LAUREA

GLOW–projekti ja sen hyväksymistestaus

Rönnquist, Olavi

2009 Leppävaara

Laurea-ammattikorkeakoulu
Laurea Leppävaara

GLOW–projekti ja sen hyväksymistestaus

Olavi Rönnquist
Tietojenkäsittelyn koulutusohjelma
Opinnäytetyö
Toukokuu, 2009

Laurea-ammattikorkeakoulu
Laurea Leppävaara
Tietojenkäsittelyn koulutusohjelma
Yritysten tietoverkot

Tiivistelmä

Olavi Rönnquist

GLOW-projekti ja hyväksymistestaus

Vuosi 2008 Sivumäärä 53

Ohjelmistojen testaus on tärkeä osa ohjelmistokehitystä. Vaikka ohjelmistojen täydellinen testaus on mahdotonta, niin hyvin suunnitellulla ja toteutetulla testauksella saavutetaan asiakasta tyydyttävä ja laadukas tuote. Siltikään ohjelmistoprojekteissa testaukselle ei usein varata tarpeeksi aikaa eikä resursseja, jolloin testaus jää liian yleiselle tasolle ja loppukäyttäjän vaivaksi jää virheiden löytäminen.

Lumenella aloitettiin syksyllä 2007 GLOW-projekti, jonka tarkoituksena oli löytää ja ottaa käyttöön Lumenen tarpeita vastaava ja yrityksen toimintaa tehostava dokumenttien ja projektien hallintajärjestelmä. Lumenella ei ollut aikaisemmin käytössä vastaavanlaista järjestelmää.

Opinnäytetyön tavoitteena oli luoda ja suorittaa hyväksymistestaus Lumenen uudelle tietojärjestelmälle opinnäytetyön tekijän toimesta ja varmistaa, että ohjelmistoyrityksen luoma järjestelmä varmasti vastaa ohjelman määrittelydokumentteja ja Lumenen tarpeita.

Opinnäytetyön teoriaosuus perustuu testausta käsittelevään kirjallisuuteen. Sen tarkoituksena on luoda kattava ja tiivis kuvaus ohjelmistojen koko testausprosessista lähtien liikkeelle uuden ohjelmiston määrittelydokumenteista ja suunnitelmasta päättyen asiakkaan suorittamaan hyväksymistestaukseen.

Lumenen suorittama ensimmäinen hyväksymistestaus ei mennyt aivan suunnitelmien mukaisesti, koska testeihin saatu ohjelma oli keskeneräinen, eikä sitä pystytty testaamaan kuten oli aiottu. Toiseen hyväksymistestaukseen saatiin jo paljon valmiimpi järjestelmä ja luodut testit pystyttiin suorittamaan. GLOW-projekti pystyttiin päättämään onnistuneesti, kun ohjelmistoyritys korjasi siinä havaitut virheet. Projektin aikataulussa ei kuitenkaan onnistuttu pysymään.

Asiasanat testaus, ohjelmistotestaus, testauksen suunnittelu, hyväksymistestaus, GLOW

Laurea University of Applied Sciences
Laurea Leppävaara
Information Technology Programme
Information Systems and Data Communications

Abstract

Olavi Rönnquist

GLOW project and acceptance testing

Year 2008 Pages 53

Software testing plays a major role in the development of quality software. Although absolute testing of software is impossible, it is still achievable to produce quality software and, thus, gain customer satisfaction through a well-planned and executed testing process.However, software companies still do not reserve enough time or resources for testing, and leave the testing of software at a too common level. As a result, users are left with the inconvenience of finding errors in the software they use.

In the autumn 2007 the Finnish cosmetics company Lumene started a project named GLOW. The goal of GLOW was to find and deploy a document and project management system that would meet the needs of the company and, at the same time, benefit Lumene's business operation. Before, Lumene had no similar system to GLOW in use.The objective of this thesis is to describe the project GLOW in which an acceptance testing for the new programme was planned and implemented by an appointed project group including the writer. The main goal of this testing procedure was to ensure that the new management system meets the needs of Lumene as well as conforms to the specification documents of the new programme itself. The programme system tested and taken into use was created by a large Finnish software company X.

The theoretical overview of this thesis is based on publications on software testing. The purpose of the overview is to give a comprehensive and concrete description of the complete process of software testing. Initially, this description concentrates on software planning and draws to its close with acceptance testing.

The first executed acceptance testing of GLOW did not come out in a satisfactory way. This was due to the fact that the management programme that Lumene received was very incomplete at first and the original test planned could, therefore, not be executed. For the second acceptance testing Lumene received a much more defined management system and the tests where executed as planned. In summary, the GLOW-project was successfully completed during the second testing round, after the software company X fixed the errors found in the first programme that, at first, turned out to be imperfect.

The objectives of the project were obtained and, thus, the research questions of this thesis (in relation to the benefits and ways of testing) became answered. However, the schedule of the GLOW-project was delayed in accordance with the original plans made by Lumene and the writer. The new document and project management system GLOW will be taken into use by Lumene in May 2008 partly owing to the eventually successful acceptance testing procedure that this thesis describes in detail.

Key words: testing, software testing, acceptance testing, test design, GLOW

Sisällys

1 Johdanto

Testaus on virheiden etsimistä, ja se on erittäin merkittävässä asemassa ohjelmistojen toimivuuden ja laadun kannalta ohjelmistojen kehitys- ja toteutusprojekteissa. Testaukseen suhtaudutaan kuitenkin usein liian vähättelevästi, ja sitä suoritetaan riittämättömällä tavalla, vaikka sillä on suora yhteys ohjelmiston toimivuuteen ja asiakkaan tyytyväisyyteen. Syy testauksen puutteellisuuteen on usein ohjelmistoprojektin liian tiukka aikataulu ja henkilöstöresurssien vähyys. Ohjelmistojen ja tietojärjestelmien tehokas ja järjestelmällinen testaus ja sen suunnittelu onkin vaativa prosessi, joka sitoo yrityksen henkilöresursseja ja on aikaa vievää.

Testauksen tulisi aina olla määrätietoista ja ennalta suunniteltua, jotta testauksesta saataisiin ohjelmiston laadun kannalta irti maksimaalinen hyöty. Ohjelmiston kattavaan testaukseen tuleekin panostaa heti ohjelmiston elinkaaren alkuvaiheessa. Samoin ohjelmistoprojektin määrittelydokumentit tulee luoda erittäin huolellisesti ja selkeästi, jotta ne varmasti pitävät sisällään asiakkaan ohjelmistolta tarvitsemat ominaisuudet ja täten vastaavat sitä tarvetta, mihin ohjelmistoa on alun perin lähdetty kehittämään.

Yksi merkittävä syy testauksen määrätietoiseen suorittamiseen ja määrittelydokumentin huolelliseen laatimiseen jo ohjelmistoprojektin alkuvaiheessa on se tosiasia, että jos ohjelmiston virheet löydetään ja korjataan ohjelmistoprojektin alkuvaiheen sijasta vasta projektin loppuvaiheessa, niiden korjaaminen tulee 50 - 200 kertaa kalliimmaksi.

Testausprosessia selkiyttämään ja helpottamaan ohjelmistoyrityksillä tulisi olla selkeät toimintamallit ja -ohjeet, jotka standardoisivat koko prosessia ja samalla pienentäisivät projektin vaatimia henkilöresursseja ja kustannuksia sekä tehostaisivat prosessin läpivientiä. Koska jokainen ohjelmistoprojekti on erilainen, toimintamallien ja ohjeiden tulisi olla luotu niin, että niitä on mahdollisimman helppo soveltaa erilaisiin projekteihin. Vaikka ohjeistukset ja mallit olisivat kuinka hyvin luotuja, täytyy jokaiseen ohjelmistoprojektiin silti luoda omat testitapauksensa, joilla testataan määrättyjä ominaisuuksia. Tällaisen laatujärjestelmän ja sen sisältämien ohjeistuksien ja mallien olemassaolo antaa myös asiakkaalle luotettavamman kuvan siitä, että ohjelmistoyritys hoitaa testauksensa huolellisesti. Tietenkin pelkkä laatujärjestelmän olemassaolo ei riitä, vaan sitä tulee myös käyttää.

Kun on kyse ohjelmistoprojektista, jossa osapuolina ovat ohjelmistoyritys ja asiakasyritys, jonka henkilöstön käyttöön jokin tietojärjestelmä luodaan, niin projektiin liittyy oleellisena osana ohjelmiston hyväksymistestaus. Tämä asiakasyrityksen suunnittelema ja suoritettava hyväksymistestaus päättää onnistuneesti läpivietynä koko ohjelmistoprojektin.

Vaikka ohjelmistoprojekti olisikin päätetty onnistuneesti, on projektien sopimuksiin yleensä kirjattu, että ohjelmistoyritys antaa tuotteelleen takuun, joka kattaa projektin päätyttyä tietyn ajan sisällä löydetyt virheet ja niiden korjauksen veloituksetta.

2 Opinnäytetyön tavoitteet ja rajaukset

Tässä opinnäytetyössä käydään läpi GLOW-projektia ja prosessia, jossa asiakasyritys Lumene on tilannut dokumentin- ja projektinhallintaan tarkoitetun tietojärjestelmän ohjelmistoyritykseltä. Opinnäytetyössä perehdytään testauksen teoriaan ja Lumenen suorittamaan ohjelmiston hyväksymistestaukseen sekä tutkitaan optimaalisia tapoja suorittaa hyväksymistestaus.

Opinnäytetyön teoriaosuuden tarkoituksena on luoda kattava ja tiivis kuvaus ohjelmistojen koko testausprosessista, lähtien liikkeelle uuden ohjelmiston määrittelydokumenteista ja suunnitelmasta ja päättyen Lumenen suunnittelemaan ja suorittamaan tietojärjestelmän hyväksymistestaukseen, järjestelmän hyväksymiseen ja käyttöönottoon valmiina ohjelmana. Testauksen teoriaosuus perustuu testauksen V-mallin eri tasoihin, joita ovat moduulitestaus, integraatiotestaus, järjestelmätestaus ja hyväksymistestaus. Teoriaosuudessa käydään myös kattavasti läpi testauksen eri tasoilla käytettäviä testaustekniikoita ja -strategioita.

Opinnäytetyön loppupuolella käydään läpi itse GLOW-projektin etenemistä, Lumenelle suunnittelemiani ja luomiani testitapauksia hyväksymistestauksen suorittamiseksi, itse testien suorittamista, saatujen tulosten arviointia, projektin hyväksymistä ja projektin nykytilaa.

3 Yrityksen ja projektin taustaa

Lumene on suomalainen kosmetiikka-alan yritys, joka on perustettu vuonna 1948. Lumene Groupilla on ulkomailla tytäryhtiöitä Virossa, Latviassa, Norjassa, Venäjällä ja Yhdysvalloissa. Lumenen pääkonttori ja tuotteiden valmistuspiste sijaitsee Kauklahdessa Espoossa. Lumenen palveluksessa työskentelee noin 1000 työntekijää, joista 730 työskentelee Suomessa. Lumene Groupiin kuluu kolme divisioonaa, joita ovat Lumene, Cutrin ja Farmos. Cutrin on keskittynyt hiusalan tuotteiden valmistukseen ja myyntiin. Farmos on puolestaan keskittynyt teollisuuskäyttöön tarkoitettujen puhdistusaineiden valmistukseen ja myyntiin. Cutrinin ja Farmoksen toimitilat sijaitsevat myös Espoon Kauklahdessa Lumenen toimitilojen yhteydessä. Farmoksella on lisäksi toimitilat Turussa.

Lumenen GLOW-projekti käynnistyi tarpeesta saada paremmin hallittua ja tehostettua Lumenen tuoteprojekteja, tuotteiden uudelleen rekisteröintiä, dokumenttipohjia, toimintakäsikirjaa ja näihin liittyviä dokumentteja, jotka sijaitsevat hajanaisesti Lumenen verkkolevyillä. Samalla haluttiin ottaa käyttöön tehokkaampia ja yhtenäisempiä työnkulkuja ja käytänteitä, jotka jo itsessään tehostavat Lumenen liiketoimintaa.

Lumenella päädyttiin siihen, että näiden edellä mainittujen asioiden toteuttamiseksi Lumene Groupille hankitaan yhtenäinen dokumentin- ja projektinhallintatietojärjestelmä. Tietojärjestelmä tulisi alkuvaiheessa vain Lumenen henkilökunnan käyttöön, mutta se tultaisiin lanseeraamaan tulevaisuudessa myös Cutrinille.

4 Tietojärjestelmien kehittäminen ja testaus

Tietojärjestelmien kehittäminen on sitä suorittavalle yritykselle osa sen oman toiminnan kehittämistä. Toiminnan kehittämisen tarkoituksena on saada aikaan toimintatavan muutos, joka tehostaa yrityksen toimintaa. Toiminnan kehittämiselle voidaan antaa muun muassa seuraavanlaisia tavoitteita:

- Se auttaa yritystä saavuttamaan tavoitteensa entistä paremmin.
- Se mahdollistaa entistä vaativampien tavoitteiden asettamisen.
- Se tehostaa jo olemassa olevia toimintatapoja.
- Se tekee mahdolliseksi jonkin uuden toiminnon.

Toiminnan kehittäminen kohdistuu yrityksissä sen henkilöstöön, järjestelmiin ja toimintoihin. Henkilöstön kohdalla kehittäminen tarkoittaa tyypillisesti henkilöstön koulutusta tai työtehtävien ja -olosuhteiden uudelleenjärjestelyä. Järjestelmien kehittäminen on puolestaan seurausta yleisen teknisen osaamisen kehityksestä ja sen tarjoamista uusista mahdollisuuksista. Myös yksittäisiä toimintoja voidaan kehittää tarkastelemalla niiden suoritustapoja ja luomalla uusia käytänteitä.

Tietojenkäsittelyn ja tietojärjestelmien kehittäminen vaikuttaa yleensä kaikkiin kolmeen edellä mainittuun osa-alueeseen. Vaikka nykyisin tietojärjestelmien kehityshankkeissa pääpaino onkin teknologian kehittämisessä, silti niillä on vaikutusta myös tietojenkäsittelykäytänteisiin ja tietojenkäsittelyä suorittaviin henkilöihin.

Kehitystyön lähtökohtana voi olla tietotekniikan hyödyntäminen yrityksen prosesseissa, mutta kuitenkin kyseessä on tietojenkäsittelytoimenpiteiden kehittäminen. Tällaisen projektin tuloksena usein tietojenkäsittelyyn liittyviä tehtäviä poistuu käytöstä ja uusia tulee tilalle. Esimerkiksi mainittakoon, että kun yritykseen otetaan käyttöön uusi tietojärjestelmä, täytyy sitä käyttävät henkilöt kouluttaa ja uusia käytänteitä aletaan soveltaa.

Keskeinen osa tietojenkäsittelyn kehittämistä on jo olemassa olevien ja uusien tietojärjestelmien kehittämisellä. Atk-sanakirja määritteleekin tietojärjestelmien kehittämisen olevan uusien tietojärjestelmien laatimista tai nykyisten oleellista muuttamista.

Parempi määritelmä olisi, että tietojärjestelmien kehittäminen on liiketoiminnan tehostamista kehittämällä uusia tietojärjestelmiä tai olemassa olevien tietojärjestelmien muokkaamista vastaamaan paremmin liiketoiminnan tarpeita. (Pohjonen 2002,14-15.)

4.1 Testaus

Testaus on virheiden etsimistä ja siihen liittyvät seuraavat työvaiheet:

- testauksen suunnittelu (testaussuunnitelma ja testitapaukset)
- testiympäristön luonti
- testin suorittaminen
- tulosten tarkastelu
- virheiden korjaus

- uusintatestaus.

Edellä listattuihin työvaiheisiin ja niihin läheisesti liittyvään virheiden jäljitykseen, korjaamiseen (debugging) ja niistä syntyvään toistoon kuluu yleisesti yli puolet ohjelmistoprojektin resursseista. Tästä syystä testauksen suunnitteluun ja itse testaukseen kannattaa kiinnittää erityistä huomiota. Testauksen yhteydessä virhe on poikkeama ohjelman spesifikaatiossa (määritelmässä) eli suunnitellussa tavasta millä ohjelman tulisi toimia. Testauksen määritelmästä seuraa, että testaus ilman spesifikaatiota on mahdotonta, koska lopputuloksen oikeellisuutta ei voi todeta.

Seuraavissa luvuissa käyn läpi testauksen teoriaa ja menetelmiä.

4.2 Testauksen määritelmä ja merkitys

Normaalissa puhekielessä testaus tarkoittaa lähes mitä tahansa kokeilua. Ohjelmistojen testauksessa testaus määritellään suunnitelmalliseksi virheiden etsimiseksi ohjelmaa tai sen osaa suorittamalla. Tässä määritelmässä keskeisiä sanoja ovat suunnittelu ja etsiminen. Virheiden etsimisen ja korjaamisen lisäksi testauksen tarkoituksena on varmistaa ja parantaa järjestelmän laatua.

Myös tarkistuksia ja ohjelmakoodin staattista analysointia kutsutaan joskus testaukseksi. Tässä opinnäytetyössä termi testaus rajataan kuitenkin tarkoittamaan ohjelmaa suorittamalla tapahtuvaa testausta.

Valitettavan usein testaus tapahtuu vain kokeilemalla ohjelmaa umpimähkäisesti jollain syöttöaineistolla, ja varsinkin jos testaaja on ohjelman tekijä, niin tavoitteeksi muodostuu helposti ennemmin ohjelman toimivuuden osoittaminen kuin sen virheiden etsiminen. Useissa aiheeseen liittyvissä kirjoissa painotetaankin, että ohjelman tekijä ei ole paras mahdollinen testaaja.

Testauksen määrä (työtunnit ja testitapausten määrä) ei välttämättä ole sama kuin testauksen tehokkuus, sillä muutaman tunnin testaus huolellisesti suunnitellulla testitapausjoukolla voi hyvin johtaa parempaan tulokseen kuin päiväkausien umpimähkäinen kokeilu. Hyvä testaaminen lähteekin huolellisesta testauksen suunnittelusta ja testauksen suorittamisesta heti ohjelmiston kehityksen alkuvaiheista lähtien. Koska mitä var-

haisemmassa vaiheessa ohjelmiston virheet löydetään, niin sitä pienemmiksi jäävät niiden korjauskustannukset ja korjauksista aiheutuva työmäärä.

Koska pientäkään ohjelmaa ei voida testata täydellisesti, niin ei koskaan voida olla täysin varmoja ohjelman virheettömyydestä. Tästä johtuen voidaankin todeta, että testauksen avulla on mahdollista osoittaa, että ohjelmassa on virheitä, mutta ohjelman täyttä virheettömyyttä sillä sen sijaan ei ole mahdollista osoittaa. Käytännössä ohjelman testauksessa pystytään kattamaan vain häviävän pieni murto-osa kaikista mahdollisista tilanteista. Tämä ei kuitenkaan tarkoita sitä, että testaukseen panostaminen olisi hyödytöntä, vaan sitä, että ohjelman toimivuuteen ei kannata luottaa liikaa hyvistä testituloksista huolimatta. (Haikala & Malijärvi 2006, 281–285; Software business competence 2006.)

4.3 Täydellisen testauksen mahdottomuus

Syy, miksi ohjelman täydellinen testaus on mahdotonta, johtuu seuraavista asioista

- Testattavien syötteiden määrä on suuri
- Tulosten määrä on suuri
- Ohjelmiston läpikulkevien polkujen määrä on suuri
- Ohjelman sisäisen tilan muutokset

Kun nämä mahdollisuudet kerrotaan yhteen, saadaan tulokseksi liian paljon eri mahdollisuuksia tai erilaisia testitapauksia ohjelmiston täydelliselle testaukselle. Otetaan esimerkiksi summausohjelma. Sen syöteavaruus muodostuu kaikista mahdollisista kelvottomista syötteistä sekä kaikista sallitulla vaihteluvälillä olevista kokonaisluvuista ja niiden kombinaatioista. Ohjelman tulosavaruus koostuu virheilmoituksista ja summista. Tällaisen ohjelman kattava testaus edellyttäisi ohjelman suorittamista ainakin kaikilla kelvollisilla syötteillä. Tässä tapauksessa kelvollisia syötekombinaatioita on noin 2 potenssiin 32 kappaletta eli noin 4 miljardia.

Lisäksi ohjelman sisäinen tila tuo lisää haastetta testaukseen. Esimerkiksi suoritettaessa sama testi kahteen kertaan tietokantajärjestelmässä saatetaan saada kaksi eri tulosta. Tämä voi johtua esimerkiksi siitä, että ohjelman käyttämä tietokanta on muuttunut. Käytännössä ohjelmien sisäisten tilojen määrä on rajaton. (Haikala & Malijärvi

2006, 290–295; Software business competence 2006.)

4.4 Virheet, viat ja häiriöt

Testauksen yhteydessä virhe (error, mistake, bug, fault, defect) on poikkeama ohjelmalle asetetusta spesifikaatiosta eli määritelmästä. Yhden yleisen tulkinnan mukaan error on ihmisen toiminto (esimerkiksi ajatusvirhe), jonka tuloksena syntyy defect. Tämän tulkinnan mukaan fault on ohjelmakoodissa oleva defect, jonka tuloksena sitten failure aiheutuu.

Yleensä ottaen ohjelmissa arvioidaan olevan ohjelmoinnin jälkeen yksi virhe muutamaa kymmentä koodiriviä kohden. Jopa pitkään käytössä olleissa ohjelmissa arvioidaan yleensä olevan noin yksi virhe tuhatta koodiriviä kohden. On myös arvioitu, että noin viisi prosenttia ohjelmavirheistä on sellaisia, ettei niitä koskaan edes havaita. Syy tälle on ohjelmien täydellisen testauksen mahdottomuus, ja vaikka virheellinen kohta suoritettaisiinkin testauksen yhteydessä, se ei välttämättä johda virhetoimintoon. Virheellisen kohdan suoritus voi aiheuttaa järjestelmään vian (fault). Vika voi korjaantua itsestään järjestelmän toisen toiminnon seurauksena tai sen vaikutus voi kumoutua toisen virheen seurauksena. Pahimmassa tapauksessa vika voi aiheuttaa häiriön (failure), joka näkyy järjestelmän ulkoisessa toiminnassa.(Haikala & Malijärvi 2006, 285-286)

4.5 Ohjelman määritelmän tekeminen

Ohjelman määritelmä voi olla ohjelman kehittäjän itsensä laatima, jos kyseessä on esimerkiksi uusi markkinoille tuleva ohjelma. Määritelmä voi myös olla ohjelman kehittäjän ja asiakkaan yhdessä tekemä, jos kyseessä on vaikkapa ohjelmointiyrityksen laatima ohjelma toisen yrityksen tiettyihin tarpeisiin. Määritelmästä seuraa, että testaus ilman spesifikaatiota on mahdotonta, koska lopputuloksen oikeellisuutta ei voida muuten todentaa. Määritelmädokumentin tärkeys nousee esiin varsinkin mahdollisissa kiistatilanteissa.

Tavallisimmin testauksessa käytettäviä määritelmiä ovat toiminnallinen ja tekninen määrittely. Näiden dokumenttien tulkinnoissa voi esiintyä erimielisyyksiä. Esimerkiksi asiakkaan mielestä selvä virhe voi olla toimittajan mielestä ohjelman ominaisuus. Vir-

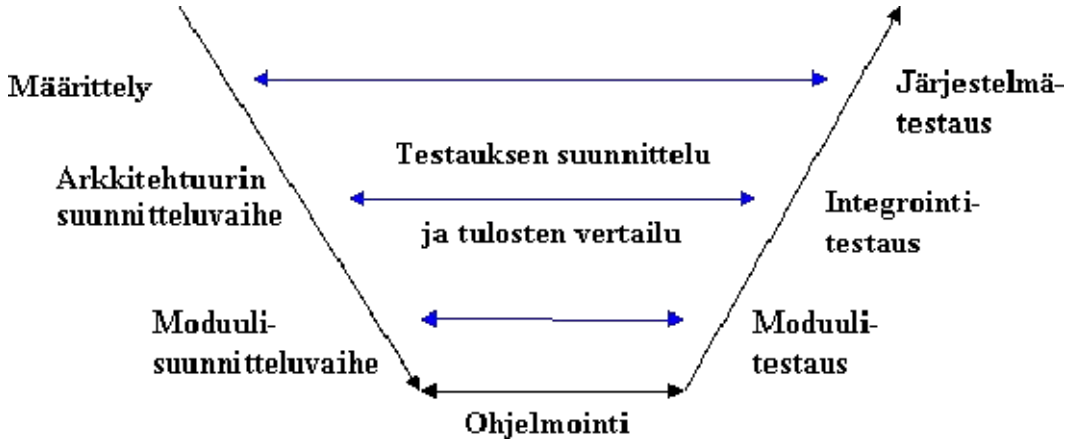
heen vakavuus voi vaihdella järjestelmän käytön kokonaan estävästä virheestä pieneen asiakasta ärsyttävään yksityiskohtaan.

Ohjelmistoprojektin sujuvuuden ja kiistatilanteiden välttämisen kannalta on erittäin tärkeää, että määritelmät tehdään mahdollisimman huolellisesti. Käytännössä kuitenkin suurin ongelma on se, että paraskin spesifikaatio on aina puutteellinen ja tulkinnanvarainen, jolloin ristiriitatilannetta ei välttämättä pystytä ratkaisemaan ohjelman spesifikaatiota tulkitsemalla. Pahimmassa tapauksessa asia joudutaan ratkaisemaan lakiteitse. Suunnitelmiin ja sopimuksiin asiakkaan ja toimittajan välillä onkin aina syytä kirjata, miten tulkinnat näiltä osin tehdään. (Haikala & Malijärvi 2006, 285–286.)

Itse olin mukana ohjelmaprojektissa, jonka määritelmää ei ollut tehty riittävän tarkasti, ja joka oli osittain melko tulkinnanvarainen. Tästä syystä jouduimmekin vääntämään kättä toimittajan kanssa ohjelmistoprojektin määritelmässä olleista asioista, tarkemmin ottaen siitä, olivatko tietyt haluamamme ominaisuudet määritelmän mukaisia. Samoin kävi myös muutamissa hyväksymistesteissämme. Meidän mieltämistämme virheistä ohjelmassa kommentoitiin toimittajan puolelta, että kyseessä on ohjelman ominaisuus, ei virhe.

4.6 Testauksen tasot

Yleisesti testaus ja sen suunnittelu etenee niin sanotun V-mallin mukaisesti, testaustasoa vastaavalla suunnittelutasolla kuten alla olevassa kuviossa 1 on osoitettu:



Kuvio 1. Testauksen V-malli.

Erilaisia testaustasoja ovat moduulitestaus (module testing, yksikkötestaus), integrointitestaus (intergration testing) ja järjestelmätestaus (system testing). Ilkka Haikala kirjoittaa kirjassaan Ohjelmistotuotanto, että järjestelmätestausta voi seurata erillinen kenttätestaus (field testing) ja/tai hyväksymistestaus. Mielestäni kuitenkin, jos kyseessä on toimittaja ja asiakas –projekti, niin kenttätestaus ja/tai hyväksymistestaus on välttämätön. Projektissa, jossa itse olin mukana, olisimme ilman hyväksymistestausta saaneet täysin keskeneräisen ja Lumenen tarpeita vastaamattoman tietojärjestelmän, jolle ei olisi juurikaan ollut käyttöä.

4.6.1 Moduulitestaus

Moduulitestauksessa testattavana on yksittäinen moduuli. Moduulilla tarkoitetaan ohjelmasta erotettavissa olevaa loogista kokonaisuutta, joka yleensä koostuu noin 100–1000 koodirivistä. Tyypillinen moduuli sisältää tietomäärittelyjä ja joukon kyseistä tietoa käsitteleviä funktioita. Moduulin toimintaa verrataan moduulisuunnittelun ja arkkitehtuurisuunnittelun tuloksiin, tavallisimmin tekniseen määrittelydokumenttiin. Testauksen suorittaa yleensä moduulin toteuttaja. Testit ovat usein mustalaatikkotestejä. Jos prosessit ovat vaikeita, myös lasilaatikkotestausta voidaan käyttää.

Testauksen suorittamiseksi voidaan joutua toteuttamaan testipetejä (test bed), joilla moduulin toimivuutta voidaan kokeilla. Testipetiin kuuluu yleensä ohjelman tulevaa ympäristöä simuloivia osia, tavallisesti testiajureita (test drivers) ja tynkämoduuleita (test stubs). Testiajurit mahdollistavat moduulin toteuttamien palveluiden kutsumisen ja tulosten tarkastelun. Tynkämoduulit puolestaan korvaavat testattavan moduulin tarvitsemat muut moduulit, jos niitä ei ole vielä olemassa.

Testeissä keskitytään sisäiseen logiikkaan ja tietorakenteisiin. Jos moduuli suorittaa vain yhden funktion, testitapausten määrä vähenee ja virheet ovat helpommin löydettävissä. Moduulitestauksen yhteydessä pyritään myös löytämään ristiriitoja testattavien moduulien määrittelyjen ja toiminnan välillä. Moduulitestaus ei sovellu ainoaksi testausmenetelmäksi, koska testimoduulit ovat ainoastaan pieniä ohjelman osia eivätkä laajoja kokonaisuuksia. (Haikala & Malijärvi 2006, 287)

4.6.2 Integrointitestaus

Integrointitestauksessa yhdistellään yhteen moduuleita tai moduuliryhmiä (osajärjestelmiä). Integraatiotestaus alkaa siitä, että moduulitestauksessa erikseen testattuja moduuleita ryhdytään testaamaan toistensa kanssa. Integrointitestaus loppuu siihen kun koko ohjelman kaikki moduulit ovat valmiita, ja niiden yhteistoiminta on kokonaisuudessaan testattu. Integrointitestauksen painopiste onkin moduulien välisten rajapintojen toimivuuden tutkimisessa: tavoitteena on löytää virheitä ohjelman rajapinnoista ja niiden keskinäisestä yhteistoiminnasta. Virheitä saattaa löytyä, vaikka yksittäiset moduulit olisivatkin osoittautuneet toimiviksi.

Testien tuloksia verrataan usein ohjelman määrittelydokumenttiin ja tämä vertailu etenee usein rinnakkain moduulitestauksen kanssa. Ohjelman määrittelydokumenttia ja testien tuloksia onkin usein tarpeetonta tarkastella erillään moduulitestauksesta. Testauksen kattavuuden kannalta moduulitestauksessa on tosin helpompi saavuttaa kunnollinen testikattavuus, koska testattavan kokonaisuuden kasvaessa on vaikea käydä kaikkea koodia läpi.

Integrointitestaus voidaan toteuttaa useammalla lähestymistavalla. Yleisin tapa on kokoava (bottom-up), jossa testaus etenee alimman tason moduuleista ylöspäin. Tämän tavan vahvuus on siinä, että kun virhe ilmaantuu, niin ainakin teoriassa vian pitäisi löytyä viimeksi testauskokonaisuuteen lisätystä moduulista. Jäsentävässä eli osittavassa (top down) tavassa testauksen etenemissuunta on päinvastainen. On olemassa lisäksi vielä kaksi tapaa (sandwich ja big-bang), joista en kuitenkaan lukemassani suomalaisessa kirjallisuudessa löytänyt mainintoja. Sandwich -mallissa yhdistellään bottom-up ja top down testausta kunnes koko ohjelma on käyty läpi. Big-bang -mallissa kaikki moduulit integroidaan kerralla yhteen ennen testauksen aloittamista. (Haikala & Mali-järvi 2006, 287-288; Tamres 2002, 221.)

4.6.3 Järjestelmätestaus

Järjestelmätestauksessa (system testing) tarkastelun kohteena on koko järjestelmä ja sen tuloksia verrataan määrittely- ja asiakasdokumentaatioon. Järjestelmätestauksen kohteena ovat toiminnalliset ja ei-toiminnalliset ominaisuudet. Kun käytössä on täysin kokonainen ja ainakin suurimmilta osin toimiva kokonaisuus, päästään testaamaan ominaisuuksia, joita ei aikaisemmin ollut mahdollista tai järkevää testata. Testauksen

tarkoituksena on siis löytää vikoja, joita ei voida moduuli- eikä integrointitestauksessa löytää.

Järjestelmätestaus koostuu useista erityyppisistä testausosioista. Eri osien tarkoituksena on virheiden löytämisen lisäksi varmistaa, että järjestelmä vastaa asiakkaan sille asettamia vaatimuksia. Näitä osia ovat muun muassa:

- Toiminnallisuustestauksen (functional testing) tarkoituksena on varmistaa, että kaikki järjestelmän toiminnot toimivat määrittelyjen mukaisesti.
- Yhteensopivuustestauksen (compatibility testing) tarkoituksena on selvittää, kuinka hyvin järjestelmä toimii määrätynlaisessa ympäristössä, kuten vaikkapa tietynlaisessa verkkoympäristössä tai käyttöjärjestelmässä.
- Konfigurointitestauksen (configuration testing) tarkoituksena on varmistaa, että ohjelma toimii eri ohjelmien, tietokoneiden ja niiden eri komponenttien kanssa, joita sen tulee tukea.
- Asennustestauksen (installation testing) tarkoituksena on testata, että ohjelman asennus, päivittäminen ja poistaminen toimivat määrittelyjen mukaisesti.
- Kuormitustestauksessa (load testing) järjestelmää käytetään niin kuin sen käyttäjä sitä käyttäisi ja varmistetaan, että sen käyttö on sulavaa. Suorituskykyä voidaan testata myös siten, että järjestelmää kuormitetaan erilaisilla lasteilla. Tällöin tarkoituksena on selvittää, minkälaisesta kuormituksesta järjestelmä selviää pysyen samalla käyttökelpoisena.
- Suorituskyvyn testauksessa (performance testing) mitataan, kuinka kauan eri työtehtävät kestävät järjestelmän normaalikuormituksen ja huippukuormituksen alaisena.
- Tietoturvatestauksessa (security testing) tavoitteena on varmistaa, että järjestelmään rakennettu suojamekanismi suojaa laittomilta tunkeutumisilta, kopioinnilta ja muokkaamiselta.
- Palautumistestauksessa (recovery testing) varmistetaan, että järjestelmä pystytään palauttamaan toimivaan tilaan sen kaatumisen, sisäisen ongelman tai palvelimen fyysisen vian jälkeen.
- Huollettavuustestauksessa (serviceability testing) varmistetaan, että järjestelmänsisäiset ylläpitoinformaatiot, kuten käyttäjistä jäävät jäljet ja diagnostiikkaviestit, toimivat.
- Käytettävyystestauksella (usability testing) halutaan arvioida järjestelmän hyödyllisyys, sopivuus ja helppokäyttöisyys. Tavoitteena on samalla tunnistaa oh-

jelman ongelmakohdat käyttäjän kannalta. Käytettävyystestaus on pääasiassa käyttöliittymän testausta. Siinä halutaan varmistaa, että järjestelmä on mahdollisimman käyttäjäystävällinen. Tämä tarkoittaa sitä, että käyttäjä pystyy selviämään mahdollisimman hyvin tehtävistä, joiden suorittamiseksi järjestelmää ollaan rakentamassa. Käytettävyystestausta voidaan yleensä suorittaa jo aikaisemmassa vaiheessa esimerkiksi järjestelmän ohjelmoijan toimesta. Yleensä käytettävyystestiä varten valitaan pieni joukko tulevia käyttäjiä ja heitä seurataan valvotussa testiympäristössä. Käytettävyystestauksen lisäksi tai sen sijaan voidaan käyttää hyväksi myös ohjelmiston arviointiin erikoistuneita ammattilaisia.

Edellä listatuilla testeillä ja niiden tuloksilla ohjelman toimittaja arvioi, toimiiko järjestelmä määritysten mukaisesti. Tällöin punnitaan, onko järjestelmä valmis julkaistavaksi tai toimitettavaksi asiakkaan hyväksymistestausta varten. Kun suoritetaan järjestelmätestejä, tulee testiympäristön olla mahdollisimman lähellä ohjelman tuotantoympäristöä: muuten ei pystytä arvioimaan lopullista tuotetta. Järjestelmätestaukseen voi lisäksi liittyä erillinen kenttätestaus (alfa testing, jonka asiakas suorittaa toimittajan tiloissa) ja hyväksymistestaus (beta testing, jonka asiakas suorittaa itsenäisesti omissa tiloissaan). Valinta kenttätestauksen ja hyväksymistestauksen välillä riippuu tietenkin siitä, tehdäänkö järjestelmä yritykselle itselleen firman omasta toimesta vai myydäänkö järjestelmä ulkopuoliselle asiakkaalle. Nämä testit suoritetaan varsinaisen järjestelmätestauksen jälkeen. (Haikala & Malijärvi 2006, 288; Tamres 2002, 222–223.)

4.6.4 Regressiotestaus

Regressiotestaus on järjestelmän uudelleentestausta virheiden löytymisen jälkeen. Voidaan sanoa, että mikään järjestelmä ei ole ensimmäisellä kerralla niin täydellinen, ettei regressiotestausta tarvitsisi tehdä. Regressiotestauksessa järjestelmätestaus tulisi suorittaa mieluiten kokonaan uusiksi, koska usein virheiden korjaus ja/tai uuden ominaisuuden lisäys aiheuttaa uuden virheen järjestelmän aikaisemmin normaalisti toimineessa osassa. Regressiotestauksen päämääränä onkin osoittaa, että aikaisemmin toimineet ominaisuudet toimivat myös järjestelmän uudessa versiossa. Uuden ominaisuuden lisääminen aiheuttaa usein sen, että aikaisempia järjestelmätestejä joudutaan päivittämään uuden ominaisuuden takia. Tämä siksi, että suorittamalla joukko vanhoja testejä ei voida todeta, että järjestelmä olisi virheetön, jos siihen lisättyjä uusia ominaisuuksia ei ole testattu.

Regressiotestausta tehdään kaikilla testauksen V-mallin tasoilla. Testaustasojen erona on se, että mitä korkeammalla V-mallin tasolla ollaan, sitä kalliimmaksi virheiden korjaus tulee. Ideaalissa tilanteessa regressiotestaus tehdään aina kun järjestelmään tulee muutos. Tämä tosin ei aina ole mahdollista, varsinkin jos testausta ei saada automatisoitua, sillä testauksen kustannukset voivat nousta erittäin kalliiksi. (Haikala & Malijärvi 2006, 288; Tamres 2002, 223.)

4.6.5 Hyväksymistestaus

Hyväksymistestaus suoritetaan yleensä asiakkaan toimesta asiakkaan tiloissa todellisessa tuotantoympäristössä. Sen tarkoituksena on varmistaa, että luotu järjestelmä on oikeasti sellainen, mitä asiakas on halunnut, ja että järjestelmä tukee asiakkaan haluamia toimintoja. Hyväksymistestausta kutsutaan myös beta-testaukseksi. Sitä saattaa edeltää alfa-testaus toimittajan tiloissa, mikä simuloi tulevaa tuotantoympäristöä. Kirjallisuudessa on ristiriitaisia väitteitä alfa-testauksen pakollisuudesta. Ainakaan Lumenen projektissa tällaista testiä ei ollut.

Hyväksymistestauksen testaajina on yleensä valittu joukko järjestelmän todellisia käyttäjiä. Hyväksymistestausta varten suunnitellaan ja luodaan testitapauksia, joiden tarkoituksena on simuloida todellisia järjestelmän käytön tilanteita, joita varten uusi järjestelmä on alun perin haluttu. Testeistä luodaan mahdollisesti melko tarkatkin paperiversiot, joita testaajat sitten käyvät läpi kohta kohdalta ja merkitsevät ylös mahdolliset virheet, puutteet, huomautukset ja kehitysehdotukset. Testit pohjautuvat yleensä projektin määrittelydokumentteihin.

Testien tulosten pohjalta luodaan testiraportti, joka käydään läpi toimittajan ja asiakakkaan edustajien kesken palaverissa. Tässä palaverissa ovat läsnä ainakin asiakkaan ja toimittajan projektipäälliköt. Erillisessä projektin päätöspalaverissa on lisäksi mukana projektin ohjausryhmä ja siinä päätetään jatketaanko vai päätetäänkö projekti.

Onnistunut hyväksymistestaus on kriteerinä projektin lopettamiselle eli laskun maksamiselle. Hyväksymistestausta saattaa edeltää ohjelmaan liittyvä henkilöstökoulutus, varsinkin jos ohjelma ei ole käyttäjille ennestään tuttu. Jos testaus ei mene odotusten mukaisesti ja korjattavaa on enemmänkin, suoritetaan hyväksymistestaus uudelleen

kun järjestelmän toimittaja on saanut korjattua havaitut puutteet. Näitä toimintoja toistetaan kunnes järjestelmä on asiakkaan määritelmien mukainen.

Pahimmassa tapauksessa voidaan joutua kääntymään lain puoleen, jos asiakkaan ja toimittajan yhteisymmärrys halutusta järjestelmästä ei kohtaa tai jos järjestelmä on kirjattu määrittelydokumentteihin liian ympäripyöreästi. Esimerkiksi ohjelman toimittaja saattaa pitää jotain korjaustyötä muutospyyntönä ja asiakas taas ajatella sen olevan määrittelyn mukainen ominaisuus. Mahdollisen riitatilanteen varalta voidaan sopia kolmas ulkopuolinen taho, joka yrittää ratkaista asian puolueettomasti ennen asian oikeuteen viemistä. Riitatilanteiden estämiseksi projektin määrittelyvaiheen määrittelydokumentit tulisi tehdä erittäin huolellisesti ja mahdollisimman seikkaperäisesti.

Hyväksymistestaus voidaan myös suorittaa kolmannen osapuolen toimesta, joka on erikoistunut testaukseen. Kolmannen osapuolen käytön etuna on se, että tieto kolmannen osapuolen käytöstä pakottaa entisestään järjestelmän toimittajaa suorittamaan omat testinsä huolellisesti, jolloin yleensä jo ensimmäinen versio järjestelmästä on virheettömämpi. Samalla asiakasyrityksen omia resursseja voidaan hyödyntää muihin tehtäviin.

Ohjelmistoprojekteille sovitaan yleensä tietyn ajanjakson pituinen takuuaika alkaen siitä kun projektin asiakas on hyväksynyt toimittajan tekemän järjestelmän. Yleensä takuuaika tarkoittaa sitä, että takuuajan aikana havaitut virheet korjataan järjestelmän toimittajan puolesta ilmaiseksi. Esimerkiksi Lumenen Sharepoint -ohjelmalle ohjelmistoyritys X antoi puolen vuoden takuun. Yleensä ohjelmistoyritys X antaa vuoden takuun, mutta nyt kyseessä oli heidän projektipäällikkönsä mukaan ensimmäinen kerta kun ohjelmiston toimittaja räätälöi Sharepointin projektin- ja dokumentinhallintaan. Rivien välistä pystyi lukemaan, että juuri tämä asia oli syy takuun lyhyydelle. Tämä asia kasvatti GLOW -projektin epäonnistumisen riskiä. (Tamres 2002, 223-224; Cybercom Plenware 2008.)

4.7 Testauksen laatu ja riittävyys

Johtuen siitä, että ohjelmiston täydellinen testaaminen on lähes mahdotonta, niin myös tarvittavan testauksen määrää on vaikea arvioida. Varsinkin järjestelmätestauksessa testausta voidaan jatkaa kunnes aika ja/tai rahat loppuvat. Tuotekehitystyössä testauk-

sen lopettaminen on usein kompromissi tuotteessa olevien vikojen aiheuttamien kustannusten ja markkinoilta myöhästymisen aiheuttaman tuoton menetyksen välillä.

Testauksen ongelmana on se, että jos kaikkea yritetään testata, niin kustannukset nousevat dramaattisesti ja ei-löydettyjen virheiden määrä laskee pisteeseen, josta ei ole enää kustannustehokasta jatkaa. Jos testaus jätetään liian lyhyeksi tai testitapaukset ovat huonoja, niin kustannukset ovat alhaiset, mutta virheitä jää paljon huomaamatta. Tavoitteena on päästä optimaaliseen testausmäärään, jolloin ei testata liikaa eikä liian vähän.

Testauksen lopettamiselle tulisikin aina asettaa hyväksymiskriteerit, jotka määritellään testaussuunnitelmassa. Muuten on vaikea sanoa, milloin testaus voidaan lopettaa ja siirtää järjestelmä eteenpäin.

Järjestelmätestauksessa kriteeri voi liittyä esimerkiksi kumulatiiviseen löydettyjen virheiden määrään; kun virhekäyrä tasaantuu, testaus voidaan lopettaa. Tarvittavaa testauksen määrää voidaan koettaa arvioida mutkikkuusmitoilla ja testauksen riittävyyttä puolestaan niin kutsutuilla kattavuusmitoilla.

Mutkikkuusmitoilla pyritään päättelemään ohjelmamoduulin monimutkaisuutta koodin rakennetta analysoimalla. Mutkikkuusmitan avulla voidaan yrittää paikantaa ohjelmistossa paljon testausta vaativat moduulit. Tunnetuimmat mutkikkuusmitat ovat Halsteadin mitta ja McCaben mitta. Kirjallisuudessa näihin mittoihin suhtaudutaan kuitenkin melko varautuneesti.

Kattavuusmitoilla voidaan yrittää varmistaa, että testiaineisto aiheuttaa testattavan ohjelman kaikkien osien kattavan suorittamisen. Kattavuusmittaa voidaan myös käyttää todisteena siitä, että testausta on suoritettu riittävästi. Kattavuusmittoja ovat lausekattavuus, päätöskattavuus, ehtokattavuus, moniehtokattavuus ja polkukattavuus.

- Lausekattavuudella tarkoitetaan suoritettuja lauseita jaettuna ohjelman sisältämillä lauseilla. Vaikka 100 % lausekattavuus saattaa kuulostaa itsestään selvältä vaatimukselta, niin käytännössä siihen on mahdotonta päästä. Käytäntö onkin osoittanut, että jos ohjelmoijaa pyydetään testaamaan pienikin koodimoduuli lausekattavasti, niin todellinen kattavuus harvoin ylittää 90 %.

- Päätöskattavuudella tarkoitetaan suoritettuja haaroja jaettuna mahdollisten haarojen määrällä, eli jokainen ehto saa testattaessa molemmat arvonsa (tosi/epätosi).
- Ehtokattavuudella tarkoitetaan, että päätöksen kaikkien ehtojen on saatava kaikki arvonsa.
- Moniehtokattavuudessa testaus suoritetaan kaikkien ehtojen kaikilla kombinaatioilla.
- Polkukattavuudessa ohjelmaa tarkastellaan suunnattuna verkkona, jonka solmuina ovat ohjelman haarautumiskohdat. Polkukattavuudessa pyritään kattamaan mahdollisimman monia erilaisia suorituspolkuja ohjelman läpi.

On olemassa myös testikattavuusanalysaattoreita. Nämä ovat työkaluja, jotka mittaavat testin kattavuutta jollain ylläkuvatulla kattavuusmitalla. Samalla työkalulla voi usein mitata ohjelmarivien suorituskertojen lukumääriä ja prosessorin ajankäyttöä.

Testauksen kattavuuden saaminen korkeaksi on helpointa moduulitestauksessa, vaikkakin siinäkään ei yleensä päästä täyteen sataan prosenttiin asti. Sääntönä voidaan pitää, että mitä korkeammalle testauksen V-mallissa edetään, sitä pienemmäksi kattavuuden prosentuaalinen määrä laskee. Esimerkiksi asiakkaalle toimitettua versiota testatessa (ns. julkaisutestaus) testikattavuus harvoin ylittää 50 prosentin rajan. (Haikala & Malijärvi 2006, 290-295; Patton 2006, 38-40; Black 2005, 298-300.)

4.8 Testauksen työkalut

Kuten edellä mainittiin, testaukseen on olemassa ohjelmallisia työkaluja, joiden tarkoituksena on automatisoida testejä. Varsinkin regressiotestauksen tulisi olla mahdollisimman automatisoitua. Testien automatisoinnin hyötyjä pitkällä tähtäimellä ovat:

- testisarjojen suorittamiseen kuluvan ajan pienentyminen
- testien suorittamisessa testaajien käytön tarpeen pienentyminen
- satojen eri käyttäjien simuloinnin mahdollistaminen
- ihmistestaajien inhimillisten virheiden, kuten näppäilyvirheiden, karsiutuminen

Testityökalujen kaksi päätoimea ovat testien suorittamien ja niistä saatujen tulosten arviointi. Näitä työkaluja ovat esimerkiksi testipetigeneraattorit, testitapausgeneraattorit, vertailijat ja testikattavuustyökalut.

Testipetigeneraattori (test bed generator) luo testattavalle ohjelmamoduulille testipedin, jolle luodaan testikuvauskielellä ajettava testi. Kielellä voidaan määrätä halutut testitulokset, jolloin testitulosten tarkastelu on automatisoitavissa.

Järjestelmätestauksessa taas käytetään ns. capture and playback -työkaluja, jotka voivat tallentaa testaajan näppäinlyönnit, hiiren liikkeet ja järjestelmän antamat syötteet. Ohjelma tallentaa nämä syötteet scriptiin, joka voidaan ajaa myöhemmin uudelleen.

Testitapausgeneraattoreiden käyttö ei ole kovin yleistä ja se on yleensä lähes mahdotonta, koska se edellyttää testattavan ohjelman formaalia spesifiointia. Joissain erikoistapauksissa sen käyttö on kuitenkin mahdollista. Esimerkiksi, jos monimutkainen käyttöliittymä on spesifioitu tilakaaviona, testiaineisto voidaan generoida tilakaavion avulla ja myös testitulosten tarkastelu voidaan automatisoida.

Vertailijaohjelmilla (comparators) etsitään eroja ohjelman antamien tulosten välillä ja erojen perusteella päätellään, ovatko testit menneet läpi onnistuneesti. Testaaja pystyy usein lisäksi määrittelemään tietyn osan testituloksista, jonka hän haluaa jättää huomioimatta. Esimerkiksi päivämäärän ja ajan huomiotta jättäminen on järkevää, koska tällöin testin lopputulos on silti positiivinen, jos testi on muilta osiltaan suoritettu onnistuneesti (paitsi silloin jos kellon aika olisi muuttunut).

Testikattavuusanalysaattorityökalut käyttävät hyväkseen jotain olemassa olevasta kattavuusmitoista. Toimintaperiaatteiltaan ne ovat yleensä esiprosessoreita, jotka instrumentoivat moduulin koodin siten, että ne mittaavat testin kattavuuden sitä suoritettaessa. Samoilla työkaluilla pystytään yleensä myös mittaamaan ohjelmarivien suorituskertojen lukumääriä ja prosessoriajan käyttöä.

Ohjelmakoodin instrumentoinnin haittapuolena on se, että se kasvattaa ohjelmakoodin kokoa ja hidastaa suoritusta. Tämän takia se rajoittaa testikattavuusanalysaattoreiden käyttöä järjestelmätestauksessa ja reaaliaikajärjestelmien yhteydessä.

Testityökalujen käyttö ei myöskään ole ilmaista ja vaatii työtä niistä saatavan hyödyn saavuttamiseksi. Pelkkä ohjelmien hankkiminen ei automatisoi testausprosessia onnistuneesti, vaan ne vaativat hyvän testaussuunnitelman. Hyödyllisen automaattisen testausympäristön luominen vie aikaa ja siitä saatavan hyödyn saaminen edellyttää useiden automatisoitujen testien suorittamista. Tämä sen takia, että automatisoitujen testien luominen vaatii testaustyökalujen hyvää tuntemusta ja niiden ohjelmointia sekä niitä käyttävien testaajien kouluttamista. Lisäksi uusia testejä joudutaan aina luomaan. Tulee myös ottaa huomioon, että testityökalut ovat ohjelmia, eli nekään eivät ole sataprosenttisesti virheettömiä.

Testaustyökalut eivät ole myöskään ongelmattomia, sillä niiden käyttö saattaa vaikuttaa ohjelman toimintaan. Esimerkiksi sulautetuissa järjestelmissä on tavallista, että kehityslaitteistossa toimiva ohjelma ei jostain syystä toimi kohdelaitteessa tai että testaustyökalun ohjelmaan liittämisen jälkeen ohjelmassa esiintynyttä ongelmaa ei saada enää esiintymään. Täytyy myös muistaa, että vaikka jokin automatisoitu testiohjelma ei löytäisikään ohjelmasta virhettä, niin se ei takaa, että ohjelma olisi virheetön. (Tamres 2002, 225–226;Haikala & Malijärvi 2006, 294–296.)

4.9 Testausstrategiat ja testitapausten valinta

Kaksi yleisintä testitapausten luomiseen käytettävää testaustekniikkaa tai lähestymistapaa ovat mustalaatikkotestaus (black-box, functional testing) ja lasilaatikkotestaus (glass/white-box testing). Tässä opinnäytetyössä käydään läpi myös harmaalaatikkotestauksen (grey-box testing), tutkivan testauksen, positiivisen testauksen ja negatiivisen testauksen lähestymistapa. Nämä kaikki neljä lähestymistapaa ovat testausstrategioita eivätkä teknisiä tai analyyttisiä menetelmiä.

Mustalaatikkotestien testitapaukset johdetaan asiakkaan kanssa tehdystä ohjelmistoprojektin määrittelydokumentaatiosta kiinnittämättä huomiota ohjelman sisäiseen tekniseen toteutukseen. Mustalaatikkotestauksessa tarkastellaan ohjelmalle annettuja syötteitä ja niistä palautuvien tulosten oikeellisuutta. Mustalaatikkotestauksessa tulee ottaa huomioon, että vaikka sen avulla voidaan löytää virheitä ja virheellisiä komponentteja, se ei välttämättä kerro virheiden syitä. Tätä varten virheiden syyt voidaan joutua selvittämään lasilaatikkotestauksella.

Mustalaatikkotestauksessa testitapausten valinta perustuu syöteavaruuden jakamiseen ekvivalenssiluokkiin (equivalence partitioning). Oletuksena on, että jos järjestelmä toimii yhdellä ekvivalenssiluokan edustajalla, niin se toimii kaikilla saman ekvivalenssiluokan edustajilla. Otetaan esimerkiksi ohjelma, joka pyytää syöttämään kokonaisluvun yhdestä kymmeneen. Tämän ohjelman ekvivalenssi luokat voisivat olla seuraavanlaiset:

1. Epäkelvot syötteet (ei kokonaisluvut ja kaikki muut näppäimistön ei numeraaliset merkit ja niiden yhdistelmät)
2. Liian pienet tai suuret kokonaisluvut
3. Kelvolliset syötteet

Tyypillisten ekvivalenssiluokkien edustajien lisäksi on hyvä ottaa testitapauksiksi luokkien rajoilla olevia mahdollisimman suuria, pieniä, pitkiä ja lyhyitä edustajia. Esimerkkitapauksessa näitä olisivat luvut 0, 1, 10 ja 11. Tällaista testitapausten valintaa kutsutaan raja-arvoanalyysiksi.

Lasilaatikkotestauksessa testitapausten valintaa käytetään hyväksi tietoa ohjelman sisäisestä teknisestä toteutuksesta. Sillä pyritäänkin testaamaan ohjelmiston sisäisten komponenttien virheettömyyttä. Harmaalaatikkotestauksessa taas käytetään hyväksi tietoa ohjelman toteutusperiaatteista. Esimerkiksi harmaalaatikkotestauksessa käytetäisiin hyväksi tietoa siitä, että ohjelma lukee lukuja merkki kerrallaan ja muuttaa ne tässä yhteydessä 32-bittisiksi kokonaisluvuiksi, kun taas lasilaatikkotestauksen testitapaukset perustuvat ohjelmakoodiin tutustumiseen.

Sekä mustalaatikko- ja lasilaatikkotestauksessa testitapaukset tulee suunnitella siten, että ne sisältävät sekä oikeita että virheellisiä syöttöarvoja. Mitä korkeammalle testauksen V-mallissa edetään, sitä enemmän testaus siirtyy lasilaatikkotestauksesta mustalaatikkotestaukseksi. (Tamres 2002, 220; Haikala & Malijärvi 2006, 289; Pohjonen 2002, 36; Kit 2005, 79–80.)

Positiivisen testauksen tarkoituksena on osoittaa, että ohjelma vastaa sille asetettuja vaatimuksia. Positiivisen testauksen testitapauksilla pyritään todistamaan, että ohjelma tekee sen, mitä sen on tarkoitus tehdä, eli ohjelma toimii odotusten mukaisesti. Näitä testejä suoritetaan sallituilla syötteillä. Negatiivisen testauksen tarkoituksena on, päinvastoin, todistaa, että ohjelma toimii niin kuin sen ei ole tarkoitus toimia. Negatiivisten

testien testitapaukset ovat yleensä monimutkaisempia kuin positiivisten testien. (Stenberg 2007; Tapola 2004.)

Testejä luotaessa tai testausammattilaisen testatessa ohjelmaa voidaan käyttää hyväksi myös virheen arvausta (error guessing), joka ei kuitenkaan ole varsinainen testaustekniikka, vaan lähinnä asiakohtainen toiminta- tai lähestymistapa. Perusideana on luoda lista mahdollisista virheistä ja virhetilanteista sekä luoda testitapaukset hyväksikäyttäen tätä listaa. Kokenut testaaja pystyy ”arvaamaan” yleisesti tunnettuja kohtia, joissa virheitä voi esiintyä ja suurella todennäköisyydellä virheitä löytyy samoista paikoista kuin ennenkin. Esimerkiksi negatiiviset numerot ovat yksi tunnetuista virhetilanteiden luojista. (Kit 2005, 87.)

Tutkiva testaus (exploratory testing) on samanaikaista oppimista, testien suunnittelua ja testien suorittamista. Toisin sanoen tutkiva testaus on mitä tahansa testausta silloin kun testaaja aktiivisesti kontrolloi testien suunnittelua samalla kun testejä suoritetaan ja käyttää hyväkseen niistä saatua tietoa suunnitellakseen uusia ja parempia testejä. Tutkivassa testauksessa voidaan käyttää hyväksi esimerkiksi määrittelydokumentteja tai standardeja, joita ohjelman tulee täyttää.

Tutkivassa testauksessa testaaja ei kysy mitä testejä tulee suorittaa, vaan mikä on paras testi, minkä hän voi suorittaa juuri nyt. Tutkivan testauksen hyöty on siinä, että niitä voidaan optimoida koko testausprosessin ajan, kun taas käytettäessä valmiiksi kirjoitettuja testejä itse testit eivät muutu ja niistä tulee tehottomampia ajan myötä.

Tutkiva testaus ei missään tapauksessa vastusta suunniteltuja kirjallisia testejä ja suunnitellut kirjalliset testit ovatkin joissain tapauksissa tehokkaampia. Esimerkiksi, jos testeistä saatavan palautteen eteenpäin meneminen on hidasta, se menettää tehonsa. Paras tapa olisi käyttää valmiiksi kirjattuja testejä ja tutkivaa testausta yhdessä ja oikeassa suhteessa.

Kaikki ohjelmistotestaajat suorittavat tutkivaa testausta ainakin jollain tasolla, vaikka he eivät loisikaan testejä. Se on vahva lähestymistapa, joka on silti laajasti väärinkäsitetty ja epäkunnioitettu: monetkaan eivät käytä tätä lähestymistapaa, eikä aihetta edes käsitellä alan kirjallisuudessa muutamaa poikkeusta lukuunottamatta.

Tutkiva testaus sopii mihin tahansa sellaiseen tilanteeseen, jossa ei selvästi tiedetä mitä testejä tulisi seuraavaksi suorittaa. Menetelmänä tutkiva testaus on paikallaan myös kun:

- Täytyy saada nopeaa palautetta uudesta ohjelmasta tai ominaisuudesta
- Tuotteen toiminta pitää opetella nopeasti
- Testaus on suoritettu tarkasti laadituilla testeillä ja sitä halutaan monipuolistaa
- Halutaan löytää suurin yksittäinen virhe mahdollisimman lyhyessä ajassa
- Halutaan tarkistaa toisen testaajan työ suorittamalla lyhyt itsenäinen tutkimus
- Halutaan tutkia ja eristää tietty yksittäinen virhe
- Halutaan tietää tietyn riskin tila, jotta voidaan arvioida tarvittavien testitapausten määrää sillä alueella

Edellä lueteltujen tapausten lisäksi tutkiva testaus sopii myös tilanteisiin, joissa testausta ei ole kokonaan ennalta suunniteltu, esimerkiksi jos:

- Halutaan improvisoida valmiiksi luotuja testitapauksia
- Tulkitaan epäselviä testiohjeistuksia
- Tehdessä tuoteanalyysiä ja testisuunnitelmaa
- Halutessa parantaa jo olemassa olevia testejä
- Uusia testitapauksia kirjoitettaessa
- Tehdessä regressiotestausta perustuen vanhoihin virheilmoituksiin
- Testauksen perustuessa ohjelman käyttäjämanuaaliin ja siinä oleviin väittämiin

Tutkivan testauksen vahvuus on informaation kulussa eli siinä, kuinka se kulkee takaisin testien suorituksesta testien uudelleen suunnitteluun. (Bach 2003.)

4.10 Riskianalyysi

Riskianalyysillä tarkoitetaan ohjelmistoprojektin mahdollisten epäonnistumisvaiheiden tunnistamista ja ennaltaehkäisyä. Riskianalyysin käytöllä heti projektin alkuvaiheessa pyritään tunnistamaan ja tuomaan esiin ohjelman toiminnan, käyttäjävaatimusten ja asiakkaan liiketoiminnan kannalta projektin ja ohjelman kriittisimmät kohdat. Riskianalyysin tekeminen auttaa ohjelmiston ja siihen liittyvien testien suunnittelussa ja tekemi-

sessä, koska sen avulla pystytään kiinnittämään erityistä huomiota ohjelman kriittisiin kohtiin.

Riskianalyysissä riski sisältää kaksi komponenttia:

1. Tapahtuman todennäköisyys
2. Epäonnistumisen vaikutus tai vakavuus

Riskianalyysissä potentiaaliset riskit listataan. Sen jälkeen jokaiselle riskille annetaan todennäköisyys ja vakavuusarvo esimerkiksi välillä 1-10. Tämän jälkeen jokaiselle riskille annetut kaksi arvoa kerrotaan keskenään, josta saadaan riskille altistumisen arvo (risk exposure). Mitä korkeampi tämä arvo on, sitä vakavampi riski on kyseessä. Mitä korkeamman arvon jokin riski on saanut, sitä tarkemmin tulee siihen liittyvä ohjelman kohtakin testata.

4.11 Laatu ja laatujärjestelmä

Tietojärjestelmistä puhuttaessa käsitteellä laatu tarkoitetaan järjestelmän kykyä täyttää käyttäjän kohtuulliset toiveet ja odotukset. Termillä laatu ei tässä yhteydessä tarkoiteta hyvää tai huippulaatua, vaan erilaisia tuotteen ja toiminnan mittaavia ominaisuuksia. Laatua tarkastellaan tuotteen laadun ja toiminnan laadun kannalta. Toiminnan laatu tarkoittaa tuotteen laatuun positiivisesti vaikuttavia toimintatapoja.

Moderni laatuajattelu lähtee siitä, että tuotteen laatuun vaikutetaan parhaiten yrityksen toiminnan laadun kautta. Tuotteen tekemisessä käytettävää yrityksen toimintatapaa kutsutaan laatujärjestelmäksi. Laatujärjestelmän tarkoituksena on taata, että yrityksen tuotantoprosessi tuottaa suunniteltua laatutasoa olevia tuotteita aikataulun ja budjetin mukaisesti. Laatujärjestelmä kuvataan yleensä yrityksen laatukäsikirjassa ja siihen liittyvissä ohjeistuksissa.

Yrityksen laatujärjestelmään liittyy toiminnan todistettavuus ja jäljitettävyys. Tarvittaessa ulkopuolisten tahojen, kuten asiakkaiden on pystyttävä varmistamaan, että yritys todellakin toimii laatujärjestelmänsä mukaisesti. Tästä syystä laatujärjestelmän mukaisesta toiminnasta on jäätävä todisteita, kuten esimerkiksi tarkastuspöytäkirjoja, palaveripöytäkirjoja, virheraportteja ja tietoa laadun mittaamisesta. (Haikala & Malijärvi 2006, 48; Pohjonen 2002, 78.)

4.12 Laadunvarmistus ja varmistukseen käytettävät tekniikat

Laatujärjestelmän toimintaan liittyy oleellisena osa laadunvarmistus. Se kohdistuu sekä toiminnan että tuotteen tekemisessä syntyvien vaihetuotteiden laadunvarmistukseen. Toiminnan laatua voidaan arvioida erilaisilla laatujärjestelmän arviointitavoilla kuten auditoinneilla.

Auditoinneissa arvioinnin suorittaa riippumaton yrityksen ulkopuolinen taho. Siinä yrityksen laatujärjestelmää tai sen osaa käydään systemaattisesti läpi ja varmistetaan, että toiminta on laatujärjestelmän mukaista. Arviointi voi perustua määrämuotoiseen tarkastuslistaan tai standardiin (esimerkiksi ISO 9001).

Tuotteen laadunvarmistuksen tarkoitus on estää virheiden pääseminen tuotteeseen ja toisaalta auttaa löytämään tehdyt virheet mahdollisimman varhaisessa vaiheessa. Tuotteen laadunvarmistusta voidaan suorittaa muun muassa testauksella, tarkastuksella ja katselmoinneilla.

Tarkastukset (inspection) kohdistuvat vaihetuotteisiin. Tarkastustilanteessa vaihetuote käydään läpi ja siitä löytyneet virheet kirjataan ylös. Tarkastusten ideana on löytää mahdollisimman paljon virheitä, jotta seuraava vaihe voi alkaa mahdollisimman vakaalta pohjalta.

Katselmoinnit kohdistuvat tuotteen dokumentteihin. Niitä järjestetään esimerkiksi kunkin vaiheen päätteeksi. Katselmuksissa vaiheen tuotteet, kuten vaikkapa määrittelydokumentti, käydään läpi ja todetaan vaihe päättyneeksi.

Tuotteen laadunvarmistuksen yhteydessä käytetään myös termejä verifiointi (verification) ja validointi (validation). Verifioinnissa varmistetaan, että tuote tai vaihetuote vastaa tuotteen spesifikaatiota. Se vastaa kysymykseen: Olemmeko me rakentamassa tuotetta oikein? Validoinnissa taas tutkitaan tuotteen sopivuutta käyttötarkoitukseen ja se vastaa kysymykseen: Olemmeko me rakentamassa oikeaa tuotetta? (Haikala & Mali-järvi 2006, 49–50; Pohjonen 2002, 78.)

4.13 Dokumentointi ja standardit

On olemassa useita julkaistuja standardeja, jotka liittyvät ohjelmistotuotantoon ja sen dokumentoitiin. Nämä standardit ovat useimmiten ohjeiden ja suositusten luonteisia. Ehkä tunnetuimpia ja laajimmin levinneitä ovat ISO- (International Organization of Standardization) ja IEEE (Institute of Electrical and Electronics Engineers) -standardit. Erilaisia standardityyppejä ovat ns. de facto -standardit, viitekehykset ja prosessistandardit.

De facto -standardit eli käytännön standardit ovat suosituksia, joiden asema perustuu niiden levinneisyyteen. Esimerkiksi IBM:n CUA -standardi määrittelee muun muassa kaikille sovelluksille yhteiset näppäintoiminnot. Viitekehykset ovat standardeja, joiden avulla määritellään alueet, joille standardeja kehitetään.

Prosessistandardit (esimerkiksi ISO 9001 ja IEEE 1074-1995) taas pyrkivät määrittelemään ohjelmistoprosessia ja asettamaan vaatimuksia koko prosessille ja sen osille (kuten tuotteenhallinta, testaus ja verifiointi). Dokumentointimalleja on lähes kaikissa standardeissa. Käytännössä niitä on hyvin eri laatuisia, joten on järkevää käyttää hyviksi todettuja malleja.

Käyttämällä tiettyjä standardeja ohjelmistoyritys pystyy osoittamaan, että se tekee ohjelmistoprojektinsa tietyn yleisesti hyväksi nähdyn tavan ja tason mukaan. Testauksessa näitä standardeja hyödynnetään niin testauksen läpiviennissä kuin testitapausten kirjaamisessakin.

ISO/IEC 12207 -standardi määrittää vaadittavat dokumentit ja sen, mitä niistä pitää ilmetä. Kyseinen standardi asettaa esimerkiksi seuraavanlaisia vaatimuksia testaukselle:

Vaadittavat dokumentit:

- Ohjelmiston laaduntakuusuunnitelma
- Testaussuunnitelma
- Testitulosten raportti
- Testitapaukset

Testiproseduureista/dokumenteista ilmenevät asiat:

- Dokumentin laatija, testitapausten laatija ja testaaja
- Päämäärät
- Testikokoonpano
- Testien syötteet ja odotetut tulokset
- Testattavat ominaisuudet
- Testin tunniste
- Vaatimukset listattuna ja testien viittaukset niihin
- Käytänteet ja data jokaista ohjelmistoyksikköä ja tietokantaa varten
- Kriteeri tulosten arvioimiseen
- Ohjeet proseduurien suorittamista varten

IEEE:N standardin asettamat vaatimukset testaukselle pitävät sisällään monia samoja vaatimuksia kuin muutkin standardit. Seuraavassa on listattu IEEE:N standardien (829–1998, 1008–1987 ja 730–1998) testaukseen liittyvät vaatimukset:

1. Testisuunnitelma, joka pitää sisällään esimerkiksi työn laajuuden, aikataulun, käytettävät henkilöstöresurssit, sekä testattavat ja ei-testattavat ominaisuudet
2. Testien suunnittelua spesifikaatio; määrittää käytettävän testauksen lähestymistavan
3. Testitapausspesifikaatio; määrittelee testitapaukset testeissä käytettävän lähestymistavan mukaan
4. Testiproseduurin spesifikaatio; määrittelee mitä seuraavat vaiheet testien suorittamiseen tarvitaan
5. Testiloki; tallentaa relevantit tiedot testien suorittamisesta
6. Testitapausten tulosten raportti; testeissä havaitut virheet, jotka vaativat lisäselvitystä.
7. Testiyhteenveto -dokumentti; yhteenveto koko testausprosessista.

(Haikala 2004, 296-297; Tamres 2002, 231-239)

5 GLOW–projekti

Projekti lähti liikkeelle Lumenen tarpeesta saada hallittua paremmin sen verkkolevyillä olevia dokumentteja ja muita tiedostoja, sekä projekteja. Haluttiin ottaa käyttöön erillinen dokumentin- ja projektinhallintajärjestelmä.

Tätä varten luotiin projektiryhmä, johon kuului erilaisissa työtehtävissä ja eri osastoilla työskenteleviä henkilöitä (kemistejä, tuotekehityksen henkilöitä, rekisteröinnin henkilöitä, markkinoinnin henkilöitä, sekä toimintakäsikirjasta vastaavia henkilöitä). Tällainen projektiryhmän henkilöiden valinta tehtiin sen takia, että haluttiin antaa eri osastojen vaikuttaa tulevaan järjestelmään ja täten varmistaa, että se varmasti vastaa eri osastojen tarpeita. Lumenen tietohallinnon olisi yksin ollut mahdotonta myöskään tietää eri osastoilla työskentelevien henkilöiden rutiineista ja toimintatavoista.

Projektiryhmän kartoitettua tulevalle järjestelmälle asetettavat vaatimukset ja ominaisuudet ryhdyttiin etsimään sopivaa ohjelmistoyritystä. Useampiakin varteenotettavia järjestelmiä ja ohjelmistoyrityksiä löytyi. Eri vaihtoehtojen vertailun jälkeen päädyttiin Microsoftin Sharepoint Server 2008 ratkaisuun, jonka toteuttajaksi valittiin ohjelmistoyritys X.

Tässä vaiheessa tulevalle järjestelmälle ja projektille annettiin nimeksi GLOW (Global Lumene Office Workspace).

5.1 Projekti ja testiryhmä

GLOW -projektin projektipäällikkönä toimi Kristina Tenqvist. Tietohallinnon johtaja Vesa Laaksonen toimi varaprojektipäällikkönä. He olivat kummatkin koko projektin ajan eniten yhteydessä yritys X:n projektipäällikköön ja ohjelmoijaan. Minä itse tulin GLOW -projektiin varsinaisesti mukaan vasta puolitoista viikkoa ennen testauksen alkua ja vastuullani oli hyväksymistestaus kokonaisuudessaan. Tultuani mukaan projektiin olin mukana kaikissa projektin sisäisissä palavereissa kuin myös ohjelmisto yrityksen X kanssa käydyissä palavereissa yhdessä projektipäälliköiden ja tulevan pääkäyttäjän Annina Kallion kanssa.

Muita projektiryhmään kuuluvia henkilöitä oli yhteensä 13. Edellä mainitut henkilöt olivat tiiviisti mukana kun määriteltiin, mitä ominaisuuksia tulevalta järjestelmältä haluttiin. Samalla tästä ryhmästä muodostui testiryhmä hyväksymistestausta varten.

5.2 Projektin aikataulu

Alla olevasta taulukosta 1 ilmenee GLOW-projektin alustava aikataulu. Projekti ei toteutunutkaan täysin aikataulun mukaan, eikä siihen ole merkitty varsinaista täysimittaista tuotantoon käyttöönottoa, joka tapahtuu alustavasti huhtikuun 2008 loppupuolella, kun tarvittava koulutus uusille käyttäjille on tehty.

Taulukko 1. Projektin aikataulu.

Id	Vaihe / tehtävät	Alkaa	Loppuu	Maksuposti/ hyväksynnän tapa
V1	Projektin käynnistys	4.10.2007	8.10.2007	N/A
V2 + V3	**Määrittely (Vaatimusmäärittely ja vaatimusanalyysi)**	8.10.2007	6.11.2007	**16.11.2007 / OR kokous**
V4	**Suunnittelu**	7.11.2007	21.11.2007	**20.12.2007/ Katselmointi**
V5	**Toteutus**	22.11.2007	17.1.2008	**17.1.2007/OR kokous**
V6	Käyttöönoton suunnittelu ja tuotantoasennukset	7.1.2008	17.1.2008	N/A
V7	Koulutukset / hyväksymistestaus	18.1.2008	23.1.2008	N/A
V8	Käyttöönotto / Tilaajan sisällöntuotanto	24.1.2008	30.1.2008	N/A
V9	**Projektin hyväksyntä ja päättäminen**	31.1.2008	6.2.2008	**6.2.2008/ OR kokous**

5.3 Sharepoint 2008

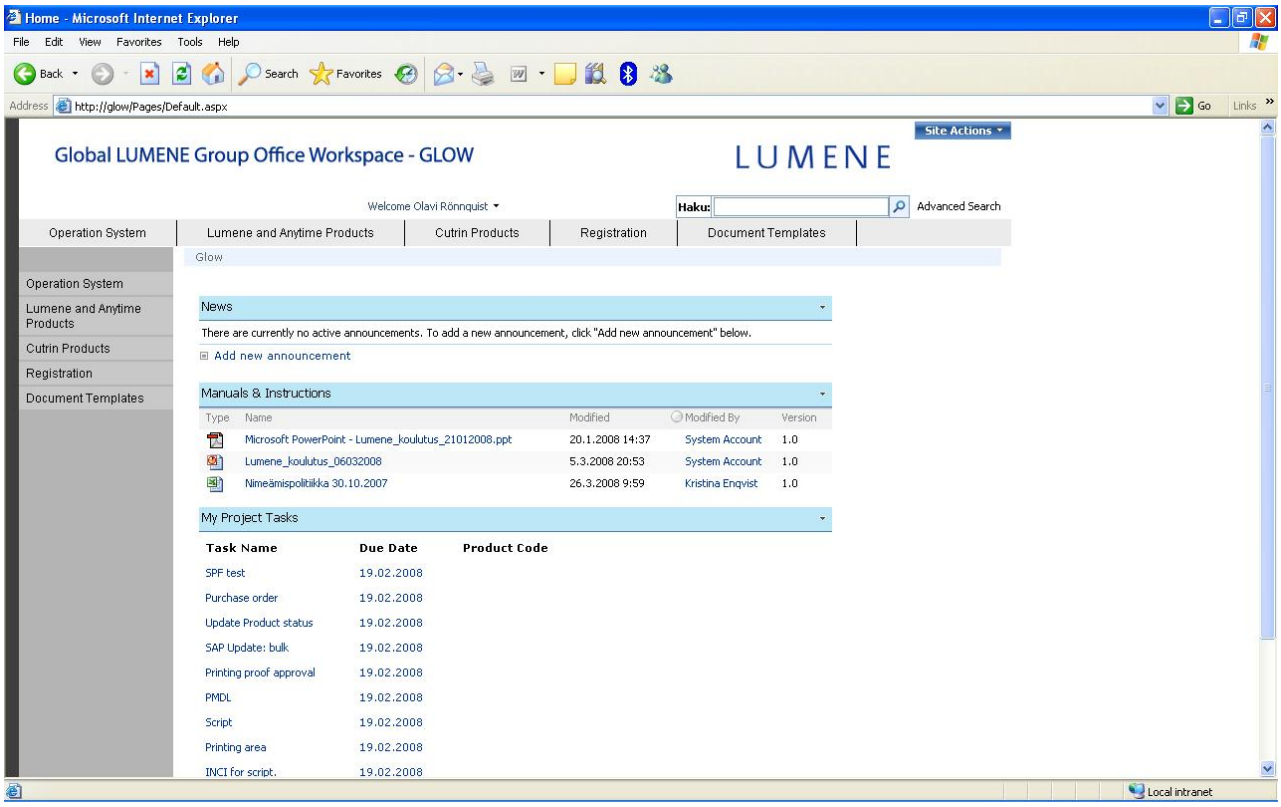
Lumenen valitsema ratkaisu pohjautuu Microsoftin SharePoint 2008-palvelimeen, joka on kyseisen ohjelman uusin versio. Se on Microsoftin toimesta testaama ja sitä käyte-

tään yleisesti esimerkiksi yritysten intranetsivujen luomiseen. Ohjelmistoyritys ei ollut ennen tätä projektia räätälöinyt Sharepointia dokumentin- ja projektin hallintaan, mutta silti kyseisen ohjelman katsottiin parhaiten vastaavan Lumenen tarpeita.

Itse fyysinen palvelin hankittiin Lumenen tietohallinnon toimesta ja se sijoitettiin Lumenen palvelinhuoneeseen. Sharepoint Server 2008 palvelinohjelma kuului Lumenen jo aikaisemmin tekemään sopimukseen Microsoftin kanssa, joten se ei aiheuttanut lisäkustannuksia projektille. Palvelimen fyysisten komponenttien ja Microsoft Windows Server 2003 asennus tehtiin Lumenen atk-osaston toimesta. Ohjelmistoyritys puolestaan asensi palvelimelle MOSSI:n (Microsoft Office Sharepoint Server).

5.4 GLOW-järjestelmä

Lumenelle räätälöidyn GLOW:n peruslähtökohtana on se, että se on jaettu viiteen eri päätasoon, joita ovat (1) rekisteröinti, (2) toimintakäsikirja, (3) Lumene-projektien hallinta, (4) Cutrin-projektien hallinta ja (5) pohjadokumenttikirjasto. Alla olevasta kuviosta 2 näkyy järjestelmän pääsivu.



Kuvio 2. Järjestelmän pääsivu.

Toimintakäsikirja ja pohjadokumenttikirjasto näkyvät kaikille käyttäjille, mutta vain näiden osioiden ylläpitäjillä on muokkausoikeudet eri osioiden sisältöön (muilla on vain lukuoikeudet ja nekin annetaan käyttäjille aina erikseen). Kaikissa näissä osioissa on perushakutoiminto, sekä myös tarkempi hakutoiminto (advanced search).

Rekisteröinnin osio näkyy vain rekisteröinnin parissa työskenteleville muutamalle henkilölle ja heillä on täydet oikeudet (luku, kirjoitus, muokkaus ja poisto) tähän osioon - lukuun ottamatta rekisteröinnin osiossa olevaan salaisempaa dokumenttikirjastoa, joka näkyy vain määrätyille henkilöille.

Lumene- ja Cutrin-projektien hallinnan osiot näkyvät vain määrätyille henkilöille ja lisäksi osioiden sisällä on aliosioita, joiden näkyvyys määritellään aina erikseen. Samoin käyttäjien oikeustasot määritellään aina erikseen.

Lumenen GLOW:n käyttäjäryhmien määrä kasvoi melko suureksi, sillä Lumenen GLOW-järjestelmä on luotu siten, että käyttäjälle määritellään aina erikseen, mihin osioihin tai alaosioihin ja millä oikeuksin hän pääsee.

Käyttäjäryhmiä oli alustavan suunnitelman mukaan noin kolmisenkymmentä. Kuitenkin testiemme edistyessä ohjelmisto yrityksen X ohjelmoija joutui antamamme palautteen pohjalta luomaan käyttäjäryhmiä kymmenkunta lisää, koska alkuperäiset käyttäjäryhmät eivät olisi täysin vastanneet tarvittavien eri käyttäjäroolien tarkoitusta.

Käyttäjien oikeuksien määrittäminen tapahtuu Sharepointin kautta pääkäyttäjien toimesta. Itse käyttäjän Sharepoint puolestaan hakee Lumenen Active Directorysta. Eli jos käyttäjää ei ole lisätty Lumenen Active Directoryyn, niin käyttäjää ei pystytä lisäämään myöskään GLOW:n käyttäjäksi. Erillistä sisäänkirjautumista selaimen kautta käytettävään järjestelmään ei ole, vaan käyttäjä tunnistetaan automaattisesti sen mukaan, millä active directoryn tunnuksella hän on kirjautunut koneelle, josta hän on menossa GLOW-järjestelmään.

5.5 Testitapausten suunnittelu

Kuten aikaisemmin mainitsin, tulin projektiin mukaan vasta puolitoista viikkoa ennen hyväksymistestauksen aloittamista. Johtuen tästä minulla oli tiukka aikataulu ensiksikin perehtyä testauksesta olevaan kirjallisuuteen sekä saada yleiskuva testauksen tasoista

ja käytännöistä. Lumenella itsellään ei ole käytössä mitään yleistä testauspolitiikkaa, standardia tai valmiita hyväksymistestauksessa käytettäviä dokumenttipohjia, joten aloitin kaiken tyhjästä. Lähtökohdaksi testitapausten luomisessa olivat määrittelydokumentit ja ajatus siitä, että testitapaukset olisivat mahdollisimman lähellä todellisia työnkulkuja ja sitä tapaa, jolla ne tullaan uudessa järjestelmässä hoitamaan.

Toinen ongelma oli se, että kukaan Lumenen projektiryhmästä ei ollut nähnyt edes keskeneräistä ohjelmaa toiminnassa, vaan ainoastaan muutaman kuvankaappauksen siitä, mitä tuleman piti. Tästä johtuen en pystynyt tässä vaiheessa suunnittelemaan kovinkaan yksityiskohtaisia ohjeita eri testitapausten suorittamiselle.

Kolmantena testien suunnittelua vaikeuttava asia oli se, että minulla itselläni ei ollut juurikaan kuvaa eri osastojen työnkuluista ja toimintatavoista, joten jouduin kiertelemään eri osastoilla selvittämässä projektiryhmäläisiltä, kuinka kukin työnsä hoitaa.

Tässä vaiheessa luodut testitapaukset perustuivat täysin määrittelydokumentteihin, keräämääni tietoon ja muutamaan kuvankaappaukseen ohjelmasta. Onneksi noin kolmea päivää ennen testauksen alkua saimme tunnukset GLOW-järjestelmään ja pääsimme kokeilemaan, kuinka se yleisperiaatteiltaan toimii. Tässä vaiheessa järjestelmä oli tosin vielä keskeneräinen. Tämä kuitenkin auttoi huomattavasti testitapausten luomisessa ja tarkentamisessa.

Testitapaukset suunniteltiin siten, että ne jaettiin kolmeen pääluokkaan, joita olivat (1) rekisteröinti, (2) Lumene-projektin hallinta ja (3) toimintakäsikirja. Aluksi tehtiin lista testattavista ominaisuuksista ja sitten ruvettiin luomaan yksityiskohtaisempia testitapauksia, jotka pitivät sisällään yhden tai useamman testattavan ominaisuuden. Opinnäytetyön liitteenä on ote testattavista ominaisuuksista (Liite 1).

Tämän jaon mukaan jaettiin myös testiryhmä, koska järjestelmän eri osuudet poikkesivat toisistaan melko paljon, vaikka yhtenäisyyksiäkin oli. Tarkoituksena oli, että jokainen ryhmä testaisi omaa osuuttaan erikseen, koska ohjelma ei ollut kenellekään testiryhmäläiselle ennestään tuttu ja näin ollen kysymyksiä ohjelman käytöstä varmasti tulisi (siitäkin huolimatta, että testiä edelsi ohjelman koulutus ohjelmisto yrityksen X kouluttajan puolesta). Jokaiselle ryhmälle oli varattu puoli päivää aikaa käydä läpi heille tehdyt testitapaukset. Järjestelmän pohjadokumenttien testauksen ja oikeustasojen testauksen yhdistin jokaisen ryhmän testitapauksiin.

Suurin osa testitapauksista oli itsenäisesti tehtäviä, mutta joukossa oli myös muutama paritesti, koska osa GLOW-järjestelmän testattavista ominaisuuksista vaati sitä. Esimerkiksi GLOW-järjestelmän projektipuolella suurin osa luoduista dokumenteista tuli hyväksyttää toisella henkilöllä. Luodut testitapaukset sisälsivät myös käyttäjien virhetoiminnan testausta, esimerkiksi epäkelvon arvon antamista dokumenttien metatietoihin.

Itse testilomakkeen perusmuoto oli yhtenäinen kaikille testi ryhmille ja siitä ilmenivät seuraavat asiat:

- Testin numero
- Testaaja
- Testaajan oikeustaso
- Testin suorituspäivämäärä
- Havaitut virheet
- Onnistuiko testi
- Mahdolliset kehitysehdotukset

Alla on toimintakäsikirjan hyväksymistestauksen ensimmäinen testitapaus. Opinnäytetyön liitteenä on suurempi ote testitapauksista (Liite 2).

Testilomake 1

Testaaja:
Päivämäärä:
Käyttäjän oikeusryhmä:
Testin numero: t1.1

Testin kuvaus: Dokumentin lisääminen toimintakäsikirjaan
1. Hae uusi kehityskeskustelulomake (kuvitteellinen) toimintakäsikirjaan GLOW-järjestelmän ulkopuolelta ja tallenna se toimintakäsikirjaan julkistettavana pääversiona
2. Tarkista, näkyykö tallentamasi dokumentti toimintakäsikirjasivuston pääsivulla

Oletettu lopputulos: Dokumentin tuominen Mossiin onnistuu ja se näkyy toimintakäsikirjasivuston pääsivulla

Toteutuiko oletettu lopputulos (Kyllä/Ei):

Sanallinen kuvaus testauksen aikana esiin tulleista asioista: Virhe/Huomautus/kehitysehdotus (ympyröi yksi tai useampi vaihtoehto).
(Jos kyseessä on virhe, niin yritä kertoa mahdollisimman tarkasti, mitä olit tekemässä virheen ilmaannuttua, esimerkiksi, painoit tiettyä nappia)

[redacted]

Testausta varten keräsimme ja siirsimme Sharepoint-palvelimelle suuren määrän Lumenella käytössä olevia pohjadokumentteja, rekisteröinnin dokumentteja ja kahden päättyneen tuoteprojektin kaikki dokumentit, jotta testiympäristö olisi mahdollisimman todenmukainen.

5.6 Testiympäristö, tila ja laitteet

Hyväksymistestaus päätettiin suorittaa Lumenen atk-koulutus luokassa. Kolmipäiväisen hyväksymistestauksen ajankohta oli 19.1.2008- 21.1.2008. Koulutusta edelsi yksipäiväinen Sharepoint-koulutus, jonka Ohjelmisto yrityksen X Sharepoint-kouluttaja piti.

Hyväksymistestaus suoritettiin oikeassa tuotantoympäristössä. Testausta varten koulutusluokan koneisiin asennettiin käsin kaksi Microsoft Officen Sharepoint lisäosaa, jotta kaikki Sharepointin ominaisuudet toimisivat kunnolla. Tulevaisuudessa kun GLOW ote-

taan käyttöön koko Lumenen laajuudessa, niin näiden lisäosien asennus tullaan tekemään etäasennustyökalujen avulla (SMS) ilman, että käyttäjä sitä edes huomaa.

Koulutusluokan laitteisto on Lumenen laitekannan keskitasoa, joka sopi hyvin hyväksymistestauksen laitteistoiksi. Koneiden oleellinen tekninen spesifikaatio on seuraava:

- Malli; HP Compaq Evo 510
- Suoritin; Intel Pentium 4 3Ghz
- Muisti; 512 MB SdRam
- Kovalevy; 60 GB

5.7 Sharepoint-koulutus

Koulutus järjestettiin Lumenen atk-koulutusluokassa ja läsnä olivat koko Lumenen GLOW-projektiryhmä, sekä Ohjelmisto yrityksen X kouluttaja ja ohjelmoija, joka oli yksin tehnyt koko Lumenen tarvitseman räätälöinnin (ohjelmoinnin) Sharepointille.

Koulutuksessa käytiin läpi ohjelman perusominaisuuksia ja toimintoja, joita lumenelaiset hakivat tältä uudelta järjestelmältä ja joita he tulevat käyttämään työssään. Koulutuksessa perustoiminnot käytiin läpi lyhyen kuvallisen monistesarjan avulla. Jo monistesarjan läpikäynnin aikana ohjelmassa ilmeni puutteita, jotka luvattiin korjata.

Suurimman osan koulutuksessa läpikäydyn monistenipun asioista olin ehtinyt opetella itsenäisesti testitapauksia luodessani. Samoin olivat tehneet projektipäällikkömme ja pääkäyttäjämme, jotka olivat myös saaneet tunnukset järjestelmään kahta päivää aikaisemmin. Tästä johtuen meillä oli enemmän kysymyksiä kuin kouluttajalla ja ohjelmoijalla vastauksia.

Tässä koulutustilaisuudessa kävi ilmi, että järjestelmä oli ainakin osittain keskeneräinen. Samalla huomattiin, että tiedonkulku Lumenelta ohjelmisto yrityksen X projektipäällikölle ja edelleen ohjelmoijalle ei ollut sujunut niin kuin piti. Ohjelmisto yrityksen X ohjelmoija ei ollut tietoinen monestakaan Lumenen pyytämästä muutoksesta, eikä ollut saanut kaikkea Lumenen lähettämää aineistoa, joka oli toimitettu ohjelmistoyritys X:lle yli kuukausi etukäteen. Ihmettelimme projektiryhmän sisällä, ettei ohjelmistoyritys X juurikaan kommentoinut tai esittänyt kysymyksiä koskien lähettämäämme aineistoa ja muutospyyntöjä. He vain tuntuivat kerralla tietävän mitä haemme takaa.

Suurin yksittäinen asia, mikä koulutustilaisuudessa ihmetytti, oli se, että eri käyttäjäryhmiä ei oltu tässä vaiheessa luotu ollenkaan, ja kuitenkin testien oli määrä alkaa jo seuraavana päivänä. Lisäksi projektin päätöspalaveri oli jo seuraavana maanantaina.

6 GLOW-hyväksymistestaus

6.1 Ensimmäinen hyväksymistestaus

Ensimmäisen hyväksymistestausryhmän oli määrä aloittaa GLOW-järjestelmän testaaminen heti aamusta, mutta koulutuksessa ilmenneistä järjestelmän puutteista johtuen päätimme Lumenen projektipäälliköiden kanssa käydä vielä yhdessä läpi ohjelman toimintoja ja työnkulkuja. Ohjelmistoyritys X:n ohjelmoija teki ensimmäisenä testipäivänä vielä muutoksia järjestelmään, joten varsinaisten testiryhmien testausta ei ollut järkevää vielä aloittaa tämänkään takia. Siirsimme ensimmäisen testiryhmän testauksen seuraavaan aamuun ja kahden muun ryhmän testirupeamat seurasivat puolen päivän välein (aamupäivä ja iltapäivä).

Käydessämme ohjelmaa läpi eli harjoittaessamme tutkivaa testausta, ohjelmassa ilmeni monenlaisia puutteita, kuten esimerkiksi metatietojen puuttumisia ja turhia pakollisia kenttiä lomakkeissa. Samalla kun kävimme läpi ohjelmaa, muokkasin testitapauksia ja ohjeistusta vastaamaan paremmin ohjelman toimintaa ja suunniteltuja työnkulkuja.

Seuraavan päivän aamuna ensimmäinen testiryhmä (Lumene-projektin hallinta) aloitti järjestelmän testaamisen. Kuten arvelinkin, kaikilta testiryhmien testaajilta tuli testien aikana melko paljon kysymyksiä siitä, kuinka jokin asia tuli tehdä, koska sitä ei ollut käyty läpi koulutustilaisuudessa. Tämä osaltaan hidasti testien kulkua, mutta pysyimme silti aikataulussa ja kaikki testiryhmät olivat suorittaneet testinsä seuraavan päivän iltapäivään mennessä.

6.2 Muut suoritetut testit

GLOW-järjestelmän hyväksymistestaukseen tehtyjen kirjallisten testitapausten lisäksi testasimme GLOW-järjestelmän toimivuuden Lumenen etäyhteyden (VPN) kautta sekä järjestelmän palautuksen Lumenen varmistusjärjestelmän varmistusnauhoilta.

Etäyhteystestien suorittajina toimivat Lumenen GLOW-projektin projektipäälliköt, pääkäyttäjä ja kirjoittaja. Jokainen suoritti testin itsenäisesti, kotonaan kannettavalla tietokoneella. Jokainen testiryhmäläisistä pystyi onnistuneesti luomaan etäyhteyden Lumenen verkkoon ja käyttämään GLOW-järjestelmää. GLOW-järjestelmän toiminnassa ei todettu mitään poikkeavaa verrattuna GLOW:n käyttöön Lumenen tiloissa. GLOW-järjestelmän palautustestaus suoritettiin Lumenen palvelinhuoneessa. Testin suoritti atk-osaston lähiverkko (LAN) spesialisti. Testissä järjestelmä pystyttiin palauttamaan onnistuneesti edellisen päivän tilaan.

6.3 Ensimmäisen hyväksymistestauksen tulokset ja raportointi

Kun ensimmäinen hyväksymistestaus oli suoritettu kokonaisuudessaan, aloin käymään läpi testaajien minulle palauttamia testitapauspapereita kootakseni ne yhdeksi kokonaisuudeksi eli testiraportiksi.

Periaatteessa yhdenkään testitapauksen ei voinut katsoa menneen hyväksytysti läpi, koska melkein jokaisessa testitapauksessa oli kohta, jota testaaja ei ollut voinut suorittaa, koska kyseinen ominaisuus ei toiminut tai puuttui järjestelmästä. Suurin ongelma ensimmäisessä hyväksymistestauksessa oli se, että kaikki testit jouduttiin tekemään admin eli pääkäyttäjän oikeuksin, koska ohjelmistoyritys X ei ollut saanut tehtyä tarvittavia käyttäjäryhmiä. Täten minkäänlaisia oikeustasojen testauksia ei voitu suorittaa. Jo tämän asian vuoksi testauksen ei voitu katsoa menneen läpi onnistuneesti.

Positiivinen asia ensimmäisessä testauksessa oli se, että testien aikana ei ilmennyt kuin muutama varsinainen odottamaton virheilmoitus ja järjestelmä oli koko ajan vakaa ja toimi nopeasti. Suurin ongelma oli toimintojen ja asioiden puuttuminen sekä monimutkaisuus. Myös käyttäjäryhmien totaalinen puuttuminen oli ongelmallista. Lisäksi raportoitiin muutama kosmeettinen epäkohta ohjelmassa.

Raportin kokoamisella oli kiire, koska projektin hyväksymistestauksen tulosten läpikäyntikokous oli jo seuraavan viikon maanantai aamuna ja testaus saatiin valmiiksi torstaina. Lisäksi GLOW-projektin ohjausryhmän päätöskokous oli alustavien suunnitelmien mukaan tarkoitus pitää jo kahden viikon päästä hyväksymistestauksesta. Raportti saatiin kuitenkin koottua perjantai-iltapäivään mennessä ja kävimme sen vielä

läpi yhdessä Lumenen projektipäälliköiden kanssa. Tässä tilaisuudessa lisäsimme raporttiin vielä muutaman heille suullisesti ilmoitetun epäkohdan.

Lisäksi kävimme läpi projektin määrittelydokumenttia varmistaaksemme, että puuttuvat ominaisuudet olivat määrittelydokumentin mukaisia eivätkä muutospyyntöjä, joista tulisi lisäkustannuksia projektille. Hankaluutena tässä oli se, että määrittelydokumentti oli osittain liian epämääräisesti laadittu ja se jättikin joitakin asioita melko tulkinnanvaraisiksi.

6.4 GLOW-projektin ensimmäisen hyväksymistestauksen tuloksien läpikäyntikokous

GLOW-projektin hyväksymistestauksen tulosten läpikäynti kokous pidettiin Lumenen tiloissa ja siinä olivat läsnä Lumenen projektipäälliköt, Lumenen pääkäyttäjä, kirjoittaja ja ohjelmistoyrityksen X projektipäällikkö.

Kokouksessa käytiin läpi testiraportin epäkohdat. Onneksi emme joutuneet vääntämään juurikaan kättä siitä, olivatko havaitut epäkohdat määrittelyn mukaisia vai muutospyyntöjä alkuperäiseen määrittelydokumenttiin. Suurin osa asioista oli määritelmän mukaisia ja vain muutama asia katsottiin muutospyynnöiksi. Kokouksen päätös oli, että projektia jatketaan, tarvittavat muutokset tehdään järjestelmään ja suoritetaan regressiotestaus eli uusintatestaus järjestelmälle. GLOW-järjestelmän uusintatestauksen pituudeksi sovittiin kaksi päivää ja ajankohdaksi 18. ja 19. helmikuuta 2008. Samalla GLOW-projektin ohjausryhmän hyväksymis- ja päätöskokous siirrettiin myöhemmin määriteltävään ajankohtaan.

Tästäkin kokouksesta jäi tuntuma, että suurin syy GLOW-järjestelmän puutteellisuuteen oli informaation huono kulkeminen Yrityksen X sisällä sekä se, ettei Yritys X ollut ottanut kunnolla selvää siitä, mitä ominaisuuksia Lumene haluamaltaan järjestelmältä oikeasti tarvitsee ja kuinka Lumenen sisäiset työkulut menevät.

6.5 GLOW-järjestelmän uusintatestaus

GLOW-järjestelmän uusintatestaus aloitettiin sovitusti ja se suoritettiin Lumene atk-koulutusluokassa, kuten aikaisempikin hyväksymistestaus. Paikalla ja testaajina testiluokassa olivat Lumenen projektipäälliköt, pääkäyttäjä ja kirjoittaja. Syy testiryhmän

pienuuteen oli se, että suurimalla osalla aikaisemman testiryhmän jäsenistä oli kädet sidottuina muihin työtehtäviin ja katsoimme Lumenen projektipäälliköiden kanssa, että testiryhmä oli näin riittävä.

Itse testaus ei alkanut toivotulla tavalla, koska yritettäessä lisätä testiryhmän jäseniä GLOW:n eri käyttäjäryhmiin järjestelmä ilmoitti, ettei tämän toiminnon suorittamiselle ollut tarvittavia oikeuksia. Tämän seurauksena emme pystyneet aloittamaan testausta heti aamusta, vaan jouduimme ottamaan yhteyttä yritys X:ään ja he rupesivat selvittämään ongelman syytä. Ongelma ei onneksi ollut kovin suuri, mutta pystyimme aloittamaan testit vasta puolenpäivän aikaan samana päivänä.

Kävimme testeissä läpi valmiiksi luodut testitapaukset ja lisäksi testasimme erikseen projektipäällikkö Vesa Laaksosen kanssa läpi kaikki eri käyttäjäryhmät ja heidän oikeutensa järjestelmän osioihin ja toimintoihin, koska yritys X:n ohjelmoija oli joutunut lisäämään yli kymmenen uutta käyttäjäryhmää ja saimme tästä lisäyksestä ilmoituksen lyhyellä varoitusajalla.

Saimme suoritettua kaikki testit kahden testipäivän aikana ja lisäksi kävimme läpi edellisessä testiraportissa ilmenneet ongelmat kohta kohdalta yhdessä projektipäälliköiden ja pääkäyttäjän kanssa.

6.6 Uusintatestauksen tulokset ja raportointi

Uusintatesteissä GLOW-järjestelmä oli jo paljon valmiimpi ja toimivampi verrattuna ensimmäiseen hyväksymistestaukseen. Tässä testissä havaitut toiminnalliset virheet eivät olleet kovinkaan vakavia ja määrällisesti niitä ei ollut montaa.

Vesa Laaksosen kanssa tekemiemme käyttäjäoikeuksien testeissä ei myöskään ilmennyt yhtään vakavaa virhettä koskien järjestelmän sisäisiä oikeustasoja ja tietoturvaa. Raportoimme ainoastaan yhden epäkohdan, jossa käyttäjäryhmältä piti mielestämme poistaa dokumenttien muokkaus- ja poisto-oikeus.

Kokosimme yhdessä regressiotestiryhmän jäsenten kanssa testiraportin, johon kirjasimme kaiken kaikkiaan yksitoista kohtaa.

6.7 GLOW-projektin toisen hyväksymistestauksen testitulosten läpikäyntipalaveri

GLOW-projektin toinen testitulosten läpikäyntipalaveri pidettiin jo regressiotestausta seuraavana keskiviikkona Lumenen tiloissa. Paikalla palaverissa olivat Lumenen projektipäälliköt, pääkäyttäjä, kirjoittaja sekä yritys X:n projektipäällikkö.

Palaverissa käytiin läpi laatimamme testitulosraportti. Osa raportissamme olleista asioista ilmeni MOSS:in (Microsoft Office SharePoint Server) ominaisuuksiksi, joille ei voinut tehdä paljonkaan. Loput raportin epäkohdat kohdat yritys X lupasi selvittää ja korjata. Näitä ei ollut kuin muutama ja nekään eivät olleet suuria.

Kokouksessa päätettiin myös, että ohjelmistoyritys X järjestää Lumenen pääkäyttäjille suunnatun koulutustilaisuuden, jossa käydään läpi MOSS:in kehittyneempiä ominaisuuksia. Uudenkoulutustilaisuuden pitäminen oli Lumenen vaatimus, joka johtui ensimmäisen koulutuksen pintapuolisuudesta.

Virallisen tilaisuuden jälkeen päätimme Lumenen projektipäälliköiden kanssa, että testaamme ne ominaisuudet, joihin korjauksia vielä tulisi heti kun ohjelmistoyritykseltä X tulisi ilmoitus, että korjaukset on tehty. Nämä korjaukset oli alustavasti luvattu tehdä seuraavan viikon aikana. Näin ei kuitenkaan tapahtunut, koska ohjelmistoyrityksen X ohjelmoija oli lomalla.

Tästä johtuen projektin päätöspalaveriakin jouduttiin siirtämään, joka oli alun perin sovittu pidettäväksi kolme viikkoa aikaisemmin. Ohjelmistoyritys X lupasi tehdä korjaukset uuteen projektin päätöskokoukseen mennessä, joka sovittiin pidettäväksi 12.3.2008. GLOW-järjestelmän toinen koulutustilaisuus sovittiin pidettäväksi viikko ennen projektin päätöskokousta.

6.8 GLOW-järjestelmän toinen koulutustilaisuus

GLOW-järjestelmän toinen koulutustilaisuus pidettiin sovitusti Lumenen tiloissa. Paikalla koulutuksessa olivat Lumenen projektipäälliköt, pääkäyttäjät ja kirjoittaja, sekä atk-osaston jäsen. Ohjelmistoyrityksen X puolelta paikalla olivat GLOW-järjestelmän ohjelmoija ja Sharepoint-kouluttaja, joka tällä kertaa oli eri henkilö kuin aikaisemmalla koulutuskerralla.

Syy atk-osaston henkilön paikallaoloon oli se, että haluttiin atk-osastolla olevan henkilö, joka tuntisi uutta järjestelmää ja pystyisi tarvittaessa auttamaan käyttäjiä GLOW-järjestelmän perusasioissa ja käyttäjien lisäämisessä järjestelmään. Tämä ratkaisu oli pakollinen jo pääkäyttäjien mahdollisten tulevien sairastapaustenkin vuoksi.

Koska ohjelmistoyritys X ei ollut ehtinyt tehdä kaikkia Lumenen vaatimia korjauksia järjestelmään ajallaan, päätimme, että käymme ne läpi koulutustilaisuuden yhteydessä. Tämä oli mielestämme järkevää, koska korjauspyyntöjä ei ollut montaa.

Itse koulutus kesti koko päivän ja siinä käytiin läpi kaikki järjestelmän perusominaisuudet mukaan lukien viimeiset Lumenen vaatimat muutokset. Pystyimme lumenelaisten kesken toteamaan, että vaatimamme korjaukset oli tehty onnistuneesti ja ne eivät vaikuttaneet järjestelmän muuhun toimintaan. Koulutuksessa käytiin läpi myös MOSS:in kehittyneempiä ominaisuuksia paksun monistenipun avulla. Monisteet sisälsivät ohjeita siitä, miten esimerkiksi uuden pääsivuston luominen tai dokumenttien metatietokenttien lisääminen tapahtuu.

Kaiken kaikkiaan olimme erittäin tyytyväisiä toiseen koulutukseen. Koulutuksessa kävi myös ilmi, että GLOW-järjestelmän järjestelmätestejä ei ollut suorittanut kukaan muu kuin ohjelman ohjelmoija, joka ei ole sopiva testaaja minkään oppikirjan mukaan. Tämän asian ohjelmisto yrityksen X ohjelmoija itse kertoi sitä häneltä kysyessäni. Syy tähän oli hänen mukaansa ajanpuute ja se, että Sharepoint on tällä hetkellä niin suosittu, ettei järjestelmän toimittajalla ole tarpeeksi Sharepoint-osaajia. Hän oli myös samaa mieltä siitä, että projektin tiedonkulussa oli ollut puutteita.

6.9 GLOW-projektin nykytilanne

GLOW-projekti päätettiin onnistuneesti 12.3.2008 Lumenen ja ohjelmistoyrityksen X yhteisessä päätöskokouksessa, jossa olivat paikalla Lumenen projektipäälliköt, projektin ohjausryhmän jäsenet ja ohjelmistoyrityksen X projektipäällikön esimies. Kokouksessa sovittiin, että ohjelmistoyritys X luo Lumenelle monistesarjan, joka kattaa GLOW-järjestelmän perustoiminnot. Tästä onnistuneesti sujuneesta päätöskokouksesta seurasi ohjelmistoyrityksen X myöntämä puolen vuoden takuu, jonka aikana GLOW-järjestelmässä mahdollisesti ilmenevät määrittelyn mukaisten toimintojen virheet korjataan veloituksetta yrityksen X toimesta.

Lumenella on järjestetty ensimmäinen GLOW-järjestelmän käyttökoulutus pääkäyttäjän toimesta. Hän suunnitteli koulutuksen ja toimi koulutustilaisuudessa kouluttajana, yhdessä GLOW -projektin projektipäälliköiden kanssa.

Lumenen oman henkilöstön koulutustilaisuuksia tullaan järjestämään ainakin yksi lisää (tarvittaessa useampi). GLOW-järjestelmä on tarkoitus ottaa täydellisesti käyttöön huhti-toukokuun 2008 tienoilla. Tarkoituksena on, että ennen kesää 2008 järjestelmästä löytyy useampikin Lumenen uutuusprojekti.

7 Johtopäätökset, kehitysehdotus ja arviointi

7.1 GLOW-projekti ja opinnäytetyö

Päättäessäni tehdä opinnäyteyöni testauksesta ja lähtiessäni mukaan Lumenen tiedoston- ja projektinhallintahankkeeseen, minulla ei ollut juuri mitään tietoa siitä, miten ohjelmistojen testaus kokonaisuudessa tapahtuu. Itse asiassa jouduin koko projektiin mukaan puolivahingossa, koska halusin saada opinnäytetyöni tehdyksi, eikä Lumenella ollut GLOW-projektiin mukaan lähtiessäni muita opinnäytetyöksi kelpaavia projekteja meneillään tai alkamassa. Aluksi vähän ajattelinkin, että itse aihe ja opinnäytetyön tekeminen tulisi olemaan melko puisevaa.

Käydessäni läpi tähän opinnäytetyöhöni liittyvää kirjallisuutta, suunnitellessani, luodessani GLOW-projektin hyväksymistestauksen testitapauksia ja suorittaessani testejä olen saanut mielestäni melko kattavan kuvan koko ohjelmistojen testausprosessista. Samalla kiinnostukseni testaukseen, sen suunnitteluun ja toteuttamiseen on kasvanut huomattavasti. Samoin mielipiteeni testauksen puisevuudesta aiheena on muuttunut huomattavasti positiivisemmaksi. Opinnäytetyön ansiosta kiinnostukseni ja tietämykseni testauksesta on kasvanut siinä määrin, että tulen varmasti tulevaisuudessa harkitsemaan työpaikan hakemista testauksen saralta.

Olen myös huomannut, että olen alkanut kiinnittää paljon enemmän huomiota esimerkiksi lehtiartikkeleihin, joissa kerrotaan ohjelmistoissa havaituista puutteista ja huomannut lukiessani pohtivani, missä ohjelman määrittelyn, toteutuksen tai testauksen vaiheessa virhe on syntynyt. GLOW-projektin ja lukemani kirjallisuuden myötä minulle on myös konkretisoitunut testauksen tärkeys.

7.2 Kehitysehdotus

Tulevaisuutta ajatellen olisi järkevää hyödyntää GLOW-järjestelmän tarjoamia projekti- ja tiedonhallinta ominaisuuksia myös muidenkin kuin projektissa mukana olleiden osastojen käyttöön. GLOW-järjestelmän laajentamista ajatellen tämä nyt läpiviety projekti antaa hyvän pohjan tuleville laajennusprojekteille. Nyt kun järjestelmä on jo olemassa ja sen ominaisuudet ovat tiedossa, on järjestelmän laajennusten suunnittelu ja haluttujen ominaisuuksien määrittely paljon helpompaa, sekä hyväksymistestauksen suunnittelu helpottuu. Siltikin määrittelyvaiheeseen ja määrittelydokumentin luomiseen tulee kiinnittää erityistä huomiota, kuten tästä projektista opittiin.

Luomaani testisuunnitelmaa ja testitapauksia pystyy melko pienillä muutoksilla käyttämään suoraan hyväksi mahdollisissa tulevissa hyväksymistesteissä. Se, mitä eniten parantaisin itse luomiini testeihin, olisi ohjeistus siitä, kuinka järjestelmän eri ominaisuuksia käytetään testien eri osioissa. Nyt kun GLOW-järjestelmä on jo käytössä, niin uusien käyttäjien koulutus onnistuu Lumenen pääkäyttäjien toimesta, ja tämä koulutus olisi tietenkin järkevää suorittaa testiryhmän jäsenille ennen testejä. Tämä osaltaan edesauttaisi pienentämään testien suorittamiseen kuluvaa aikaa ja samalla säästäisi rahaa, koska ulkopuolisen yrityksen koulutusta ei tarvittaisi.

Uskon, että järjestelmän muokkaaminen laajempaan käyttöön Lumenella helpottaisi ja samalla tehostaisi työntekijöiden tehokkuutta. Järjestelmän laajentamisesta muodostuvat kustannukset eivät olisi kovinkaan suuria, koska tämä vaatisi vain järjestelmän muokkausta ilman laitteisto- ja lisenssikustannuksia.

7.3 Arviointi

Kuten olen jo aikaisemmin todennut, niin tämä projekti ei ollut mikään malliesimerkki onnistuneesta projektista ja sen toteutuksesta. Tästä johtuen myöskään GLOW-järjestelmän hyväksymisteustausta ei pystytty suorittamaan aivan oppikirjojen mallien mukaisesti. Suurimmat syyt tähän olivat testiin saadun järjestelmän ominaisuuksien ja järjestelmätestauksen puutteellisuudet sekä Lumenen ja ohjelmistoyrityksen yhdessä tekemän järjestelmän määrittelydokumentin osittaiset näkemyserot siitä, mitä järjestelmältä haluttiin ja siitä, miten sen tulisi toimia. Se, etten päässyt näkemään testattavaa järjestelmään kuin vasta muutama päivä ennen testien aloitusta, ei myöskään helpottanut testien suunnitelua.

Tällaisessa kahden osapuolen ohjelmistoprojektissa erityistä huomiota pitää kiinnittää ohjelmiston määrittelyvaiheeseen. Ohjelmiston määrittelydokumenttiin tulee kirjata mahdollisimman selkeästi, mitä ohjelmistolta halutaan ja miten sen tulisi toimia, jotta vältytään erimielisyyksiltä ja kiistatilanteilta. Hyvin tehty määrittely auttaa myös aikataulussa pysymiseen ja korjauspyyntöjen minimoimiseen, jolloin projektin kustannukset pysyvät sovitussa budjetissa.

Hyväksymistestejä suunnitellessa ja luodessa on tärkeää kiinnittää huomiota siihen, että testit kattavat mahdollisimman laajasti testattavan ohjelman eri ominaisuudet. Varsinkin ohjelman määrittelydokumenttiin kirjatut ominaisuudet tulee testata huolellisesti, jotta ohjelma vastaa sille asetettuja vaatimuksia. Testitapausten tulee olla selkeästi ohjeistettuja, jotta testit pystyy suorittamaan kuka tahansa. Jos käytössä on paperinen testilomake tulee siinä olla tarpeeksi tilaa mahdollisten vieheiden yksityiskohtaiseen kirjaamiseen. Hyvin luotu määrittelydokumentti auttaa myös testien luomista.

GLOW-projektissa määrittelydokumentin selkeyteen olisi pitänyt kiinnittää enemmän huomiota, sillä se sisälsi liikaa tulkinnan varaisia kohtia. Tosin osa syy näkemyseroihin oli se, että määrittelyvaiheessa oli mukana ohjelmistoyrityksen henkilö, joka tuntui ymmärräntävän hyvin, mitä Lumene halusi. Valitettavasti tämä henkilö ei ollut enään mukana projektissa, kun järjestelmää alettiin luomaan.

GLOW-projekti ei ollut myöskään mikään malliesimerkki hyvin toteutetusta testauksesta ohjelmistoyrityksen taholta, pikemminkin päinvastoin. En voi sanoa, että olisin itsekään toiminut täysin lähdekirjallisuuden oppien mukaan, mutta lopputulos oli mielestäni hyvä. Tavallaan pääsin näkemään ja oppimaan enemmän tässä projektissa kuin sellaisessa projektissa, missä kaikki olisi mennyt oppikirjojen mukaan.

Kokonaisuutena GLOW-projekti oli erittäin opettavainen, koska en ole aikaisemmin ollut mukana ohjelmistoprojektissa. Nyt minulla on paljon parempi tietämys ohjelmistoprojekteista ja niiden eri vaiheista.

LÄHTEET

Bach, J. 2003. Exploratory Testing Explained. Tulostettu 19.3.2008. http://www.satisfice.com/articles/et-article.pdf

Black Rex. 2004. Critical Testing Processes. Boston, USA: Pearson Education INC.

Cybercom Plenware. Viitattu 7.3.2008. http://www.plenware.fi/liitteet/150.pdf

Haikala, I & Malijärvi, J. 2006. Ohjelmistotuotanto. 11. painos. Jyväskylä: Gummerrus Kirjapaino OY.

Kit, E. 2005. Software testing in the real world. Edinburg, Great Britain: Pearson Education Limited.

Pohjonen, R. 2002. Tietojärjestelmien kehittäminen. 1. painos. Jyväskylä: Docendo Finland Oy.

Software Business Competence. Viitattu14.2.2008. http://www.oamk.fi/sbc/testaus/testauksen_tavoitteet.htm

Stenberg, A. 2007. Ohjelmiston testaus 2007. Viitattu 14.3.2008. http://www.pori.tut.fi/~stenberg/index_files/johdanto.pdf

Tamres, L. 2002. Introducing Software Testing. Edinburg, Great Britain: Pearson Education Limited

Tapola, V. 2004. Testaaminen ohjelmiston kehitysprosessin aikana. Viitattu 14.3.2008 http://www.pori.tut.fi/~stenberg/index_files/johdanto.pdf

LIITTEET

Liite 1: Ote testattavista ominaisuuksista

Testin numero	Toiminnon tarkoitus	Testattava toiminto	Ryhmä	Oletettu lopputulos
t1.1	Dokumentin lisääminen toimintakäsikirjaan	Käytännön työt		Dokumentin lisääminen onnistuu
t1.2	Menettelyohjeen päivittäminen ja hyväksyttäminen	Käytännön työt		Päivittäminen ja hyväksyttäminen onnistuu
t1.3	Uutisen luominen toimintakäsikirja sivustolle	Käytännön työt		Uutisen luominen onnistuu ja se näkyy announcemets listalla
t1.4	Uuden auditoinnin lisääminen	Käytännön työt		Auditoinnin lisääminen onnistuu ja se näkyy toimintakäsikirjan pääsivulla.
t2.1	Rekisteröitävän tuotteen kansion luominen ja dokumentin lisääminen pohjadokumentista	Käytännön työt		Rekisteröitävän tuotteen kansion luominen ja dokumentin lisääminen onnistuu
t2.2	Tuotteen uudelleenrekisteröinnin hälytyksen määrittäminen	Käytännön työt		Uusinta rekisteröinnin määritteleminen tuotteelle onnistuu ja se näkyy rekisteröinnin kalenterilistalla. Käyttäjä saa lisäksi hälytyksen sähköpostiin.
t2.3	INCI dokumenttien hakeminen laajennetulla haulla ja sen lisääminen rekisteröinnin sivustolle	Käytännön työt		INCI dokumenttien hakeminen ja lisääminen rekisteröinnin sivustolle.
t2.4	Rekisteröinnin laajennetun haun hälytyksen asettaminen INCI dokumenteille	Käytännön työt		Hälytyksen asettaminen onnistuu ja käyttäjä saa sähköpostin uusista dokumenteista
t2.5	Hälytyksen saaminen kun dokumenttityyppi VEDOS muuttuu julkaistavaksi versioksi.	Käytännön työt		Käyttäjä saa hälytyksen Vedoksen muuttumisesta julkaisuversioksi.

Liite 2: Ote testi tapauksista

Testilomake 2

Testaaja:
Päivämäärä:
Käyttäjän oikeusryhmä:
Testin numero: t2.1

Testin kuvaus: Rekisteröitävän tuotteen kansion luominen ja dokumentin lisääminen pohjadokumentista

1. Luo tuotteen x rekisteröinnin kansio
1.Lisää tuotteen x rekisteröintiin liittyvä uusi dokumentti rekisteröinnin pohjadokumentista
2. Mene rekisteröinnin laajennettuun hakuun
3. Tarkista, että automaattisen haun dokumentti listasta löytyy äsken luomasi dokumentin.

Oletettu lopputulos: Rekisteröitävän tuotteen kansion luominen ja Dokumentin lisääminen onnistuu

Toteutuiko oletettu lopputulos (Kyllä/Ei):

Sanallinen kuvaus testauksen aikana esiin tulleista asioista: Virhe/Huomautus/Kehitysehdotus (ympyröi yksi tai useampi vaihtoehto).
(Jos kyseessä on virhe, niin yritä kertoa mahdollisimman tarkasti mitä olit tekemässä virheen ilmaannuttua, esimerkiksi painoit tiettyä nappia)

Testaaja:
Päivämäärä:
Käyttäjän oikeusryhmä:
Testin numero: t2.2

Testin kuvaus: Tuotteen uudelleen rekisteröinnin hälytyksen määrittäminen
1.Määritä uusinta rekisteröinti hälytys tuotteelle x täksi päiväksi

2.Tarkista, näkyykö määrittelemäsi tuotteen uusintarekisteröinti uusinnat-listalla

3. Tarkista, että sait sähköpostiin hälytyksen uudelleenrekisteröinnistä

Oletettu lopputulos: Uusinta rekisteröinnin määritteleminen tuotteelle onnistuu ja se näkyy rekisteröinnin task listalla. Käyttäjä saa lisäksi hälytyksen sähköpostiin.

Toteutuiko oletettu lopputulos Kyllä/Ei

Sanallinen kuvaus testauksen aikana esiin tulleista asioista: Virhe/Huomautus/kehitysehdotus (ympyröi yksi tai useampi vaihtoehto).
(Jos kyseessä on virhe, niin yritä kertoa mahdollisimman tarkasti mitä olit tekemässä virheen ilmaannuttua, esim. painoit tiettyä nappia)

Testaaja:
Päivämäärä:
Käyttäjän oikeusryhmä:
Testin numero: t2.3

Testin kuvaus: INCI dokumenttien hakeminen laajennetulla haulla ja sen lisääminen rekisteröinnin sivustolle

1.Mene rekisteröinnin laajennettuun hakuun rekisteröinnin pääsivulta.
2. Hae rekisteröintiin tarvittavat INCI dokumentit käyttäen laajennettua hakua ja lisää rekisteröinnin sivustolle..

Oletettu lopputulos: INCI dokumenttien hakeminen ja lisääminen rekisteröinnin sivustolle.

Toteutuiko oletettu lopputulos Kyllä/Ei

Sanallinen kuvaus testauksen aikana esiin tulleista asioista: Virhe/Huomautus/kehitysehdotus (ympyröi yksi tai useampi vaihtoehto).
(Jos kyseessä on virhe, niin yritä kertoa mahdollisimman tarkasti mitä olit tekemässä virheen ilmaannuttua, esim. painoit tiettyä nappia)

Testaaja:
Päivämäärä:
Käyttäjän oikeusryhmä:
Testin numero: t2.4

Testin kuvaus: Rekisteröinnin laajennetun haun hälytyksen asettaminen INCI dokumenteille

1.Mene rekisteröinnin laajennettuun hakuu
2.Määritä INCI dokumenteille hälytys
3. Lisää järjestelmään inci dokumentti
4. Tarkasta sähköpostista saatko hälytyksen uudesta INCI dokumentista.

Oletettu lopputulos: Hälytyksen asettaminen onnistuu ja käyttäjä saa sähköpostin uusista dokumenteista

Toteutuiko oletettu lopputulos Kyllä/Ei

Sanallinen kuvaus testauksen aikana esiin tulleista asioista: Virhe/Huomautus/kehitysehdotus (ympyröi yksi tai useampi vaihtoehto).
(Jos kyseessä on virhe, niin yritä kertoa mahdollisimman tarkasti mitä olit tekemässä virheen ilmaannuttua, esimerkiksi painoit tiettyä nappia)

Testaaja:
Päivämäärä:
Käyttäjän oikeusryhmä:
Testin numero: t2.5

Testin kuvaus: Hälytyksen saaminen kun dokumenttityyppi VEDOS muuttuu julkaistavaksi versioksi.

Oletettu lopputulos: Käyttäjä saa hälytyksen Vedoksen muuttumisesta julkaisuversioksi.

Toteutuiko oletettu lopputulos Kyllä/Ei

Sanallinen kuvaus testauksen aikana esiin tulleista asioista: Virhe/Huomautus/kehitysehdotus (ympyröi yksi tai useampi vaihtoehto).
(Jos kyseessä on virhe, niin yritä kertoa mahdollisimman tarkasti mitä olit tekemässä virheen ilmaannuttua, esimerkiksi painoit tiettyä nappia)