Opinnäytetyö (AMK)

Tietojenkäsittelyn koulutusohjelma

Sähköisen liiketoiminnan järjestelmät

2012

Henna Eriksson

TESTAUSPROSESSIN KEHITTÄMINEN OSANA LAATUTYÖSKENTELYÄ

- Case: Dimenteq Oy

TURUN AMMATTIKORKEAKOULU
TURKU UNIVERSITY OF APPLIED SCIENCES

OPINNÄYTETYÖ (AMK) | TIIVISTELMÄ

Turun ammattikorkeakoulu

Tietojenkäsittelyn koulutusohjelma | Sähköisen liiketoiminnan järjestelmät

2012 | 35 + 2 liitettä

Minna-Kristiina Paakki

Henna Eriksson

TESTAUSPROSESSIN KEHITTÄMINEN OSANA LAATUTYÖSKENTELYÄ – CASE: DIMENTEQ OY

Ohjelmistotuotannon osana testaus on yksi laadunvarmistuksen osa, jonka tarkoituksena on todentaa ja varmistaa järjestelmälle asetettujen määritysten ja vaatimusten täyttyminen. Testaus on myös osa Dimenteq Oy:ssä tuotettavien järjestelmien laadunvarmistusta. Vuonna 2007 perustettu paikkatietoalan yritys on ottanut asiakaslähtöisten projektien yhdeksi kulmakiveksi testauksen ja sitä kautta yrityksen ja järjestelmien varmistettavan laadun.

Opinnäytetyön taustalla on Dimenteq Oy:lle kehitetyn testausprosessin tuloksena tuotettu testauksen käsikirja ja sen kehitysprosessin vaiheet. Käsikirja on yrityksessä testausprosessin ensimmäinen kokonaisuutta kuvaava opas, jonka tarkoituksena on jakaa testaustietoutta yrityksen sisällä. Testauksen käsikirjan lisäksi yrityksen sisäisissä toimissa on kehitetty työntekijöiden välistä kommunikaatiota ja palautteenantoa osana testausprosessia.

Testausprosessi kehittyi harjoittelujakson aikana ketteränä ja edelleenkehittyvänä prosessina, jonka tavoitteet muuttuivat tehdyn työn kehittyessä. Tavoitteena oli dokumentoida ja kehittää yritykselle toimiva testausprosessi, joka palvelee sekä testaajia työn tekemisessä että muita yrityksen työntekijöitä testauksen ymmärtämisessä. Testauksen suorittamiseen lisättiin vähitellen erilaisia palasia ja niistä syntyi ketterästi tutkien ja yhdistellen testauksen toiminnot ja testauksen käsikirjan teoria.

Testausprosessin tämänhetkinen toimivuus ja sen kuvaus takaa yrityksessä tasalaatuisemman ja yleisesti laadukkaamman testauksen ja sitä kautta tuotettavien järjestelmien laadukkuuden. Koska yksittäinen testaaja ei voi olla vastuussa tuotettavien järjestelmien laadusta, voidaan prosessin avulla jakaa ja ylläpitää laatuajattelua yrityksen sisällä.

ASIASANAT:

testausprosessi, ohjelmistotuotanto, testaus, testausmenetelmät, ketterät menetelmät, motivointi, laatu, yritykset

BACHELOR´S THESIS | ABSTRACT
TURKU UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology | E-Business Systems

2012| 35 + 2 appendices

Minna-Kristiina Paakki

Henna Eriksson

DEVELOPING OF TESTING PROCESS AS PART OF QUALITY WORKING – CASE: DIMENTEQ OY

One part of the software production testing refers to quality assurance which verifies and ensures definitions and requirements of the system. Testing is also part of software production and quality assurance at Dimenteq Oy, established in 2007. The company makes customer-oriented systems in geographic information. One focus of the company's software production is quality assurance of producable software and the company itself.

This thesis describes the process of planning testing manual for the Dimenteq and results of it. The manual is the first part of the unit that includes the company's whole testing process. The main function of the manual is to share the testing information inside the company. In addition, inside the company advanced communication and feedback aspects as part of testing process.

The testing process was developed within the author's practical training period as a program tester. The process cycle behind the testing manual was agile and it is still developing. The process cycle was developed with several sections which are inserted in part by part. The main target behind the process was to document and develop functional testing process. The process developed by collecting and connecting the information from the company's former documentation and common testing practices to action.

The testing process is currently working to assure the uniform quality of testing. The good quality of testing process also improves the quality assurance of software production. The tester cannot be in charge of the quality of software but with the manual can project team and company share and uphold quality thinking.

KEYWORDS:

Testing process, Testing, Testing method, Software production, Agile Software development, Motivation, Quality, Company

SISÄLTÖ

LIITTEET

KUVIOT

KÄYTETYT LYHENTEET JA SANASTO

ekvivalenssiluokka	joukko ekvivalentteja eli toisiaan vastaavia alkioita (tässä: syötteitä)
integraatiotestaus	usean eri ohjelmakoodin osan integroinnin eli yhdistämisen jälkeinen testaus
JIRA	Atlassianin tuottama tehtävienhallintajärjestelmä
järjestelmätestaus	kokonaisuuden testausta todentaen asetetut vaatimukset ja määritykset
lasilaatikkotestaus	testaajalla on käytössään järjestelmän kooditason tiedot ja rakenne
mustalaatikkotestaus	testaaja ei hyödynnä järjestelmän kooditason tietoa ja rakennetta
regressiotestaus	kertaalleen testattujen järjestelmän osien testaamista uudelleen kokonaisuutena
Scrum	ketterän ohjelmistotuotannon eräs projektinhallintamalli
TDD	testivetoinen testaus, test driven development
vesiputousmalli	vaiheellinen ohjelmistotuotantoprosessi
V-malli	testauksen vaiheet sisältävä ohjelmistotuotantomalli, joka perustuu vesiputousmalliin
yksikkötestaus	pienimmälle ohjelmakoodin palaselle kuten funktiolle suoritettava testaus

1 PROJEKTIN ALOITTAMINEN JA LÄHTÖKOHDAT

Opinnäytetyön tavoitteena oli kehittää Dimenteq Oy:n testausprosessia ja dokumentoida se tavalla, josta on hyötyä sekä yrityksen testaajille työssään että muille yrityksen henkilöille testausprosessin ymmärtämisessä. Lisäksi asiakkaalle toimitettavana laatukäsikirjan liitteenä tuotetun testauksen käsikirjan tavoitteeksi asetettiin testauksen merkityksen ymmärtäminen yrityksen sisäisen kokonaisprosessin osana.

Dimenteq Oy on vuonna 2007 perustettu paikkatietomarkkinoiden pieni, mutta kilpailukykyinen yritys. Yrityksen perustajilla on taustallaan vankka osaaminen paikkatietoalan eri tehtävistä ja organisaatioista kymmenien vuosien ajalta. Yrityksen palveluksessa toimii tällä hetkellä 14 henkilöä. Dimenteq Oy:n henkilöstöllä on laaja ymmärrys ja osaaminen eri teknologioista, joita hyödynnetään teknologiariippumattomien ja asiakaslähtöisten tietojärjestelmähankkeiden toteuttamisessa. Useimmat tuotettavat tietojärjestelmät toteutetaan selainpohjaisina. Dimenteq Oy on tuplannut vuodessa sekä työntekijämääränsä että liikevaihtonsa ja kasvun odotetaan jatkuvan edelleen seuraavan vuoden kuluessa. (Dimenteq Oy 2011.)

Dimenteqillä käytetään ketterää ja sopeutuvaa projektimallia, joka mukautuu toimeksiannon koon ja tavoiteasetannan mukaan. Projektien primäärinä ohjelmistotuotantomenetelmänä käytetään menetelmää, joka yhdistää ketterien ja suunnittelupohjaisten mallien parhaiksi todetut käytännöt. Mallissa yhdistyy yleisesti käytössä olevat ja käytännössä hyviksi todetut alan parhaat käytännöt. (Dimenteq Oy 2012.)

Testaus kuuluu yleisesti yhtenä osana ohjelmistotuotantomenetelmiin, niin myös Dimenteqillä. Yrityksen toiminnot testausprosessia lukuun ottamatta olivat tarkoin dokumentoituja ja huomioin harjoittelujaksollani testaajana, kuinka tärkeä osa dokumentointi on myös testausprosessia ja sen vakiointia.

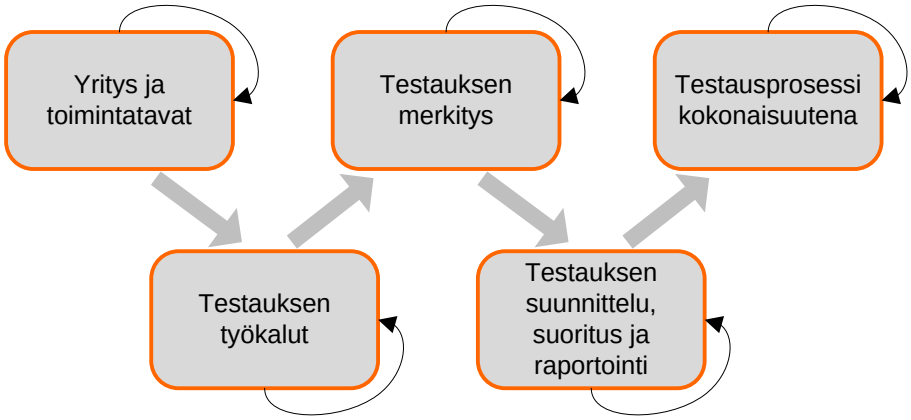
Dokumentoimattomalla testauksella ajankäytön suuntaaminen sekä työn laadunvarmistus ja mielekkyys eivät ole hallittavissa. Dokumentoitu

testausprosessi vaikuttaa sekä yrityksen tulevaisuuteen, testaajien työn kuvaan että tämän hetken testaajan työn laadun ja hallinnan tarkkailuun. Dokumentoinnin perusteella tehtyä työtä voidaan tarkastella ja arvioida myöhemmin sekä kehittää eteenpäin (Pyhäjärvi & Pöyhönen 2004).

1.1 Testausprosessin kehittyminen

Testausprosessi testauksen käsikirjan taustalla alkoi kehittyä, kun pohjatieto yrityksen toiminnasta, testauksen silloisesta nykytilasta ja niihin liittyvistä toiminnoista alkoi hahmottua ja kerääntyä tietooni harjoittelujakson aikana. Harjoittelujaksollani toimin Dimenteqillä järjestelmätestaajana. Hiljainen taustaprosessi kaiken kerätyn ja muistiinkirjoittamani tiedon sekä yrityksen silloisen testauksen yhteensaattamisesta toimivaksi prosessiksi alkoi harjoittelun alkumetreiltä. Harjoittelun aikana lisäsimme yhdessä toimitusjohtaja Teemu Virtasen ja teknisen johtajan Jan Lindbomin kanssa vähitellen erilaisia palasia testauksen toteutukseen. Testausprosessin kehittäminen oli lopulta osa harjoittelujakson omia ja yrityksen tavoitteita.

Itse testausprosessi kehittyi ketterästi inkrementaalisena prosessina ja sen kehitys on yhä käynnissä. Inkrementaalisella prosessilla tarkoitan syklistä kehitystä jokaisen osakokonaisuuden eli kehitysvaiheen ympärillä. Vaikka kuviossa 1 olen kuvannut kunkin kehitysvaiheen aloituksen kronologisen etenemisjärjestyksen, ei se kuvaa sitä kehitysjärjestystä, mikä kussakin vaiheessa on sen jälkeen tapahtunut.



Kuvio 1. Testausprosessin kronologinen ja syklinen kehittyminen.

Ensimmäisenä testausprosessin kehittymisen osana pidän yritykseen ja sen toimintatapoihin tutustumista. Toimintatavat kuten muukin yrityksen toiminta, kehittyvät jatkuvasti, ja mainittavana osana tätä kehitystä on sisäinen ketterä kehitysmalli, jonka esittelen myöhemmin. Ensimmäisenä varsinaisena testaukseen liittyvänä tehtävänäni harjoittelujaksolla oli tutustua testaustyökaluihin ja vanhoihin dokumentteihin testauksesta. Siitä alkoi testauksen merkityksen ymmärtäminen sekä yrityksen toiminnan että testaustyön kokonaisuuden osana.

1.2 Testauksen käsikirjan suunnittelu

Koska testauksen merkitys on huomioitu Dimenteq Oy:llä, oli sen dokumentointi seuraava tärkeä osa tämän projektin osan onnistumisessa. Jotta testauksen kautta saatavia tuloksia on helpompi saada, tarkastella, verrata ja analysoida, on taustalla oltava joukko noudatettavia ohjeita ja tietoa sen toteuttamisesta. Tavoitteeksi asetettiin taustaprosessin kuvaus käytettävien dokumentointivälineiden ja taustajärjestelmien sekä toteutettavien toimenpiteiden osalta.

Lisäksi yksi käsikirjan lisäajatus ja tukipilari oli pureutua mielikuvaan, etteivät kehittäjät yleisesti pidä testaajista, sillä testaajat löytävät heidän tekemästään koodista virheitä ja osoittavat niitä sormella (Katara 2011, 30). Tämä ajatus ei ole kokemuspohjainen, mutta sen käsitteleminen ja esiintuominen käsikirjan ja

tekemämme työn osana on mielestäni tärkeää. Testaajan tekemän työn ymmärtäminen on yksi tärkeä osa yhtenäistä ymmärrystä ja palautteen vastaanottamista (Vuori 2010b, 3–6).

Työn toteuttaminen alkoi alkutalvella 2012 harjoittelujakson aikana keräämieni muistiinpanojen ja testauksen yleisen taustatiedon yhteensovittamisella. Dimenteqillä oli jo olemassa testaustyön helpottamiseksi valittuja työvälineitä ja toimintatapoja, joiden dokumentaatio oli ympätty yhteen lyhyeen PowerPoint-esitykseen ja yhteen A4-pituiseen Word-tiedostoon. Näiden dokumenttien ja harjoittelujaksoltani kerätyn tiedon yhdistäminen testauksen yleiseen pohjatietoon tuottivat Dimenteqin testauksen käsikirjan ensimmäisen version.

Dimenteqin testauksen käsikirjan julkaistu versio sisältää sekä virallisen kuvauksen yrityksen testausprosessista liitettäväksi osaksi laatukäsikirjaa että testaajan työohjeistuksia. Työohjeistuksissa on käsitelty yrityksen testaajan työssään käyttämien työkalujen ohjeistus ja muu prosessin kuvaus. Testauksen käsikirjan avulla pystytään perehdyttämään uusia testaajia yritykseen ja osoittamaan tuleville asiakkaille testauksen merkitys ja laatu.

Testauskäsikirjan liitteeksi tuotettiin seuraavat dokumenttipohjat: testitapauspöytäkirja, järjestelmätestausraportti ja virheraportti. Tuotettujen dokumentointipohjien avulla pyrittiin helpottamaan testauksen raportointia, kuten testitapausten kirjaamista projekteittain yhteneviin välineisiin. Toimivien dokumentointipohjien avulla niiden käyttämiseen ei kulu turhaan ylimääräistä aikaa, joten sitä jää enemmän varsinaiseen järjestelmän testaukseen. Lisäksi toteutettujen dokumentointipohjien avulla projektikohtaisen testauksen laatutaso pystytään pitämään tarvittavalla tasolla.

2 TESTAAJAN ROOLIN JA TESTAUKSEN MERKITYKSEN YMMÄRTÄMINEN

Testausprosessin kokonaiskuva alkoi hahmottua, kun ymmärrys testauksen merkityksestä yritykselle alkoi valjeta. Vaikka yleisesti tiedetään, että testauksesta saatavilla hyödyillä voidaan parantaa sekä yrityksessä tuotettavia järjestelmiä että sen sisäistä toimintaa (Pyhäjärvi 2004), oli prosessoimattoman testauksen suunnittelun aloittaminen haastavaa opintojaan päättävälle testaajaharjoittelijalle. Prosessin suunnittelun taustalla työ alkoi kuitenkin kantaa hedelmää havaittujen onnistumisten ja nopean ymmärryksen edistymisen myötä pian harjoittelujakson aloittamisen jälkeen. Tämä luku kuvaa testaajan käsittelemiä ajatuksia työn tekemisen taustalla ja testauksen merkityksen yrityksen toiminnan osana.

2.1 Testaus laadunvalvojana

Dimenteqin ohjelmistotuotantoprosessin osana testauksella pyritään todentamaan, että ohjelma toimii virheettömästi ja lopputulos vastaa toiminnallisia ja ei-toiminnallisia vaatimuksia (Pyhäjärvi & Pöyhönen 2004). Testausprosessin toimivuudella ja sen edistämällä järjestelmän laadulla pyritään ylläpitämään ja jakamaan laatuajattelua yrityksen toiminnan osana.

Muiden toimien ohella testauskaan ei kuitenkaan todenna järjestelmää täydellisesti toimivaksi. Järjestelmät ovat ihmisten tekemiä, ja ihmiset tekevät virheitä (Katara 2011, 27). Kaikkien virheiden löytyminen laajasta järjestelmästä on ensinnäkin lähes mahdotonta, ja toisaalta turhaan aikaa vievää, joka osaltaan nostaa kustannuksia. Kataran (2011, 51) mukaan suuri osa virheistä löytyy suppeasta järjestelmän osasta, jota ei alun perinkään ole tehty kunnolla. Mahdollisten järjestelmän käyttöä häiritsevien virheiden löytyminen myöhemmin on kuitenkin kiusallista ja niiden korjaaminen kallista. Katara (2011, 52) toteaakin, että jopa puolet ohjelmistotuotannon resursseista voi kulua testaukseen, josta odotetaan merkittäviä tuloksia.

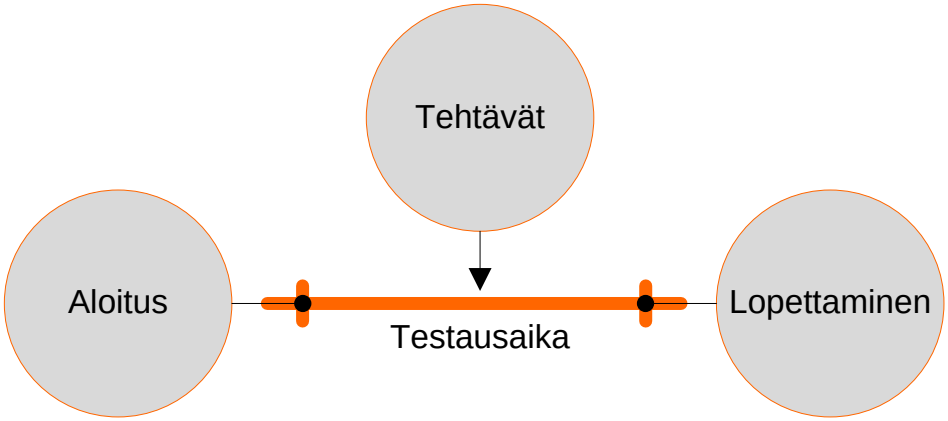
Testauksella pyritään tilanteeseen, jossa voidaan todeta järjestelmän vakuuttava toimivuus ja sen mahdollisesti sisältävien virheiden olematon vaikutus järjestelmän normaaliin toimivuuteen. Testauksen laatuun vaikuttavia tekijöitä on useita, ne voidaan kuitenkin jakaa neljään pääkohtaan, joita ovat suunniteltu prosessi, sen suorittajat ja aika sekä testauksesta syntyvän dokumentoinnin kuten testiraporttien laatu ja hyödynnettävyys (Pyhäjärvi 2004).

Jotta testauksen avulla pystytään todentamaan laadukkaan järjestelmän ominaisuuksia, on huomioitava, että itse testauksen laatu vaikuttaa sen tuloksiin. Hyvän testauksen laadun saavuttamiseksi projekteissa käytetään erityyppisiä toimenpiteitä (Pyhäjärvi 2004). Dimenteqillä on käytössään seuraavat Pyhäjärveä mukailevat toimenpiteet:

- testauksen suunnittelu
- testitapausten määrittely ja suunnittelu
- testien suoritus
- tutkiva testaus
- katselmoinnit
- välineet, työkalut
- automaatio, uudelleenkäyttö (testitapaukset, dokumenttipohjat).

Toimenpiteiden lisäksi testauksen laatuun vaikuttaa niiden suorittaja. Testaajan ammattitaito, motivaatio ja kokemus sekä mielentila vaikuttavat osaltaan testauksen laatuun. Lisäksi yhtenä laadun mittarina voidaan pitää tuotettavien raporttien sisältöä ja niiden lukemisen mielekkyyttä eli esimerkiksi sitä, miten jokin epämukava virheen korjausehdotus ”myydään” kehittäjälle korjattavaksi (Katara 2011, 35). Testaajan ammatillinen kehittyminen ja sitoutuminen sekä löydetyt virheet ja onnistumiset vaikuttavat motivaatioon testata järjestelmää yhä monipuolisemmin ja -ulotteisemmin (Vuori 2010b, 11–12). Hyvällä testaajalla on vainu löytää virheitä tiedostettujen ja vanhojen ongelmatapausten pohjalta (Vuori 2010a, 19). Mielentilaan saattaa vaikuttaa myös aika- ja olosuhdehaasteet, kuten kiire, joka tuntuu eri ihmisillä eri tavalla. Testaajan motivaatio tai mielentila ei saa kuitenkaan olla este testauksen tai järjestelmän laadulle.

Seuraavana esiteltävänä testauksen laatuun vaikuttavana osana on aika- ja laatupaine, jota on kuvattu kuviossa 2 (Pyhäjärvi & Pöyhönen 2004). Testauksen toteutuksen suunnitteleminen voidaan aloittaa tarvittavien dokumenttien valmistuttua tai toteutuksen ollessa tarvittavan pitkällä. Vaikka testauksen suunnittelu on aloitettu hyvissä ajoin, ei varsinaisen testauksen aloittamista voida aikaistaa. Kehittäjä saattaa ilmaista asian: ”odota, sillä tiedän tämän sisältävän vielä paljon virheitä” (Pyhäjärvi & Pöyhönen 2004).



Kuvio 2. Testaukseen kohdistuva aika- ja laatupaine (Pyhäjärvi & Pöyhönen 2004).

Testaukseen kohdistuu aikahaaste myös projektin lopusta päin, jolloin projektipäällikkö ilmoittaa järjestelmän julkaisupäivän ja toivoo kaiken olevan valmiina ilmoitettuun päivään mennessä. Lisäksi ajankäyttöön vaikuttavat tehtävät eli itse testitapausten suorittaminen, raportoiminen, uudelleentestaus ja mahdolliset odottamattomat viivästykset. Mikäli testaus ei mahdu suunniteltuun aikaraamiin, on vaihtoehtoina lykätä järjestelmän julkaisua, julkaista virhepitoinen tai karsittu versio järjestelmästä. (Pyhäjärvi & Pöyhönen 2004.)

Testauksen dokumentointi on varmasti tärkein testauksen ja sen tulosten laadun mittari (Stenberg 2007, 14). Testauksen ja testien dokumentoinnilla voidaan varmistua testattavaan järjestelmään tehtyjen testien kattavuuksista, ajettujen testitapausten määrästä ja löydettyjen virheiden määrästä (Pyhäjärvi 2004). Näiden avulla testauksen laatua voidaan tarkastella. Lisäksi

dokumentoinnin laatuun vaikuttaa dokumenttien sisällön luettavuus, itse sisältö ja sisällön hyödynnettävyys.

Testauksen laatua voidaan mitata myös laskemalla. Esimerkiksi testauksen laatua voidaan todentaa laskemalla prosenttiosuus löydettyjen virheiden määrästä suhteutettuna joko odotettuun järjestelmän laadun lähtötasoon tai testitapausten määrään tietyssä järjestelmän osassa. Lisäksi testauksen laatua voidaan mitata tutkimalla testauksen toimintaa kylvämällä virheitä järjestelmään tai taustadokumentaatioon ja laskemalla testauksen löytämien virheiden määrä suhteutettuna tunnettujen virheiden määrään. Tällaista virheiden kylvämisestä tai muuten tunnettujen virheiden löytämisestä saatavaa tulosta kutsutaan virheiden havaitsemisprosentiksi. Myös löydettyjen virheiden jakautuminen koko testausjakson ajalle on eräs huomioitava asia laadun tasaisuudesta. (Pyhäjärvi 2004.)

2.2 Testauksen roolit

Testaus on prosessi, jonka toteuttajina on useita osapuolia, kuten Vuori (2010b, 8) ja Katara (2011, 51) toteavat omissa pohdinnoissaan. Testauksella todennetaan järjestelmän laatua ja laadusta vastaavat projektitiimin jäsenet. Erillisen testaajan yhtenä pääperiaatteena on pitää yllä laadukkuutta, ja orientoida muut projektitiimin jäsenet tähän yhteiseen päämäärään (Vuori 2010b, 8-–10).

Laatuajattelun esiintuominen kaikkien yhteiseksi päämääräksi vähentää testauksen virheraporteista mahdollisesti aiheutuvia asenneongelmia ja niiden tuomia ristiriitoja yrityksen sisällä. Testaajan on osattava asennoitua muihin laatua edistäviin osapuoliin oikein yhteisen päämäärän saavuttamiseksi (Vuori 2010a, 3). Toisaalta jo löytyneiden virheiden oikeanlainen perusteleminen ja muiden osapuolien hyväksyminen osana omaa laadunvalvontaa tulee riittää molemminpuoleiseen ymmärrykseen.

Testaajan tai testaustiimin vastuulla on kokonaisuuksien testaus. Testaajan testaustyö alkaa, kun ensimmäinen järjestelmän dokumentti on valmis.

Valmistuvien dokumenttien pohjalta luodaan suunnitelmia ja lopulta testitapauksia. Varsinaisen testaustyön aloittaa kehittäjä, joka ajaa tekemäänsä sovelluskoodia kehitysympäristössä varmistaakseen sen toimivuuden. Testaus aloitetaan, kun ensimmäinen metodi tai muu sovelluskoodin osa on valmis. (Katara 2011, 52–60.)

Varsinainen testaustyö antaa erilliselle testaajalle erilaisia rooleja (Pyhäjärvi & Pöyhönen 2004; Katara 2011, 32):

- testitapauksia ajava robotti; suunnittelija, suorittaja.
- salapoliisi; tutkija, analysoija, vainukoira.
- hyväksy -leimasimen käyttäjä; päätöksentekijä.
- vikojen siivooja; virheiden raportoija, sormella osoittaja, vikojen myyjä.
- neuvonantaja; motivoija, kokonaisuuden hallitsija, muutoksentekijä.

Muita testaukseen erilaisin roolein osallistuvia osapuolia on esimerkiksi asiakkaissa, kuten projektin hyväksymistestaajat sekä muut olemassa olevat ja tulevat ostajat. Lisäksi projektipäälliköt peilaavat dokumentteja ja toteutusta asiakkaan kanssa, mikä voidaan mieltää eräänlaiseksi testaukseksi tai laadunvarmistamiseksi. Testaukseksi voidaan mieltää myös muun yrityksen johdon ja työntekijöiden suorittama projektitiiminjäsenten testaaminen ja haastaminen jokapäiväisessä tekemisessä.

3 OHJELMISTOTUOTANTOMALLIIN JA TOIMINTATAPOIHIN TUTUSTUMINEN

Testausprosessin kehittymisen ensimmäisenä vaiheena oli ymmärtää kyseisen yrityksen toiminnan perusteet. Tähän tarpeeseen yrityksen laatukäsikirja antoi ensimmäisen perusteellisen kuvauksen. Dimenteqin laatukäsikirjassa kuvataan muun tarpeellisen tiedon ohella käytettävät ohjelmistotuontantomallit ja mainitaan testauksen osuus niiden toteutuksessa. Laatukäsikirja ei kuitenkaan antanut testaajalle hänen tarvitsemiaan ohjeita työn suorittamiseen.

3.1 Ohjelmistotuotantomallia hahmottamassa

Laatukäsikirjan mukaisesti todellisuudessa toteutettavien projektien ohjelmistotuotantomallina käytetään mainitusti suunnittelupohjaiseen vesiputousmalliin ja ketteriin menetelmiin perustuvaa sopeutuvaa projektimallia. Projektimalli mukautuu projektikohtaisesti asiakkaan toiveiden ja tarpeiden mukaisesti. Yrityksen sisäisessä toiminnassa pyritään kehitys pitämään mahdollisimman ketteränä riippumatta asiakkaalle toteutettavasta projektimallista. (Dimenteq 2012a.)

Dimenteqin sisäisen ketteryyden sovelluskehitysmalli perustuu Scrumiin, jonka avulla kehitys vaiheistuu erimittaisiin sykleihin eli sprintteihin. Dimenteqillä yhden sprintin pituus on yleensä kahdesta neljään viikkoa (Dimenteq 2012a). Tällaisen ketterän menetelmän tarkoituksena on pystyä vastaamaan projektin kontrolloidun etenemisen ja vaiheistamisen haasteisiin, jotka ovat alalla yleisiä tuotannon ongelmakohtia. (Sininen meteoriitti 2011.)

Sisäinen ketterä kehitysmalli antaa työntekijöille erilaisia rooleja perinteisten yrityksen roolien lisäksi. Dimenteqillä on eroteltuna projekteittain vaihtuva projektipäällikkö, tiimimestari ja tiimi. Tiimi koostuu tiimiä vetävän ja ohjaavan tiimimestarin lisäksi testaajasta ja kehittäjistä. Lisäksi ketterällä projektilla on aina omistaja, joka omistaa päätösvallan järjestelmään tehtävistä

toiminnallisuuksista. Dimenteqillä omistaja on usein järjestelmän tilaajan projektipäällikkö. (Dimenteq 2012a; Sininen meteoriitti 2011.)

3.2 Testausmenetelmiä tutkimassa

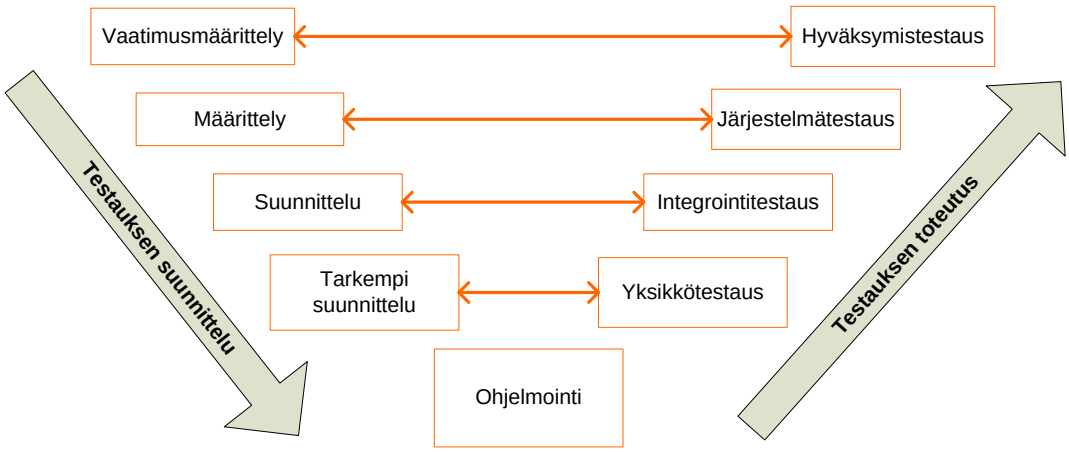
Ketterän ohjelmistotuotannon projekteissa syklinen kehitys ja jatkuva muutostarve ohjaa ja muovaa testaustakin ketteräksi ja nopeasyklisemmäksi (Vuori 2010a, 8). Testauksen on vastattava nopeisiin muutoksiin ja järjestelmän kehitykseen. Erilaisista projektiin vaikuttavista muutoksista johtuen ketterän kehityksen dokumentointi ei ole yhtä kattavaa ja pilkuntarkasti järjestelmän toimintoja tai toteutusta kuvaavaa (Vuori 2010b, 92). Dimenteqillä on käytössään JIRA-tehtävienhallintajärjestelmä, jonne jokaisesta toteutettavan järjestelmän toiminnosta kirjoitetaan toimintokuvaus. Toimintokuvaus on kertonut kehittäjälle, mitä oli tarkoitus saada tehtyä ja testaajalle toimintokuvaus ilmaisee, millainen järjestelmän toiminto tulisi pystyä todentamaan.

Dimenteqillä ketterässä ohjelmistotuotannossa testauksella pyritään todentamaan projektipäällikön ja projektin omistajan järjestelmälle määriteltyjen ohjeellisten käyttötapausten eli toimintokuvausten toimivuus järjestelmässä. Tiedetään millaisiin käyttötapauksiin järjestelmän tulisi vastata ja testauksella todennetaan, kuinka järjestelmän toteutus täyttää vaatimukset toiminnallaan (Vuori 2010a, 23–29). Ketterän testauksen yhteydessä voidaan lisäksi varmistua järjestelmän käytettävyydestä, helppokäyttöisyydestä ja käyttöliittymän tarkoituksenmukaisesta toimivuudesta riippumatta projektissa käytettävästä ohjelmistotuotantomallista (Vuori 2010a, 71).

Tutkiva testaus on yksi ketterästä testauksesta eroteltavissa oleva menetelmä. Tutkivalla testauksella kerätään tietoa järjestelmän toiminnasta käyttämällä järjestelmää eli tutkimalla järjestelmän käyttäytymistä ja antamalla järjestelmän ohjata testauksen kulkua. Tällaisen testauksen avulla voidaan järjestelmästä löytää sellaisia ongelmakohtia tai piilotettuja toiminnallisuuksia, jotka muuten saattaisivat jäädä todentamatta. Tutkivan ja yleisesti ketterän testauksen sivutuotteena voi syntyä myös jatkokehitysideoita ja parannusehdotuksia, joita

ennalta suunniteltujen testitapausten noudattamisella ei havaittaisi. (Vuori 2010a, 10–12.)

Perinteiseen vesiputousmalliin perustuvissa projekteissa testaus nojautuu enemmän dokumentteihin. Testaaja hyödyntää työssään eritoten vaatimusmäärittely- ja määrittelyvaiheiden dokumentteja. Testauksen yleisesti tunnettu malli on vesiputousmallin vaiheisiin perustuva V-malli, joka ottaa tarkemmin kantaa erityisesti testauksen eri kehitysvaiheisiin, joissa kunkin vaiheen rooli- ja päämääräjako vastaa kohtaavaa projektin kehitysvaihetta (kuvio 3). Jokaisen projektin kehitysvaiheen lopputuloksena syntyvä dokumentointi vastaa kohtaavan testausvaiheen toteutuksen sisältöä ja tarkoitusta. (Katara 2004.)



Kuvio 3. Testauksen V-malli (Stenberg 2007, 9).

Dimenteqillä V-mallin vaiheiden ohella suoritetaan myös ketterää testausta. Ketterällä testauksella pystytään poistamaan jäykkyyttä V-mallin suunniteltujen testitapausten ja järjestelmän välistä luomalla todenmukaisempi kuva järjestelmästä (Vuori 2010a, 13). Toisaalta myös ketterän testauksen puutteiden, kuten dokumentoinnin vähyyden korvaaminen V-malliin pohjautuen yhtenäistää mallien parhaita käytäntöjä (Vuori 2010, 24–27). Dimenteqillä parhaiksi todettuja käytäntöjä hyödynnetään testauksessa riippumatta ohjelmistotuotantomentelmästä samoin kuin muissakin ohjelmistotuotannon osissa (Dimenteq 2012a).

3.3 V-mallin testauksen vaiheet

Dimenteqin testauksen käsikirjaan toteutettiin taulukkomuotoisena V-mallin yksikkö-, integraatio- ja järjestelmätestauksen kuvaukset. Taulukkomuotoisena asiat ovat selkeästi otsikoiden alta luettavissa samanmuotoisina eri vaiheissa ja tiedot helposti päivitettävissä. Taulukko järjestelmätestauksesta on liitteenä 1.

Yksikkötestaus

Yksikkötestaus on varsinaisen testauksen aloittaja. Yksikkötestauksessa kehittäjä ajaa tekemäänsä ohjelmakoodia lasilaatikkomenetelmällä kehitysympäristössä yrittäen havaita siitä ongelmia tai virhekäyttäytymistä samalla varmistaen, että koodi toimii halutulla tavalla (Katara 2011, 60–61). Yksiköiden toiminta tulee vastata V-mallin tarkemmassa suunnittelussa kirjattuja vaatimuksia. Vaatimusten perusteella ohjelmakoodin palaselle kuten metodille luodaan yksikkötesti, joka testaa palasen toimintaa (Katara 2011, 62–71). Jokaisen yksikön toiminta varmistetaan kehitysympäristössä ennen sen liittämistä eli integrointia muihin palasiin. Ohjelman yksiköiden alkuperäiset virheet jäävät usein vain kehittäjän tietoon (Katara 2011, 60).

Yksikkötestit voidaan luoda myös ennen varsinaisen ohjelmakoodin toteutusta. Tällöin kehittäjä toteuttaa koodin, joka pääsee läpi esimerkiksi toisen tiimin jäsenen, kuten pääkehittäjän, luomasta yksikkötestistä. Tällaista testausta kutsutaan testivetoiseksi (Test Driven Development, TDD) ja sen uskotaan vähentävän virheiden määrää, mutta myös lisäävän toteutettavan koodin pituutta (Katara 2004).

Integrointitestaus

Integrointitestauksessa kehittäjä todentaa suunnitteluvaiheessa kuvattujen vaatimusten vastaavuuden toteutukseen. Integraatiotestauksessa liitetään yksikkötestattuja ohjelmakoodin palasia toisiinsa ja testataan niiden toimivuus yhdessä. Näiden vaiheiden suoritus sijoittuu osittain päällekäin ilman selkeää

rajaa. Integraatiotestaus toteutetaan yleisesti sekä lasi- että mustalaatikkotestauksena, jolloin pyritään löytämään mahdollisia ongelmia kutsuttavista rajapinnoista ja komponenttien välisestä kommunikoinnista sekä kooditasolla että rakentuvan sovelluksen näkökulmasta. (Katara 2011, 81–84.)

Kattavassa integraatiotestauksessa joudutaan usein tekemään puuttuvien ohjelmaosien tilalle tynkiä ja ajureita. Tyngät korvaavat testattavan osan kutsumia komponentteja ja ajurit testattavaa osaa kutsuvia komponentteja. Näiden avulla voidaan ohjelma koota pala palalta kokonaisuudeksi niin, että jokainen ohjelman osa on varmistettu ja testattu toimivaksi. (Katara 2011, 91-–95.)

Dimenteqillä sekä yksikkö- että integraatiotestauksen toteuttaa kehittäjä kehitysympäristössään. Kehitysympäristössä kehittäjä testaa omaa koodiaan, ja aika-ajoin yhdistää eli integroi myös muiden tiimissä työskentelevien kehittäjien yksikkö- ja integraatiotestatut komponentit omaansa versionhallinnnan kautta. Jokainen kehittäjä ylläpitää omaa kehitysympäristöään omalla koneellaan. Kokonaisuuden integroinnissa käytetään sekä katselmointeja että koodin testaamista yhdessä lopulta palvelimelle luodussa testiympäristössä. (Dimenteq 2012a.)

Järjestelmätestaus

Järjestelmätestauksella pyritään todentamaan Järjestelmän määrittelyiden vastaiset toiminnot ja löytämään mahdolliset ongelmakohdat ennen sen luovuttamista asiakkaalle (Katara 2011, 101). Dimenteqillä järjestelmätestauksen suorittaa erillinen testaaja, joka käyttää aikaa järjestelmälliseen toteutuksen läpikäyntiin. Testaaja käsittelee toteutettua järjestelmää kokonaisuutena testiympäristössä. Kokonaisuudesta voi puuttua vielä osia, mutta testaus aloitetaan, kun yksi toimintokokonaisuus on valmis testattavaksi (Dimenteq 2012a). Kokonaisuudenhallintaan kuuluu järjestelmän laadunvarmistaminen, tietoturva, käytettävyys, ulkoasu ja muut vastaavanlaiset

seikat (Katara 2011, 102). Näihin pyritään ottamaan kantaa järjestelmätestauksen yhteydessä.

V-mallin mukaisessa järjestelmätestauksessa testaus toteutetaan testitapausten perusteella, jotka muodostetaan määrittelyvaiheen pohjalta dokumentoinnin valmistuttua. Muodostettavien testitapausten tulisi kattaa kaikki määrittelyvaiheessa kuvattujen toiminnallisuuksien tarpeet ja erityispiirteet. (Katara 2004.)

Järjestelmätestauksessa tai sitä myöhemmin löydettyjen virheiden korjaaminen on yleisesti kallista. Toteutus on edennyt niin pitkälle, että virheet saattavat esiintyä niin, ettei niiden alkuperää ole helppoa todentaa. (Katara 2004.) Kaikenlaiset virheet on kuitenkin hyvä kirjata ylös jatkotoimenpiteitä ja myöhemmin mahdollisesti ilmenevien lisävirheiden korjauksen helpottamista varten. Virheiden löytyminen ja realisoiminen on parempi tehdä tässä vaiheessa kuin myöhemmin.

Järjestelmätestauksen lopuksi koko järjestelmän kertaalleen testatut osakokonaisuudet testataan läpi yhdessä varmistaen, ettei niiden integrointi ole muuttanut tai rikkonut rakenteita. Tällaista testausta kutsutaan regressiotestaukseksi. Testien toistamisen vuoksi jokainen osakokonaisuuksien testitapaus on nauhoitettu automaatiotyökalun avulla. Nauhoitusten luominen ja niiden ajaminen vähentävät toistossa kuluvaa aikaa ja varmistaa järjestelmän toimivuuden (Katara 2011, 311).

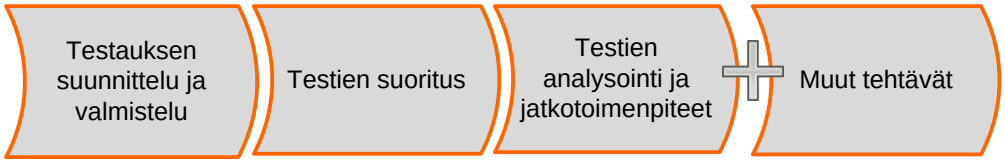
Hyväksymistestaus

Hyväksymistestauksen tarkoituksena on saada tilaajan hyväksyntä järjestelmälle asetettujen vaatimusten täyttymisestä (Katara 2011, 110). Vesiputousmalliin perustuvissa projekteissa Dimenteqillä tilaaja suorittaa yleensä 30 kalenteripäivän mittaisen hyväksymistestauksen aikana oman järjestelmätestauksen omassa testi- tai tuotantoympäristössä, johon on asennettu viimeisin versio hyväksytyn järjestelmätestauksen läpäisseestä järjestelmästä. Ketterissä ohjelmatuotannon projekteissa tilaaja suorittaa

jokaisen sprintin jälkeen toteutettujen toimintojen hyväksynnän ja projektin päätteeksi järjestelmän lopullisen hyväksymistestauksen. Toimintojen hyväksyminen sprintin päätteeksi antaa sekä tilaa nopeaan jatkokehitykseen että varmistaa projektin etenemisen oikeaan suuntaan ajallaan varmistaen, ettei toimintojen viimeistely jää projektin loppuun. Lisäksi toiminnoista sprintin päätteeksi saadut hyväksynnät vähentävät järjestelmän hyväksymistestauksessa ilmenevien virheellisten toimien määrää ja niiden korjauksia, sillä toimintojen rakenteet ovat jo hyväksyttyjä. Hyväksymistestausta edeltää järjestelmän käytön mahdollinen koulutus ja teknisen käsikirjan toimitus. Teknisen käsikirjan sisältö toimii yhtenä hyväksymistestauksen testitapauksena. (Dimenteq 2012a.)

4 TESTAUSPROSESSIN ETENEMINEN

Testaus on prosessi, joka etenee tilanteesta toiseen ilman suurempia raja-aitoja, ja on käynnissä lähes koko ohjelmistotuotantoprosessin läpi. Suoritusjärjestys voidaan kuitenkin pilkkoa suunnittelu, suoritus ja analysointi järjestykseen. Lisäksi testaukseen kohdistuu useita muita tehtäviä, kuten kokonaisuuden ja viivästymisten hallintaa sekä nk. asiakaspalvelua. Tämän kappaleen tiedot perustuvat pääasiallisesti sekä Pyhäjärven & Pöyhösen (2004) mietelmiin kuviosta 4 että omiin pohdintoihini siitä.



Kuvio 4. Testauksen suoritusjärjestys.

4.1 Testauksen suunnittelu ja valmistelu

Testauksen suunnitteluun ja valmisteluun vaikuttaa projektissa käytettävä ohjelmistotuotantomalli. Kuten todettua, suunnittelupohjaisissa projekteissa testauksen suunnittelu voidaan aloittaa perusteellisesti aikaisemmin kuin ketterien menetelmien projekteissa verrattuna varsinaisen testauksen aloitus ajankohtaan. Testauksen suunnitteluun kuuluu olennaisena osana testaussuunnitelma, testitapausten suunnittelu ja yleinen projektiin tutustuminen. Dimenteqillä projekteihin tutustuttaessa omaksutaan kokonaisvaltaista tietoa tilaajan järjestemälle asettamista toiveista, ongelmanasettelusta ja yleistä tietoa tilaajan työskentelyalasta. Tällaisen taustatiedon ja faktojen perusteella testaussuunnitelman ja testitapausten suunnitteleminen on mahdollista projektikohtaisesti.

V-mallin mukaisesti edettäessä testauksen suunnittelu aloitetaan samalla, kun projektin suunnittelu aloitetaan. Testaussuunnitelma on ensimmäinen testausprosessin osa, joka valmistuu lopulta suunnitteluvaiheen lopputuloksena. Testaussuunnitelmaan liitetään jokaisen toteutusvaiheen päätteeksi luodut

erilliset testaus- ja testisuunnitelmat (Katara 2004). Testaussuunnitelma toteutetaan Dimenteqillä projektipäällikön ja tiimimestarin kanssa yhteistyössä.

Testaussuunnitelmassa esitetään järjestelmälle toteutettavan testauksen pääpiirteet ja tavoitteet. Testaussuunnitelma vastaa kysymyksiin, mitä testataan, koska testaus aloitetaan, missä ympäristössä testataan, millaisilla välineillä ja kuinka paljon testataan sekä kuka testauksen suorittaa (Stenberg 2007, 30). Testaussuunnitelma on projektikohtainen sisältäen kuvauksen projektissa käytettävistä työvälineistä sekä toteutettavan ohjelmistokehitysmallin mukaisesta testauksesta.

Projektin toteutukseen tutustuessa on ainutlaatuinen tilanne tutkivan testauksen avulla todentaa ns. ummikkotilassa, mitä järjestelmässä tapahtuu ja millaisiin tilanteisiin ajaudutaan ilman tietoa siitä, mitä varsinaisesti pitäisi tapahtua. Ummikkotila katoaa nopeasti, kun järjestelmä tulee tutummaksi käyttää (Vuori 2010a, 10, 36). Kehittäjän on vaikea päästä toteuttamassaan järjestelmässä mainittuun ummikkotilaan, sillä kehittäjä vaistomaisesti testaa omassa koodissaan toimivia ja jo testattuja osia ja syötteitä. Tästä syystä yrityksen ja järjestelmän kannalta onkin tärkeää yksittäisestä kehittäjästä riippumaton testaaja. Testaajan kadotettua ummikkotilansa tulee hänestä askel askeleelta järjestelmän ammattimainen käyttäjä, joka tuntee kokonaisuuden. Testaajalla on lopuksi hyvä tietämys järjestelmän toiminnasta (Katara 2011, 305). Hän tietää toiminnallisuudet, keskeneräisyydet, korjaamattomat virheet ja yleiskuvan tilanteesta.

Testitapaukset ohjaavat testausta (Katara 2011, 42). Dimenteqillä testitapaukset luodaan projektin dokumentaation pohjalta, kuten suoraa vesiputousmallia noudatettaessa vaatimus- ja määrittelyvaiheen dokumenteista. Testitapausaineistoa täydennetään tutkivan testauksen avulla. Lisäksi toimintokohtaiset testitapaukset luodaan JIRA:an syötettyjen tehtäväkuvausten pohjalta riippumatta käytettävästä ohjelmistotuotantomenetelmästä.

Testitapausten tarkoitus on ryhmitellä ja jäsentää järjestelmään tehtävien testien suoritusta. Testitapauksilla tunnistetaan järjestelmän ominaisuudet,

joista tarvitaan laatuun liittyvää tietoa (Katara 2011, 11). Dimenteqillä testitapaukset dokumentoidaan testitapauspöytäkirjaan. Ketterän testauksen testitapauspöytäkirja kuvaa toimintojen oikeanlaista toimivuutta ja testaus varmentaa näitä kuvauksia. V-mallin mukaisessa testitapauspöytäkirjassa testitapaukset ovat tarkemmin yksilöityjä syötteitä myöten.

Testitapausten luomisen haasteena on niiden kattavuus, oikeellisuus ja tuottavuus. Testitapausten suorittamiseen kuluvaa aikaa minimoitaessa hyvä testitapaus kattaa mahdollisimman monta järjestelmän skenaariota hankaloittamatta testien tulosten analysointia. Huonosti suunniteltu testitapaus ei saavuta sille asetettuja tavoitteita. (Katara 2011, 43–46.) Hyvässä testitapauksessa otetaan huomioon yleisesti järjestelmässä tarkistettavat syötteet tietotyypin mukaan, kuten syötteen pituus, ekvivalenssiluokat ja tietoturva.

Testitapauksia voidaan luoda myös erilaisten testikerrosten avulla, jolloin samaa testitapausta suoritetaan rakenteellisesti eri näkökulmista. Tällaisia kerroksia voi olla esim. kokeileva, virheitä tekevä, analysoiva, käyttäjärooleja, helppokäyttöisyyttä tai selainten toimivuutta tutkiva testikerros. Testikerrosten avulla pystytään testaamaan ensin järjestelmän kriittisimmän toiminnan taso ja sitten toimintojen syvemmät tasot. Perustestitapauksia voidaan laajentaa testikerroksen avulla ja järjestelmää testata eri näkökulmista. (Vuori 2010a, 38–43.)

4.2 Testien suoritus

Testitapausten ja testauksen suorituksessa käytetään yleisesti pätevää suoritusjärjestystä, joka jaottelee arkipäiväisessä tekemisessä automaattisesti tapahtuvia toimia kronologiseen järjestykseen (Pyhäjärvi & Pöyhönen 2004). Dimenteqillä toimintokohtaisille testeille tehtävien toimenpiteiden suoritusjärjestys on Pyhäjärveä & Pöyhöstä (2004) mukaileva:

1. Testit suoritetaan manuaalisesti testiympäristössä.
2. Arvioidaan tuloksia.
3. Suoritetaan testit uudelleen varmistaen saadut tulokset.

4. Selvitellään ongelmia ja havaittuja epäselvyyksiä (vaatimukset, kehittäjä).
5. Dokumentoidaan testien suoritus (testauspöytäkirja, automatisointi).
6. Kirjoitetaan havaintoraportti JIRA:an.
7. Tehdään uusintatestaus korjausten jälkeen.
8. Havainnoidaan mahdollisia korjauksesta aiheutuneita ei-toivottuja sivuvaikutuksia.
9. Dokumentoidaan uusintatestien tulokset.
10. Informoidaan kehittäjää ja projektipäällikköä testauksen kulusta.
11. Päivitetään tarvittaessa suunnitelmia ja testauspöytäkirjaa.
12. Automatisoitu järjestelmätestaus suoritetaan tilaajan testiympäristössä.

Suoritusjärjestykseen kuuluvien kohtien toisto on projektikohtaisesti muuttuva. Jokaisessa projektissa määräytyy omanlaisensa suoritusaikataulu eteen tulevien asioiden summana. Osassa toiminnoista inkrementaalikierroksia on enemmän kuin toisissa.

4.3 Testien analysointi ja jatkotoimenpiteet

Useissa testin suoritusjärjestyksen kohdissa kuvataan erilaisia testauksen raportointiin liittyviä tehtäviä. Raportointi tapahtuu sekä kirjallisesti että suullisesti (Katara 2011, 210–211). Raportointi eli tulosten analysointi ja kirjaaminen on osa testauksen laadunvarmistamista ja testauksen jatkotoimenpiteitä kuten edelleenkehitystä.

Testauksen suorituksen laatua on mahdotonta mitata ilman syntyvää dokumentaatiota (Pyhäjärvi & Pöyhönen 2004). Dimenteqillä dokumentointia suoritetaan eri tavalla riippuen projektin ohjelmistotuotantomallista. Suora vesiputousmalli vaatii järjestelmällisen dokumentoinnin myös testauksen osalta, mutta ketterien menetelmien dokumentointi on kevyempää, projektille räätälöityä; päätarkoituksena testitulosten kirjaaminen, analysointi ja testauksen mitattavuus.

Dimenteqillä yksi raportoinnin osa on testitapauspöytäkirja, jossa hallitaan suunniteltuja, suoritettavia ja todennettuja testitapauksia.

Testitapauspöytäkirjaan merkitään jokaisesta testitapauksesta kuvaus, odotettu tulos, suoritusajankohta, käytettävä selain ja tulokset. Vesiputousmallin mukaisessa testauksessa testitapaukset kirjoitetaan ennen testauksen aloitusta määrittelyiden perusteella. Testitapauksia täydennetään testitapauspöytäkirjaan testauksen aikana ilmenevien testaustarpeiden ilmentyessä.

Testauksen päätteeksi yrityksessä tuotetaan järjestelmätestausraportti järjestelmään tehtyjen testien ja niistä saatujen havaintojen perusteella. Järjestelmätestausraportista käy ilmi, mitä on testattu ja minkälaisilla välineillä sekä millaisia tuloksia suoritetuista testeistä on saatu. Raportissa kuvataan testauksen pääpiirteet ja suoritukset sekä mahdolliset ongelmat ja virheet. Testiraportin liitteeksi laitetaan kuormitustestauksen ja automaatiotestien html-muotoiset raportit. Järjestelmätestausraportin liitteeksi voidaan myös toteuttaa JIRA:an syötetyt toimintokuvaukset kommentteineen eli testauksen toimintokohtaiset raportit. Järjestelmätestausraportin katselmoi projektipäällikkö projektin lopussa ja raportti toimitetaan sopimuksesta myös tilaajalle.

4.4 Muut tehtävät

Testauksen etenemiseen ja ajankäyttöön vaikuttaa myös useita muita tekijöitä. Tällaiset tekijät hidastavat testauksen aloitusta, etenemistä ja lopettamista. Tekijöitä, joihin testaaja voi esimerkiksi törmätä prosessin aikana, ovat odottamattomat viivästykset toimintojen valmistumisessa, testausympäristön ongelmat, virheraporttien analysointi ja tutkiminen kehittäjien kanssa, käyttäjäkoulutukset sekä yleisesti väärinkohdistettu ajankäyttö projektin testauksessa (Pyhäjärvi & Pöyhönen 2004).

Testauksen aloitukseen vaikuttaa mainittu aika- ja laatupaine, joka aiheutuu usein odottamattomista viivästyksistä. Viivästymistä voi aiheuttaa esimerkiksi resurssiongelmat, ongelmat toimintojen teknisessä valmistumisessa tai alun perin liian tiivis aikataulutus. Työn edetessä viivästymisiä voi tapahtua aiemmin kuvatun testien suoritusjärjestyksen eri vaiheissa. Työn eteneminen saattaa hidastua testiympäristön ongelmista, usean projektin samanaikaisesta resursoinnista hajanaisine tehtävineen tai vaikka uusien testaustekniikoiden

samanaikaisesta perusteellisesta haltuunotosta. Projektin testauksen loppupuolella haasteena on hallita testauksen lopettamisen ajoitus. Ajoitukseen liittyy kuitenkin useita erilaisia tehtäviä, kuten muuttuneiden asiakastarpeiden hallinta ja mahdolliset käyttäjäkoulutukset. (Pyhäjärvi & Pöyhönen 2004.)

5 TESTAUSTYÖKALUJEN HALTUUNOTTAMINEN

Yleisen yrityksen toiminnan ja toimintatapoihin tutustumisen ohella seuraavana tehtävänäni oli saada hyvä ymmärrys testaustyökalujen toiminnasta, hyödyntämisestä ja etenkin lopputulosten generoinnista. Työkalujen tarkoitus on helpottaa testauksen suorittamista ja tukea sitä poistamalla turhaa toistoa (Katara 2011, 311). Dimenteqille oli jo määriteltynä kaksi testaustyökalua, jotka edellinen testaaja ja tekninen johtaja olivat yhdessä valinneet. Valituiksi olivat tulleet Selenium-testausautomaatio- ja Jmeter-kuormitustestaustyökalu, joihin tehtäväni oli tutustua. Työ ei kuitenkaan osoittautunut niin yksinkertaiseksi kuin aluksi olisi voinut kuvitella.

Tutustuessani työkaluihin minulla oli käytössäni mainitsemani yrityksen kaksi testauksen dokumenttia ja työkalujen virallisia ohjeistuksia Internetin syövereistä sekä keskustelupalstoilla satojen samaisiin virhetilantesiin ajautuneiden käyttäjien vinkkejä. Tietolähteistä saamieni pohjatietojen avulla ratkaisin harjoittelujaksollani lukuisia ongelmia, jotka muun perustiedon ohella dokumentoin testauksen käsikirjaan. Käsikirjan asiakasversioon työkalujen ohjeistukset eivät kuitenkaan kuulu ilman erillistä sopimusta.

Esittelen kappaleessa lyhyesti Jmeter-kuormitustestaustyökalun ja Selenium IDE-toistotyökalun perusperiaatteet. Toistotyökalun käsikirjaan toteuttamani ohjeistus on opinnäytetyön liitteenä (liite 2).

5.1 Kuormitustestaustyökalu

Dimenteqillä käytetään järjestelmätestauksen osana toteutettavassa kuormitustestauksessa Apache Jmeter -työkalua, joka on open source ja Java -pohjainen (Apache Software Foundation 2012b). JMeter-kuormitustestaustyökalulla voidaan testata selainpohjaisen järjestelmän suorituskykyä tietyllä käyttäjämäärällä ja valituilla järjestelmän osilla. JMeter simuloi oikeaa käyttäjää, joka tekee määriteltyjä asioita järjestelmässä. Työkalun avulla testataan järjestelmän käyttäytymistä, toipumista ja toimimista,

kun järjestelmällä on valittu määrä käyttäjiä tietyllä aikavälillä (Apache Software Foundation 2012b).

Kuormitustestaukseen valitaan yhdessä kehittäjien ja mahdollisesti projektipäällikön kanssa sovitut testitapaukset. Kuormitettaviksi tapauksiksi määritellään eniten järjestelmää kuormittavat ns. pullonkaulat. Mahdollisten ongelmatilanteiden raportointi on erityisen tärkeää ainoastaan kuormitustestauksesta löydettyjen virheiden osalta.

Apache Ant on Java-kirjasto ja komentorivi -pohjainen työkalu, jonka avulla Jmeterin kuormitustestausten tulokset voidaan muodostaa html-muotoiseksi tiedostoksi. Ant-paketti sisältää pohjan raportin luomiseksi. Raportin sisältöä voidaan muokata normaalissa tekstieditorissa. Perusraporttipohja on englanninkielinen ja sisältää taulukkopohjan testiajon ominaisuuksille, jotka ovat testien yhteismäärä, mahdollisten virheiden määrä, onnistumisprosentti sekä suoritusajat. Raportista nähdään myös mahdolliset virheilmoitukset sekä yksityiskohtaisesti kunkin suoritetun testin suoritustiedot. Antin avulla voidaan Jmeter-testit ajaa tiedostoon ja muodostaa raportti tai Jmeteristä voidaan kirjoittaa tulokset suoraan tiedostoon (.jtl), joka ajetaan Antin avulla html-raportiksi. (Apache Ant 2012a.)

5.2 Automaatiotestaustyökalu

Selenium IDE on Mozilla Firefox -selaimen liitännäinen, jolla pystytään vähentämään manuaalisen toiston määrää testauksessa. Selenium-testausautomaatiotyökalua käytetään testitapausten suoritusten nauhoittamiseen ja nauhoitusten ajamiseen yhä uudelleen (Selenium Project 2012a). Työkalun käytön tarkoituksena on tukea järjestelmä- ja regressiotestausta.

Työkalun avulla voidaan todentaa kertaalleen testattujen testitapausten toimivuus toteutuksen edetessä ja toistaa yksinkertaisella tavalla toisiinsa liittyviä osakokonaisuuksia (Katara 2011, 313). Toteutin myös ohjeistukseen

muistilistan Seleniumista, jotta toiston merkitys testauksen edetessä ei unohtuisi (liite 2, 4).

Selenium Remote Control (RC) avulla saadaan Selenium-testitapausajosta muodostettua html-muotoinen raportti (Selenium Project 2012b). Suoritusten html-muotoisesta raportista voidaan todeta testien suoritusaika, testien määrä, läpäistyt testit ja epäonnistuneet testit sekä jokaisen testitapauksen suorituslokit. Selenium RC:n avulla Selenium IDE:llä nauhoitettuja testitapauksia ja testiryppäitä voidaan ajaa myös eri selaimilla (Selenium Project 2012b), mutta tämän ominaisuuden käyttöönotto on viivästynyt useista yrityksistä huolimatta.

5.3 Yhteenveto

Mainittujen testaustyökalujen tarkoitus on tukea järjestelmätestauksen suorittavaa testaajaa. Testaustyökaluista yksinkertaisesti ja siististi saatavat html-muotoiset raportit tukevat ajatusta testauksen dokumentoinnin tärkeydestä. Työkaluilla tehtyjen testien raportit liitetään osaksi lopullista järjestelmätestausraporttia. Tilaajalle toimitettavana järjestelmätestausraporttina html-muotoiset testiliitteet ovat helposti ymmärrettävä testaustulos ja siksi toimiva osa raportointia.

Testaustyökalujen haltuunottaminen oli aluksi haasteellista useiden eteentulleiden ongelmien osalta, mutta prosessina se on ollut kehittävä. Työkalujen toimivuuden eteen on joutunut tekemään töitä jopa pienen paineen alaisena ja lopputuloksista on saanut iloita oikeasti. Yhtenä esimerkkinä voisi mainita Jmeter-kuomitustestaustyökalun raportin generoimisen. Raportin generoimiseksi Antin avulla tarvittiin useita tunteja, mutta lopulta vika löytyi väärintulkitusta ohjeistuksesta. Ohjeistuksen englanninkieliset tekstit olivat lyhennetty niin, ettei koko lauseen tarkoitusta ollut helppo ymmärtää ja siitä syystä eräs komponentti sijoitettiin väärään kansioon.

Testaustyökalujen käytön vaikeutena on myös niihin kuluva aika. Testien suunnitteluun, tekemiseen, nauhoittamiseen, ajamiseen, muokkaamiseen ja

uusintatoistoon voi tuhraantua aikaa, jota on vaikea mitata. Lopputuloksena generoituvat tulosraportit antavat työkalujen käytöstä yksinkertaisen ja helpon kuvan. Tulosten saamisen taustalla voi kuitenkin olla paljon teknisiä tai muita ongelmia, joita lopputulos ei näytä eikä tulkitse. Tasapainoilu ajankäytön ja hyödyllisten tulosten välillä on ratkaistava tapauskohtaisesti.

6 TESTAUKSEN SUUNNITTELUPROSESSIN ARVIOINTI JA JOHTOPÄÄTÖKSET

Dimenteq Oy:lle toteutetun testausprosessin ensimmäinen käsikirjaversio auttaa ymmärtämään testauksen monipuolisen hyötynäkökannan yritykselle. Se on kuitenkin suhteellisen nuoren ja edelleen kehittyvän yrityksen ensimmäinen testauksen dokumentoinnin kokonaisvaltainen tietolähde eikä se ota kantaa kaikkiin testausprosessin osiin yksityiskohtaisesti.

Testausprosessi on kuitenkin jatkanut kehittymistä myös opinnäytetyössä kuvatun stabiloidun tilan jälkeen. Esimerkkinä tästä voin mainita perustestitapausten ja testauskohteiden tarkistuslistan. Testaussuunnittelun osana tällaisen perusdokumentin käyttö tuo huomattavan edun suunniteltuun ajankäyttöön. Tällaisia perustestitapauksia, kuten erityyppisten syöttökenttien testaamista, voidaan hyödyntää monessa projektissa sellaisenaan.

Suunnitteluprosessin ja testauksen käsikirjan lopputuloksena voidaan todeta sen oikeanlaisen työn jäsentelyn ja testauksen laadunvarmistamisen selkeyttäminen. Testien suoritusjärjestys, testaustyökalujen toimintaohjeet ja testauksen dokumentointipohjat järjestelevät arkipäiväisen toiminnan. Näiden parannusten avulla testaajan työ on yrityksessä mielekkäämmin prosessoitua eikä testaajan tarvitse ajatella, ettei suorita tarpeeksi monipuolisesti tai yksinkertaisesti testausta tai sen dokumentointia.

6.1 Suunnitteluprosessin haasteet

Testauksen suunnitteluprosessin kehittymisen ensimmäisenä haasteena oli ymmärtää kokonaisuus, johon minut oli asetettu päävastuuseen. Testausprosessia ei oltu kuvattu yrityksen dokumenteissa kovinkaan tarkasti ja testauksen aloittaminen harjoittelijana ilman yksityiskohtaisia ohjeita tai suurempaa alan kokemusta oli haaste sinänsä. Lähdin kuitenkin pelottomasti kokoamaan vähitellen erilaisia osioita testausprosessin osaksi, joita tunsin tarvitsevani prosessin tukemiseksi ja laadunvarmistamiseksi. Tavoitteena oli

työn helpottaminen ja yksinkertaistaminen sekä jonkinlainen stabiloitu tila. Tällaisesta stabiloituneesta tasanteesta on jatkossa helpompi kehittää lisäominaisuuksia testauksen suorittamiseen askel kerrallaan, ja uskon onnistuneeni siinä tähän mennessä.

Toinen haaste prosessin kehittymisessä, joka juolahtaa mieleeni aika-ajoin, on testausprosessia tukevat henkilöt. Yrityksessä oikeastaan kukaan ei tuntenut sen kummemmin testauksessa käytettäviä työvälineitä, joten niihin tutustuminen oli todella omissa hyppysissäni. Työkaluista käsikirjaan dokumentoidut hyödylliset piirteet tuskin kattavat koko työkalun ominaisuuksien kirjoa. Uskon kuitenkin, että nykyisissä ohjeistuksissa mainituilla tiedoilla pystytään luomaan niin kattavia ja yksityiskohtaisia testejä kuin on tarpeellista tällä hetkellä.

Kolmantena haasteena voisin mainita yrityksen sisällä tapahtuvan muutoksen hallinnan ja eteenpäin viennin. Koska testaus työnä vaikuttaa lähes kaikkiin yrityksen työntekijöihin, on sen merkityksen ymmärtäminen ja sisäistäminen muun työn osana tärkeää. Pahimmassa tapauksessa kukaan ei luota testaajaan ja testaaja ei luota esimerkiksi kehittäjien tekemään työhön. Dimenteqillä jokainen työntekijä ymmärtää, ettei testaaja yritä arvostella heitä ihmisinä vaan parantaa tekemämme työn laatua, arvoa ja esiintuomista yhteisenä projektina.

6.2 Tulevaisuuden pohdintoja

Kuten jo mainitsin testausprosessin kehittäminen on tästä eteenpäin osa yrityksen kokonaisprosessia ja mielestäni sen edistäminen ei ole enää kovin laajasti mahdollista itsenäisenä prosessina. Tällaista peruskehitystä, jota testauksen käsikirja on teorioineen, pystyy tekemään itsenäisenä testausyksikkönä, mutta jatkuva koulutus ja jonkinlainen testauksen kehitystiimi toisivat mielestäni paremmat avaimet oikeanlaisen yritykselle sopivan testauksen toteuttamiseen. Dimenteqillä tämä näkökulma on otettu jo työn alle.

Lisäksi testaajan ammattitaidon jatkuva kehitys oikeaan suuntaan muuttuvien toteutusten ja uusien teknologioiden alla on pidettävä balanssissa. Mielestäni testaajaksi kuten useaksi muuksikaan työntekijäksi, ei voi oppia pelkästään

kirjoja lukemalla. Työ tekijäänsä opettaa, mutta uusien tuulien hakeminen esimerkiksi testauskoulutuksista ja muista oman alan henkilöiden kokoontumistilaisuuksista on myös merkittävä osa ammattitaidon ja prosessin kehitystä. Uuden oppiminen ja tuoreen tiedon ymmärtäminen on yksi testaajan perusominaisuuksista (Vuori 2010a, 92).

Jos tekisin opinnäytetyöni uudelleen samasta aiheesta, korostaisin tekstissä enemmän yhtenä perusajatuksena ollutta lausahdusta: ”Kehittäjät eivät yleisesti pidä testaajista, sillä testaajat löytävät heidän tekemästään työstä virheitä ja osoittavat niitä sormella” (Katara 2011, 30). Näkökulmaan voisi nostaa esille työhyvinvoinnin, muutosjohtamisen ja muita työntekijöiden väliseen toimintaan liittyviä teemoja. Muutosjohtamisen muutosvastarinta ja sen tuomien ongelmien käsittely tuntuisi jollakin tavalla hyvältä opinnäytetyöidealta, vaikkei varsinaisesti alaan liitykään.

LÄHTEET

Apache Software Foundation 2012a. Apache Ant. Viitattu 29.4.2012 http://ant.apache.org/.

Apache Software Foundation 2012b. Apache Jmeter. Viitattu 29.4.2012 http://jmeter.apache.org/.

Bartlett, Nick 2008. Must known commands for Selenium IDE. Viitattu 13.4.2012 http://nickbartlett.com/wordpress/must-know-commands-for-selenium-ide/.

Dimenteq Oy 2012a. Dimenteq Oy – Laatukäsikirja 1.4. Dimenteq Oy.

Dimenteq Oy 2012b. Dimenteq Oy – Testauksen käsikirja 1.0. Dimenteq Oy.

Dimenteq Oy 2011. Viitattu 9.4.2012 http://www.dimenteq.com > Yritys.

Katara, Mika 2004. Kehittyneet testausmenetelmät. Viitattu 13.4.2012 http://www.cs.tut.fi/~clark/testausseminaari/alustus.pdf.

Katara, Mika 2011. Ohjelmistojen testaus. Viitattu 9.4.2012 http://www.cs.tut.fi/~testaus/s2011/luennot/.

Pyhäjärvi, Maaret 2004. Alihankittu testaus. Viitattu 14.4.2012 http://testausosy.ttlry.fi/webfm_send/117.

Pyhäjärvi, Maaret & Pöyhönen, Erkki 2004. Laadun ja kiireen haasteet testauksessa. Viitattu 9.4.2012. http://testausosy.ttlry.fi/webfm_send/102.

Selenium Project 2012a. Selenium IDE. Viitattu 29.4.2012 http://seleniumhq.org/docs/02_selenium_ide.html.

Selenium Project 2012b. Selenium RC. Viitattu 29.4.2012 http://seleniumhq.org/docs/05_selenium_rc.html.

Sininen meteoriitti 2011. Ketteryys haltuun: Scrum pähkinänkuoressa. Viitattu 13.4.2012 http://www.meteoriitti.com/fi-FI/tiedotteet/ajankohtaista/ketteryys-haltuun-scrum-pahkinankuoressa.

Stenberg, Arto 2007. Ohjelmiston testaus 2007 – testauksen hallinta. Viitattu 13.4.2012 http://www.pori.tut.fi/~stenberg/index_files/hallinta.pdf.

Vuori, Matti 2010a. Ketterä testaus ja testaus ketterässä ohjelmistokehityksessä. Viitattu 13.4.2012 http://www.mattivuori.net/julkaisuluettelo/liitteet/kettera_testaus.pdf.

Vuori, Matti 2010b. Testaajan eettiset periaatteet. Viitattu 25.4.2012 http://www.mattivuori.net/julkaisuluettelo/liitteet/testaajan_eettiset_periaatteet.pdf.

Ote testauksen käsikirjasta: V-mallin järjestelmätestaustaulukko

Järjestelmätestaus

Testausvaihe:
Järjestelmätestaus

Vaiheen tarkoitus:
Järjestelmätestauksella pyritään todentamaan järjestelmän määrittelyiden vastaiset vaatimukset ja löytämään mahdolliset ongelmakohdat käsiteltäessä järjestelmää kokonaisuutena.

Mitä todennetaan?
• Määrittelyvaiheen vaatimukset • Kuvattujen toimintojen tarpeet ja erityispiirteet • Käytettävyys • Käyttöliittymän ulkoasu • Käyttöopasteet ja -dokumentaatio • Kuormitus • Luotettavuus • Käyttöliittymän tietoturva

Kenen vastuulla:
Testaaja. Tiimimestari ja kehittäjät avustavat.

Milloin?	**Millä tavoin?**
• Ensimmäisen toimintokokonaisuuden valmistuttua • Kokonaisuuden valmistuessa • Muutostilanteissa	• Testausympäristössä • Lähinnä mustalaatikkomenetelmällä • Luomalla testitapauksia sekä määrittelyiden pohjalta että tutkivan testauksen avulla • Toistamalla testitapauksia • Kuormittamalla järjestelmää

Riskit:
Testauksen laadun tulee olla tasaista huolimatta testauksen suorittajasta, testattavan koodin kirjoittajasta tai projektista. Aika- ja olosuhdehaasteet.

Tulevaisuutta:
Tietoturvatestauksen mallintaminen ja työkalun haltuunotto. Ketterän testauksen menetelmien lisäkehitys ja dokumentointipohjat.

Ote testauksen käsikirjasta: Selenium IDE - testausautomaatiotyökalun ohjeistus

Testitapausten toistaminen

Testitapausten toistaminen on suuri osa testausprosessia. Testauksessa järjestelmän osia valmistuu eri aikaan, niitä liitetään korjattuina ja muokattuina takaisin järjestelmään. Toimintojen lisäykset ja muokkaukset saattavat muuttaa jo olemassa olevaa järjestelmää ja sen osien toimintaa. Järjestelmän oikeanlainen toiminta on pystyttävä varmistamaan tehtyjen muutosten jälkeen. Testitapausten toistamiseksi käytetään toistotyökalua.

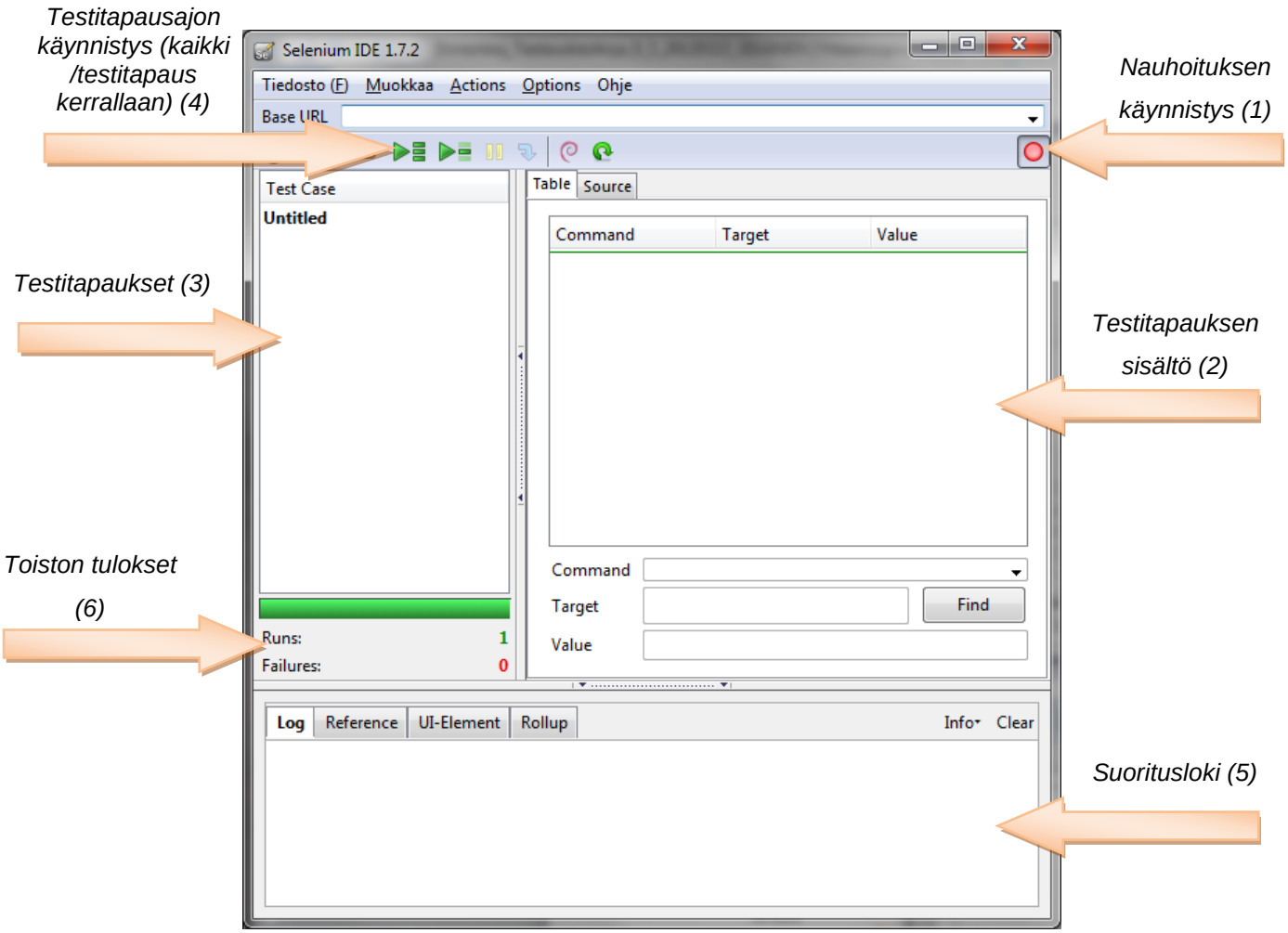
Toistotyökalun avulla jo testattujen toimintojen uudelleenvarmistus käy nopeasti ja luotettavasti. Testitapauksen manuaalisen toiminnan testauksen jälkeen toiminnallisuus nauhoitetaan toistotyökalun avulla, josta se voidaan tallentaa ja avata koska tahansa uusintatoistoon. Varsinaisen uusinta-ajon suorittaminen on yksinkertaista, ja tapahtuu käynnistyspainikkeesta.

Selenium IDE

http://seleniumhq.org/docs/02_selenium_ide.html

Selenium IDE on Mozilla Firefox -selaimen liitännäinen, jolla pystytään vähentämään manuaalisen toiston määrää testauksessa. Selenium-testausautomaatiotyökalua käytetään testitapausten suoritusten nauhoittamiseen ja nauhoitusten ajamiseen yhä uudelleen. Työkalun käytön tarkoituksena on tukea järjestelmätestausta.

Työkalun avulla voidaan todentaa kertaalleen testattujen testitapausten toimivuus toteutuksen edetessä ja toistaa yksinkertaisella tavalla toisiinsa liittyviä osakokonaisuuksia. Osakokonaisuuksien muutosten ja lisäysten jälkeen tulee jokainen testitapaus tarkastaa mahdollisten puutteiden ja virheiden varalta.



Kuva 1. Selenium IDE ikkuna.

Selenium IDE -ikkuna avataan sen latauksen jälkeen Mozilla Firefox -selaimen Työkalut-valikosta (Ctrl+Alt+S komento). Nauhoittaminen aloitetaan selaimen ikkunasta, kun nauhoitus-painike (1) on alaspainettuna Selenium IDE -ikkunassa.

Testitapausten nauhoitettu sisältö näytetään sille varatulla alueella (2). Uusi testitapaus voidaan luoda valikosta tai klikkaamalla hiiren oikealla painikkeella testitapaukset-osiossa (3) ja valitsemalla avautuvasta valikosta *New Test Case*. Testitapausta voi vaihtaa kaksoisklikkaamalla sen nimeä.

Testitapaukset tulee tallentaa kuvaavalla nimellä aloittaen projektin nimestä <projektin nimi>_<testitapauksen nimi>_< mahdollinen versio>.html. Projektin kaikki testitapaukset tulee tallentaa myös testiryppääseen (*Test Suite*), jolloin testitapaukset pysyvät yhdessä projektikohtaisesti ja jatkotyöstäminen on helpompaa.

Testitapausajoa voidaan hallita työkalupalkin toimintojen avulla (4):

 testitapausajon käynnistys kaikilla testitapauksilla

 testitapausajon käynnistys valitulla testitapauksella

 testitapausajon pysäytys ja uudelleen käynnistys

 testitapausajon suorittaminen komento kerrallaan (komentojen toiminnan testaus).

Komentokohtaisen suorituslokin (5) viestit näkyvät sivun alalaidassa. Lisäksi alalaidan osioon saa näkyviin yksittäisen komennon ohjeen (*Reference*). Testitapausajon pienimuotoiset tulokset (6) näkyvät Testitapaus-osion alla.
Nauhoitettujen komentojen lisäksi usein on tarpeellista lisätä ja muokata manuaalisesti testitapauksia. Nauhoitettujen komentojen väliin voidaan lisätä *pause*-komento, joka pysäyttää ajon aikana komentojen suorituksen määritellyksi ajaksi. On kuitenkin hyvä huomata, että *pause*-komennon lisäksi myös *WaitFor*-komennoilla voidaan odottaa esimerkiksi elementin latausta ja ilmestymistä ruutuun (*WaitForElementPresent*) ilman ajon varsinaista pysäytystä.

Selenium IDE:n hyödyllisiä komentoja:

Komento	**Kuvaus**
open	Selain avaa määritellyn sivun
assert	Vahvistaa esim. elementin olemassaolon
verify	Varmistaa esim. elementin olemassaolon.
click	Klikataan määriteltyyn elementtiin esim. syöttökenttään.
type	Kirjoittaa määritellyn tekstin kenttään
typeKeys	Kirjoittaa määritellyn tekstin kirjain/numero/merkki kerrallaan.
select	Valitsee määritellyn vaihtoehdon valintalaatikosta.
addSelection	Korostaa määritellyn vaihtoehdon järjestelmän valintalaatikosta.
waitFor	Odottaa kunnes toteutuu. Esim. WaitForElementVisible
pause	Keskeyttää suoritusajon määritellyksi ajaksi. Esim. value 3000=3s.

store	Tallettaa esim. elementin arvon valitun nimiseen muuttujaan.
echo	Tulostaa lokiin tekstiä ja muuttujien arvoja.

Taulukko 1. Selenium IDE:n hyödylliset komennot (Bartlett 2008).

Seleniumin muistilista:

- Selenium ei pysty toistamaan kaikkia kartan toimia kuten sen raahausta.
- Tarkista, että olet nauhoittanut kaikki tarpeelliset toiminnallisuudet. Myös virheet ja virheelliset toiminnot ovat tarpeellisia.
- Älä tyydy korjaamaan hajonnutta testiä toimivaksi, vaan tutki miksi testiä ei läpäistä.
- Tarkista, että olet tallentanut sekä testitapaukset (*Test Case*, ei *-merkintää nimen perässä) että testiryppään (*Test Suite*) niitä kuvaavilla nimillä.
- Käytä aikaa oikeanlaisten testinauhoitusten tekemiseen. Järjestelmän jälkitarkastus automaation avulla on yksinkertaisempaa hyvin tehtyjen nauhoitusten perusteella.

Taulukko 2. Seleniumin muistilista.

Selenium RC

http://seleniumhq.org/docs/05_selenium_rc.html

Selenium Remote Control (RC) avulla saadaan Selenium-testitapausajosta muodostettua html-muotoinen raportti. Raportti muodostetaan command prompt:n kautta tarvittavien asennusten jälkeen. Suoritusten html-muotoisesta raportista voidaan todeta testien suoritusaika, testien määrä, läpäistyt testit ja epäonnistuneet testit sekä jokaisen testitapauksen suorituslokit.

Selenium RC:n avulla Selenium IDE:llä nauhoitettuja testitapauksia ja testiryppäitä voidaan ajaa myös eri selaimilla.

Raportin muodostamisen komento Firefox-selaimella command promt:n kautta:
java -jar <C:\polkusi\selenium-server*.jar> -htmlsuite
"*firefox <P:\olkusi\firefox.exe>" http://suoritettavan.sivun.osoite.fi
"<P:\olkusi\suoritettavaan\testiryppääseen(*Test Suite*).html>"
"<P:\olkusi\kirjoitettavaan\tulostiedostoon.html>"