joita myös tutkielmassa tuodaan esille.

Tutkielmassa pyritään selvittämään, miten GUI-automatisointi tulisi toteuttaa. Tarkoituksena on kartoittaa millaisia ratkaisuja GUI-automatisointiin on esitetty ja miten tällaista automatisointia kannattaa lähestyä. Tutkielmassa haetaan vastausta myös siihen, missä vaiheessa manuaalisen testin muuttamista automatisoiduksi kannattaa harkita.

Automaattista GUI-testausta tehdään yleensä niin sanotuille nauhoita ja toista –työkaluilla (Capture and replay –tool). Koska nämä testaustyökalut ovat oleellinen osa automaattista testausta, niin tutkielmassa keskitytään myös sellaisiin ominaisuuksiin sekä vaatimuksiin, joita nauhoita ja toista –työkaluille voidaan nykypäivän automaattista GUI-testausta suunniteltaessa asettaa. Tutkielmassa esitellään tarkemmin erään tällaisen nauhoita ja toista –työkalun, TestCompleten, toimintaa peilaten sen ominaisuuksia niihin vaatimuksiin, joita tutkielmassa löydetään testaustyökaluille.

Automaattisen testauksen ja testien nauhoittamisen kannalta on tärkeä ymmärtää käyttöliittymäkomponentin käsite. Tällä tarkoitetaan yksittäistä graafisen käyttöliittymän elementtiä, jolla on sovelluksen toiminnan kannalta jokin tarkoitus. Muun muassa tekstikentät, sekä muokattavat että staattiset, ja erilaiset painikkeet ovat käyttöliittymäkomponentteja. Näillä komponenteilla on kaikilla tyypistään riippuen tiettyjä ominaisuuksia, kuten nimi, korkeus, leveys sekä mahdollisesti erilaisia tapahtumia, jotka tapahtuvat, kun niihin vaikutetaan esimerkiksi painamalla painiketta tai muokkaamalla jonkun kentän arvoa. Näiden komponenttien avulla toteutetaan automaattista testausta, kertoen testaustyökaluille mitä ja miten eri komponenttien kanssa tulee vaikuttaa haluttuun lopputulokseen pääsemiseksi.

2 Automaattiseen GUI-testaukseen liittyvää kirjallisuutta

Tutkielmassa sovellustestaus on oleellisessa osassa. Testausta on tutkittu paljon ja siitä on olemassa useita, hyviä teoksia. Glenford J. Myersin, Tom Badgettin ja Corey Sandlerin teos ”The Art of Software Testing” on erittäin kattava testausta esittelevä kirja, jonka kolmas versio on julkaistu vuonna 2012. Myös varsinaista automaattista testausta käsitteleviä teoksia on paljon. On olemassa paljon erilaisia lähdekoodin yhteyteen kirjoitettaviin testeihin (code-driven testing) liittyviä teoksia sekä jonkun verran erilaisiin neuroniverkkotesteihin liittyviä artikkeleja. Näitä aiheita tutkielma ei kuitenkaan esittele, vaan tutkielmassa pyritään pureutumaan nimenomaan käyttöliittymän päällä toimivaan GUI-testaukseen, jota on tutkittu yllättävän vähän.

Eräs varteenotettava teos on "Automated GUI performance testing", jonka ovat kirjoittaneet Andrea Adamoli, Dmitrijs Zaparanuks, Milan Jovic sekä Matthias Hauswirth. Se käsittelee nimenomaan GUI-testausta, esitellen sen haasteita ja ongelmia. Lisäksi artikkelissa on testattu viittä erilaista ilmaita nauhoita ja toista –työkalua, joilla pystytään totetuttamaan automaattista GUI-testausta. Valitettavasti teoksessa esitellyt testit painottuvat Java-ohjelmointikielelle, mutta toimintatavat ja vaatimukset ovat kuitenkin yleistettävissä muillekin ohjelmointikielille. Artikkeli pitää sisällään myös kirjoittajan mielestä parhaan mallin suunnitella ja toteuttaa automaattista testausta, sisältäen myös useita havainnollisa taulukoita aiheeseen liityen.

Tuore artikkeli "A Systematic Capture and Replay Strategy for Testing Complex GUI based Java Applications", kirjoittajinaan Omar El Ariss, Dianxiang Xu, Santosh Dandey, Brad Vender, Phil McClean ja Brian Slator, käsittelee edellisen artikkelin tapaan automaattista testausta Java-sovelluksiin. Tämä artikkeli esittelee kehyksen käyttää nauhoita ja toista –testausta käyttöliittymäläheiseen testaukseen, funktionaaliseen testaukseen sekä regressiotestaukseen käyttäen hyödyksi joko luotettua, toimivaa järjestelmää tai valmiita määrityksiä. Artikkelissa on lisäksi esitelty, miten heidän esittelemäänsä tapaa käytetään sovelluksen testaukseen.

Myös artikkeli "A Graphical User Interface (GUI) Testing Methodolody" tarjoaa hyviä näkökulmia automaattiseen testaukseen. Artikkelin ovat kirjoittaneet Zafar Singhera, Ellis Horowitz sekä Abad Shah. Tämäkin artikkeli käsittelee automaattista testausta tuoden uusia näkemyksiä aiheeseen. Siinä esitellään ohjeita GUI-testaukseen sekä esitellään menetelmiä toteuttaa automaattista testausta. Lisäksi siinä kuvataan, kuinka menetelmiä sovelletaan erääseen sovellukseen.

TestComplete-sovellusta on tutkittu vähän. Artikkelissaan “Comparative Study of automated Testing Tools: TestComplete and QuickTest Pro” Manjit Kaur ja Raj Kumari vertailevat TestCompletea sekä toista maksullista nauhoita ja toista –sovellusta, Quick Test Prota ottamalla kantaa muun muassa nahoituksen tehokkuuteen, scriptien tuottamiseen sekä automatisoitujen testien toistamisen nopeuteen. Valitettavasti TestCompleten aivan uusinta versiota ei vertailussa ollut, mutta melko tuore versio kuitenkin. Artikkeli on julkaistu vuonna 2011 ja uusin versio TestCompletesta, TestComplete 9 julkaistiin vuonna 2012.

3 Testaus

Tässä luvussa esitellään avainkäsitteitä, joiden ymmärtäminen on välttämätöntä automaattisen GUI-testauksen kannalta. Tutkielmassa keskitytään vain sovelluksen toiminnallisuuden testaamiseen. Erilaisen käyttäjäläheisten toteutuksen tai sovelluksen tehokkuuden testaaminen ei kuulu tutkielman sisältöön, eikä niitä automaattisella GUI-testauksella varsinaisesti pystytäkään testaamaan.

Testaus lienee käsitteenä kaikille tuttu. Sille löytyykin useita määritelmiä, mutta yleensä näiden määritelmien sisältö on samantyyppinen. Testaus kuitenkin on lopulta juuri sitä, että sovelluksesta pyritään löytämään ja sitä kautta poistamaan virheet (Myers, Badgett & Sandler 2012).

3.1 Testitapaus

Testaamiseen liittyvät oleellisesti testitapaukset (test case). Ne ovat kehyksiä testeille ja niiden rakenne on yleensä sellainen, että ensin rakennetaan tietty ympäristö, jonka jälkeen suoritetaan työvaiheet ja lopulta tarkastetaan, että halutut tulokset on saavutettu (IEEE Standard for Test Documentation 1998).

Testitapauksista rakentuu yleensä kattava kokonaisuus ohjelman testaamista varten. Testitapaukset suunnitellaankin yleensä projektiryhmän kanssa ja niissä keskitytään kattamaan koko sovelluksn toiminta siten, että mitään oleellista ei jää testaamatta. On suositeltavaa, että valmiita testitapauksia ei heitettäisi pois, vaikka sillä hetkellä testattava sovellus toimisi, kuten pitääkin (Myers ym. 2012). Testitapauksia pystytään käyttämään hyväksi regressiotestauksen yhteydessä.

3.2 Regressiotestaus

Regressiotestaus on testausta, joka suoritetaan aina, kun ohjelmaa on muutettu. Joskus ohjelmaa muokattaessa tapahtuu ymmärrettävästi virheitä. Nämä virheet voivat vaikuttavaa välillisesti johonkin toiseen ohjelman osaan ilman, että virhe paljastuu itse

muutetun osion toiminnallisuutta varmistaessa. Regressiotestauksella pyritään varmistamaan, että tällaisia virheitä ei ole tullut mihinkään osaan testattavaa sovellusta suorittamalla sovelluksen oletettavasti muuttumattomien osien testitapauksia (Myers ym. 2012).

3.3 Savutestaus

Sovellustestauksessa savutestauksella (smoke testing) tarkoitetaan sitä, että varmistutaan, että sovellus toimii, eikä siinä ole mitään vakavia, sovellusta kaatavia ongelmia. Tällaisessa testauksessa ei ole erityistä laajamittaista testaussuunnitelmaa vaan sillä pyritään yksinkertaisesti varmistamaan sovelluksen ja sen eri osien toiminnallisuus (Memon, Nagarajan & Xie 2005).

Automaattinen testaus palvelee savutestausta. Kun testaussovellus suorittaa käyttäjän toimia, paljastuvat samalla ohjelman vaaralliset virheet. Nämä virheet voidaan automaattisen testauksen jälkeen raportoida suoraan kehittäjille, jotka pystyvät korjaamaan virheet ilman, että jouduttaisiin aloittamaan uutta testaussykliä manuaalisessa testissä havaitun virheen perusteella.

4 Automaattinen GUI-testaus

Tässä luvussa esitellään tarkemmin käyttöliittymän päällä toimivaa automaattista testausta. Luvussa esitellään ensin suositeltuja toimintatapoja sille, miten automaattista GUI-testausta kannattaa lähestyä. Tämän jälkeen kerrotaan hyväksi havaittuja toimintatapoja toteuttaa tällaista testausta. Lopuksi tuodaan esille automaattisen GUI-testauksen haasteita ja tiettyjä vaatimuksia sekä testattavalle sovellukselle että valitulle testausohjelmalle keskittyen erityisesti nauhoita ja toista –työkaluihin. Tässä yhteydessä täytyy muistaa, että testaustyökalu on syytä valita tarkasti käyttötarkoitusta varten. Usea työkalu tukee vain tiettyjen alustojen tiettyä ohjelmointikieltä ja esimerkiksi web-sovelluksille on olemassa omia testityökaluja, joita ei voi soveltaa työpöytäsovelluksiin.

4.1 Idea

Käyttöliittymän päällä toimiva automaattinen testaus suorittaa tietyt käyttäjän toiminnot automaattisesti. Näitä toimintoja voivat olla muun muassa hiiren liikkeet sekä näppäinsyötteet. Automaattisessa testaamisessa olisi hyvä myös olla erilaisia tarkastuspisteitä, joissa pystytään joko lukemaan jonkun käyttöliittymäkomponentin arvo tai tarkastamaan jopa tietokannasta tai muista tietueista, kuten tulostettavista tiedostoista, että toiminnot ovat onnistuneita. Automaattiset testit kirjoitetaan tai nauhoitetaan yleensä scripteihin, joissa niitä päästään muokkaamaan ja päivittämään haluttuun suuntaan. Myös testien tarkentaminen onnistuu näillä scripteillä.

Automaatitisen testauksen päämääränä on eliminoida ihmisen tekemän työn tarve testauksen aikana suorittaen automaattiesesti käyttäjän toimintoja (Adamoli ym. 2011). Käyttäjän itse suorittamaan testaukseen verrattuna automaatitinen testaus suoriutuu McCaffreyn (2006) mukaan paremmin seuraavissa asioissa:

- Automaattiset testit ovat nopeampia (Speed)
- Automaattisissa testeissä on vähemmän virheitä (Accuracy)
- Automaattiset testit ovat tarkempia (Precision)
- Automaattiset testit ovat tehokkaampia (Effigency)

- Automaattiset testit kehittävät niiden tekijää enemmän, kuin manuaalinen testaus (Skill-building)

Testin toistaminen on merkittävästi nopeampaa kuin manuaalisessa testauksessa. Kun testi on hyvin toteutettu, eri automatisoitujen tapahtumien välinen viive saattaa olla vain sekunnin murto-osia. Esimerkiksi teksti lisätään kenttiin kokonaisena merkkijonona ja hiiren liikkeitä ei tarvitse suorittaa, vaan klikkaaminen tapahtuu automaattisesti.

Automaattisessa testauksessa suurin työ tehdään testien suunnittelussa ja toteuttamisessa. Artikkelissa "Improving the Maintainability of Automated Test Suites" Cem Kaner (1997) arvioi yhden testitapauksen automatisointiin kuluvan noin 3-10 kertaa sen ajan, joka kuluisi koko testitapauksen suorittamiseen manuaalisesti yhden kerran. Tämä on kuitenkin vanha arvio ja sen paikkansapitävyys on saattanut vuosien saatossa muuttua. Kuitenkin automaattiset testit tekevät testauksesta merkittävästi tehokkaampaa ja nopeampaa, kunhan ne suunnitellaan kunnolla ja automatisoitavaksi valitaan testitapauksia, joita tullaan todennäköisesti ajamaan monta kertaa. Pelkästään muutamaa testikertaa varten automaattista testiä ei kannata suunnitella.

Automaattisilla testeillä pystytään myös luomaan kattavia testitapauksia, jolloin testattavaan sovellukseen jää vähemmän virheitä. Sovelluksesta voidaan testata melko isoja kokonaisuuksia joiden yhteydessä automaattiset testit havaitsevat virhetilanteita, joita manuaalisella testillä ei ehkä sillä kerralla havaittaisi. Esimerkkinä toimii regressiotestaus, jossa manuaalisesti testattaessa ei välttämättä avattaisi jotain dialogia lainkaan, vaan virhe löytyisi automaattisen testin yhteydessä.

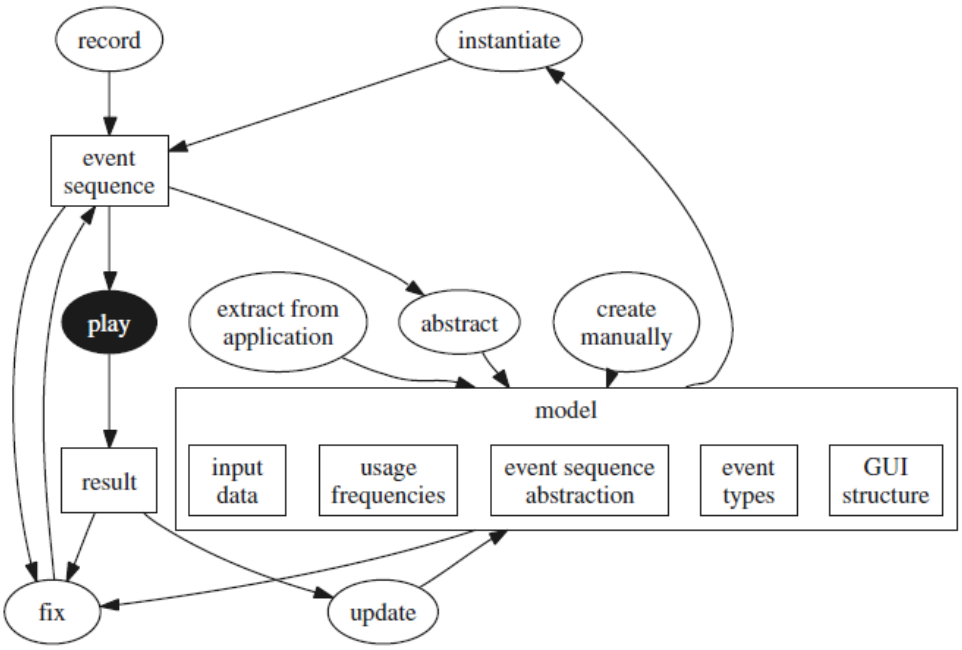
Tarkkuus korostuu myös erilaisissa vertailuissa. Jos käyttöliittymäkomponentissa on esimerkiksi rajoitus jollekin syötteen pituudelle, voidaan kehittyneellä testaustyökalulla tarkastaa aivan tietokannasta asti se, että koko merkkijono on oikein. Lisäksi pitkien lukujen vertailu on automaattisessa testauksessa huomattavasti helpompaa, koska laskeminen tapahtuu automaattisesti, eikä käyttäjän tarvitse huolehtia esimerkiksi pienistä laskuvirheistä. Tässäkin esimerkissä sivutaan myös automaattisten testien vaikutusta testauksen nopeuteen.

Fewsterin (1999) mukaan automatisoituja testejä käyttämällä voi kustannuksissa säästää jopa 80% verrattuna manuaaliseen testaukseen (Pei, Yuewei, Wu & Wu 2009). Tämä arvio en esitetty vuonna 1999, joten sekin saattaa olla hieman vanhentunut. Säästöjä automaattisella testuksella kuitenkin saa aikaiseksi, jos kiinnittää huomioita testeihin, joita automatisoidaan.

4.2 Toimintatavat

Tekstissään Adamoli ym. (2011) esittelevät automaattisen testausmallin pääpiirteet. He ovat keränneet malliinsa vaikutteita useasta eri lähteestä ja pyrkivät omassa mallissaan kokoamaan useiden eri kirjoittajien valinnoista parhaat puolet esiin. Kuvassa 1 on esitelty automaattisen testauksen suunnitteluprosessi. Malli koostuu kahdenlaisista solmuista. Ellipsit ovat aktiviteettejä ja suorakulmiot kuvaavat datan keräämistä tai tallennusta. Kuvassa 1 esitetyt tilat ovat seuraavat:

- Tapahtumajärjestys (Event Sequence)
- Nauhoitus (Record)
- Malli (Model)
- Intstanssointi (Instantiate)
- Toisto (Play)
- Tulos (Result)
- Korjaa (Fix)
- Päivitä (Update)



Kuva 1: Automaattisten testien suunnitteluprosessi (Adamoli ym. 2011)

Tapahtumajärjestys oleellinen osa automaattista testausta. Tilaan on tallennettu tapahtumajärjestys, joka automaattisessa testauksessa halutaan suorittaa. Tapahtumajärjestyksen voi luoda usealla eri tavalla, kuten esimerkiksi nauhoittamalla. Tavallisin tapa käyttää Nauhoita ja toista –työkalua ja käyttää siinä mukana tulevaa nauhoitusominaisuutta, jolla tallennetaan käyttäjän suorittamat toiminnot.

Monet testit lähestyvät testausta myös erilaisten mallien kautta, joilla pyritään kuvaamaan mahdollisia tapahtumia nauhoittamatta niitä. Malli koostuu useista erilaisista tiedoista, kuten syöttödatasta (input data), käytön yleisyydestä (usage frequencies), tapahtumajärjestyksen yksinkertaistamisesta (event sequence abstraction), tapahtumatyypeistä (event types) sekä käyttöliittymän rakenteesta (GUI structure). Malli voidaan luoda ja ylläpitää usealla eri tavalla. Erilaisia tapoja on Adamolin ym. (2011) mukaan kolme:

- Osa malleista voidaan tuottaa suoraan sovelluksesta käyttäen staattista tai dynaamista analyysiä. **(**Extract from application)

- Osa malleista voidaan tehdä yksinkertaistamalla konkreettisista tapahtumajärjestyksestä. (Abstract)
- Malli voidaan luoda myös kokonaan manuaalisesti, jopa ennen kuin sovellus on edes valmis. (Create Manually)

Mallin valmistuttua suoritetaan sille instanssointi, jolloin se muutetaan kuvaamaan tiettyä tapahtumajärjestystä. Kun tapahtumajärjestys on tavalla tai toisella saatu valmiiksi, pystytään se suorittamaan käyttäen testaustyökaluja tai scriptejä. Toistamisen yhteydessä tapahtumajärjestys käydään läpi, jolloin päästään johonkin tulokseen. Tämä tulos voi olla se, mitä haluttiin tai jotain, joka vielä vaatii korjausta. Tuloksen pertusteella joko korjataan vanhaa testiä sitä voidaan päivittää vastaamaan testin tarpeita vielä enemmän.

Automaattisen testauksen suunnittelu on siis sykli, jossa testiä kehitetään jatkuvasti. Se, onko automaattinen testi koskaan täysin valmis, riippuu halutuista tuloksista ja siitä, kuinka paljon automaattiseen testaukseen halutaan panostaa.

4.3 Milloin automatisoida?

Testien automatisointi on haastavaa ja melko monimutkaista puuhaa. Myös testien suunnitteluun menee oma aikansa. Sovelluksen testausta suunnitellessa onkin hyvä kiinnittää huomiota seuraaviin kysymyksiin:

- Yhden automaattisen testin tekeminen ja ajaminen maksaa enemmän kuin sen tekeminen manuaalisesti. Kuinka paljon enemmän se maksaa? (Marick 1998.) Kattaako automaattisen testin elinaika sen kustannukset?
- Vaikka automaattisia testejä ajetaan usein, niin täytyy miettiä, kuinka monta kertaa automatisoitua testiä lopulta ajetaan. Millaiset asiat voivat johtaa kyseisen testin poistamiseen tai poistumiseen? Tapahtuuko tämä mahdollisesti lähitulevaisuudessa vai vasta projektin päätyttyä?
- Kuinka todennäköistä on, että automatisoitu testi löytää virheitä ensimmäisen ajokerran jälkeen? Onko automatisoinnista hyötyä, jos virheitä ei löydy kuin harvoin? (Marick 1998.)

- Kauanko aikaa kuluu testin automatisointiin?
- Jos testin automatisoi, niin mitkä manuaalisest testit se tulee kattamaan? Löytääkö automatisoitu testi mahdollisesti kaikki ne virheet, jotka löytyisivät manuaalisella testaamisella? (Marick 1998.)

Vaikka moni näistä kysymyksistä esitettiinkin jo vuonna 1998, ovat ne edelleen ajankohtaisia. Sovelluskehityksessä tehokkuus on avainasemassa ja automaattinen testaus tuo ratkaisuja tietynlaisiin ongelmiin. On syytä harkita tarkkaan, tarjoaako automaattinen testaus tarpeeksi hyötyjä suhteessa siihen, kuinka paljon sen käyttöön ottaminen ja toteuttaminen tuo haasteita.

4.4 Vaatimukset

Automaattista testausta tutkineet ovat yksimielisiä siitä, että automaattinen testaus ei ole helppoa. Selkeitä suuntaviivoja ja suoria vastauksia automaattisten testien ongelmiin ei ole, vaikka kaikilla on jonkinlainen ratkaisu. Selkeiten asian tuovat esille Singhera, Horowitz ja Shah (2008), jotka erittelevät artikkelissaan eräitä vaatimuksia automaattisille testeille. Heidän listaamiaan vaatimuksia ovat muun muassa seuraavat:

- Jokainen käyttöliittymäelementti on määriteltävä yksilöllisellä nimellä. Kaikilla elementeillä tulisi olla oikein määritellyt parametrit ja niiden tapahtumat pitäisi olla hyvin määritelty.
- Testit tulisi organisoida hierarkkisesti siten, että osa jäljittelisi käyttäjän toimintaa ja toinen osa varmistaisi tulokset. Tämän hierarkian pitäisi myötäillä testattavan sovelluksen objektihierarkiaa. Jokainen script käsittelisi vain tiettyä objektia ja sen jälkeläistä.
- Automaattisen testin scriptien pitäisi olla mahdollisimman yksinkertaisia ja pieniä. Kuitenkin samantyyppiset scriptit pitäisi pystyä niputtamaan yhdeksi yleiskäyttöiseksi scriptiksi.
- Automaattisen testin scriptit pitäisi muodostaa siten, että ensin kommentoidaan toiminta hyvin, jonka jälkeen alustetaan ympäristö testille. Tämän jälkeen testataan

toiminta ja tarkastetaan sen oikeellisuus sekä lopuksi palautetaan ohjelma siihen tilaan, josta testi lähti liikkeelle.

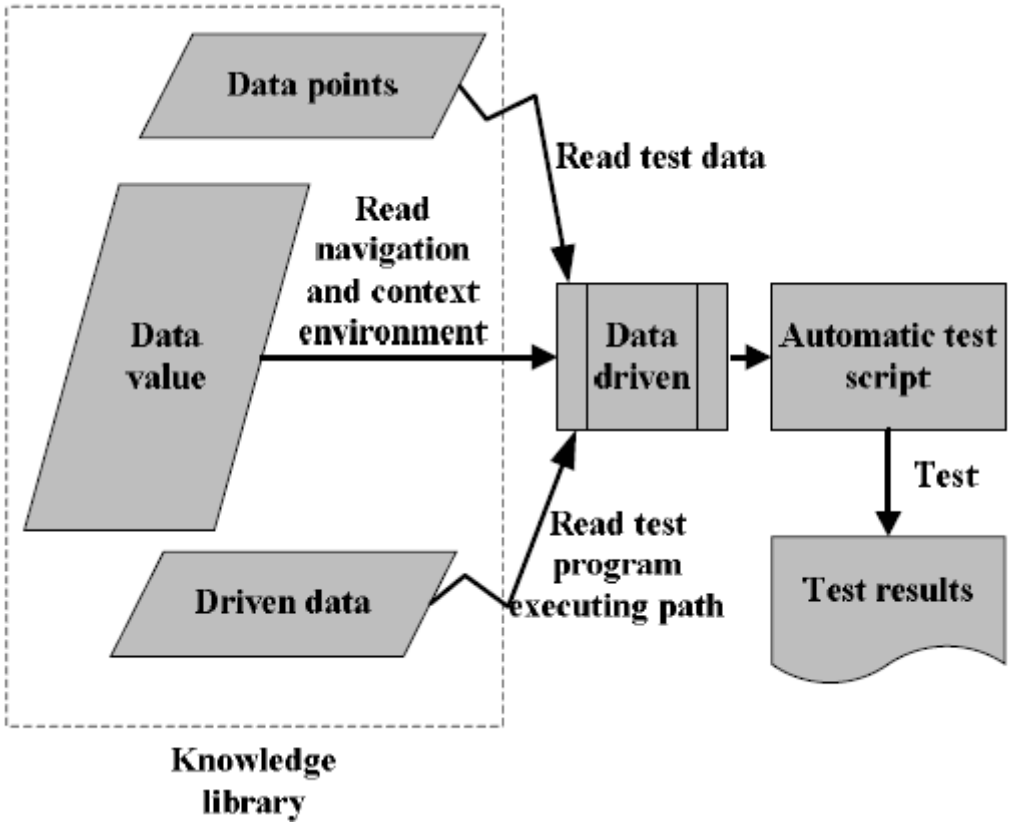
- Testit tulisi tehdä uudelleenkäytettäväksi siten, että niitä voidaan kutsua myös muualta.
- Tiedon tarkastaminen olisi syytä tehdä siten, että tarkastetaan vain oleelliset tiedot ja objekteilta saataisiin tiedot silloin, kun niitä tarvitaan. Tässä asiassa korostuu jälleen oikeanlaisen nimeämisen tarve.
- Scriptit, jotka simuloivat käyttäjän toimintaa sekä scriptit, joissa testataan datan oikeellisuutta pitäisi pitää erillään.

Lista on kattava, mutta se ei pidä sisällään aivan kaikkia vaatimuksia. Testejä laatiessa on syytä muistaa, että vaikka testi toimisikin hyvin koneella, jolla testi laaditaan, saattaa ongelmia ilmetä kuitenkin muissa koneissa. Esimerkiksi erilaiset näyttöjen resoluutiot voivat aiheuttaa sen, että johonkin ikkunaan ei mahdukaan niin paljon infoa kuin testiä laadittaessa tai jokin ikkuna onkin toisen päällä ajettaessa automaattista testiä. Testaustyökalut eivät välttämättä osaa itse osaa kasvattaa tai siirtää ikkunoita saati käyttää vierityspalkkeja joten näidenkin toimintojen automatisointi jää testauksen laatijalle.

4.5 Aineisto-ohjattu testaus

Automaattinen testaus halutaan joskus toteuttaa siten, että sama testi ajetaan useaan kertaan erilaisilla arvoilla. Jos näin halutaan toimia, kannattaa käyttää niin kutsuttua aineisto-ohjattua testausta (Data-driven testing). Tällaisella testauksella tarkoitetaan sitä, että testien automatisoinnin lisäksi luodaan taustadata. Esimerkiksi tietokantataulu tai erilainen datatiedosto, kuten excel- tai CSV-tiedostot voivat olla luettavana tietona (TestComplete 9 Documetation 2012). Nämä tiedostot pitävät sisällään testeissä käytettävät arvot eri parametreina.

Kun testit ajetaan, käydään läpi joko kaikki arvot tai ennalta määritelty arvojoukko ja testataan sovellusta näillä arvoilla. Wu, Wan, Xi ja Chen (2009) esittelevät artikkelissaan seuraavan yksinkertaisen mallin aineisto-ohjatulle testaukselle. Kuvassa 2 on esitelty tämän mallin toiminta.



Kuva 2: Malli aineisto-ohjatulle testaukselle (Wu ym. 2009)

Kuvassa esitetään niin sanottu tietokirjasto (knowledge library) josta haetaan suoritettava testi. Tämän kirjaston perusteella kasataan tieto, jonka perusteella muodostuu automatisoitavan testin scripti. Tämä voidaan toteuttaa esimerkiksi siten, että data pisteinä (data points) ovat muuttujat, datan arvoina (data value) nauhoitettu scripti ja ajettavana datana (driven data) testattavan ohjelman polku. Näistä kootaan siis ajettava data (Data driven) ja siitä muodostetaan automaattinen testiscripti (automatic test script), joka ajetaan. Kun testi on toistettu, tarkastetaan tulokset (test results), raportoidaan ne ja haetaan uudet arvot seuraavalle testille. Näin jatketaan, kunnes tietokirjastossa ei enää ole haettavia arvoja.

4.6 Nauhoita ja toista –työkalut

Nauhoita ja toista –työkalu on nimensä mukaisesti sellainen testaustyökalu, jolla pystyy nauhoittamaan käyttäjän toimintoja sovelluksen päällä. Työkalu ei siis varsinaisesti tarvitse

sovelluksen lähdekoodia, vaan osaa toimia itsenäisesti käyttäen hyväkseen sovelluksen käyttöliittymälle tarjoamia tietoja. Testityökalut keskittyvät yleensä johonkin tiettyyn käyttöjäjrjestelmään ja tiettyihin ohjelmointikieliin, joten testityökalun valinnassa täytyy olla tarkkana.

4.6.1 Vaatimuksia työkalulle

Käyttöliittymän päällä toimiminen luo haasteita työkalulle. Alkeelliset nauhoita ja toista – työkalut tallensivat vain hiiren painalluksen paikan koordinaatit (Adamoli ym. 2011), jolloin jos uudelleen ajettaessa jokin haluttu komponentti sattui olemaan eri paikassa, epäonnistuivat testit täysin. Uudemmat ja kehittyneemmät testaustyökalut pystyvät hakemaan käyttöliittymäkomponentin nimen ja suorittamaan sen metodeja halutulla tavalla. Tämä taas luo uuden ongelman. Koska kehittyneemmät työkalut noutavat toistovaiheessa käyttöliittymäkomponenttien nimen, on tärkeää, että nimet olisivat uniikkeja ja niitä ei vaihdettaisi sen jälkeen, kun automaattinen testaus on aloitettu. Jos nimen vaihtaa toiseksi tai mahdollisesti jopa nimeää toisen komponentin aiemmin olemassa ollella nimellä tulee automaattisessa testauksessa varmasti virheitä.

Jos nauhoitustyökalu ei ole riittävän tarkka, esimerkiksi hiirenn toiminta voi olla vaativaa nauhoittaa. Tajuaako nauhoittava työkalu, että tapahtuuko hiirtä kaksi kertaa painettaessa yksi klikkaus, jonka jälkeen toinen vai osaako se rekisteröidä klikkaukset tuplaklikkaukseksi? Tällaisia ongelmia ilmeni paljon vanhemmissa nauhoita ja toista – työkaluissa (Adamoli ym. 2011).

Myös synkronisaatio-ongelma on ilmeinen (Adamoli ym. 2011). Miten toteutetaan automaattisesti testitapaus, joka toimii sellaisen sovelluksen päällä, joka muuttuu dynaamisesti esimerkiksi päivämäärän tai kellonajan mukaan? Pelkästään nauhoittamalla tätä ongelmaa ei pysty ratkaisemaan, vaan on tarkennettava automatisointia käyttäen hyväksi esimerkiksi scriptejä.

Poikkeustilanteista tai vaihtoehtoisista tilanteista toipuminen on yksi mahdollinen ongelma. Miten toimitaan, jos testattavassa sovelluksessa ikkuna esiintyy joskus ja toisinaan kyseinen ikkuna ei ilmestykään? Kuvitteellisena esimerkkinä sovellus voisi

esimerkiksi joka maanantai ilmoittaa, että uusi viikko on alkanut. Muuten ilmoitusta ei tulisi. Koska nauhoita ja toista –työkalut suorittavat käyttäjän toiminnan tallennetussa järjestyksessä, täytyy poikkeukset käsitellä jotenkin. Tässäkin tapauksessa scriptaus on hyvä vaihtoehto, jos sellainen mahdollisuus työkalusta löytyy.

4.6.2 Erilaisia nauhoita ja toista –työkaluja

Nauhoita ja toista –työkaluja on tarjolla useita ja niiden taso vaihtelee merkittävästi. Ilmaistyökalut ovat toiset puutteellisia ja toiset taas ajavat asiansa välttävästi. (Adamoli ym. 2011). Maksullisissa sovelluksissa sen sijaan löytyy jo paljon hyviä ominaisuuksia ja niillä voikin tehdä jo monimutkaisia automaattisia testejä.

Ilmaisia, open source -työkaluja Adamoli ym. (2011) esittelee muutaman. Näitä ovat Abbot, Jacareto, Pounder, Marathon sekä JFCUnit. He löytävät sovelluksista paljon puutteita liittyen juuri edellisessä kappaleessa esiteltyihin asioihin. Muita ilmaissovelluksia ovat muun muassa Selenium sekä Dojo Object Harness, joilla testataan Web-sovelluksia.

Maksullisia sovelluksia on markkinoilla useita. Esimerkkejä näistä työkaluista ovat esimerkiksi TestPartner QuickTest Pro sekä tässä tutkielmassa tarkemmin esiteltävä TestComplete. Maksullisille sovelluksilla pystytään yleensä tekemään kattavampaa testausta sekä testauksen raportointi on toteutettu paremmin.

4.7 Variaatiot ja pidemmälle viedyt toteutukset

Kuten jo mainittu, automaattinen testaus on haastava toteuttaa. Ongelmia syntyy muun muassa siitä, miten testejä lähdetään rakentamaan. Mahdollisia ratkaisuja on esitelty paljon, joista osassa keskitytään vain kuvaamaan sopiva malli automaattisten testien generoimiselle ja osassa lisäksi annetaan esimerkkejä testien toteuttamisesta.

Triou, Abbas ja Kothapalle (2009) esittelevät artikkelissaan tavan, jossa testit suoritetaan sillä periaatteella, että ensin määritetään lopputila ja sen jälkeen suoritus, jolla tähän tilaan päästään. Lopuksi tarkastetaan, päästiinkö haluttuun lopputulokseen nimenomaan testauksen kannalta. Tämän toimintatavan etuna on se, että se keskittyy siihen, mihin

halutaan sen sijaan kuin siihen, miten haluttuun lopputulokseen päästään, ottaen huomioon erilaiset mahdolliset siirtymät testattavan sovelluksen sisällä.

Memon ym. (2005) esittelevät tekstissään DART-mallin (Daily Automated Regression Tester). Tämän mallin ideana on se, että testattavan sovelluksen käyttöliittymästä erotetaan ja generoidaan automaattisesti sekä testit että testien kattavuusarviot. Vaikka edellä mainitut asiat luodaan automaattisesti, joudutaan ne kuitenkin vielä varmistamaan ja korjaamaan mahdolliset virheet ja puutteet käsin. DARTiin on luotu matemaattisia malleja testeille ja DART onkin hyvin teoriapainotteinen. Kuitenkin tekijät ovat onnistuneet toteuttamaan testien automatisoinnin myös käytännössä.

Kolmas hieman pidemmälle viety tapa on käyttää agenttipohjaista mallia. Yin, Miao, Miao ja Shen (2005) esittelevät tekstissään tällaisen agenttipohjaisen kehyksen, jossa eri käyttötarkoituksia varten kehitetyt agentit hoitavat automaattisessa testauksessa suoritettavia asioita aina käyttäjän toimien lukemisesta testien toistamiseen. Kaiken kaikkiaan käytettäviä agentteja heidän mallissaan on viisi kappaletta.

Automaattiseen ei ole löytynyt yhtä oikeaa tapaa toimia. Kuten jo edellä esitellyistä esimerkeistä käy ilmi, automaattista testausta voidaan toteuttaa todella erilaisilla tavoilla. Oikeaa tapaa ei varmasti ole olemassa, koska kaikissa tavoissa on puolensa ja vaikeutensa. Monenlaisista tavoista käy kuitenkin ilmi se, että automaattinen testaus on vaikea aihe ja parhaista toimintamalleista ei vielä ole päästy yksimielisyyteen.

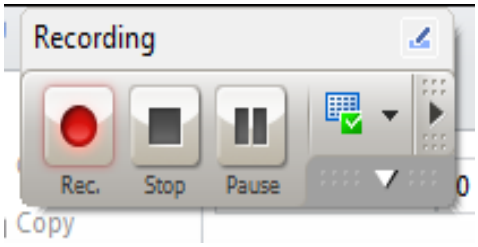
5 TestComplete

Tässä luvussa käsitellään SmartBear Softwaren sovellusta TestComplete. Luku esittelee TestCompleten tiettyjä ominaisuuksia ja käyttöä peilaten juuri tutkielmassani esiteltyihin toimintoihin ja vaatimuksiin. TestComplete on todella kattava sovellus ja tässä tutkielmassa esitellyt asiat ovat vain pintaraapaisu sen käyttämiseen. Tarkoituksena onkin tuoda esille sen mahdollisuuksia ja käyttötapoja yleisellä tasolla. TestCompletessa projektin hallinta on toteutettu hierarkkisena puumallina, jossa yhteen projektijoukkoon (Project Suite) voi sisältyä monia eri projekteja. Nämä projektit sisältävät taas omat scriptinsä, tietokantansa sekä muut projektiin liittyvät tiedostot, joita voivat olla muun muassa erilaiset tapahtumat tai projektin asetukset.

5.1 Nauhoitus

TestComplete tarjoaa muiden nauhoita ja toista –työkalujen tapaan mahdollisuuden nauhoittaa käyttäjän toimintaa sovelluksessa. Nahuoitus voidaan tehdä joko täysin uuteen testiin tai vaihtoehtoisesti jo olemassa olevaa testiä voidaan laajentaa.

Nauhoituksen yhteydessä esiin ilmestyy erillinen ikkuna nauhoituksen hallintaa varten. Tämä hallintaikkuna on esitelty kuvassa 3.



Kuva 3: Nauhoitusikkunan ulkoasu.

Nauhoitusikkunalla on oletuksena neljä pääpainiketta.

- Rec.-painike. Vilkkuva valo kertoo, että nauhoitus on käynnissä. Oletuksena TestComplete nauhoittaa kaiken, mitä käyttäjä näytöllä tekee. Jos nauhoitus on

jostain syystä pysäytetty Pause-painikkeella, saa nauhoituksen uudelleen käyntiin Rec-painikkeella.

- Stop-painike. Lopettaa nauhoittamisen ja palaa TestCompleteen.
- Pause-painike. Keskeyttää nauhoituksen väliaikaisesti. Nauhoitusta on mahdollista jatkaa samasta kohtaa painamalla Rec –painiketta.
- Checkpoint-painike. Painamalla tätä painiketta avautuu alasvetovalikko, josta on mahdollista valita erilainen tarkastuspiste (checkpoint), joka lisätään testin yhteyteen. Tarkastuspisteen lisäämisen aikana nauhoitus on kesketetty.

Kun nauhoittaminen on ohi, siirtyvät nauhoitetut työvaiheet editoriin, keyword-testien tapauksessa sille sopivaan editoriin ja scriptejä nauhoittaessa script-editoriin. Nauhoitus myös tekee karkeaa dokumentaatiota käyttäjän tekemisistä.

5.2 Testit

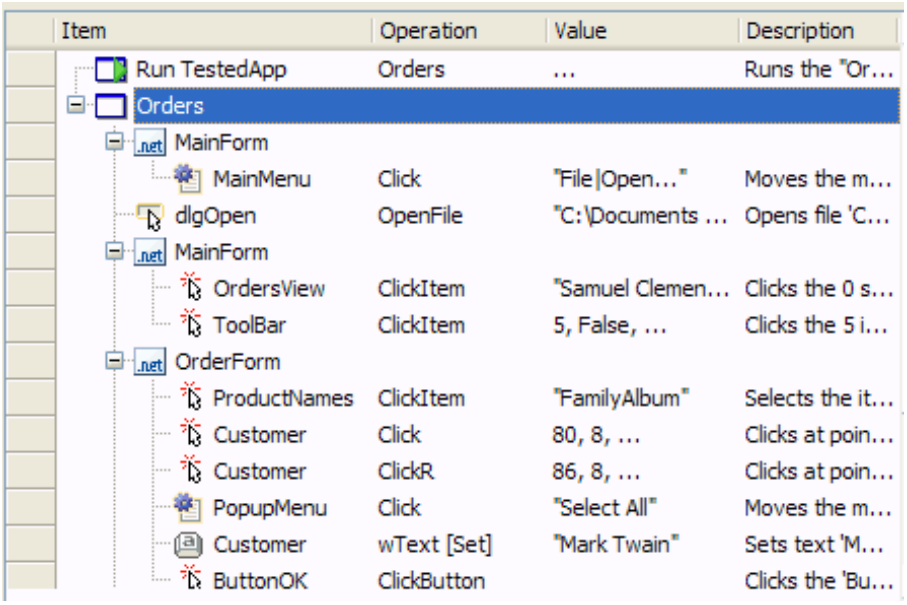
TestComplete tarjoaa testien kehittämiselle kaksi tapaa, jotka tukevat toisiaan. Toinen tapa on nauhoittaa testit scripteiksi ja toinen tapa niin kutsutuiksi keyword-testeiksi.

5.2.1 Keyword-testit

Keyword-testien suuri etu on se, että niitä voi nauhoittaa ja muokata tuntematta ollenkaan sctiptaamiseen käytettyjä kieliä. TestCompleten keyword-testit koostuvat nimensä mukaan keywordeista. Jokainen keyword on yksi tapahtuma, kuten painikkeen painaminen tai tekstikenttään kirjoitus. Nämä toiminnot ovat TestCompletessa nimeltään operaatioita, joille voidaan antaa parametreja määrittelemään kuinka ne tulevat toimimaan.

Perusperiaate keyword-testeissä on sellainen, että ensin määritellään kohde (item), sen jälkeen operaatio (operation), joka kohteelle suoritetaan ja sen jälkeen arvo (value). Myös kuvauksen (description) voi antaa. Nämä kuvaukset toimivat eräänlaisena kommentointina. Kuvassa 4 on esitelty keyword-testien rakennetta. Kuvasta ilmenee hyvin keyword-testien rakenne. Kuvassa kannattaa huomioida, että vaikka klikkaus operaatioille on arvoiksi

koordinaatit, ovat nämä koordinaatit nimeomaan käsiteltävän kohteen parametrit, ei koko ikkunan tai näytön parametrit.



Kuva 4: Esimerkki keyword-testien rakenteesta (TestComplete Documentation 2012).

Keyword-testeille muiden testien tapaan asettamaan parametreja ja muuttujia. Nämä tapahtuvat keyword-testien yhteydessä omissa dialogeissaan ja niiden asettaminen on melko yksinkertaista.

5.2.2 Script-testit

Scriptit ovat yleisemmin TestCompletessa käytetty tapa ohjelmoida automaattista testausta. Testien nauhoittaminen tapahtuu täysin samalla tavaoin kuin keyword-testien tapauksessakin. Nauhoittamisen jälkeen kuitenkin testejä pääsee muokkaamaan erilliseen editoriin valitsemallaan scriptikielellä. Mahdollisia scriptauskieliä ovat VBScript, JavaScript, C++Script, C#Script ja DelphiScript. Scriptikielen valinnalla ei ole merkitystä testattavan sovelluksen ohjelmointikielen kanssa vaan molemmat toimivat itsenäisesti.

Scriptien kirjoittaminen on periaatteellisella tasolla yksinkertaista. TestCompleteen on luotu tehokas scriptikirjasto ja dokumentaatio on kattava. Scriptit ovat siinä mielessä

helpompia ja monipuolisempia luoda, sillä niissä muuttujien ja parametrien käsittely tapahtuu suoraan koodissa ja pienelläkin ohjelmointitaustalla saa helposti monipuolisempia testejä kuin keyword-testeillä. Lisäksi keyword-testejä pystyy muuttamaan suoraan scriptiksi, jolloin TestComplete kääntää tallennetun keyword-testin valitulle scriptikielelle. Myös molempien tapojen yhdistely onnistuu. Scriptistä voi kutsua keyword-testiä ja keyword testiin pystyy lisäämään ”Run Script Routine” tai ”Run Code Snippet” osan. Ruotine suorittaa koko tallennetun scriptin ja code snippet sen sijaan mahdollistaa jonkun scriptin pätkän lisäämisen keyword-testeihin.

5.3 Tarkastuspisteet ja tietokannat

TestCompletessa pystytään lisäämään erilaisia tarkastuspisteitä. Tarkastuspisteitä pystyy lisäämään testien nauhoittamisen yhteydessä tai niitä voi lisätä sekä keyword- että script-testeihin joko manuaalisesti tai erillisellä wizardilla. TestCompletessa on kattava valikoima erilaisia testejä. Tarjolla olevat tarkastuspisteet ovat seuraavat:

- Ominaisuustarkastuspiste (Property Checkpoint)
- Objektitarkastupiste (Object Checkpoint)
- Taulutarkastuspiste (Table Checkpoint)
- Tiedostotarkastuspiste (File Checkpoint)
- XML-tarkastuspiste (XML Checkpoint)
- Tietokantatarkastuspiste (Database Checkpoint)
- Aluetarkastuspiste (Region Checkpoint)
- Leikepöytätarkastuspiste (Clipboard Checkpoint)
- Manuaalinen tarkastuspiste (Manual Checkpoint)
- Verkkopalvelu tarkastuspiste (Web Service Checkpoint)
- Verkon saavutettavuustarkastuspiste (Web Accessibility Checkpoint)
- Verkon vertailutarkastuspiste (Web Comparison Checkpoint)

Tavallisin tarkastuspisteistä on ominaisuustarkastuspiste. Sen avulla pystytään tarkastamaan, että kyseisellä, tällä hetkellä olemassa olevalla objektin ominaisuudella on tietty arvo. Tällä tarkastuspisteellä pystytään tarkastamaan esimerkiksi jonkun

käyttöliittymäelementin, kuten merkinnän (label) teksti. Objektien laajempi tarkastelu sen sijaan onnistuu käyttäen objektitarkastuspistettä. Tällä tarkastuspisteellä voidaan edellisen ominaisuustarkastuspisteen tapaan tarkastaa jonkun objektin arvo, mutta objektitarkastuspiste tarjoaa mahdollisuuden tarkastaa kerralla useampia arvoja. Myös valmiin ikkunan tai komponentin tarkastamiselle on tarkastuspisteensä. Jos halutaan tarkastaa näytetäänkö ikkuna tai komponentti oikein, käytetään aluetarkastuspistettä, joka vertaa näkyvillä olevaa ikkunan tai komponentin kuvaa tallennettuun kuvaan. Tämä on myös yleisesti käytetty tarkastuspiste.

Taulutarkastuspisteet ovat taulukkomuotoisen tiedon tarkastamista varten. Näille tarkastuspisteille täytyy määritellä ensin jokin taulupohja, johon automaattisen ajamisen yhteydessä taulua verrataan. Taulupohjaa varten luodaan malli joko silloisella hetkellä näytöllä olevasta taulusta tai muokataan taulun rivit ja sarakkeet käsin. Jos käytetään olemassa olevaa mallia, voidaan osoittaa näytöltä taulukkomuotoista objektia, jonka jälkeen TestComplete tarjoaa mahdollisuuden valita, mitä taulun kenttiä halutaan verrata.

TestCompleten tarkastuspisteiden avulla pääsee tarkastelemaan myös tietokannassa olevaa tietoa. Tällaisia tarkastuspisteitä ovat tietokantatarkastuspisteet. Nämä tarkastukset suoritetaan ennalta määrättyyn tietokantaan, johon TestComplete osaa ottaa automaattisesti yhteyden tietokantaan ja tarkastaa, vastaavatko tietokannassa olevat arvot tallennettua taulua tai sen osaa.

Myös erilaisten tiedostojen tarkastelulle on olemassa erilaisia tarkastuspisteitä. Sekä tiedostotarkastuspisteet että XML-tarkastuspisteet pystyvät vertaamaan tiedostoja määriteltyihin mallitiedostoihin. Tällaiset testit ovat käytännöllisiä esimerkiksi silloin, jos testattava sovellus luo tiedostoja ja halutaan tarkastaa, toimiiko tälläinen tiedostojen luominen kuten pitääkin.

Web-sovelluksille on olemassa omat tarkastuspisteensä. Verkkopalvelu tarkastuspisteellä voidaan tarkastaa vastaus, jonka sovellus saa ottaessa yhteyttä verkossa olevaan palveluun. Verkon saavutettavuustarkastuspisteellä voidaan tarkastaa verkkosivujen tyypillisiä arvoja. Esimerkiksi objektien korkeuden ja leveyden voi tarkastaa, jotta varmistutaan siitä, että ne todella näkyvät verkkosivulla. Myös linkkien oikeellisuuden tarkastaminen onnistuu

käyttäen tätä monikäyttöistä tarkastuspistettä. Verkon vertailutarkastuspiste sen sijaan mahdollistaa esitettävän verkkosivun vertaamisen tallennettuna olevaan. Tällä tarkastuspisteellä pystyy vertaamaan myös HTML-tiedostoja.

Leikepöytätarkastuspiste on yksinkertainen tarkastuspiste, jolla tarkastetaan, että leikepöydällä oleva tieto vastaa testin tekemisen yhteydessä tallennettua tietoa. Jos mikään edellä olevista tarkastuspisteistä ei tarjoa haluttua vertailua, voi TestCompletella luoda myös manuaalisia tarkastuspisteitä. Nämä tarkastuspisteet toimivat siten, että kun automatisointi pääsee manuaaliseen tarkastuspisteeseen asti, automaattinen testin suoritus keskeytyy. Tämän jälkeen joudutaan manuaalisesti tarkastamaan testin tila. Manuaaliseen tarkastuspisteeseen voi tallentaa yksityiskohtaiset ohjeet tarkastavalle testaajalle, jolloin testaaja suorittaa manuaalisen tarkastuspisteen yhteydessä olevan tarkastuksen ja saa valita joko hyväksytyn tai hylätyn arvon tarkastuspisteelle. Tämä tarkastuspiste on hieman kyseenalainen, sillä se vastaa suoraan manuaalista testausta ja vaatii, että käyttäjä käy tarkastamassa automaattisen testin kulun. Manuaalista tarkastuspistettä kannattaakin käyttää vain tilanteissa, joissa muita tarkastuspisteitä ei syystä tai toisesta ole mahdollista käyttää.

5.4 Raportointi

Jossain vaiheessa TestCompleten suorittama testi loppuu. Käyttäjän valinnoista riippuen testien automaattinen suorittaminen voidaan lopettaa vasta silloin, kun kaikki tapahtumaketjut on käyty läpi ja testit ovat suoritettu. Testeihin voi ohjelmoida tilanteita, jolloin suoritus keskeytyy tai vaihtoehtoisesti voi valita, että testit keskeytyvät, jos jokin tapahtuma tai tarkastuspiste epäonnistuu.

TestComplete pitää kattavaa lokia testien tapahtumista. Jokainen vuorovaikutus käyttöliittymän kanssa taltioidaan ja lisätään lokiin. Jos tapahtuu virhe tai testi ei suoriudukaan niin kuin pitäisi, lokiin tallennetaan tieto kohdasta, jossa virhe tapahtui, sekä mahdollisesti kuva tilanteesta. Lisäksi lokiin tulee tieto siitä, minkälainen tilanteen olisi pitänyt olla ja minkälainen tilanne testiä ajettaessa oli. Lokitiedot ovat todella kattavia ja lokien tarkasteluun tarkoitettu käyttöliittymä toimii hyvin.

Testitulosten vieminen ulos TestCompletesta onnistuu. Lokit voi muodostaa joko PDF-, XML-, HTML-, tai MHT-tiedostoksi. Näitä tiedostoja voi sen jälkeen käsitellä tallennusmuodosta riippuen esimerkiksi verkkoselaimella. Lokien vieminen tapahtuu joko itse testeissä tai vaihtoehtoisesti sen voi tehdä manuaalisesti testituloksia tarkasteltaessa. TestComplete osaa myös lähettää testituloksia sähköpostilla. Lokeista ei tarvitse viedä kaikkea tietoa, vaan ulos vietävän datan voi rajata koskemaan tiettyä testiä tai jopa testin osaa.

5.5 Automaattinen ajaminen

Valmiita testiprojekteja voi suorittaa myös komentoriviltä. Tämä avaa mahdollisuuden automatisoida myös testien aloittaminen, jolloin esimerkiksi öisin julkaistavien päivitysten regressiotestaaminen onnistuu haluttuna kellonaikana. TestCompleten käyttäminen normaalisti vaatii sen, että sillä on käyttöliittymään esteetön pääsy. Tämä vaatii sen, että joku on kirjautuneena koneelle, jolla automatisointi suoritetaan. Tämä taas ei ole järkevää, sillä koneen jättäminen auki on tietoturvariski.

Toinen vaihtoehto on käynnistää TestComplete komentoriviltä niin kutsuttuun hiljaiseen moodiin (Silent mode), jolloin TestComplete suorittaa hyvin yksinkertaista testausta halutulle sovellukselle kokeillen käyttöliittymän toimintaa näyttämättä dialogeja oikeasti. Tämän moodin yhteydessä TestComplete kokeilee dialogeja ja valitsee aina oletuspainikkeen. Tieto näin käsitellyistä dialogeista, ilmoituksista ja virheistä tallennetaan erilliseen silent.log tiedostoon. Tälläinen hiljainen moodi ei kuitenkaan suoriudu automaattisesta testaamisesta niin kattavasti kuin olisi syytä, vaan kattavampien automaattisten testien tapauksessa on vielä käytettävä normaalia moodia.

Toimivaan ja turvalliseen automaattiseen ajamiseen ei ole vielä löytynyt ratkaisua. Kenties jossain vaiheessa TestCompleteen tai muuhun testaustyökaluun saadaan ominaisuus, jonka avulla automaattista GUI-testausta voidaan suorittaa turvallisesti myös lukitulla koneella.

6 Yhteenveto

Automaattinen GUI-testaus on vaikea aihe, koska siihen ei ole, ainakaan vielä, muodostunut vakituista ja helppoa tapaa. Automaattiseen GUI-testaukseen on kuitenkin olemassa hyviä ohjeita ja toimintamalleja, joita noudattamalla testauksesta saadaan suhteellisen tehokasta. Testien automatisointiin menee oma aikansa, mutta pidemmän päälle automaattisen testauksen avulla testaajien työpanos voidaan aikaavievien ja itseään toistavien testien sijaan ohjata tärkeämpiin huomiota vaativiin asioihin. Adamolin ym. (2011) mukaan automaattisella testauksella ei ole vielä mahdollista testata pitkiä, realistisia tapahtumaketjuja, vaan testaaminen rajoittuu tiettyyn pieneen osaan sovellusta.

Automaattista GUI-testausta harkitessa onkin syytä punnita tarkkaan, tuoko se riittävästi etuja suhteessa siitä muodostuviin kuluihin ja haasteisiin. Oikein toteutettuna automaattinen GUI-testaus ratkaisee monia ongelmia, mutta jos sen toteuttaa väärin, voi eteen tulla useita ja yllättäviäkin ongelmia.

Nauhoita ja toista –työkalut ovat kehittyneet viime aikoina. Ilmaissovellukset ovat tasoltaan yksinkertaisempia kuin maksulliset. Maksullisilla työkaluilla on mahdollista suorittaa melko kattaviakin testejä. Kuitenkin sovelluksissa on vielä kehitettävää, eikä selkeää, parasta nauhoita ja toista –työkalua ole vielä olemassa. Tällaisen sovelluksen kehittäminen on varmasti haaste tulevaisuudessa.

Useiden haasteiden takia automaattisessa GUI-testauksessa riittää vielä kehitettävää. Kehityskohteita löytyy oikeastaan joka osa-alueelta. Tutkielman perusteella ainoaa oikeaa mallia automaattiselle testaamiselle ei ole olemassa, vaan oikea malli on löydettävä tilanteen mukaan. Hyville ja toimiville malleille löytyisi käyttöä. Lisäksi testien toteuttamiselle on omat haasteensa, koska sekä testattava sovellus että testaustyökalu asettavat omat vaatimuksensa ja rajoitteensa automatisoinnille. Näiden ongelmien ratkaiseminen tai ainakin tilanteen parantaminen on varmasti yksi tulevaisuuden kehityskohde testaamisen saralla.

Kiitokset

Kiitokset Tieto Oyj:lle ja erityisesti Tieto Healthcare & Welfare Oy:lle automaattisen GUI-testauksen ehdottamisesta tutkielmani aiheeksi sekä kannustuksesta ja tuen tarjoamisesta tutkielman tekemisen aikana.

Lähteet

Adamoli, A., Zaparanuks, D., Jovic, M. & Hauswirth, M. 2011. Automated GUI Performance Testing. Software Quality Journal, 19. 801–839.

El Ariss, O., Xu, D., Dandey, S., Vender, B., McClean, P. & Slator, B. 2010. A Systematic Capture and Replay Strategy for Testing Complex GUI Based Java Applications. In Information technology: New generations (ITGN), 2010 seventh international conference. 1038–1043.

IEEE Standard for Software Test Documentation 1998. IEEE Std 829-1998.

Kaner, C. 1997 Improving the Maintainability of Automated Test Suites. Software QA 4, no. 4, 1997.

Kaur, M., & Kumari, R. 2011. Comparative Study of Automated Testing Tools: TestComplete and QuickTest Pro. International Journal of Computer Applications, 24(1), 1–7. (Published by Foundation of Computer Science)

Marick, B. 1998. When Should a Test Be Automated. Viitattu 17.12.2012. http://www.wuxin-housekeeping.cn/Test/pdf/automate.pdf

McCaffrey, J. 2006. .NET Test Automation Recipes: A Problem Solution Approach, Apress Publishing, New York.

Memon, A., Nagarajan, A. & Xie, Q. 2005. Automating regression testing for evolving GUI software. Journal of Software Maintenance and Evolution, 17(1), 27–64.

Myers, G. J., Badgett, T. & Sandler, C. 2012. Art of Software testing. Third Edition. John Wiley & Sons, Inc. Hoboken, New Jersey.

Pei S., Yuewei, D., Wu, B. & Wu, C. 2009. Effective GUI-oriented Automated Testing Platform for Pervasive Computing Software Environment, Pervasive Computing (JCPC), 2009 Joint Conferences.

Singhera, Z, Horowitz, E. & Shah, A. 2008. A Graphical User Interface (GUI) Testing Methodology. International Journal of Information Technology and Web Engineering, 3(2), 1–8, 10–17.

SmartBear TestComplete 9 Documentation. Viitattu 17.12.2012. http://support.smartbear.com/viewarticle/32760/

The Linux Information Project 2004. GUI Definiton. Viitattu 17.12.2012. http://www.linfo.org/gui.html

Triou, E., Abbas, Z. & Kothapalle, S. 2009. Declarative testing: A Paradigm for Testing Software Applications. In Information Technology: New generations, 2009. ITGN ’09. Sixth international conference. 769–773.

Wu, T., Wan, Y., Xi, Y. & Chen, C. 2009. Study on the Automatic Test Framework Based on Three-Tier Data Driven Mechanism. In Computer and information science, 2009. ICIS 2009. eighth IEEE/ACIS international conference. 996–1001.

Yin, Z., Miao, C., Miao, Y. & Shen, Z. 2005. Actionable Knowledge Model for GUI Regression Testing. In Intelligent agent technology, IEEE/WIC/ACM international conference. 165–168.