Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Laila Aimo

Opinnäytetyö

Dynaamisen ohjelmistotestauksen automatisointi: Windows-sovelluksen testaus ja TestPartner-ohjelma

Työn ohjaaja lehtori Jyrki Vehmas
Työn tilaaja Yritys X, tuotekehitysyksikkö X
Tampere 11/2010

Tampereen ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma

Tekijä	Laila Aimo
Työn nimi	Dynaamisen ohjelmistotestauksen automatisointi: Windows-sovelluksen testaus ja TestPartner-ohjelma
Sivumäärä	47
Valmistumisaika	marraskuu 2010
Työn ohjaaja	lehtori Jyrki Vehmas

TIIVISTELMÄ

Yritys X:n tuotekehitysyksikössä X ylläpidetään ja kehitetään ohjelmistoja, joiden sovellusalue kuuluu kriittiselle taloushallinnon alueelle. SEPA (Single Euro Payments Area) -muutosten takia taloushallinnon ohjelmistoihin tehdään parhaillaan paljon muutostöitä. Samaan aikaan ohjelmistojen korkea laatu on pystyttävä säilyttämään, mihin tarvitaan paljon regressiotestausta.

Tämän opinnäytetyön tavoitteena oli selvittää, onnistuuko testauksen automatisointi toimeksiantajan tuotekehitysyksikössä jo yrityksessä käytössä olevalla TestPartner-ohjelmistolla. Testaustyökaluohjelmiston käyttö ei ollut suoraan kopioitavissa yrityksen muilta tuotekehitysyksiköiltä erilaisten toiminnallisuuksien ja ohjelmistoarkkitehtuurien vuoksi. Tarkoituksena oli myös lisätä tietämystä testausautomaatiosta toimeksiantajan yksikössä.

Toimeksiantaja voi käyttää tätä opinnäytetyötä taustatietona, kun päättää testauksen automatisoinnin kehittämisestä TestPartner-työkalua hyödyntäen. Opinnäytetyötä kannattaa tuotekehitysyksikössä lukea kaikkien niiden, jotka testaavat ohjelmia, vetävät tuotekehitysprojekteja ja ovat tekemässä päätöksiä testauksen automatisointiprojektin toteuttamisesta.

Toimeksiantajan yksikössä testausprosessi on toimiva. Siitä voisivat ottaa mallia ne, jotka vasta suunnittelevat ohjelmistotestauksen järjestämistä tai parantamista omassa tuotekehityksessään. Tämä opinnäytetyö toimii myös ohjelmistotestauksen teorian pikakurssina ja testauksen osuuden tärkeyden korostajana ohjelmistonkehitysprojektissa.

Avainsanat Ohjelmistotestaus, testiautomaatio, TestPartner, Windows-ohjelmistot.

Tampere University of Applied Sciences
Business Information Systems

Writer	Laila Aimo
Thesis	Automation of Dynamic Software Testing: Windows Application and TestPartner Software
Pages	47
Graduation time	November 2010
Thesis Supervisor	Jyrki Vehmas

ABSTRACT

Company X's co-operative unit is developing and maintaining financial management software of a critical area. SEPA (Single Euro Payments Area) is causing major alterations to this kind of software. However, the high quality of the software must be able to be maintained and thus lot of regression testing is needed.

One of the goals of this thesis was to clarify if it is possible to automate testing in the co-operative unit with the TestPartner software, which was already used by the other units of the Company X. The use of this automation tool could not be copied as it was from the other units, because of different functionalities and architectures. There was also purpose to add knowledge of test automation in the co-operative unit.

This thesis can be used as background information in the co-operative unit, when making decision of starting to develop test automation with the TestPartner automation tool. It is worthwhile that this thesis is read in the co-operative unit by the people, who are testing software, manage projects of software development and are making decisions of implementing test automation projects.

The process of testing in the co-operative unit is working so well that it could be an example for those, who plan to organize or improve software testing on their product development. This thesis also functions as a short introduction of the theory of software testing and as an emphasizer of the importance of testing on a software project.

Keywords Software testing, test automation, TestPartner, Windows-software.

Sisältö

Termit ja lyhenteet

BGC	Bankgirocentralen BGC AB hoitaa Ruotsin sisäistä maksujen välitysjärjestelmää pankkien ja niiden asiakkaiden välillä.
C2B	*Corporate-to-Bank,* Yrityksille tarkoitettu maksujen sanomakuvaus, jossa huomioidaan ISO 20022 -standardi ja SEPA:n vaatimat tietoelementit.
EPC	*European Payments Council*. Eurooppalaisten pankkien yhteistyöelin on laatinut yhtenäisen euromaksualueen säännöt ja standardit tilisiirroille ja suoraveloituspalveluille. (Finanssialan keskusliito 2010).
FTP	*File Transfer Protocol.* Tiedostonsiirtomenetelmä, jonka avulla voidaan siirtää tiedostoja kahden tietokoneen välillä.
GUI	*Graphic User Interface*. Graafinen käyttöliittymä.
IDE	Integrated Development Environment. Ohjelmointiympäristö. (Sainio 2009, 164.)
Moduuli	Moduuli on ohjelmiston yksittäinen osa, jossa voi olla 100 -1000 koodiriviä (Haikala & Märijärvi 2006, 289).
.NET	Microsoft on kehittänyt .NET-arkkitehtuurin tukemaan seuraavan sukupolven (next generation) sovellusten ja verkkopalveluiden kehittämistä varten. (MSDN Library).
ODBC	*Open Database Connectivity* on määritetty yleinen ohjelmistorajapinta, jonka avulla sovelluksilla on pääsy tietokannan tietoon.
PATU	PATU tarkoittaa pankkien asiakasyhteyksien tietoturvaa (Finanssialan keskusliito 2010).
PKI	*Public Key Infrastructure* on toimintamalli, joka mahdollistaa luottamuksellisen viestinnän ja digitaalisen allekirjoituksen. PKI perustuu julkisten avainten ja varmenteiden hallinnointiin. (Viestintävirasto, 2010).
RAM	*Random Access Memory*. Tietokoneen työmuisti.
SEPA	*Single Euro Payments Area*. Eurooppalaiset pankit, Euroopan keskuspankki ja Euroopan komissio ovat luoneet Euroopan talousalueelle maksuliikennettä koskevan kotimarkkina-alueen, josta käytetään nimitystä yhtenäinen euromaksualue. (Finanssialan keskusliito 2010).
SOA	*Service Oriented Architecture.* Palvelukeskeinen arkkitehtuuri, jonka yksi tekninen toteutus on Web Services.

TDD	*Test Driven Development*, testauslähtöinen ketterä ohjelmiston kehitysmenetelmä.
TPI-analyysi	*Test Process Improvement* -analyysin avulla selvitetään testausprosesssin tila ja tunnistetaan testausprosessin vahvat ja heikot puolet.
UI	*User Interface*. Käyttöliittymä.
UML	Unified Modeling Language. Graafinen mallinnuskieli, jota käytetään ohjelmistojen ja järjestelmien kuvaamiseen.
VBA-kieli	*Visual Basic for Applications* on Microsoftin sovellusohjelmissa makrokielenä käytetty ohjelmointikieli (MSDN Library 2010).
Vesiputousmalli	Vaihejakomalli, jolla kuvataan ohjelman kehitystyötä tai koko elinkaarta. Yleensä siinä erotetaan esitutkimus- tai tarvekartoitus-, määrittely-, suunnittelu- ja toteutusvaiheet. (Haikala & Märijärvi 2006, 37.)
Virtuaali PC	(*Virtual PC)* on Microsoft virtuaalikoneteknologiaa, jossa ohjelmallisesti luodaan virtuaalikone, joka emuloi fyysistä konetta ja on loogisesti eristetty isäntäkoneesta. Tällä tavoin voidaan suorittaa useita käyttöjärjestelmiä yhtäaikaa samalla fyysisellä koneella. (technet.microsoft.com)
Web Services	Web services on ohjelmistojärjestelmä, joka on suunniteltu tukemaan tietokoneiden välistä vuorovaikutusta tietoverkon yli. Sille on määritelty ohjelmoitavat rajapinnat, joihin perustuu muiden järjestelmien vuorovaikutus sen kanssa. Vuorovaikutuksessa käytetään XML-kielisiä viestejä. (W3C Web Services Architecture 2004.)
XML	Extensible Markup Language on metakieli, jolla voidaan määritellä rakenteellisia merkkauskieliä (W3C Recommendation 2008).
XML-skeema	(XML schema) XML-skeema kuvaa XML-dokumentin rakenteen. Skeemalla voidaan määrittää dokumentin elementit ja attribuutit, määrittää elementtien ja attribuuttien sijainnit dokumentissa ja määrittää elementtien ja attribuuttien sisältöjä. Skeema on sekä ihmisen luettavissa että koneellisesti käsiteltävissä. (W3C XML Technology 2010.) Esimerkiksi voidaan määrittää mitkä maksutapakoodit ovat sallittuja.
XML-validointi skeemaa vasten	Tarkistetaan vastaako XML-dokumentti skeemaa, jossa sen rakenne on määritelty. Tämä voidaan suorittaa koneellisesti.

1 Johdanto

1.1 Tausta

Minulla on noin 25 vuoden työhistoria ohjelmistonkehityksen parissa. Työtehtäviini on kuulunut eniten ohjelmointia, jonkin verran määrittelytyötä ja testausta. Työkokemukseni ansiosta tiedän, ettei laadukkaan ohjelmistotuotteen kehitys ole mahdollista ilman testaamista. Vaikka minulla on käytännön kokemusta ohjelmistotestauksesta, en ole aikaisemmin tutustunut siihen liittyvään teoriakehykseen. Tämä on ollut mielenkiintoinen matka ohjelmistotestauksen, sen automatisoinnin ja testausautomaatio-ohjelmiston maailmaan.

Nykyisin työskentelen kansainvälisen ohjelmistoyrityksen tuotekehitysyksikössä. Ylläpidämme ja kehitämme ohjelmistoja, joiden sovellusalue kuuluu kriittiselle taloushallinnon alueelle. SEPA (Single Euro Payments Area) -muutosten takia taloushallinnon ohjelmistoihin tehdään parhaillaan paljon muutostöitä. Samaan aikaan ohjelmistojen korkea laatu on pystyttävä säilyttämään.

Ohjelmistojen eri piirteiden on toimittava vielä muutostöidenkin jälkeen, siksi tarvitaan paljon regressiotestausta eli vanhojen ominaisuuksien uudelleentestausta. Regressiotestaus vie aikaa ja on työlästä, koska koko ohjelmisto ja sen eri piirteet on testattava alusta loppuun. Tämä on kallista ilman testauksen automatisointia.

Yrityksessä tehdyssä TPI (Test Process Improvement) -analyysissä havaittiin, että testauksen työkalujen (Test tools) käyttö oli hyvin vähäistä juuri tässä yksikössä. Yrityksen muilla tuotekehitysyksiköillä on testausautomaatiota kuitenkin jo sovellettu ja heillä on käytössä testaustyökaluohjelmisto TestPartner.

1.2 Tavoitteet

Tavoitteena on selvittää onnistuuko testauksen automatisointi jo yrityksessä käytössä olevalla automaatiotyökalulla tuotekehitysyksikössä, jossa työskentelen. Testaustyökaluohjelmiston käyttö ei ole suoraan kopioitavissa yrityksen muilta tuotekehitysyksiköiltä, erilaisten toiminnallisuuksien ja ohjelmistoarkkitehtuurien vuoksi.

Tarkoituksena on myös lisätä tietämystä testausautomaatiosta tuotekehitysyksikössäni. Keskusteluissa esimieheni kanssa on selvinnyt, että testauksen automatisoinnista tiedetään hyvin vähän eikä siitä ole kokemusta.

Toimeksiantajatuotekehitysyksikössä työskentelee ohjelmistokehityksen ammattilaisia, silti olen asettanut yhdeksi tavoitteeksi kirjoittaa tätä opinnäytetyötä niin, ettei sitä lukevan tarvitse ymmärtää ohjelmistotekniikoita.

1.3 Sisällöstä

Toisessa luvussa esittelen keskeisimpiä testaukseen liittyviä käsitteitä. Näiden ymmärtäminen auttaa tämän työn loppuosan lukemista ja helpottaa yleensäkin testausta koskevan teorian ymmärtämistä. Lisäksi korostan testauksen tärkeyttä laadun näkökulmasta. Toisessa luvussa on myös lyhyt katsaus yleiseen ohjelmistotestauksen teoriaan.

Kolmannessa luvussa kerron mitä tarkoitetaan ohjelmistotestauksen automatisoinnilla, miten ohjelmistotestausta voidaan automatisoida, ohjelmistotestauksen automatisoinnin työkaluista, ohjelmistotestauksen eduista ja ongelmista. Kolmannen luvun lopussa pohdin millaisia testitapauksia kannatta automatisoida.

Neljäs luku kertoo testauksen automatisoinnista TestPartner ohjelmistolla. Aluksi esittelen sovelluksen, jonka testausta on tarkoitus automatisoida. Ennen kuin voidaan automatisoida onnistuneesti ohjelmistotestausta, pitää manuaalisen testausprosessin olla kunnossa. Siksi kerron testattavaa sovellusta kehittävän tuotetiimin mallikkaasti toimivasta testausprosessista. Seuraavaksi esittelen TestPartner ohjelman. Sitten valitsen automatisoitavat testitapaukset, suunnittelen testejä ja kokeilen testauksen automatisointia TestPartner ohjelmalla. Lopuksi arvioin TestPartner työkalun soveltuvuutta ja kerron toimeksiantajan yksikössä järjestämästäni TestPartner-ohjelman esittelytilaisuudesta.

Viides luku sisältää yhteenvedon ja lyhyen päätelmän TestPartner ohjelman soveltuvuudesta testattavan sovelluksen testauksen automatisointiin.

2 Ohjelmistotestaus

2.1 Käsitteitä

Dynaamisessa testauksessa suoritetaan ohjelmaa (Ohjelmisto- ja algoritmitestaus 2003).

Staattinen testaus on katselmointia eli siinä ei suoriteta ohjelmaa (Ohjelmisto- ja algoritmitestaus 2003). Staattista testausta on lähdekoodin lukeminen ja tarkastelu yksin, pareittain tai ryhmässä.

Lasilaatikkotestaus (glass/white box testing, structural testing). Testaus perustuu ohjelman toteutukseen ja rakenteeseen eli käytännössä testin suunnittelussa tulee olla käytössä ohjelman tai sen osan lähdekoodi ja toiminnallinen määrittely. (Haikala & Märijärvi 2006, 291.)

Mustalaatikkotestaus (black box, functional testing). Mustalaatikkomenetelmä perustuu ohjelman toiminnallisiin ja laadullisiin vaatimuksiin. Ohjelmaa testataan ilman tietoa ohjelman sisäisestä rakenteesta, sen lähdekoodista yleensä tai algoritmista, jonka pohjalta ohjelma on toteutettu. (Haikala & Märijärvi 2006, 291.)

Harmaalaatikkotestaus Gray box testing. Harmaalaatikkotestaus yhdistää lasilaatikko- ja mustalaatikkotestauksen ominaisuudet. Testaus perustuu sekä koodikatselmointiin että ohjelman toiminnallisiin ja laadullisiin vaatimuksiin. (Haikala & Märijärvi 2006, 291.)

Integrointitestaus on moduulien tai ohjelmien osaryhmien välisten rajapintojen testausta (Haikala & Märijärvi 2006, 290).

Järjestelmätestauksessa testataan koko ohjelmistoa ja verrataan määrittelydokumentteihin. Tässä testataan myös ei-toiminnalliset osuudet kuten asennus-, suorituskyky- ja käytettävyystestit. (Haikala & Märijärvi 2006, 290.)

Moduulitestaus, Unit testing, Yksikkötestaus. Moduulitestaus on yhden moduulin testausta, jota varten saatetaan tarvita testipetiä (test bed) simuloimaan ohjelmiston muita osia. (Haikala & Märijärvi 2006, 289.)

Regressiotestaus on uudelleentestausta, jossa ennestään testattuja moduuleja joudutaan testaamaan yhteen moduulin tehtyjen muutosten tai korjausten vuoksi (Haikala & Märijärvi 2006, 290).

Savutestit Smoke Test, positiivisia testitapauksia, testijoukko jonka tarkoitus on tarkistaa version perustoiminnallisuudet (Sainio 2009,123).

Syöteavaruus tarkoittaa testattavalle ohjelmalle tulevia ulkopuolisia ärsykkeitä esimerkiksi Tulosta-painikkeen painallus. (Haikala & Märijärvi 2006, 285.)

Tulosavaruus sisältää ohjelmiston ulkopuolella havaittavissa olevat toiminnot esimerkiksi raportin tulostuminen. (Haikala & Märijärvi, 2006, 285.)

Testipetin (test bed) avulla testataan moduulin toimivuus. Testipeti simuloi ohjelmiston muita osia, joita moduuli tarvitsee toimiakseen. (Haikala & Märijärvi, 2006, 289.)

Testiskriptit muodostuvat sarjasta käskyjä ja ohjeita testaustyökalulle. Käskyjä toteuttamalla automaattinen testi tulee suoritettua (Pohjalainen 2003,24). Testiskripti koostuu koodiriveistä.

Validoinnissa tarkastetaan ohjelman soveltuvuutta käyttötarkoitukseensa (Haikala & Märijärvi, 2006, 51). Tämä tarkoittaa sen varmistamista, että ohjelmisto vastaa asiakkaan todellisia tarpeita.

Verifiointi on katselmointia, jolla varmistetaan, että ohjelmisto vastaa määrittelyitään ja siinä käydään läpi määrittelydokumentteja. (Haikala & Märijärvi, 2006, 51.)

Virhe (error, mistake, bug) ohjelmassa on poikkeama spesifikaatiosta. Tämän määritelmän mukaan testaus ilman spesifikaatiota on mahdotonta, koska tällöin ei voida todeta lopputuloksen oikeellisuutta. Tavallisesti testauksessa käytetään spesifikaationa toiminnallista määrittelyä ja teknistä määrittelyä. Erilaisista määrittelydokumenttien tulkinnoista johtuen asiakkaan ja ohjelmiston tuottajan välillä on tyypillisesti erimielisyyksiä siitä, mikä on ohjelman virhe ja mikä on ohjelman piirre tai ominaisuus. (Haikala & Märijärvi, 2006, 287.)

2.2 Testaus ja laatu

Ohjelmiston testaus tuo ensimmäisenä mielikuvan virheiden etsinnästä. Laadun mittaamisen näkökulmasta testauksen katsotaan käsittävän kaikki ne menetelmät, joilla pyritään mittaamaan ja parantamaan ohjelman laatua (Haikala & Märijärvi, 2006, 284).

Yleisesti laatu ymmärretään virheettömyytenä. Asiakkaan kannalta ohjelmistotuotteissa laatu tarkoittaa myös käytettävyyttä, joka sisältää muun muassa soveltuvuuden käyttötarkoitukseen ja suorituskyvyn. Asiakas voi myös kokea puutteet käytettävyydessä ohjelmiston virheinä. Ohjelmiston kehittäjän kannalta laadukkaalla ohjelmistotuotteella on myös selkeä arkkitehtuuri ja sen ylläpito on nopeaa ja helppoa.

Ohjelmistoja kehittävän yrityksen kannalta laatu voidaan myös määrittää asiakkaan tarpeiden täyttämisenä tehokkaalla ja kannattavalla tavalla. Yrityksen toiminta ei voi olla laadukasta, jos kannattavuus kärsii asiakastarpeiden täyttämisen takia. (Haikala & Märijärvi 2006,192,193.) Ohjelmistotestauksen määrä on siis pyrittävä optimoimaan riittäväksi.

Ohjelmistotuotteiden laadun mittaamisessa prosessin eri vaiheissa löytyneet virheet lienevät keskeisimpiä mittareita. Asiakkaan löytämä virhe voi saada vakavuudessa suuremman painoar-

von kuin testausvaiheessa sisäisesti löydetty virhe. Sillä pahimmillaan asiakkaan havaitsemat virheet voivat vaikuttaa tuotteen maineeseen ja sitä kautta koko yrityksen imagoon.

Tosiasia on myös, että mitä aikaisemmassa vaiheessa tuotekehitysprosessia virhe löytyy, sitä vähemmän kustannuksia sen korjaamisesta aiheutuu. Tuotekehitysprosessin edetessä aikajanalla, virheen kustannukset kasvavat kymmenkertaisesti. Määrittelyvaiheessa löydetty ja korjattu virhe ei maksa mitään tai korkeintaan yhden laatukustannusyksikön, kun sama virhe maksaa 10 yksiköstä 100 yksikköön koodaus ja testausvaiheeseen päätyessään. Asiakkaan löytämänä tuon virheen kustannukset saattavat kivuta tuhansiin, jopa miljooniin kustannusyksiköihin (Patton, 2006, 18.) Näin kävi tamperelaiselle pörssiyhtiölle Solteqille, jonka kassaohjelman pieni ohjelmavirhe aiheutti lopulta miljoonaluokan kustannukset oikeudenkäyntikuluineen (Kauppalehti 2010). Parhaimmillaan tehokas testaus siis vähentää tuotekehityskustannuksia.

2.3 Testaus ohjelmistonkehitysprosessissa

2.3.1 Testauksen vaiheet

Ohjelmistojen tehokas testaaminen on oltava suunnitelmallista. Suunnitteluvaiheesta syntyvät testaussuunnitelma ja testitapaukset. Seuraavat vaiheet ovat testiympäristön luonti, testin suorittaminen ja tulosten tarkastelu. Näihin työvaiheisiin sekä niihin liittyviin ohjelmiston korjaamiseen kuluu noin puolet ohjelmistoprosessin resursseista. (Haikala & Märijärvi 2006,283).

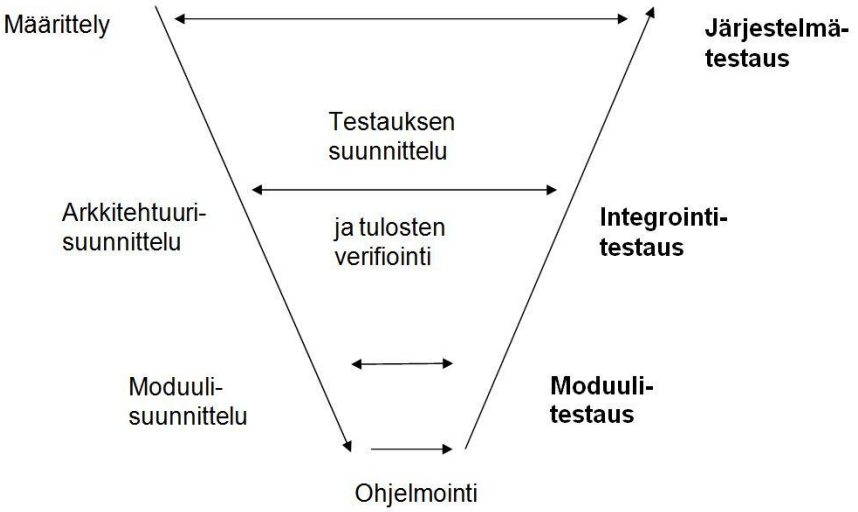
2.3.2 Testausstrategiat

Testauksessa on kaksi peruslähestymistapaa: lasilaatikkotestaus ja mustalaatikkotestaus. Lasilaatikkotestauksessa käytetään hyväksi tietoa ohjelman toteutuksesta eli käytännössä testin suunnittelussa tulee olla käytössä ohjelman tai sen osan lähdekoodi ja toiminnallinen määrittely. Testitapaukset pyritään suunnittelemaan niin, että kaikki koodin funktiot ja haarat tulisi testatuiksi. Mustalaatikkomenetelmä perustuu vain ohjelman toiminnallisiin ja laadullisiin vaatimuksiin. Ohjelmaa testataan ilman tietoa ohjelman sisäisestä rakenteesta tai lähdekoodista. (Haikala & Märijärvi 2006,291.)

Harmaalaatikkotestaus (gray box) yhdistää lasilaatikko- ja mustalaatikkotestauksen ominaisuudet. Siinä hyödynnetään tietoa ohjelman toteutusperiaatteista. Harmaalaatikkotestaus tarkistaa käyttäjälle näkyvän lopputuloksen ja arvioi järjestelmän eri osien sopivuutta kokonaisuuteen. Tämä periaate soveltuu hyvin erityisesti web–sovellusten testaamiseen, koska ne koostuvat useista erilaisista komponenteista. (Software Business Competence 2006.)

2.3.3 Testaustasot ja V-malli

Testauksen V-mallissa erottuvat eri testaustasot, joita ovat moduulitestaus (yksikkötestaus, unit testing), integrointitestaus ja järjestelmätestaus (kuvio 1). (Haikala & Märijärvi 2006,288.)



KUVIO 1. Kuvassa on testauksen V-malli. (Haikala & Märijärvi 2006, 289).

V-malli näyttää selkeästi miten eri testaustasojen testauksen suunnittelu liittyy ohjelmistoprosessin eri suunnitteluvaiheisiin. Kunkin testaustason tulokset todetaan oikeiksi vertaamalla niitä vastaaviin suunnitteludokumentteihin eli spesifikaatioihin. Näin V-mallin mukaan testaus perustuu spesifikaatioihin, jotka muodostuvat ohjelmiston eri kehitysvaiheissa.

Moduulitestaus on yhden moduulin testausta, jota varten saatetaan tarvita testipetiä (test bed) simuloimaan ohjelmiston muita osia. (Haikala & Märijärvi 2006,289.) Moduulisuunnitteluvaiheen spesifikaatio on tekninen suunnitelma.

Integrointitestaus on moduulien tai ohjelmien osaryhmien välisten rajapintojen testausta. Järjestelmätestauksessa testataan koko ohjelmistoa ja verrataan määrittelydokumentteihin. Tässä testataan myös ei-toiminnalliset osuudet kuten asennus-, suorituskyky- ja käytettävyystestit. (Haikala & Märijärvi 2006,289.)

Siirryttäessä V-mallissa ylöspäin testaus muuttuu lasilaatikkotestauksesta enemmän mustalaatikkotestaukseksi (Haikala & Märijärvi 2006,289). Tällöin tietämys ohjelman teknisestä toteutuksesta vähenee.

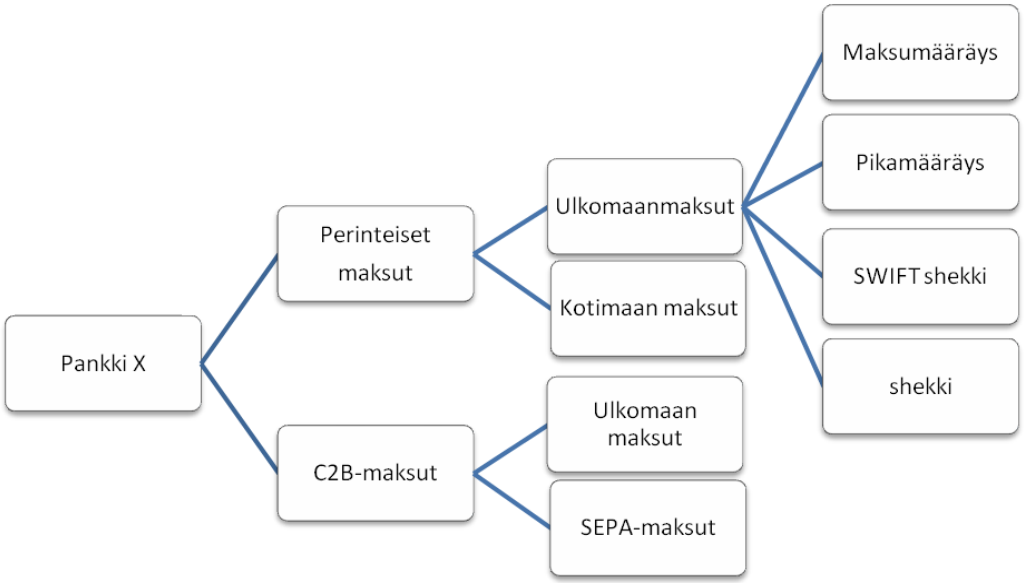
Testauksen V-malli sopii hyvin kuvaamaan testausprosessia sellaisessa ohjelmistonkehitysprosessissa, josta erottuvat selvästi määrittely-, suunnittelu- ja toteutusvaiheet kuten esimerkiksi vesiputousmallissa. Näin ei välttämättä ole ketterissä menetelmissä. TDD (Test Driven Development) ketterässä ohjelmiston kehitysmenetelmässä etukäteen kirjoitetut testitapaukset voivat korvata vaatimusmäärittelyt, sillä ne kertovat yksiselitteisesti kuinka ohjelman tulisi toimia (Sainio 2009,27).

2.3.4 Testitapaukset

Dynaamisessa lasilaatikkotestauksessa testitapaukset valitaan lähdekoodin perusteella niin, että eri funktiot ja ehtohaarat tulevat testatuiksi mahdollisimman kattavasti. (Haikala & Märijärvi 2006,291,292).

Dynaamisessa mustalaatikkotestauksessa testitapaukset suunnitellaan ohjelmiston määrittelydokumenttien perusteella. Määrittelyistä päätellään oikeat lopputulokset. Testitapausten valinta aloitetaan jakamalla syöteavaruus ekvivalenssiluokkiin. Ekvivalenssi tarkoittaa samanarvoisuutta ja rinnasteisuutta. Testitapaukseksi riittää yksi edustaja aina yhdestä ekvivalenssiluokasta, sillä yhden ekvivalenssiluokan syötteillä ohjelmisto toimii samoin eli tuottaa saman tulosavaruuden. (Haikala & Märijärvi 2006,291,292.)

Esimerkiksi toimeksiantajatuotekehitysyksikössä pankkiyhteysohjelman maksujen testauksessa voitaisiin aloittaa ekvivalenssiluokkiin jakaminen eri pankkiryhmien mukaan. Maksut jaettaisiin Nordean, OP Pohjolan, Sampo Pankin ja Samlink-pankkien maksuihin. Oy Samlink Ab ylläpitää useamman eri pankin IT-palveluita. Tässä esimerkissä törmättäisiin heti osituksen oikeellisuusongelmaan. Todellisuudessa jaottelu ei ole läheskään riittävä. Samankin pankin kotimaan- ja ulkomaanmaksuilla on eri maksutapoja ja kulutyyppejä. Ohjelmasta lähetettävillä perinteisillä maksuilla on erilaisia ascii-tiedostoformaatteja, kun taas siirryttäessä SEPAan, maksujen formaatti on XML-kielinen C2B. Siirtymäaikana voidaan periaatteessa lähettää maksuja sekä perinteistä tiedonsiirtokanavaa pitkin että uutta pankin Web Services -palvelua hyödyntäen. Käytännössä yhden pankin maksut testataan kerrallaan. Seuraavaksi valitaan testataanko perinteisiä maksuja vai C2B-maksuja. Lopulta luokittelu hahmottuu hierarkkiseksi malliksi (kuvio 2).



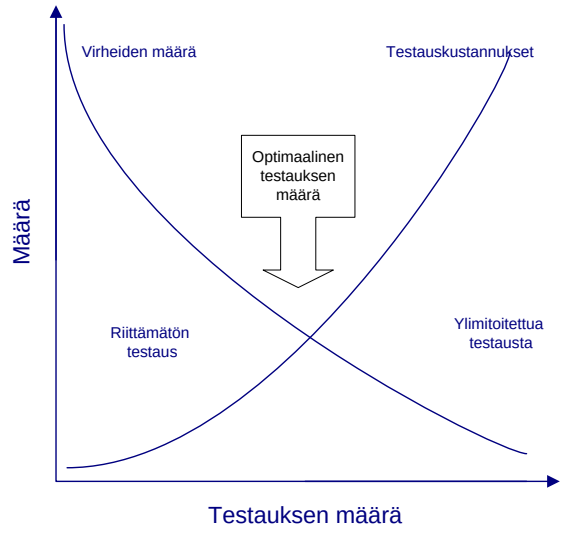
KUVIO 2. Maksujen jaottelua testitapauksiin. Selvyyden vuoksi kaikkia alakohtia ei ole kuvattu.

Luokkien rajoilla olevien arvojen valintaa testitapauksiksi kutsutaan raja-arvoanalyysiksi (Haikala & Märijärvi 2006,291,292). Raja–arvoja on yleensä vaikea löytää kuten myös edellisessä maksujen testausesimerkissä. Kuviossa 2 C2B-maksujen alta löytyvät ulkomaan- ja SEPA-maksut. Tässä voisivat SEPA-maksut luokkaan kuuluvat Suomen rajan yli lähetettävät maksut eli SEPA-alueen maihin lähetettävät maksut kuulua raja-arvoihin. Käytännössä raja-arvo esimerkkien hakeminen edellä mainitusta maksuesimerkistä on keinotekoista. Kotimaiselle maksun saajalle lähetettävien SEPA-maksujen voisi myös ajatella edustavan raja-arvoja, sillä ne sisältävät omia erityispiirteitä. Nämä ansaitsevat kuitenkin oman testitapausluokkansa, jota ei ole kuviossa 2 merkitty näkyviin.

2.3.5 Testauksen riittävyys

Testauksen riittävyyttä on vaikea arvioida. Ohjelman täysin kattava testaus on mahdotonta, sillä silloin pitäisi testata kaikki mahdolliset syötteet, niiden kaikki yhdistelmät, tulokset ja ohjelman kaikki toimintoketjut. Käytännössä resurssit eivät koskaan riitä täysin kattavaan testaukseen. Tämän lisäksi ohjelmistomääritykset voivat olla tulkinnanvaraisia ja näin ollen myös ohjelman tietynlaisen toiminnan määrittäminen virheeksi voi olla kiistanalaista. Tällöin ei pystytäkään määrittämään tarkasti onko ohjelma täysin kattavasti testattu virheettömäksi.

Ohjelmistotestauksessa on löydettävä optimaalinen työmäärä, on päätettävä mitä ja miten kannattaa testata riskien minimoimiseksi (kuvio 3).



KUVIO 3. Testauksen optimaalinen määrä (Patton 2006, 40, suomennos LA).

Testauksen lopettamiskriteerit määritellään testaussuunnitelmassa. Järjestelmätestauksessa lopettamiskriteerinä voi olla tietty määrä löydettyjä virheitä niiden vakavuusasteet huomioiden. Moduuli- ja funktiotestauksen riittävyyden arvioinnissa voidaan käyttää apuna erilaisia koodin rakenteen tutkimiseen perustuvia mutkikkuus- ja kattavuusmittoja. Polkutestauksessa pyritään testaamaan mahdollisimman monta toimintoketjua ohjelman läpi (Haikala & Märijärvi 2006,293).

3 Automaattinen ohjelmistotestaus

3.1 Mitä on ohjelmistotestauksen automatisointi?

Automatisointi on manuaalisen testauksen suorittamista koneellisesti, jonkin testausohjelman avulla (Pohjalainen 2003,23). Näin väljästi ohjelmistotestauksen automatisointi määriteltiin Suomessa vuonna 2003. Amerikkalainen Patton (2006) asettaa ohjelmistotestauksen automatisoinnille kovemmat vaatimukset väittämällä, että ohjelmistotestaus on automatisoitu vasta, kun testausohjelma suorittaa testit, etsii virheet, analysoi näkemäänsä ja kirjoittaa lokia tuloksista.

Tyypillisiä testausautomaation tehtäviä ovat testiskriptien kehittäminen, ylläpito ja suoritus sekä testitulosten verifiointi. (Software test automation in practice: empirical observations, 2009.)

Automaatiolla ei voida täysin korvata kaikkia manuaalisia testejä. Aina tarvitaan myös testaajia, jotka osaavat analysoida testien tuloksia. (Sainio 2009, 120.)

3.2 Miten ohjelmistotestausta voidaan automatisoida?

3.2.1 Makrojen nauhoitus

Perustapa automatisoida ohjelmistotestausta on nauhoittaa ensimmäisellä kerralla testaajan tekemät toimenpiteet näppäimistöllä ja hiirellä ohjelmiston käyttöliittymässä ja ajaa ne uudestaan playback-toiminnolla, kun tarvitaan uusintatestausta. (Patton 2006, 240.)

3.2.2 Testaus ohjelmoitavilla makroilla

Askel eteenpäin makrojen nauhoituksesta ovat ohjelmoidut makrot, joissa annetaan yksinkertaisia käskyjä playback-toiminnolle. Näiden etuna verrattuna pelkkään nauhoitukseen on se, että niissä voidaan välillä näyttää testitulos ja pyytää testaajalta kuittausta testin onnistumisesta. Automaattista testitulosten verifiointia näillä ei kuitenkaan voi tehdä. Ohjelmoiduilla makroilla voidaan ratkaista monia nauhoitettujen makrojen ajastusongelmia, sillä niissä voidaan viiveelle asettaa tietyn ehdon täyttyminen ennen suorituksen jatkumista sen sijaan, että viive perustuisi tiettyyn absoluuttiseen aikaan. (Patton 2006, 242.)

Ohjelmoitavat makrot ovat rajoittuneet komentojen peräkkäiseen suorittamiseen ja kykenevät vain koodisilmukoiden ja toistojen suoritukseen. Perinteisten ohjelmointikielien muuttujat ja päättelylausekkeet puuttuvat niistä. (Patton 2006, 243.)

3.2.3 Testaus täysin ohjelmoitavilla automaattisilla testaustyökaluilla

Täysin ohjelmoitava automaattinen testaustyökalu sisältää perinteisen ohjelmointikielen, makrokomennot, makrojen nauhoituksen ja tulosten verifioinnin. Tärkein ominaisuus näissä automaattisissa testaustyökaluissa on tulosten verifiointi. Tämä voi perustua näyttökaappauksiin, arvojen tarkistuksiin ja tiedostovertailuihin. (Patton 2006, 243.)

Näyttökaappausvertailu toteutetaan niin, että ensimmäisellä testauskerralla otetaan talteen näyttöjen kuvat oikeista lopputilanteista. Testaustyökalu vertaa uusintakierroksilla näyttöjä tallennettuihin ja raportoi virheen, jos eroja löytyy. Näyttökaappauksiin perustuvaa vertailua on työlästä ylläpitää ja se sopii ainoastaan ohjelmistoille, joiden käyttöliittymä muuttuu harvoin. (Patton 2006, 244.) Arvojen vertailussa automaatiotyökalu vertaa käyttöliittymän yksittäisten elementtien kuten tietokenttien arvoja. (Patton 2006, 244.)

Tiedostovertailussa automaatiotyökalu lukee ohjelman tallentaman tiedoston ja vertaa sitä aiemmin oikein muodostettuun tiedostoon. Automaatiotyökalu täytyy erikseen ohjelmoida ohittamaan vertailussa päiväykset, laskurit ja muut muuttuvat tiedot tiedostoissa, joiden vuoksi vertailu voi epäonnistua. (Patton 2006, 245.)

3.2.4 Satunnaistestaus: test monkey

Yksi automaatiotyökalun muoto on test monkey (testiapina), jonka tarkoituksena on simuloida käyttäjien toimenpiteitä. Apina-nimitys tulee ajatuksesta, että tilastollisesti miljoona apinaa näppäilemässä miljoonalla näppäimistöllä miljoonassa vuodessa voisi kirjoittaa Shakespearen näytelmän veroisen teoksen. Test monkeyn avulla voitaisiin teoriassa löytää ne ohjelmistossa piilevät virheet, jotka tulevat esiin vasta tuhansien tai miljoonien ihmisten käyttäessä ohjelmistoa. (Patton, 2006, 246.)

Dumb monkey klikkaa ja näppäilee satunnaisesti eikä tiedä mitään testattavasta ohjelmistosta. Tällainen satunnainen näytölle klikkailu voidaan toteuttaa automaatiotyökalun koodi-silmukassa. Dumb monkey jatkaa klikkailua ja näppäilyä kunnes tulee ulos koodi-silmukasta tai testattava ohjelma kaatuu. (Patton, 2006, 247.) Semi-Smart monkey saadaan aikaan lisäämällä lokin kirjoitus dumb monkey:iin (Patton, 2006, 248).

Smart monkey tietää missä on, mitä voi siellä tehdä, minne voi mennä, missä on ollut ja onko se mitä on nähnyt oikein. Smart monkey tietää mitä painikkeita voi painaa, mitkä valikon valinnat voi valita ja mihin syöttää tietoa. Smart monkey voi tarkistaa toimintansa tuloksia ja verifioida niitä. Kun testitapaukset on ohjelmoitu, smart monkey voi suorittaa ne satunnaisesti, etsiä virheitä ja kirjoittaa lokia tuloksista. (Patton 2006, 249.) Smart monkeyn toiminta perustuu sille syötettävään testattavan ohjelmiston tilakarttaan, joka sisältää kaikki ohjelman tilat, tilassa sallitut käyttäjän toimenpiteet ja näiden oikeat tulokset. (Patton 2006, 249,250.)

3.3 Työkaluista

Dynaamisen testauksen automatisointiin voidaan käyttää testipetigeneraattoreita, testitapausgeneraattoreita, makroja nauhoittavia -työkaluja, täysin ohjelmoitavia testaustyökaluja, laajoja testiautomaatiosovelluksia, vertailijoita, testikattavuustyökaluja ja IDE:jä (Integrated Development Enviroment) eli ohjelmointiympäristöjä.

Yksikkötestauksessa käytettävä testipetigeneraattori (Test bed generator) luo testattavalle moduulille testipetin, jolle voidaan kuvata testikuvauskielellä ajettava testi sekä halutut testitulokset. (Haikala & Märijärvi 2006,297.)

Testitapausten automaattinen generointi voi olla mahdollista, jos käyttöliittymä on määritelty tilakaaviona. Tilakaavion perusteella voidaan generoida testitapaukset ja automatisoida testitulosten tarkastelu. (Haikala & Märijärvi 2006,297.)

Suunnittelussa käytettävissä UML-työkaluissa saattaa olla koodin generointia ja testitapausten johtamista UML-mallista. Tällaisia työkaluja ovat esimerkiksi NetBeansIDE ja Rational Rose. StarUML on avoimen lähdekoodin UML-mallinnustyökalu. (Sainio 2009, 147.)

Makrojen nauhoitus -testaustyökaluja voidaan nimittää myös GUI–ajureiksi. Näillä voidaan automatisoida graafisen käyttöliittymän testausta nauhoittamalla GUI-karttoja (Sainio 2009, 154). Myös täysin ohjelmoitavat testaustyökalut sisältävät GUI-ajureiden ominaisuuksia sen lisäksi, että niillä on mahdollista myös vertailla tuloksia. Aikaisemmin Microsoftin kehittämä, myöhemmin IBM:n ylläpitämä Visual Test on esimerkki tällaisesta työkalusta. (Patton 2006, 243.) Laajoja testiautomaatiosovelluksia ovat HP QualityCenter ja Micro Focuksen TestPartner.

Vertailijoita ovat tiedostorakenteiden vertailutyökalut, dokumenttien sisällön vertailutyökalut ja tietokantojen vertailutyökalut. Tiedoston vertailutyökalu on esimerkiksi Comparator ja WinMerge. Dokumenttien sisältöä voidaan vertailla Diff Doc –työkalulla. Tietokantojen rakennetta ja sisältöä voidaan verrata CompareDatabase-työkalulla, joka toimii Access- ja SQL Server -tietokantojen kanssa. (Sainio 2009, 160.)

Kattavuustyökaluilla voidaan mitata ohjelmiston koodin kattavuuslukuja tai testauksen kattavuutta. (Sainio 2009, 152). Esimerkiksi PartCover on testikattavuustyökalu .NET-kielille. GTC on avoimen lähdekoodin kattavuustyökalu C- ja C++-kielille. (Sainio 2009, 152,153.)

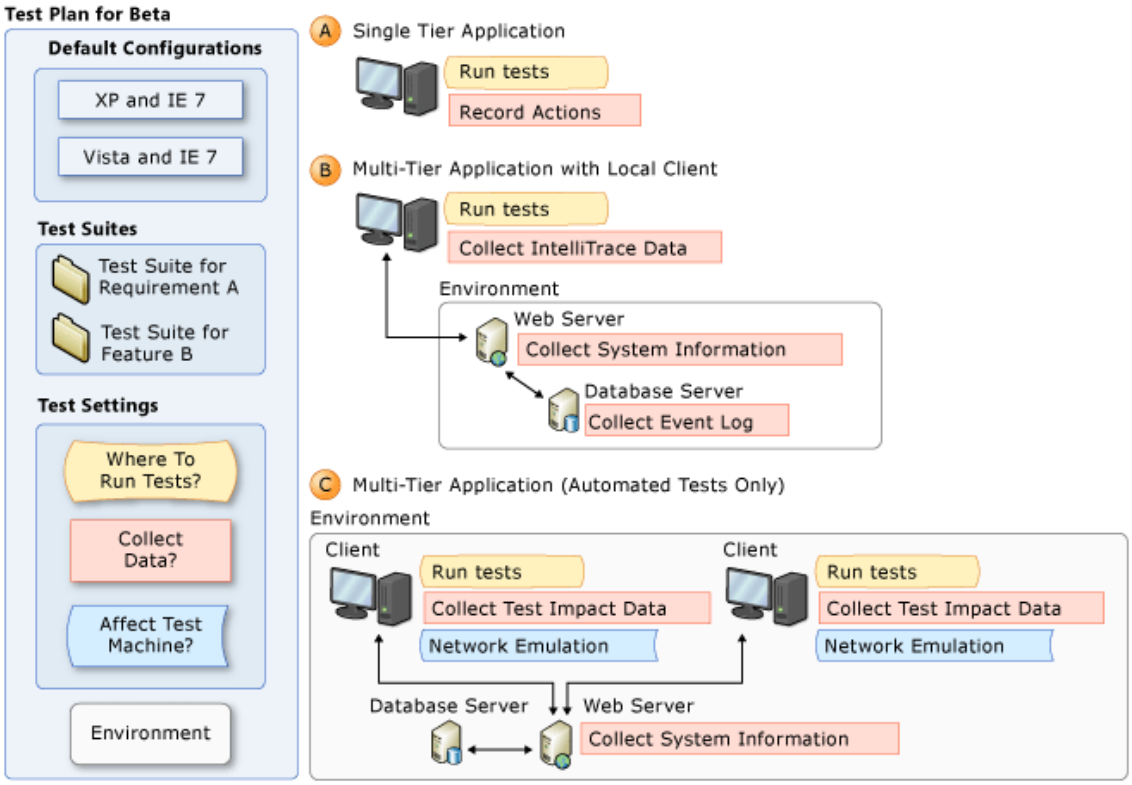
Ohjelmointiympäristöt eli IDEt sisältävät testaus- ja analysointityökaluja. Niihin voidaan asentaa myös testauksen automatisointia varten lisäosia. (Sainio 2009, 164.) Tällainen ohjelmointiympäristö on esimerkiksi Microsoft Visual Studio.

Microsoft Visual Studio

Microsoft Visual Studio 2008 Professional ympäristössä voidaan suorittaa testitapauksia järjestettyinä joukkoina. Siinä löytyy myös työkalut web-sovellusten yksikkötestien automatisointia varten. Esimerkiksi Object Test Bench (OTB) oliotestipenkillä voidaan testata yksinkertaisia luokkia ja niiden metodeita (Microsoft Visual Studio 2008 Documentation).

Microsoft Visual Studio kehitysympäristöön voidaan integroida myös muiden toimittajien kaupallisia ohjelmistoja. Tällainen on esimerkiksi Testwellin CTC++ kattavuusanalysaattori. (Sainio 2009, 165.)

Microsoft Visual Studio 2010 ympäristössä on uusi sovellus nimeltään Microsoft Test manager, jonka puitteissa voidaan luoda testisuunnitelmat, testisarjat (test suites), testien kokoonpano ja testitapaukset yksityiskohtaisesti (kuvio 4). Automatisoidut testit voidaan käynnistää testisuunnitelmasta Test Manager -sovelluksessa, ne voidaan käynnistää paikallisesti tai etänä Microsoft Visual Studiossa tai ne voidaan käynnistää komentoriviltä. Microsoft Visual Studiossa automatisoidut testit voidaan käynnistää myös ajastetusti ohjelmakoodin käännöksen ja linkkauksen (build) jälkeen. **(**MSDN Library 2010.)



KUVIO 4. Kuvassa on kolme testausesimerkkiä, jotka käyttävät eri ympäristöjä Microsoft Test Manager -sovelluksesta käsin **(**MSDN Library 2010)**.**

Microsoft Visual Studio 2010 Ultimate tai Visual Studio 2010 Premium ympäristöissä voidaan luoda automatisoituja UI (User Interface) -testejä, jotka sisältävät käyttöliittymän toiminnallisen testauksen ja käyttöliittymän kontrollien validoinnin. Näiden automatisoitujen UI-testien avulla on mahdollista testata käyttöliittymän toimivuus koodimuutosten jälkeen nopeammin kuin manuaalitestauksella. (MSDN Library 2010.)

Muita automatisoivia testaustyökaluja

Web Service-rajapintojen testaukseen käytetään SOA (Service Oriented Architecture) -testereitä. Ne automatisoivat rajapintoja ja generoivat niiden perusteella testiympäristön. Tällaisia ovat esimerkiksi Rational Tester for SOA Quality ja avoimen lähdekoodin ohjelma Testmaker. (Sainio 2009, 158.)

3.4 Edut ja hyödyt

Automaation avulla nopeutetaan ja tehostetaan ohjelmistotestausta. Esimerkiksi laskimen tyyppisen ohjelman automaattitestauksessa saadaan laskettua 10, 100 tai jopa 1000 testitapausta, kun samaan aikaan manuaalitestauksella saadaan laskettua yksi testitapaus (Patton 2006,232.)

Automatisoinnilla voidaan testata ohjelmistoja kattavammin kuin manuaalitestauksella. Automaatiolla voidaan suorittaa testejä, joita manuaalisesti olisi mahdotonta toteuttaa. Esimerkiksi kuormitustesteissä voidaan simuloida useita satoja yhtäaikaisia käyttäjiä.

Koko ohjelmiston kehitysprosessia voidaan nopeuttaa testausautomaatiolla. Uudesta ohjelmistoversiosta saadaan virheet esiin jo varhaisessa vaiheessa, automatisoitujen yksikkö ja savutestien avulla. Savutesteillä varmistetaan ohjelmiston perusominaisuuksien toimivuus ohjelmistoa koostettaessa. (Sainio 2009, 123.) Kustannussäästöjä syntyy, kun virheet löydetään varmimmin jo ohjelmiston kehitysvaiheessa. Kone ei väsy eikä tee inhimillisiä erehdyksiä tai unohda testata jotain osa-aluetta.

Automaatio voi lisätä testaajan työn mielekkyyttä vähentämällä yksitoikkoisia, toistuvia työtehtäviä. Testaaja voi keskittyä enemmän testien suunnittelutyöhön ja esimerkiksi käytettävyystesteihin. Näin automaatio tehostaa testausta välillisesti, sillä hyvin suunniteltu testaus on myös tehokasta.

3.5 Ongelmat ja haasteet

Automatiikkaa käytetään usein toistettavaan yksikkötestaukseen tai regressiotestaukseen. Tällöin testit suoritetaan joka kerta, kun muutoksia tehdään. Automaation toteuttaminen on työlästä ja kannattaa toteuttaa vain usein toistettaviin testeihin. Kuitenkaan erottelu manuaalitestaukseen ja automatisoitavaan testaukseen ei ole ihan näin yksinkertaista, sillä automatisoinnissa suuri huolenaihe voi olla ohjelmiston testattavuus. Tällöin ohjelmiston toteutustavan eli huonon koodin takia ohjelmiston testausta on mahdotonta automatisoida luotettavasti.
(Software test automation in practice: empirical observations, 2009.)

Testauksen automatisointi vie alussa enemmän aikaa ja resursseja ja siihen kohdistuu helposti epärealistisia odotuksia. Luullaan, että automaatio voidaan toteuttaa pysyväksi pienellä työmäärällä. Todellisuudessa testauksen automatisointi on verrattavissa tuotekehitysprojektiin. Testiautomaatiota täytyy jatkuvasti ylläpitää ja päivittää testattavan ohjelmiston muuttuessa. Automatisoinnin toteuttajilla on usein oltava myös ohjelmointitaidot. Ohjelmistokehityksen testausprosessin pitää olla kunnossa ja toimiva, ennen kuin voidaan onnistuneesti toteuttaa automatisointiprojektia.

Automatisointia ei kannata lähteä toteuttamaan kehitteillä olevaan ohjelmistotuotteeseen, jossa käyttöliittymä vielä jatkuvasti muuttuu. Kuitenkin ohjelmiston arkkitehtuurin suunnittelussa pitäisi pystyä luomaan jo edellytykset testauksen automatisoinnille. Toisaalta testausautomaatio pro-

jektin aloittaminen ei ole enää järkevää ohjelmistotuotteeseen, joka on elinkaaren loppuvaiheessa.

3.6 Automatisoitavat testitapaukset

Testauksen automatisointi sopii parhaiten V-mallin alimmilla tasoilla olevaan testaukseen. Järjestelmä- ja hyväksyntätestausta ei kannata automatisoida. Niissä on niin paljon käsitteellistä vertailua ohjelmiston spesifikaatioihin. Käytettävyyttä lienee mahdotonta arvioida koneellisesti. Tähän vaadittaisiin jo tekoälytyyppinen järjestelmä, jolla olisi hankalaa korvata ihmisaivojen toimintaa.

Kuten edellä on jo mainittu, usein toistettavat testit kannattaa automatisoida. Tällaisia ovat esimerkiksi yksikkö- ja savutestit. Savutesteillä tarkoitetaan testijoukkoa, jonka tarkoitus on tarkistaa version perustoiminnallisuudet. Ihanteellista olisi, jos savutestit käynnistyisivät automaattisesti ohjelmiston uuden asennuspaketin valmistuttua. Automatisoidut yksikkötestit toimivat myös ohjelmoijien tukena, sillä niillä testataan alimman tason funktioita ja luokkia. (Sainio 2009, 123, 124.)

Suorituskyky- ja kuormitustestit suoritetaan aina samoilla testitapauksilla uusille ohjelmistoversioille ja tuloksia verrataan edellisistä versioista saatuihin tuloksiin. Nämä testit ovat otollisia automatisoinnin kohteita myös usein toistuvina testeinä. Automaatiolla voidaan toteuttaa myös kestävyystestausta, jossa selvitetään miten ohjelmisto kestää viikkojen tai kuukausien ajan kestävää yhtäjaksoista käyttöä (Sainio 2009, 125).

4 Testauksen automatisointi TestPartner työkalulla

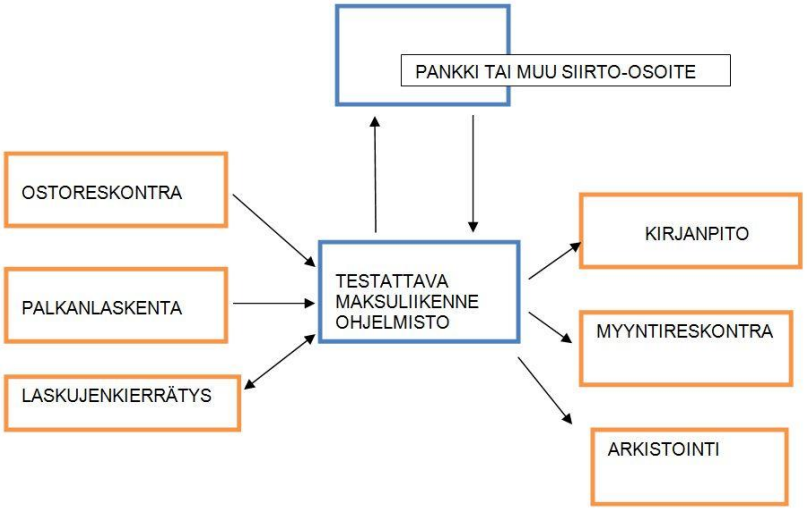
4.1 Automaattisesti testattava ohjelmisto

4.1.1 Ohjelmiston yleiskuvaus

Tässä opinnäytetyössä on tarkoitus selvittää kotimaan ja pohjoismaiden markkinoille suunnatun taloushallinnon ohjelmiston testauksen automatisointia. Tällä Windows-ohjelmistolla yritykset ja yhteisöt hoitavat niiden ja pankkien välisen maksuliikenteen kotimaassa sekä pohjoismaissa. Ohjelmisto sisältää yhteysmahdollisuuksia myös muihin siirto-osoitteisiin kuin pankkeihin, kuten verkkolaskuoperaattoreihin ja eläkevakuutusalan palvelukeskuksiin (kuvio 5).

Ohjelmistossa on liittymiä yrityksen muihin taloushallinnon ohjelmistoihin, joita ovat esimerkiksi reskontra-, kirjanpito-, palkanlaskenta-, arkistointi- ja laskujen kierrätys -ohjelmistot (kuvio 5). Liittymät perustuvat lähinnä tiedostojen siirtoon. Ohjelmistoon on integroitu ostoreskontran perustoiminnot, joka sisältää kotimaan ostovelkamaksujen käsittelyn sekä valuuttaostoreskontran.

Tämän ohjelmiston ja pankkien välinen maksuliikenne sisältää esimerkiksi maksujen, palkkojen ja verkkolaskujen lähettämistä, tiliotteiden, maksuerittelyiden, maksupalautteiden ja saapuvien viitesuoritusten noutamista. Ohjelmistolla voidaan tehostaa yritysten maksuliikenneprosessia automatisoinneilla. Tiliotteiden noudot pankista voidaan automatisoida ajastamalla ne tapahtuviksi aina samaan aikaan vuorokaudesta. Tiliotteiden saavuttua ohjelmiston tietokantaan, ohjelmisto käynnistää tapahtumien automaattitiliöinnin. Tämän jälkeen tiliöinnit voidaan siirtää kirjanpitoon. Ohjelmistossa voidaan automatisoida myös pankista noudon jälkeen käynnistyväksi viitesuoritusten jako myyntireskontriin siirrettäviin tiedostoihin.



KUVIO 5. Testattava maksuliikennesovellus on kaaviossa keskellä.

Ohjelmistosta on yhteydet seuraaviin Suomessa toimiviin pankkeihin:

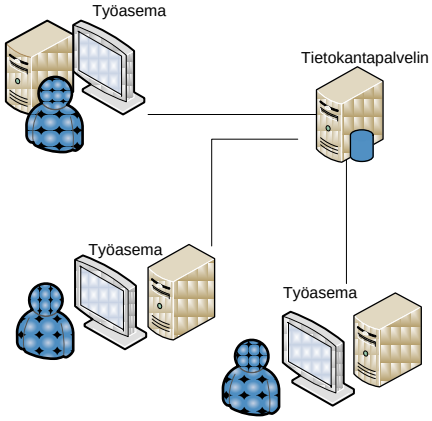
- Nordea
- OP-Pohjola
- Sampo Pankki
- Paikallisosuuspankit
- Säästöpankit
- Tapiola Pankki
- S-Pankki
- Ålandsbanken
- Handelsbanken
- SEB
- Danske Bank
- Swedbank Suomi
- DnB NOR

Edellä lueteltujen lisäksi ohjelmistossa on myös suoria yhteyksiä ulkomaisiin pankkeihin ja muun muassa Ruotsin Bankgirocentraleniin (BGC), joka hoitaa maan sisäistä maksujen välitysjärjestelmää.

Tiedonsiirto ohjelmiston ja pankkien välillä on perustunut FTP (File Transfer Protocol) -tiedostonsiirtomenetelmään ja PATU (Pankkien turvamenettely) -tietoturvaan. Nykyisin pankit ovat siirtymässä tiedonsiirtoon, joka perustuu Web Services- ja PKI (Public Key Infrastructure) -standardeihin. Ohjelmistossa voi siis olla käytössä yhtä aikaa kaksi erilaista tiedonsiirtokanavaa. SEPA on tuonut uudet XML-tiedostoformaatit perinteisten ascii peräkkäistiedosto -formaattien rinnalle.

4.1.2 Järjestelmäarkkitehtuurista

Ohjelmiston palvelinversio voidaan asentaa työasema – palvelin (client–server) arkkitehtuurin mukaisesti, jolloin yhtäaikaisia käyttäjiä voi olla useita. Palvelinversio voidaan asentaa myös kokonaan samalle palvelinkoneelle (kuvio 6). Yksittäiskäyttöversiossa ohjelmassa voi olla vain yksi käyttäjä kerrallaan.



KUVIO 6. Testattavan ohjelmiston työasema-palvelinversio arkkitehtuuri**.**

4.1.3 Toiminnallisuuksia

Testattavaan ohjelmistoon on vuosien varrella kertynyt runsaasti erilaisia piirteitä ja toimintoja. Seuraavassa luettelossa on keskeisempiä toimintoja:

- Tiedonsiirto
    - Aineistojen lähetys
    - Aineistojen nouto
    - SEPA-muunnin
- Maksun tallennus
- Liikekumppanitietojen tallennus
- Maksuun valinta
- Maksujen tulostus
- Maksut kirjanpitoon
- Maksuerittelyiden tulostus
- Reskontraote
- Laskujen käsittely
- Palkkojen tallennus
- Ajantasasaldot
- Tiliotteet
    - Tiliotetiliöinti
    - Tiliotteen automaattitiliöintisääntöjen tallennus
    - Automaattinen tiliöinti
    - Tiliotteet kirjanpitoon
    - Tulostus
- Viitesuoritukset
    - Viitteiden jakosääntöjen tallennus
    - Viitteiden jako
    - Automaattinen viitteiden jako
- Ulkomaanmaksujen tallennus
- Ulkomaan liikekumppanitietojen tallennus
- Ulkomaan maksujen maksuun valinta
- Ulkomaan maksujen tulostus
- Ulkomaanmaksujen tiliöinti
- Käyttäjähallinta
- Yritystietojen ylläpito
- Kirjanpitoliittymien ylläpito
- Asetustiedot

4.1.4 Tekniikkaa ja järjestelmävaatimuksia

Ohjelmointiympäristönä on Microsoft Visual Studio 2008 ja ohjelmointikielenä on C++.

Raportointi on toteutettu Crystal Reports -raportointiohjelmistolla. Ohjelmiston tietokantajärjestelmä on IBM SolidDB.

Ohjelmisto toimii seuraavissa käyttöjärjestelmissä:

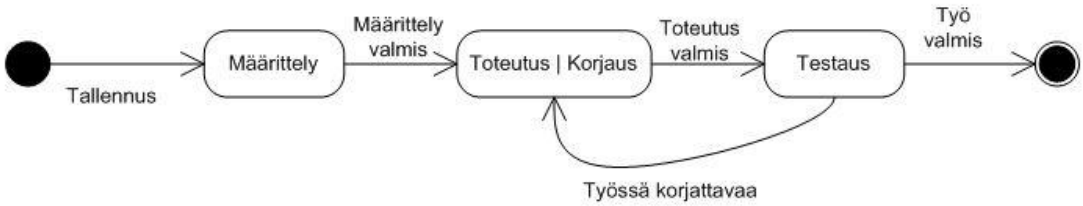
- Windows XP (vähintään SP 3)
- Windows Vista (32- tai 64 bittinen)
- Windows 7 (32- tai 64 bittinen)
- Windows Server 2003 (32- tai 64-bittinen)
- Windows Server 2003 R2 (32- tai 64-bittinen)
- Windows Server 2008 (32- tai 64-bittinen)
- Windows Server 2008 R2

Tietokantapalvelimella voi olla vielä edellä lueteltujen käyttöjärjestelmien lisäksi Linux (Intel) -käyttöjärjestelmä.

4.2 Nykyinen testausprosessi

Toimeksiantajatuotekehitysyksikkö voisi toimia laadukkaan ohjelmistotuotannon mallina. Testattava ohjelmisto on elinkaarimallin ylläpitovaiheessa. Ohjelmiston kehitys etenee syklisesti eli kaksi kertaa vuodessa julkaistaan versio, jossa on uusia ominaisuuksia ja korjauksia. Kuukausittain julkaistaan tarvittaessa päivitysversioita. Korjausversio tehdään välittömästi, jos julkaistussa ohjelmistossa löytyy niin vakava virhe, että se estää ohjelmistoa käyttävän asiakkaan toimintaa. Uuden tuotekehitysjakson työt etenevät pääpiirteittäin kuin vesiputousmallissa: alkaen töiden määrittelyvaiheista, siirtyen toteutusten kautta testaukseen ja valmiiksi töiksi.

Tuotekehitysyksikössä on käytössä projektinhallintaohjelmisto, jossa töille kiinnitetään määrittelijä, toteuttaja ja testaaja. Työn tila muuttuu ja työ siirtyy automaattisesti seuraavan vaiheen toteuttajan työlistalle, kun edellisen vaiheen toteuttaja kuittaa oman vaiheensa valmiiksi (kuvio 7). Kunkin vaiheen dokumentit liitetään työlle projektinhallintaohjelmassa, joka alustaa kaikille dokumenteille valmiin pohjan niitä avattaessa ensimmäistä kertaa. Määrittelijä aloittaa työn tarvittaessa tarvemäärittelyllä ja yleensä aina toiminnallisella määrittelyllä. Toteutusvaiheeseen kuuluvat teknisen suunnitelman kirjoittaminen, ohjelmointi ja ohjelmoijan oma testaus, joka voitaisiin sijoittaa V-mallissa (kuvio 1) moduulitestaukseksi.



KUVIO 7. Työnkulku projektinhallintaohjelmistossa UML-tilakaaviona.

Toteuttaja siirtää valmiin yksittäisen työn testattavaksi. Tämä testausvaihe sijoittuu V-mallissa (kuvio 1) moduulitestauksen ja integrointitestauksen väliin. Siinä on iteraatioita, sillä virheitä löydettyään testaaja palauttaa työn toteuttajalle korjattavaksi. Testaaja kirjoittaa testiraporttia valmiille vakiopohjalle jokaisesta testausiteraatiosta.

Testiraportti laaditaan niin, että sen perusteella kuka tahansa pystyy toistamaan testin. Uuden toiminnon tai piirteen määrittelydokumentin kaikista kohdista kirjataan testauksen lopputulos testiraportille. Virhekorjauksissa testaaja toteaa ensin virheen vanhalla versiolla ja testaa sitten uuden korjatun ohjelmaversion. Testissä käytettyjen aineistojen sijainti linkitetään myös testiraportille. Kuten V-mallin järjestelmätestauksessa, testaaja voi jo tässä vaiheessa verrata tuloksia työn toiminnalliseen määrittelyyn. Toiminnalliseen määrittelyyn on voitu kirjata erikseen testaussuunnitelma, jos työn testaus ei ole kokonaan pääteltävissä toiminnallisesta määrittelystä. Testaaja kuittaa työn projektinhallintaohjelmassa valmiiksi, kun ei enää löydä siitä virheitä.

Kaksi kertaa vuodessa ilmestyvälle versiolle varataan testausaikaa kuukausi. Kuukautta ennen tulee kaikkien kyseiseen versioon mukaan tulevien töiden olla valmiina. Ohjelmistosta tehdään asennuspaketti, jolle tehdään integraatio- ja järjestelmätestaus. Nämä testaustyöt syötetään projektinhallintaohjelmaan ja asetetaan suoraan testaustilaan. Jos näissä testaustöissä havaitaan korjattavia virheitä, testaaja syöttää uudet korjaustyöt projektinhallintaohjelmistoon ja määrittelee niille vakavuusasteet. Osa virheistä saattaa olla niin vakavia, että ne korjataan heti. Tällöin korjaukset tulevat mukaan julkaistavaan versioon. Järjestelmä- ja integraatiotestauksessa voi löytyä myös niin lieviä virheitä, että ne korjataan vasta seuraaviin versioihin. Jos virheen vakavuusaste on epäselvä, testaaja selvittää oikean vakavuusasteen kehitystiimin ja tuotepäällikön kanssa.

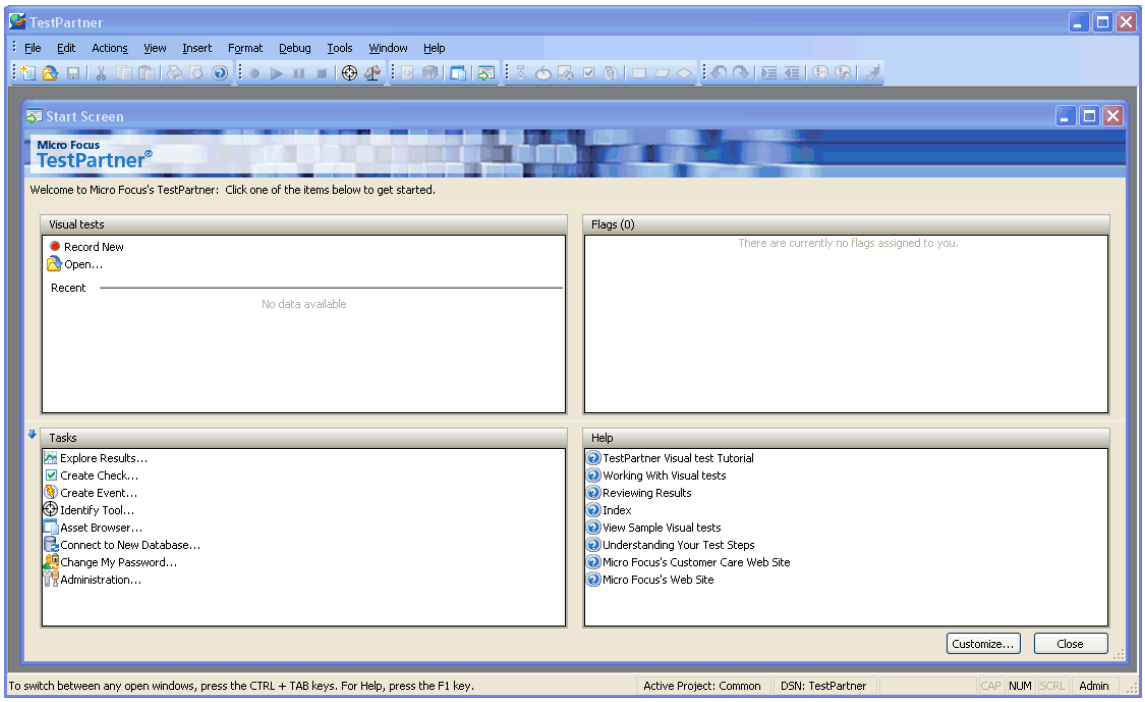
Kuukauden testausjaksossa saatetaan tehdä useampia asennuspaketteja, joissa on mukana järjestelmä- ja integraatiotestauksessa löydettyjen vakavampien virheiden korjauksia. Jokaisen asennuspaketin asennus testataan kaikkiin niihin käyttöjärjestelmiin, joissa ohjelmiston edellytetään toimivan.

Integraatio- ja järjestelmätestauksia varten laaditut testaussuunnitelmat liitetään näiden testaustöiden määrittelyiksi projektinhallintaohjelmistoon. Tarvittaessa päivitetään uudet toiminnot tai piirteet integraatio- ja järjestelmätestaussuunnitelmiin.

4.3 TestPartner

TestPartner on ollut aiemmin osa ohjelmistojätti Compuwaren QACenter-laadunvarmistuspakettia. Tästä kertoo Mikropc.net artikkelissaan vuodelta 2001:”*Älä hutiloi web-sovelluksen käyttöönotossa. Robotti helpottaa testausta”.* Artikkelissa TestPartner ohjelmaan liitetään automaattinen robottitestaus. Lisäksi siinä kerrotaan, että TestPartnerilla voi nauhoittaa minkä tahansa Windows-sovelluksen suoritusta tai ohjelmoida itse testiajoja VBA (Visual Basic for Applications) -kielellä. Ohjelmassa on Visual Basic for Applicationsin -kehitysympäristö testiskriptien kirjoittamiseen. Artikkelissa robotilla tarkoitetaan käyttöliittymän testausta automatisoivaa käyttöliittymätapahtumien nauhoitusta kuten näppäin- ja hiirikomentojen nauhoitusta. Artikkeli esittelee myös Rational Robot -ohjelmaa samankaltaisena testaustyökaluna kuin TestPartner.

Monikansallinen ohjelmisto- ja IT-yritys Micro Focus osti Compuwaren testausohjelmistot ja ASQ (Automated Software Quality) -sovellusratkaisut 2009 kesällä. (Microfocus.com 2009.) Micro Focus mainostaa TestPartnerin (kuvio 8) olevan automatisoitu testaustyökalu, joka on erityisesti suunniteltu Microsoft teknologioita hyödyntävien sovellusten testaukseen.



KUVIO 8. TestPartner-ohjelman aloitusikkuna.

4.3.1 Mikä TestPartner on?

TestPartner on automaattinen testaustyökalu, jolla voi tehostaa Microsoft-, Java-, Web-, SAP-, Oracle- ja monilla muilla teknologioilla kehitettyjen monimutkaisten sovellusten toiminnallista testausta. TestPartner-ohjelmalla voidaan luoda testejä nauhoittamalla käyttäjän istunnot testattavassa sovelluksessa. Nauhoitettuja testejä voidaan kehittää lisäämällä niihin validointia ja logiikkaa. Testejä toistamalla varmistetaan, että testattava sovellus toimii niin kuin sen halutaan toimivan. (TestPartner 6.3 Ohje.)

TestPartner tukee useilla erilaisilla kehitystyökaluilla toteutettujen sovellusten testausta. Näissä kehitystyökaluissa voi olla Java-, Visual Basic- tai Visual C++ -kieliä, selainpohjaisia web-sovelluksia, COM komponentteja, ActiveX kontrolleja ja automaatio-olioita (automated objects). (TestPartner 6.3 Ohje.)

4.3.2 TestPartner-ohjelman järjestelmävaatimuksia

TestPartner versio 6.3 voidaan asentaa koneelle, jossa on vähintään Pentium 4 suoritin tai vastaava. RAM (Read Alter Memory) -muistia pitää olla vähintään 512 MB. Järjestelmälevyllä (System disk) oltavaa levytilaa 300 MB. Oletustietokanta SQL Server 2005 Express vaatii 220 MB lisätilaa. (Microfocus.com 2009.)

TestPartner toimii seuraavissa käyttöjärjestelmissä:

- Microsoft Windows 7 Professional, Enterprise ja Ultimate (32-bit & 64-bit) Editions
- Microsoft Windows Vista Business, Enterprise ja Ultimate SP1, SP2 (32-bit) Editions
- Microsoft Windows XP Professional SP3 (32-bit)
- Microsoft Windows Server 2008 Standard ja Enterprise SP1, SP2 ja R2 (32-bit & 64-bit) Editions
- Microsoft Windows Server 2003 Standard ja Enterprise SP1, SP2 ja R2 (32-bit) Editions

(Microfocus.com 2009.)

TestPartner 6.3 -ohjelmalla on oma tietokanta, johon käyttäjätiedot, testiajot, testiskriptit ja tulokset tallentuvat. Tietokantana voi olla jokin seuraavista:

- Microsoft SQL Server 2008 SP1
- Microsoft SQL Server 2008 Express Edition SP1
- Microsoft SQL Server 2005 SP3
- Microsoft SQL Server 2005 Express Edition SP3
- Microsoft Access 2000
- Oracle 11.1, 10.2, or 10.1 RDBMS tietokanta Microsoft tai Unix/Linux alustoilla

(Microfocus.com 2009.)

4.3.3 Mihin TestPartner-ohjelman toiminta perustuu?

TestPartner-ohjelmisto matkii testaajan näppäimistön painalluksia, hiiren käyttöä, valintoja valikoista ja listoista. TestPartner-ohjelman näppäin- ja hiiritoiminta testattavassa ohjelmistossa on ohjelmallisesti aikaansaatua, kun taas käyttäjä tekee sen laitteiston (hardware) kautta näppäimistöllä ja hiirellä. Visuaalisissa testeissä ja testiskripteissä voidaan käyttää tarkistuksia (checks) testitulosten ja odotettujen tulosten vertailussa. TestPartner pystyy päättelemään mikä toiminto pitää suorittaa seuraavaksi edellisen toiminnon tulosten perusteella. TestPartner-ohjelmalla on myös pääsy PC:n kelloon ja kovalevylle, joten se ymmärtää ajankulun ja pystyy kirjoittamaan ja lukemaan tiedostoja. (TestPartner 6.3 Ohje.)

TestPartner käyttää menetelmää, jota kutsutaan kiinnittymiseksi (attaching) ollessaan vuorovaikutuksessa testattavan ohjelman kanssa. Kiinnittyminen ikkunaan on verrattavissa ikkunan aktivointiin manuaalisesti klikkaamalla, jolloin kohdistin tulee ikkunalle ja ikkuna aktivoituu. (TestPartner 6.3 Ohje.)

Windows-sovelluksessa on erilaisia dialogi-ikkunoita, valikoita, tietokenttiä, listoja, painikkeita ja muita kontrolleja. TestPartner liittää kuhunkin dialogiin ja kontrolliin yksilöivän kiinnittymisnimen (attach name) oliokarttojen (Object Maps) avulla. TestPartner käyttää kiinnittymisnimeä ikkunan paikallistamiseen ja kiinnittyy siihen. (TestPartner 6.3 Ohje.)

Object-termi voidaan tulkita olioksi, vaikka tässä yhteydessä oliolla on suppeampi merkitys kuin yleensä ohjelmistotuotannossa. Yleiskielessä Object voitaisiin myös ajatella objektiksi eli kohteeksi. Tällainen *object map* on käytössä muissakin vastaavissa automaatiotyökaluissa. TestPartner ohje selittää object termin seuraavasti:”*ikkuna tai lomake sovelluksessa, ikkuna tai kontrolli dialogissa”*. Ikkuna koostuu tyypillisesti useista olioista (objects*):*

- Ulommaisesta (parent) ikkunasta, joka sisältää dialogin otsikon, pienennä-, suurenna- ja sulje-painikkeet.
- Sisemmästä (child) ikkunasta, jossa on muut ikkunakontrollit.
- Kehyksistä, jotka rajaavat toisiinsa liittyvät kontrollit samoihin ryhmiin
- Staattisista teksteistä, joilla tunnistetaan ikkunan kontrollien nimet tai toiminnot
- Editoitavista kentistä, listoista, alasvetolistoista ja grafiikoista

Kiinnittymisnimi (Attach name) on ikkunaolion nimi. Ikkunaolioita ovat esimerkiksi dialogi-ikkuna, lista, kuvake tai kenttä. Täydellinen kiinnittymisnimi muodostuu tyypillisesti useista osista kuten sovelluksen nimestä, kontrollin tai kuvakkeen nimestä tai elementin tyypistä (esimerkiksi HTMLCombobox, EditBox).

Esimerkkejä kiinnittymisnimistä:

> Application=IEXPLORE.EXE Caption='NuBid Electronics - Microsoft Internet Explorer' TypeName=Window\TypeName=HTMLCheckBox Name=checkbox
>
> Application=NOTEPAD.EXE Caption='newfile.txt-Notepad' TypeName=Window\TypeName=EditBox Index=1

(TestPartner 6.3 Ohje.)

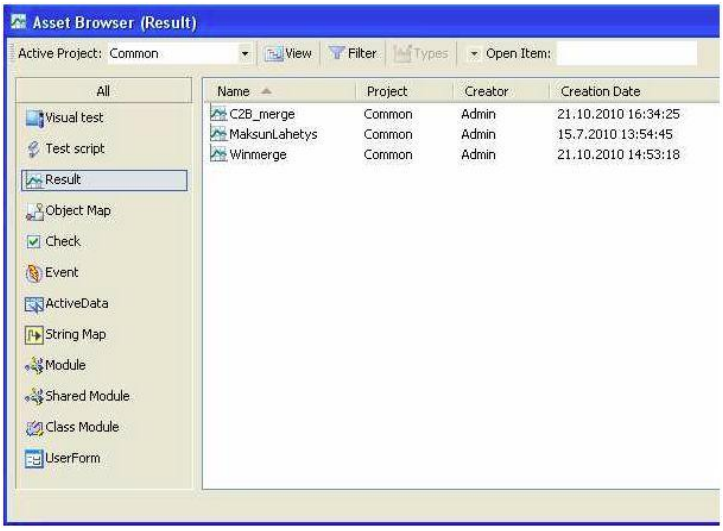
Oliokartta (Object Map) on yksi TestPartner-ohjelman peruselementti (asset), joka sisältää ikkunaan tai kontrolliin liitetyn loogisen nimen (alias). TestPartner käyttää yksilöivää nimeä (raw attach name) kontrollin paikallistamiseen ja siihen kiinnittymiseen. Oliokartassa kontrollin tai ikkunan yksilöivä nimi korvautuu loogisella alias nimellä. Alias nimi edustaa kontrollin tai ikkunan tarkkaa määritystä tietokannassa. Kaikki viittaukset kontrolliin tapahtuvat alias nimellä sen jälkeen, kun alias nimi on ensimmäisen kerran rekisteröity oliokarttaan. (TestPartner 6.3 Ohje.)

4.3.4 Ohjelman esittelyä

Seuraavaksi esittelen lyhyesti TestPartner-ohjelman toimintaa, elementtejä ja käsitteitä. Tiedot pohjautuvat TestPartner 6.3 kokeiluversion englanninkieliseen online-ohjeeseen.

Assets

Asset on testausprojektin perusrakenne, testauselementti, joka määritetään ja tallennetaan TestPartnerin tietokantaan. Asset Browser -toiminnolla voidaan katsella ja hallita näitä perusrakenteita (kuvio 9). Erilaisia asset-tyyppejä ovat visuaaliset testit (visual tests), testiskriptit, moduulit (modules), tarkistukset (checks), tapahtumat (events) ja tulokset (results).
Useimpia perusrakenteita voidaan käyttää uudelleen tai muuttaa ilman, että muutetaan muita niihin viittaavia rakenteita. (TestPartner 6.3 Ohje.)



KUVIO 9. Asset Browser.

TestPartner-projektit

TesPartner-projekti koostuu peruselementeistä (assets), joihin voidaan liittää TestPartner käyttäjien käyttöoikeuksia. Käyttäjä näkee vain ne projektin elementit, joihin hänelle on myönnetty käyttöoikeudet. (TestPartner 6.3 Ohje.)

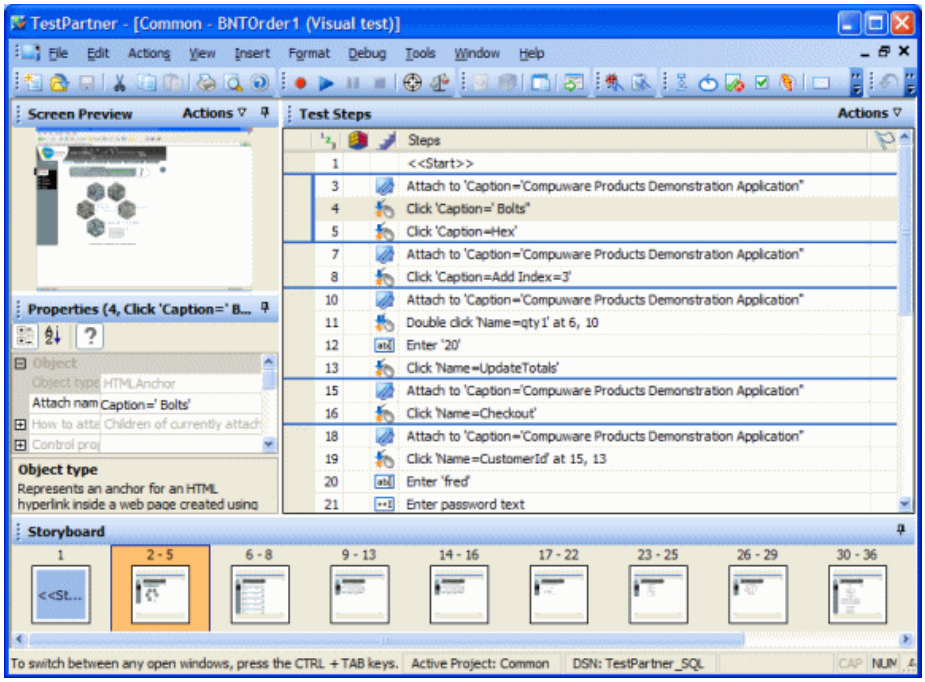
Järjestelmänvalvojan (administrator) oikeuksilla varustettu käyttäjä voi luoda ja lisätä uuden projektin TestPartner-ohjelman tietokantaan. Projektien avulla voidaan tallentaa saman sovelluksen tai version peruselementit (assets) samaan loogiseen yhteyteen. Käyttäjille voidaan myöntää eri projekteihin eri tasoisia käyttöoikeuksia joko täydet oikeudet tai vain lukuoikeudet tai kieltää oikeudet kokonaan jostain yksittäisestä projektista. (TestPartner 6.3 Ohje.)

Visual tests

Visuaalinen testi (Visual test) sisältää komentoja, jotka jäljittelevät ohjelmistontestaustoimenpiteitä kuten menuvalintoja ja tiedon syöttämistä. Visuaalinen testi voidaan nauhoittaa automaattisesti, editoida manuaalisesti tai luoda molempia tapoja yhdistäen. Kun testiskripti sisältää koodirivejä, vastaavasti visuaalinen testi sisältää sarjan testattavaan ohjelmistoon kohdistuvia yksittäisiä toimenpiteitä (steps). (TestPartner 6.3 Ohje.)

Visual navigator

Visual navigator esittää visuaalisen testin graafisesti. Visual navigator koostuu neljästä pääalueesta: *Test steps*, *Screen Preview, Properties* ja *Storyboard* (kuvio 10). *Test Steps* -alueella on lista, joka koostuu visuaalisen testin yksittäisistä toimenpiteistä helposti ymmärrettävällä kielellä kuvattuna. *Screen Preview* näyttää pikakuvan (Snapshot) testattavan ohjelman käyttöliittymästä sillä hetkellä, kun visuaalisen testin yksittäistä toimenpidettä (step) suoritetaan. *Properties* näyttää yksittäisen toimenpiteen ominaisuuksia. *Storyboard* on kuin kuvakäsikirjoitus parhaillaan meneillään olevasta visuaalisesta testistä. *Test Step* -alueella toimenpiteet (steps) voidaan erotella sinisellä viivalla loogisiin ryhmiin, joita *Storyboard*-ikkunassa edustaa yksi kuvake. (TestPartner 6.3 Ohje.)



KUVIO 10.Visual Navigator. (TestPartner 6.3 Ohje.)

Test scripts

TestPartnerin scriptikieli on Microsoft's Visual Basic for Applications (VBA), joka on Microsoftin sovellusohjelmissa makrokielenä käytetty ohjelmointikieli. Testiskriptejä (Test scripts) voidaan käyttää Visual Test -testien kanssa tai yksin monimutkaisempaan testaukseen. Testiskriptejä voidaan luoda nauhoittamalla tai koodata manuaalisesti. Kätevä tapa on luoda nauhoittamalla testiskripti ja koodata siihen lisää manuaalisesti. (TestPartner 6.3 Ohje.)

Test results

Visuaalisen testin tai testiskriptin suorittamisesta syntyviä testituloksia voidaan tarkastella heti toiston jälkeen tai myöhemmin valitsemalla halutut tulokset Asset Browser -toiminnossa. Testituloksiin tallentuvia tietoja ovat esimerkiksi visuaalisen testin tai testiskriptin nimi, ajonumero, päiväys, suoritetun testitoimenpiteen (step) aika ja tila (pass/fail). Testituloksia voidaan tarkastella jälkeenpäin ja analysoida kohtia, joissa testi on voinut epäonnistua. (TestPartner 6.3 Ohje.)

Verifications

Verifikaatio (verification) tarkistaa käyttäjän määrittämät ehdot ja lähettää viestin epäonnistumisesta tai onnistumisesta visuaalisen testin tai testiskriptin tuloksiin. Verifikaation ehdossa vertaillaan kahta arvoa ja palautetaan vertailun tulos. Vertailtavat arvot voivat olla testattavan sovelluksen kontrollien arvoja, visuaalisessa testissä käytetyn muuttujan arvoja tai testitoimenpiteisiin (steps) käytettyä aikaa. (TestPartner 6.3 Ohje.)

Verifikaatio logiikka koostuu seuraavasti:

```
If <condition=TRUE> Then
  <Send pass message to results>
Else
  <Send fail message/flag to results>
End If
```

(TestPartner 6.3 Ohje.)

Test Checks

Testattavan sovelluksen toiminta voidaan tarkistaa käyttämällä TestPartner check-elementtiä (asset). Tarkistus (check) määrittelee testattavan sovelluksen odotetun tilan ja vertaa odotettua tilaa testin suorituksessa syntyneeseen tilaan. Check-elementillä voidaan varmistaa, että testattava sovellus tarkistaa syötteet, suorittaa laskennat oikein, tallentaa tietoa luotettavasti ja raportoi tarkasti. Tarkistuksia (checks) voidaan käyttää uudelleen useimmissa visuaalisissa testeissä tai testiskripteissä. (TestPartner 6.3 Ohje.)

Erilaisia tarkistus tyyppejä voidaan määrittää:

- Bittikarttatarkistukset (bitmap checks) vertaavat todellista bittikarttaa määritettyyn bittikarttaan. Bittikarttatarkistuksia käytetään tarkistamaan työkalurivien, työpöydän ja sellaisten ikkunoiden ulkoasua, joissa on muuta kuin tekstiä.
- Kellotarkistukset (clock checks) mittaavat aikaa, joka järjestelmältä menee prosessin suorittamiseen. Kellotarkistukset auttavat määrittämään miten järjestelmä suoriutuu erilaisten prosessoreiden tai verkkokuorman kanssa.
- Sisältötarkistukset (content checks) tarkistavat taulukoiden ja listakontrollien sisältöjä ikkunalla tai web-sivulla.
- Kenttätarkistuksilla (field checks) voidaan tehdä tietyn tyyppisiä tekstivertailuja, myös päiväyksen ja kelloajan vertailua yksittäisissä kentissä ikkunalla tai tietyllä alueella.
- Ominaisuustarkistukset (property checks) tarkistavat jokaisen kontrollin ominaisuudet dialogissa tai web-sivulla.
- Tekstitarkistukset (text checks) mahdollistavat ikkunan tai yksittäisen alueen tekstin tarkan vertailun määritettyyn tekstiin. Verrattaessa koko näyttöä, sallitusti muuttuvat tiedot kuten päiväykset ja kirjautumistunnukset voidaan ohittaa.
- Käyttäjätarkistus (user check) on VBA-kielen metodi, jota käytetään testiskripteissä. Siinä pysähdytään testiajossa kysymään testaajan kuittausta ja mahdollisesti kommenttia. Testikäyttäjän vastaus tallentuu testiajon tuloksiin (results). (TestPartner 6.3 Ohje.)

Events

Tapahtumat (events) ovat odottamattomia ilmiöitä tai ehtoja, joihin testattavan sovelluksen halutaan reagoivan määritetyllä tavalla. Tapahtuma-avustajalla (Event wizard) luodut tapahtumat tallennetaan tietokantaan peruselementeiksi (assets). TestPartner tukee kahdenlaisia tapahtumia:

- Wait
- Whenever

Wait-tapahtuma käskee TestPartnerin odottamaan jotain ilmiötä tietyn pituisen ajan. Wait on hyödyllinen tilanteissa, joissa ei voida ennakoida vasteajan kestoa. Esimerkiksi verkkopohjaisissa sovelluksissa vaihtelee aika, jona kirjautumiskehote tulee esiin. Testiajossa ennen käyttäjätunnuksen ja salasanan syöttämistä automaattisesti, TestPartner voidaan käskeä odottamaan kirjautumiskehotteen ilmestymistä. Wait-tapahtumaan liittyy kuitenkin aikarajaus, jota ei Whenever-tapahtumassa ole. Wait-tapahtuma sopii paremmin ilmiöihin, jotka voidaan olettaa tapahtuvaksi. Whenever-tapahtuma käskee TestPartner-ohjelman tarkkailemaan tiettyä ilmiötä ja reagoimaan, jos tuo ilmiö esiintyy. (TestPartner 6.3 Ohje.)

ActiveData
Käyttöliittymään tulevat syötteet voidaan tallentaa tiedostoihin, joista ne testitilanteessa saadaan käyttöön. ActiveData ominaisuutta voi käyttää visuaalisissa testeissä siellä missä paikallisia muuttujiakin voi käyttää. Yksittäisen kiinnittymisnimen tai syötteen voi korvata ActiveData tiedostolla visuaalisessa testissä. Testiskripteihin ActiveData-tiedostot saadaan tehtyä lukuisten avustajien (wizards) avulla. Tällöin ei tarvitse suoraan muuttaa visuaalisia testejä tai testiskriptejä, jolloin myös vähemmän tekniikka ymmärtävät testaajat pystyvät tekemään monimutkaisiakin testiskriptejä. (TestPartner 6.3 Ohje.)

4.3.5 TestPartner ja Visual Studio

TestPartner integroituu Visual Studio 2005 Team Systems (VSTS) -ohjelmistokehitysympäristön kanssa. Testiskriptejä voidaan ajaa VSTS-ympäristössä, mutta ei visuaalisia testejä.
TestPartner tukee testiskriptien suorittamista etäkoneella verkon yli. Tällöin on mahdollista tehdä sovelluksen toiminnallista testausta paikallisella koneella ja käyttää tähän testaukseen muilla tietokoneilla sijaitsevia testiskriptejä. (TestPartner 6.3 Ohje.)

TestPartner-ohjelmalla voi testata Visual Studio 2008 sovelluksia TestPartner-ohjelman käyttöliittymässä, mutta tietoa samanlaisesta integraatiosta kuin Visual Studio 2005 Team Systems –ympäristön kanssa ei löydy.

4.4 Automatisoitavien testitapausten valinta

4.4.1 Valintaperusteita

Testitapauksia ja testausta suunniteltaessa pitää muistaa, ettei vain automatisoida olemassa olevia manuaalisia testejä. Tällöin jää käyttämättä automatisoinnin tarjoamat uudet mahdollisuudet, joita ei manuaalisessa testauksessa pystyä hyödyntämään. (Sainio 2009, 121.)

Tiivistäen automaation kohteiksi sopivat usein toistettavat testit:

- yksikkötestit
- savutestit
- suorituskyky- ja kuormitustestit
- kestävyystestaus

Ohjelmoijan tueksi tehtäviä yksikkötestejä ei ole ollut tarvetta enää rakentaa testipetien kanssa testattavan sovelluksen tuotetiimissä, koska tehdyt muutokset on helppo liittää valmiiseen sovellukseen kehitysympäristössä. Tämän jälkeen ohjelman testaaminen onnistuu samassa kehitysympäristössä.

Toimeksiantajatuotekehitysyksikön testattava sovellus on nykyisin lukuisten muutosten kohteena, sillä se on SEPAn ytimessä maksuliikenne sovelluksena. Siinä on runsaasti erilaisia piirteitä, joiden toimivuus edellyttää tässä muutosten tulvassa paljon myös regressiotestausta. Tällöin automaation kohteena voisivat olla savutestit eli yksinkertaiset toimintojen läpikäynnit käyttöliittymässä, kuten ikkunoiden avaukset, tiedon syöttäminen, tiedon tallentaminen ja ikkunan sul-

keminen. Tarkastelen tätä sekä tuotetiimissäni kehitettyjä rutiinitestejä jäljempänä Savutestit ja Rutiinitestaus-otsikon alla.

Nykyisessä testausprosessissa on kuukauden mittainen testausjakso ennen puolivuosittaisen version julkaisua, ehkä tätä jaksoa voitaisiin lyhentää testausautomaation avulla. Tuossa jaksossa suoritetaan myös suorituskykytestejä, jotka samanlaisina toistuvina testeinä ovat suositeltavia testausautomaation kohteita.

Kestävyystestit olisivat paikallaan silloin, kun ohjelmistoon tehdään laajoja useisiin toimintoihin vaikuttavia muutoksia. Tällaisia ovat esimerkiksi tietokantaohjelmiston tai raporttiohjelmiston version vaihtaminen. Nämä vaikuttavat miltei kaikkiin ohjelmiston toimintoihin. Näiden testaukset voitaisiin toteuttaa niin, että samaan ohjelmiston asennukseen kohdistuvaa automaattista iteratiivista savutestausta käynnistetään päivittäin vähintään viikon ajan. Toisaalta tarvetta näin laajojen muutosten testaamiselle on harvoin, joten on syytä arvioida tarkkaan kuluuko siinä enemmän resursseja automaation toteuttamiseen kuin manuaaliseen testaamiseen.

Seuraavissa luvuissa esittelen tarkemmin muutamia testattavalle sovellukselle tyypillisiä testitapauksia, joihin automaatio olisi toivottavaa.

4.4.2 Pankkiin lähetettävät tiedostot

Toimeksiantajatuotekehitysyksikön testattavan ohjelmiston pitää osata kirjoittaa lukuisia määrämuotoisia ascii- ja XML-kielisiä tiedostoja. Tällaisia tiedostoja ovat esimerkiksi kotimaisiin pankkeihin lähetettävät maksutiedostot, jotka kuuluvat ohjelmiston perustoiminnallisuuksiin. SEPAn myötä tulevat uudet XML-kieliset tiedostoformaatit vähitellen käyttöön. Testattavaan ohjelmistoon tulee muutoksia sitä mukaa kuin pankit ilmoittavat, että heillä on valmiudet uuteen tiedostoformaattiin. Vaikka uudet formaatit perustuvat EPC:n (European Payments Council) julkaisuihin, niihin tulee pankkienvälisiä eroavaisuuksia (European Payments Council, 2010). Ohjelmistoon on toteutettu myös SEPA-muunnin, joka muuntaa reskontraohjelmista tulevat perinteiset aineistot XML-kielisiksi.

Pankkiin lähetettävät maksutiedostot pitää testata aina ennen uuden version julkaisua, samoin jo toteutetut tiedostojen käsittelyt. Testauksessa pitää pystyä verifioimaan maksutiedostojen olevan pankin vaatimaa muotoa, muutoin pankki hylkää maksutiedoston. Jos pankkiin lähetettyjen tiedostojen jälkikäsittelyt ohjelmistossa eivät toimi, saattavat samat maksut mennä uudestaan lähetettäväksi. Tällaisia virheitä ei saa jäädä julkaistavaan ohjelmistoversioon. Jälkikäsittelyissä voidaan lähetetty tiedosto poistaa tai nimetä uudelleen ja siihen voi sisältyä palvelimelle siirtoa. Näiden testaus on työlästä, sillä ohjelmistossa on yhteydet useaan eri pankkiin ja näillä on useita eri maksu-aineistoja kuten SEPA-alueen maksut, palkkamaksut ja ulkomaanmaksut.

Tiedostojen tarkastaminen on hidasta ihmisen tekemänä. Ensin eri maksujen tiedot on syötettävä ohjelmistoon ja tallennettava tietokantaan. Kaikkiin pankkeihin ei ole olemassa testiyhteyksiä, jolloin maksutiedostoja ei voi tarkistaa lähettämällä niitä pankkiin. XML-validointi skeemaa vasten ei löydä pankkienvälisiä eroavaisuuksia, sillä C2B-maksuaineiston skeema

pain.001.001.02.xsd on kansainvälisen ISO-organisaation määrittelemä ja on yleisempää tasoa. Tästä nousi mieleeni idea testauksen automatisoinnin kehittämiseksi: voitaisiinko muodostaa itse pankkikohtaiset skeemat ja suorittaa validointia näitä vasten jollain validointityökaluohjelmalla. Ehkä tuon voisi liittää osaksi automaatiotyökalulla tehtävää testausta. Tuossa ideassa on vielä kuitenkin yksi heikkous: XML-validoinnilla skeemaa vasten ei pystytä tarkistamaan tiedostoja täydellisesti. Esimerkiksi XML-kielinen tiedosto saatetaan monesti muotoilla tabulaattorimerkein sisentämällä. Skeema-tarkistus ei löydä tabulaattorimerkkejä. Kuitenkin eräs pankkiryhmittymä hylkää C2B-maksuaineiston, jos se sisältää yhdenkin tabulaattorimerkin.

Pankkiin lähetettävien tiedostojen ja niiden käsittely on hyvä testauksen automatisoinnin kohde, sillä niiden testaaminen toistuu aina uutta versiota tehtäessä. Tämän lisäksi nämä ovat ohjelmiston keskeistä ja kriittistä aluetta, virheet niissä voivat muodostua vakaviksi. Tiedostojen testaamista voidaan toteuttaa lähettämällä niitä pankkien testausympäristöihin tai vertaamalla uuden ohjelmistoversion tuottamia tiedostoja ohjelmiston edellisistä versioista syntyneisiin tiedostoihin, joiden oikeellisuus on jo aikaisemmin todennettu. Validointi pankkikohtaista skeemaa vasten voisi olla myös yksi keino toteuttaa pankkiin lähetettävien tiedostojen tarkistusta.

4.4.3 Savutestit ja rutiinitestaus

Kuukausittain julkaistaviin päivitysversioihin tai korjausversioihin ei ole omaa testausjaksoa vaan ne toteutetaan ja julkaistaan nopeasti. Testausautomaation tultua esille kehitettävänä asiana, toimeksiantajatuotekehitysyksikössä tehtiin ohjeistus rutiinitestauksesta järjestelmätestauksen testaussuunnitelmadokumenttiin. Nopea rutiinitestaus sisältää joukon selkeitä testitapauksia kuten esimerkiksi maksun syöttö, maksun lähetys pankkiin, tiliotteen nouto pankista ja tiliotteen siirto kirjanpitoon. Tämä rutiinitestaus on erinomainen automaationkohde, koska toistuu aina samanlaisena.

Rutiinitestauksen testitapauksina on ohjelmiston keskeisimpiä toimintoja. Tämän lisäksi voitaisiin automatisoida toimintojen läpikäynnit käyttöliittymässä, kuten ikkunoiden avaukset, tiedon syöttäminen, tiedon tallentaminen ja ikkunan sulkeminen. Nämä voitaisiin toteuttaa niin, että ohjelmiston lähes kaikki toiminnot tulisi läpikäytyä. Asennuspaketin tekijä voisi käynnistää testit heti paketin valmistuttua ja testit olisivat käytettävissä jokaista julkaistavaa versiota varten. Tämä säästäisi muiden testaajien aikaa, sillä asennuspaketti asetettaisiin testattavaksi vasta sen läpäistyä nämä savutestit.

4.4.4 Suorituskyky- ja kuormitustestaus

Testattavan sovelluksen suorituskykytestauksia on tehty tietovarastojen suurilla tietomäärillä. Testausta varten on luotu tietokanta, jossa on suuri määrä esimerkiksi tiliotteita ja maksuja. Automatisoinnin ansiosta voitaisiin testata ohjelmiston työasema-palvelinversiota helpommin useilla yhtäaikaisilla käyttäjillä.

4.4.5 Asennustestit

Toimeksiantajatuotekehitysyksikön toinen hyvä testausautomaation kohdealue teoriassa voisi olla ohjelmiston asennuksen testaaminen eri käyttöjärjestelmiin. Nämä testit suoritetaan aina

samanlaisina, jokaisen ohjelmistoversion asennuspakettia tehtäessä. Lisäksi asennusohjelmaan tulee harvoin muutoksia. Asennustestit eri käyttöjärjestelmiin suoritetaan virtuaali PC:llä.

Jokaiseen testattavaan käyttöjärjestelmään testataan ohjelmiston uusi asennus ja ohjelmiston päivitys vanhemmasta versiosta uuteen. Uusi asennus tarkoittaa ohjelmistoa käyttöönotettaessa tehtävää ensimmäistä asennusta, kun ohjelmistoa ei ole ollut aiemmin asennettu työasemalle tai palvelimelle. Kun testataan versiopäivitystä, täytyy ohjelman vanha versio aina ensin asentaa virtuaali PC:lle. Yleensä vanhana versiona käytetään asennustesteissä aina samaa versiota, jonka täytyy olla riittävän vanha testaamaan kattavasti tietokannan päivittymistä uuteen versioon. Asennustestaus tehdään myös erikseen ohjelmiston palvelinversioille ja yksittäiskäyttöversioille.

TestPartneria muissa yrityksen yksiköissä käyttävien kollegoiden haastatteluissa ilmeni, että hekään eivät olleet automatisoineet testausta virtuaali PC:llä. TestPartner pitäisi myös asentaa virtuaalikoneelle, sillä nauhoituksessa kontrollien tunnistaminen etäyhteyden yli ei onnistu. Ajatus tuollaisten asennustestien automatisoinnista ei ollut haastateltavien mielestä mahdotonta, mutta saattaisi olla työlästä toteuttaa. Jos vielä toteutuksesta tulee monimutkainen, se on hankala ylläpitää.

4.5 Testien suunnittelu

Automatisoidut testit ja testausympäristö pitää suunnitella niin, että testit ovat toistettavissa.
On määritettävä mitä halutaan testata, miten testitapaus päättyy ja mikä on odotettu testin tulos. Suunnitellaan testin alkukohta; onko testattava sovellus testin alkaessa jo käynnissä vai pitääkö se käynnistää ja suorittaa kirjautuminen. Määritetään tarkkaan toimenpiteet ja vaiheet sekä testissä syötettävä tieto.

Toimeksiantajan yksikkö kuuluu yritykseen, jonka toisessa testien automatisointia jo vuosia hyödyntäneessä yksikössä testaajat suunnittelevat testitapaukset testaamalla ne ensin manuaalisesti. Automatisoitujen testien toteuttaja ohjelmoi testit testaajien suunnitelmien pohjalta.

Verrattaessa manuaaliseen testiin, automatisoidun testin toteutus ei saisi olla hitaampaa. Automaattisia testejä pitäisi pystyä muuttamaan helposti ja nopeasti. Hyvin suunniteltua testiautomaatiota ei tarvitse kokonaan muuttaa, kun testattava sovellus muuttuu. Testien tulee olla niin joustavia, etteivät ne kaikki heti pysähdy ja raportoi virhettä aikaviiveistä tai uudesta dialogista, joka viime hetkellä lisätään ohjelman toimintalogiikkaan. Tähän tavoitteeseen pääsyä auttaa testien muodostaminen modulaarisesti ja tietoa uudelleen käyttämällä. Myös testissä käytettävät syötteet voidaan pitää eri tiedostoissa kuin testiskriptit. Esimerkiksi testattavaan ohjelmiston käynnistämisestä ja siihen kirjautumisesta voidaan tehdä oma moduuli, joka liitetään muihin testeihin. (TestPartner 6.3 Ohje.)

TestPartner-ohjelman ohjeessa annetaan esimerkki skriptejä kokoavasta testiskriptistä (driver test script), jolla voidaan ajaa peräkkäin erilliset visuaaliset testit ja testiskriptit:

```
Sub Main()
This is an example of a driver test script that runs separate test scripts
 to test the online ordering system.
Run "OnlineLogOn_Script"
Run "CreateProfile_Script"
Run "PlaceOrders_Script"
Run "OnlineLogOff_Script"
End Sub
```

(TestPartner 6.3 Ohje.)

Kokoava testiskripti käynnistää ja tarkkailee muita testiskriptejä, jotka testaavat kohdesovellusta. Kokoava testiskripti ei varsinaisesti suorita testausta. (TestPartner 6.3 Ohje.)

Suunnittelin erillisten esimerkkitestien kaavat testattavaa sovellusta varten (kohdat X1 – X9 saavat tarkat arvot lopullisia testejä suunniteltaessa):

- Testi 1: Testattavan sovelluksen käynnistys ja kirjautuminen
    - avataan resurssienhallinta, avataan X1 hakemistopolku, klikataan testattava.exe (tämä käynnistää testattavan ohjelman).
    - syötetään salasana X2 (käyttäjätunnus on oletuksena).
    - klikataan OK-painiketta.
- Testi 2: Maksun tallennus sovelluksessa
    - Avataan valikosta Maksun syöttö -ikkuna (Testi 2:n alussa sovellukseen on jo kirjauduttu ja sen pääikkuna on auki)
    - Valitaan maksulle saaja X3 selausikkunasta
    - Valitaan maksulle tositelaji X4
    - Valitaan maksutapa X5
    - Valitaan maksun tilaksi X6
    - Tiliöidään arvolla X7
- Testi 3:Maksun tai maksujen lähetys pankkiin
    - Avataan Tiedonsiirto-ikkuna
    - Valitaan pankki X8
    - Valitaan lähetettävä aineisto X9 (voi olla Testi 2. aikana tallennettu maksu tai muusta järjestelmästä tullut maksuaineisto)
    - Klikataan Yhteys-painiketta
    - Hyväksytään aineiston lähetys OK-painikkeella
    - Lähetyksen jälkeen klikataan Sulje-painiketta, jos ohjelma ilmoittaa Tiedonsiirto OK.
- Testi 4:Maksutiedoston tarkistus
    - Verrataan Testi 3:n suorittamisesta syntynyttä maksuaineistoa odotetun malliseen maksuaineistoon.

4.6 Testaus TestPartner työkalulla

4.6.1 TestPartner-ohjelman asennuksesta

TestPartneriin tutustuminen käytännössä alkoi 30 päivän kokeiluversion asentamisella. Lisenssi asennettiin myöhemmin erillisen ohjelman avulla. TestPartner 6.3 Asennuksessa ja ohjelman käynnistysyrityksestä tuli seuraava virheilmoitus:"*TestPartner is not compatible with the currently installed version of VBA(6.5.1053)*". Sovellusta jo käyttävien kollegoiden haastattelujen avulla selvisi, että Micro Focuksen sivuilta löytyvä korjauspäivitys ratkaisee ongelman. Koneella oli jo ennestään asennettuna Microsoft Office 2007 -paketti, jonka ohjelmistoissa myös käytetään VBA-kieltä makrojen ohjelmointiin. Tässä ensimmäisessä asennuksessa ilmeni VBA-makrokielen yhteensopivuusongelma.

TestPartner määrittää automaattisesti ODBC (Open Database Connectivity) -tietolähteet tietokoneelle, kun se asennetaan käyttämään SQL Server 2005 Express tai Access tietokantaa. Jos TestPartnerin halutaan käyttävän muita tietokantoja, on luotava ja määritettävä ODBC-tietolähde ja ODBC-tietokanta manuaalisesti tietokoneelle.

4.6.2 Kirjautuminen ja tietokantayhteyden valinta

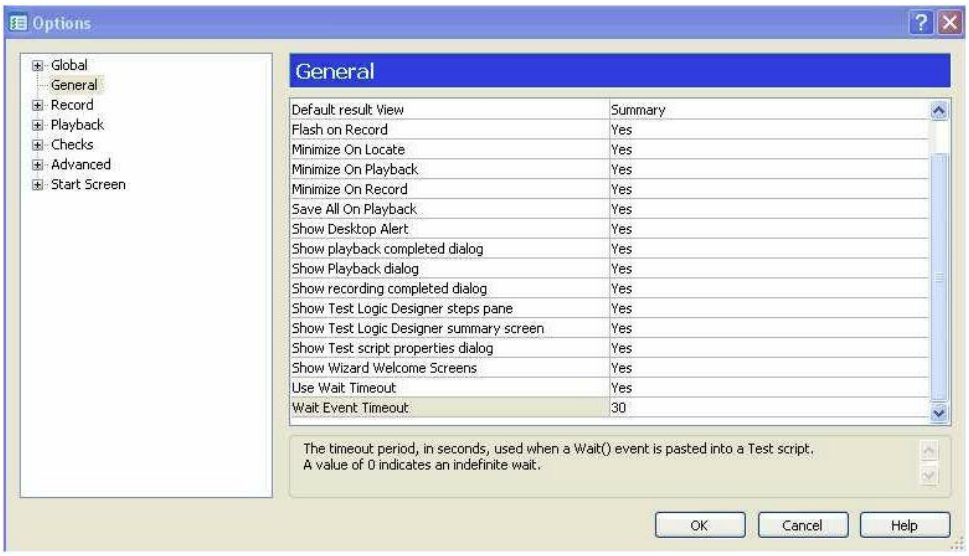
Kirjautumiseen tarvitaan käyttäjätunnus ja salasana. Näiden jälkeen valitaan tietokantayhteys alasvetolistasta, johon tulevat näkyviin TestPartnerille asennetut ODBC (Open Database Connectivity) -tietolähteiden nimet (kuvio 11). Kirjautumisikkunalla voidaan siten vaihtaa TestPartnerin käyttämä tietokanta. Jokaisen testattavan tuotteen eri versioiden testejä varten voidaan perustaa oma tietokanta.

KUVIO 11. TestPartner-ohjelman kirjautumisikkuna**.**

4.6.3 Asetuksia

TestPartner sisältää joukon asetustietoja, joiden avulla sen toimintaa voi säädellä. Klikkaamalla työkaluvalikosta Tools>Options saadaan esiin eri asetusluokat (kuvio 12). Kohdasta *General* löytyy esimerkiksi tärkeä asetus *Wait Event Timeout,* joka ilmaisee sekunteina ajan joksi Wait-komento pysäyttää toiston ennen sen jatkumista. Arvolla ”0” määritellään ääretön odotusaika.

Kohdan *Playback_Results, Save all information,* YES-valinta kasvattaa turhaan TestPartnerin tietokantaa. NO-valinta tallentaa tietokantaan tuloksista yhteenvedot ja virheilmoitukset, jotka ovat täysin riittävää informaatiota testeistä.



KUVIO 12. TestPartnerin asetustietoja.

4.6.4 Visuaalisen testin nauhoitus ja toistaminen

Testattava ohjelmistotuote on jo elinkaaren ylläpitovaiheessa eikä käyttöliittymään tule enää suurempia muutoksia. Testauksen automatisointi nauhoittamalla käyttöliittymätoimenpiteitä ei ole tavoite vaan enemmänkin keino saada aikaan vertailtavat tiedostot. Kuitenkin samalla tulee testattua myös keskeisiä toimintoja kuten Maksun syöttö ja tallennus tietokantaan. Nauhoituksella saataisiin myös tarvittaessa nopeasti tehtyä kaikkien käyttöliittymän toimintojen läpikäynti.

Nauhoitettua visuaalista testiä voidaan muokata jälkeenpäin ja lisätä siihen ehtoja ja toistosilmukoita sekä viesti-ikkunoita. Ensimmäiset visuaalisten testien nauhoituskokeilut tein testattavan sovelluksen maksun syötössä. Nauhoitus tapahtuu seuraavasti:

TestPartner-ohjelman aloitusikkunan Visual tests -paneelissa tai työkalurivillä klikataan punaista pyöreää Record New -kuvaketta (kuvio 13).

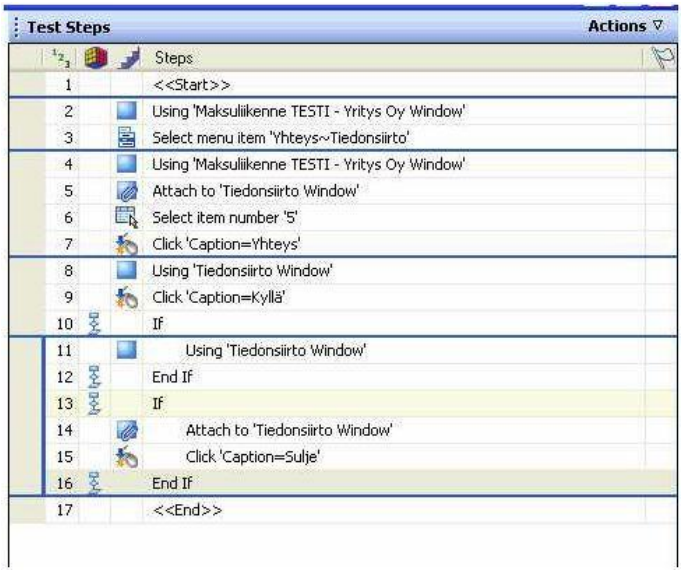
KUVIO 13. Record New -kuvake.

TestPartner-ohjelman ikkuna pienenee alapalkkiin ja näkyviin ilmestyy ponnahdusikkuna osoittamaan, että nauhoitus on alkanut (kuvio 14). Tällöin TestPartner nauhoittaa kaiken mitä käyttäjä tekee työpöydällä lukuunottamatta vuorovaikusta TestPartnerin kanssa. TestPartner osaa myös toistaa, jos käyttäjä avaa resurssien hallinnan, hakee koneen Program Files -hakemistosta ohjelman asennuskansion ja käynnistää ohjelman tai avaa tiedoston.

KUVIO 14. Ponnahdusikkuna, joka ilmoittaa nauhoituksen alkaneen.

Uuden maksun tallennus onnistui toistamalla tuo ensimmäinen nauhoitus. TestPartner toimii nopeammin kuin ihmiskäyttäjä. Toiston seuraaminen oli kuin seuraisi nopeutettua videota. Toiston lakattua ohjelmiston tietokantaan oli ilmestynyt uusi maksu.

Jatkoin nauhoituskokeiluja testattavan ohjelmiston tiedonsiirto toiminnolla, jossa lähetetään esimerkiksi maksuja pankkiin. Tässä toisto ei enää onnistunutkaan, koska toiston aikana lähetys kesti kauemmin kuin nauhoituksessa. TestPartner eteni jo seuraavaan toimenpiteeseen (step), kun testattava ohjelma odotti vielä kuittausta pankista. Tällaiseen toimintoon voidaan luoda odotusaikaa Tapahtuma-avustajalla (Event Wizard) tai luoda testiskriptillä tauko toistotoimintoon tai kääriä odotusajan jälkeen tulevat testitoimenpiteet (steps) if-ehtojen sisään (kuvio 15).



KUVIO 15. If- ehdot visuaalisessa testissä.

If-ehtojen lisääminen onnistui visuaaliseen testiin Maksun lähetys -testissä valitsemalla ensin testitoimenpiteet ja hakemalla hiiren oikean painikkeen valikosta if-ehto. Tämän jälkeen testin toistaminen onnistui, vaikka lähetystilanteessa tuli viivettä ennen kuin pankin kuittaus lähetyksestä saapui ja testattava ohjelma antoi käyttäjälle tiedon lähetyksestä.

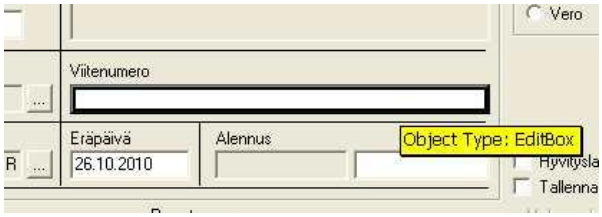
4.6.5 Testikriptien tekeminen

Testiskriptejä voidaan käyttää visuaalisten testien kanssa tai yksin monimutkaisempaan ohjelmistotestaukseen. Myös testiskriptit matkivat käyttäjän toimenpiteitä testattavassa sovelluksessa. Testiskriptejä voidaan luoda nauhoittamalla, koodata manuaalisesti tai luoda molempia tapoja yhdistäen. (TestPartner 6.3 Ohje.)

Yrityksen toisessa testiautomaatiota käyttävässä yksikössä on yksi kokopäivätoiminen testiautomaation tekninen toteuttaja, joka suosii skriptien koodaamista ennemmin kuin GUI-nauhoitusta. Hänen mielestään skriptit ovat luotettavampia monimutkaisten testien tekemiseen eikä nauhoittamalla saa aikaan yleiskäyttöistä koodia. Kahden vuoden aikana hän on koodannut noin 250 testiskriptiä. Yhden testitapauksen koodaamiseen kuluu keskimäärin yksi työpäivä.

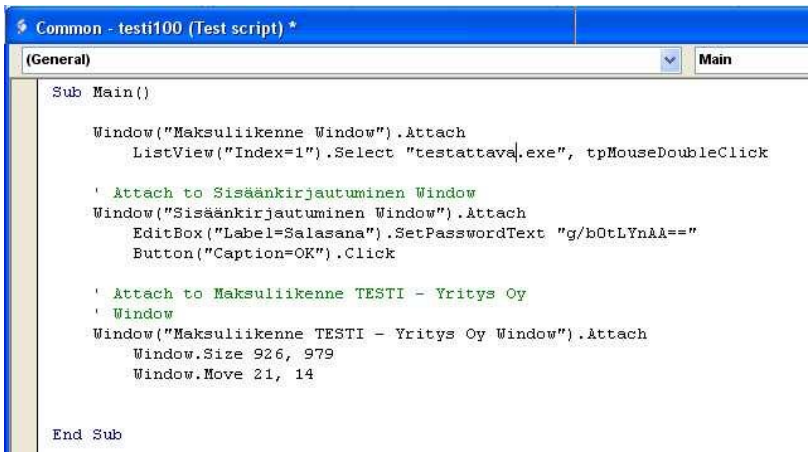
Hän hyödyntää TestPartnerin Identify-työkalua, jolla hän kaappaa testattavasta sovelluksesta kontrollien pitkät kiinnittymisnimet VBA-koodiin. Identify-toiminnon voi käynnistää monesta eri paikasta; esimerkiksi klikkaamalla sen pyöreää kuvaketta työkalurivillä (kuvio 16). Sillä voi kaapata kiinnittymisnimen suoraan testattavasta sovelluksesta viemällä hiiren kontrollin päälle (kuvio 17). Tunnistaminen onnistuu myös visuaalisen testin testitoimenpiteen (step) ominaisuuksista (Properties).

KUVIO 16. Identify-kuvake.



KUVIO 17. Identify-työkalu tunnistaa tiedon syöttökenttää testattavassa sovelluksessa.

Nauhoittamalla saa aikaan testiskriptin, kun ensin avaa uuden testiskriptin pohjan ja asettaa kohdistimen oikeaan paikkaan. Tämän jälkeen käynnistetään nauhoitus. TestPartner generoi skriptin siihen kohtaan missä kohdistin on, kun nauhoitus alkaa. Seuraavassa kuvassa on skripti (kuvio 18), joka syntyi nauhoittamalla testattavan sovelluksen käynnistämistä klikkaamalla ohjelman exe-tiedostoa hiirellä ja ohjelmaan kirjautumista. Ajamalla tätä skriptiä saadaan testattava sovellus käynnistymään ja TestPartner kirjautumaan siihen. Kuvaan on muutettu oikean ohjelmatiedoston nimi testattava.exe nimiseksi.



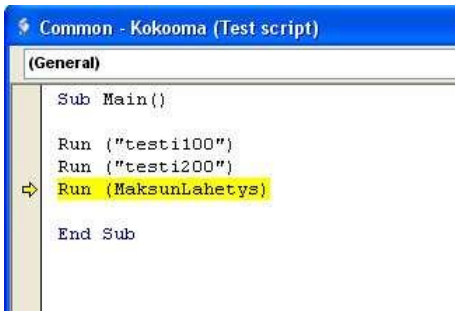
KUVIO 18. Testattavan sovelluksen käynnistys ja siihen kirjautuminen testiskriptinä.

Nauhoitin skriptit kohdan 4.5 testisuunnitelman mukaisesti:

- Testi 1: Testattavan sovelluksen käynnistys ja kirjautuminen
- Testi 2: Maksun tallennus sovelluksessa
- Testi 3: Maksun tai maksujen lähetys pankkiin

Näiden jälkeen kirjoitin kokoomaskriptin, joka ajaa peräkkäin nauhoittamani erilliset skriptit. Tämän jälkeen testasin testin eli ajoin kokoavan testiskriptin.

Ohjelman käynnistys ja siihen kirjautuminen onnistuivat toistossa, samoin maksun tallennus. Kun testiajo pysähtyi virheeseen, TestPartner näytti selkeästi virheellisen kohdan kokoavassa testiskriptissä (kuvio 19).



KUVIO 19. Virhe skriptin suorituksessa.

4.6.6 Testien ajo komentokehotteesta

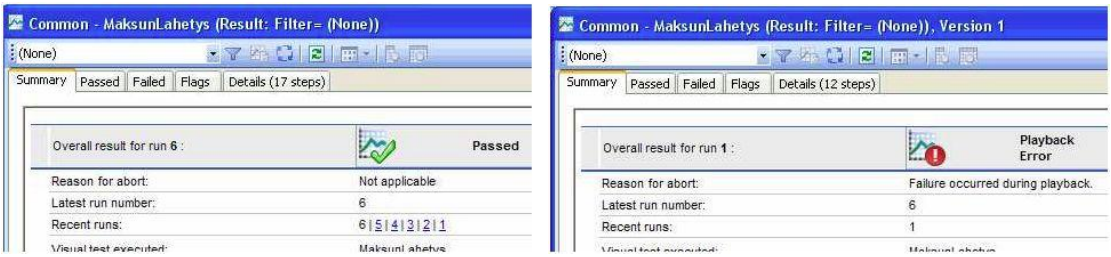
TestPartner komentoriviohjelmalla (TP.EXE) voidaan TestPartner käynnistää automaattisessa moodissa ja käynnistää näin visuaalinen testi tai testiskripti komentokehotteesta. Tämä komento voidaan myös sijoittaa komentojonotiedostoon automaattista testien toistamista varten. Komentokehotteesta käynnistetty testin toisto voi pysähtyä antamatta ilmoitusta, jos virhekäsittelyä ei ole rakennettu testiin. (TestPartner 6.3 Ohje.) Komennon syntaksi on seuraava:

tp -u <*Username*> -p <*Password*> -d <*ODBC DSN*> -r <*Project name*> -s <*list of test script names delimited by space*> (TestPartner 6.3 Ohje).

Yrityksen X testiautomaatiota eniten hyödyntävässä yksikössä ajetaan öisin web-sovelluksen kokoamisajot (build). Näiden jälkeen käynnistyvät automaattiset TestPartner-testiajot. Aamulla voidaan tarkastaa testiajojen status työtilaan korkealle sijoitetun tietokoneen näytöltä. Iso vihreä kuvake osoittaa testiajojen menneen virheittä läpi. Punainen kuvake tarkoittaa, että virheitä on löytynyt, jolloin testaajat aloittavat testitulosten läpikäynnit.

4.6.7 Testitulosten analysointia

Kun visuaalinen testi tai testiskripti on ajettu, tuloksia voidaan tarkastella tulos (Result) –ikkunalta (kuvio 20). Visuaalisen testin tuloksia voidaan tarkastella myös kohdassa 4.3.1 esitellyllä Visual Navigator-ikkunalla, jossa voidaan nähdä miltä testattavan sovelluksen ikkunat näyttivät testiajon aikana. TestParner tarjoaa monipuolisia testituloksia visuaalisten testien ja testiskriptien ajamisesta. Tulokset ovat myös muokattavissa tarpeiden mukaisiksi asetustiedoissa kohdassa *Playback Results*. (TestPartner 6.3 Ohje.)



KUVIO 20. Kuvassa vasemmalla on Result-ikkuna onnistuneen testin jälkeen. Oikean puoleisessa kuvassa testi on keskeytynyt virheeseen.

Yhteenveto (Summary) -välilehti aukeaa oletuksena, kun Result-ikkuna avataan (kuvio 20). *Passed*-välilehti näyttää testiajossa läpimenneet tarkistukset (checks) ja verifikaatiot (Verifications). *Failed*-välilehti näyttää epäonnistuneet tarkistukset (checks) ja verifikaatiot (Verifications). *Flags*-välilehti näyttää kaikki lipulla (flag) merkityt testitoimenpiteet (steps). Lipuilla voidaan antaa testeihin muistutuksia käyttäjille. Nämä näkyvät esimerkiksi visuaalisen testin Test Steps –paneelissa omassa sarakkeessaan lipun kuvakkeena. Flags-ilmoituksia voidaan asettaa suoraan testituloksiin tai verifikaatio logiikan avulla testiajon aikana. ikkunan *Details*-välilehdellä voidaan tarkastella jokaista visuaalisen testin testitoimenpidettä (steps) tai testiskriptin koodiriviä. (TestPartner 6.3 Ohje.)

4.6.8 Tiedostovertailu

Edellä kohdassa Testitapausten valinta ja sen alakohdassa Pankkiin lähetettävät tiedostot kerroin toimeksiantajayksikön tarpeesta tiedostovertailuun. Tuohon on olemassa erilaisia enemmän tai vähemmän automatisoituja ratkaisuja.

TestPartnerilla tiedostovertailun toteutus onnistuu ohjelmoimalla testiskripti, joka vertaa tiedostoja rivi riviltä koodisilmukassa. Toisen yksikön testiautomaation toteuttaja oli tehnyt tämän tyyppisen skriptin, jolla hän vertasi Excel-taulukoita riveittäin ja sarakkeittain. XML-tiedostojen vertailua varten tuota skriptiä täytyy vähän muuttaa, siinä ei tarvita sarakkeittain vertailua. Tämä skripti voidaan ajaa kohdassa 4.5 suunnittelemieni esimerkkitestien viimeisenä testinä (Testi 4:Maksutiedoston tarkistus).
Kokeilin myös nauhoittamalla TestPartnerilla tiedostonvertailija ohjelman toimintaa. Käytin tässä Winmerge-ohjelmaa. Tuon nauhoituksen toistaminen onnistui. Testin lopussa näkyi Winmergen tiedostojen vertailunäkymä, johon voidaan määrittää tekstitarkistuksia (text checks). Vaikka vertailtavat tiedostot voivat sijaita eri hakemistoissa, niiden täytyy aina löytyä testiajoja toistettaessa uudestaan samasta hakemistosta kuin nauhoitusta tehtäessä. Myöskään tiedostojen nimet eivät saa muuttua uudella testauskierroksella. Tässä sekä testiskriptillä tehdyssä vertailussa tulokset ovat tarkasteltavissa TestPartnerin testituloksissa samoissa yhteyksissä kuin ne testit, joissa vertailtavat tiedostot ovat syntyneet.

Tiedostovertailutyökaluja on mahdollista käynnistää myös komentokehotteesta, jolloin vertailun tulokset voidaan tallentaa omiin tiedostoihinsa. Tällaista vertailua voidaan ketjuttaa komentojonotiedostoon.

4.7 TestPartner työkalun soveltuvuuden arviointia

Kohdassa 4.1 esitellyn Windows-sovelluksen testauksen automatisointi onnistuu teknisesti TestPartnerilla. Tämä vaatii kuitenkin yhden henkilöresurssin kouluttamista ja kiinnittämistä testiautomaation toteutukseen. TestPartneria muualla yrityksessä käyttävät yksiköt kehittävät myös testattavia sovelluksiaan Visual Basic -kielellä, joten VBA-makrokieli on heille tuttua. Toimeksiantajayksikössä Windows-sovelluksia kehitetään C++ -kielellä ja VBA-makrokieltä tunnetaan hyvin vähän. Toisaalta opastusta, käytännön kokemuksia ja jopa mallitestiskriptejä on saatavilla.

Jokaisesta tuotetiimistä osallistui yksi henkilö järjestämääni TestPartner-ohjelman tutustumistilaisuuteen. Tilaisuudessa TestPartneria esitteli yrityksen toisesta yksiköstä kokopäivätoiminen testiautomaation tekninen toteuttaja. Osallistujat kertoivat tilaisuuden jälkeen saaneensa hyvän käsityksen siitä mitä TestPartnerilla voi tehdä. Itsekin sain tilaisuudessa uuden tiedon lisäksi vahvistusta aikaisempiin havaintoihini. Todettiin testiautomaation toteuttamisen TestPartnerilla olevan toimeksiantajan yksikössä resursoinnin kannalta haasteellista. Yksi tuotetiimeistä kehittää web-sovellusta .NET-tekniikalla ja he ovat jo aloittaneet testauksen tehostamista Microsoft Visual Studio 2008 -työkalulla. Heidän ei kannata enää ottaa käyttöön erillistä testausautomaatiotyökalua vaan siirtyä Visual Studio 2010 -kehitysympäristöön ja saada sitä kautta mahdollisuudet kehittyneempään testiautomaatioon.

Yrityksen muissa web-sovelluksia kehittävissä yksiköissä käytetään jo Visual Studio 2010 -kehitysympäristöä. Tällöin saavutetaan etua sillä, että kehitysympäristö ja laaja testausautomaatiosovellus ovat samassa työkalussa. Esimerkiksi resursoinnin kannalta tämä voi tuoda etuja, joita ei tämän opinnäytetyön puitteissa ole päästy tutkimaan.

Toimeksiantajayksikössä on nyt kehitysympäristönä Visual Studio 2008. Vaikka TestPartner voi olla käyttökelpoinen Windows-sovellusten testauksessa, toimeksiantajayksikön kannattaa tutkia tarkemmin myös Visual Studio 2010:n tarjoamat mahdollisuudet testausautomaatioon.

5 Yhteenvetoa ja päätelmiä

Tämän opinnäytetyön yhtenä tavoitteena oli selvittää onnistuuko kohdassa 4.1 esitellyn sovelluksen testauksen automatisointi yrityksen muissa yksiköissä jo käytössä olevalla TestPartner automaatiotestaustyökalulla. Toinen tavoite oli nostaa tietämystasoa testausautomaatiosta toimeksiantajatuotekehitysyksikössä. Opinnäytetyö on myös kirjoitettu siten, että sitä ymmärtäisivät myös vähemmän ohjelmistotekniikoita tuntevat henkilöt. Joissakin kohdin tämän viimeisen tavoitteen saavuttaminen on ollut haasteellista, sillä tämän työn puitteissa ei ole ollut tarkoituksenmukaista selittää kaikkia käsitteitä tyhjentävästi.

Tähän opinnäytetyöhön kuului TestPartner-ohjelman tutustumistilaisuuden järjestäminen. TestPartneria esitteli yrityksen toisesta yksiköstä kokopäivätoiminen testiautomaation tekninen toteuttaja. Osallistujat saivat tilaisuudessa hyvän käsityksen siitä mitä TestPartnerilla voi tehdä.

Kauttaaltaan kaikessa testausautomaatiosta kertovassa kirjallisessa materiaalissa kerrotaan, että testauksen automatisointiin kohdistuu helposti epärealistisia odotuksia. Luullaan, että automaatio voidaan toteuttaa pysyväksi pienellä työmäärällä. Todellisuudessa testauksen automatisointi on verrattavissa tuotekehitysprojektiin. Testiautomaatiota täytyy jatkuvasti ylläpitää ja päivittää testattavan ohjelmiston muuttuessa.

Yksinkertaisempia testejä pystyvät tekemään nauhoittamalla testaajat, joilla ei ole ohjelmointitaitoja. Testiskriptien avulla voidaan kuitenkin luoda monimutkaisempia ja yleiskäyttöisempiä testejä. Huonosti toteutettu testausautomaatio saattaa pikemminkin kuluttaa resursseja kuin vapauttaa niitä. Tosiasia kuitenkin on, että testiautomaation toteutus alussa vie paljon resursseja. Vasta myöhemmin sillä saavutetaan säästöjä.

Ohjelmistokehityksen testausprosessin pitää olla kunnossa ja toimiva, ennen kuin voidaan onnistuneesti toteuttaa automatisointiprojektia. Automatisoinnin toteuttajien pitää tuntea hyvin automatisoinnin työkalut sekä sovellus, jonka testausta automatisoidaan. Toimeksiantajayksikön testausprosessi on toimiva ja voisi toimia jopa malliesimerkkinä hyvästä testausprosessista.

Päätelmäni on että, kohdassa 4.1 esitellyn sovelluksen testauksen automatisointi onnistuu TestPartnerilla. Kuitenkin resursoinnin kannalta tämä on haasteellista. Visual Studio 2010 tarjoamat mahdollisuudet testiautomaatioon kannattaa myös tutkia. Kehitysympäristöön integroitu testaussovellus voi tuoda uusia etuja mukanaan.

Automatisoinnin kohteeksi sopivat usein toistettavat testit. Tällaisia ovat savutestit, asennustestit, suorituskyky- ja kuormitustestit. Tässä työssä esitellyn testattavan ohjelmiston erikoispiirre on pankkiin lähetettävät tiedostot. Näiden testauksen automatisoinnissa voidaan käyttää apuna tiedostovertailua ja XML-validointia skeemaa vasten.

Lähteet

Kirjallisuus:

Haikala, I. & Märijärvi, J. 2006. Ohjelmistotuotanto. Helsinki: Talentum Media Oy

Patton, R. 2006. Software Testing. United States of America:Paul Boger.

Pohjolainen, P. 2003. Ohjelmiston testauksen automatisointi. Pro gradu -tutkielma. Kuopion yliopisto. Tietojenkäsittelytieteen laitos. Kuopio.

Sainio, L. 2009. Ohjelmistotestauksen menetelmät ja työvälineet. Opinnäytetyön lukumateriaali. Saimaan ammattikorkeakoulu. Tietotekniikan koulutusohjelma. Lappeenranta.

Digitaaliset ja verkkolähteet:

European Payments Council. [www-sivu]. [Viitattu 21.11.2010]. Saatavissa: http://www.europeanpaymentscouncil.eu/

Finanssialan keskusliito. [www-sivu]. [Viitattu 3.10.2010]. Saatavissa: http://www.fkl.fi/

Kauppalehti.[www-sivu]. [Viitattu 23.8.2010]. Saatavissa: http://www.kauppalehti.fi/

Microfocus.com. [www-sivu]. [Viitattu 11.10.2010]. Saatavissa: http://www.microfocus.com/

technet.microsoft.com [www-sivu]. [Viitattu 23.10.2010]. Saatavissa: http://technet.microsoft.com/

Mikropc.net. [www-sivu]. [Viitattu 11.10.2010]. Saatavissa: http://mikropc.net/nettilehti/pdf/pc1406200140.pdf

Microsoft Visual Studio 2008 Documentation. [Ohjelman online-ohje].

MSDN Library. [www-sivu]. [Viitattu 6.9.2010]. Saatavissa: http://msdn.microsoft.com/

Ohjelmisto- ja algoritmitestaus. [www-sivu]. [Viitattu 29.8.2010]. Saatavissa: http://www.cs.hut.fi/Opinnot/T-106.850/PMRG/k2003/luennot/luento_1802_2003/testaus-kalvot.pdf

Software Business Competence. [www-sivu]. [Viitattu 26.8.2010]. Saatavissa: http://www.oamk.fi/sbc/testaus/index.htm

Software test automation in practice: empirical observations. [www-sivu]. [Viitattu 2.9.2010]. Saatavissa: http://www.hindawi.com/journals/ase/2010/620836.html
TestPartner 6.3 Ohje. [Ohjelman online-ohje].

Viestintävirasto. [www-sivu]. [Viitattu 3.10.2010]. Saatavissa: http://www.ficora.fi/index.html

W3C Web Services Architecture 2004. [www-sivu]. [Viitattu 3.10.2010]. Saatavissa: http://www.w3.org/TR/ws-arch/#whatis

W3C Recommendation 2008. [www-sivu]. [Viitattu 29.8.2010]. Saatavissa: http://www.w3.org/TR/REC-xml/

W3C XML Technology. [www-sivu]. [Viitattu 23.10.2010]. Saatavissa: http://www.w3.org/standards/xml