Jani Leppäniemi

TESTAAMISEN KEHITTÄMINEN YRITYKSEN OHJELMISTOKEHITYKSESSÄ

Tietojenkäsittelyn koulutusohjelma

2014

samk
Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

TESTAAMISEN KEHITTÄMINEN YRITYKSEN OHJELMISTOKEHITYKSESSÄ

Leppäniemi, Jani
Satakunnan ammattikorkeakoulu
Tietojenkäsittelyn koulutusohjelma
Huhtikuu 2014
Ohjaaja: Nieminen, Hans
Sivumäärä: 40
Liitteitä: 0

Asiasanat: testaus, testitapaus, laatu, Modultek

Tämän opinnäytetyö aiheena oli Modultek Oy:n PDM-tuotekehitysosaston testauksen kehittäminen. Tavoitteena oli luoda yhtenäinen testitapauskirjasto ja valita yrityksen käyttöön soveltuva testitapaustenhallintajärjestelmä. Projektin osana oli myös valitun järjestelmän asennus ja käyttöönotto.

Työssä kuvataan projektin kulku ja perehdytään testauksen teoriaan yleisellä tasolla. Lisäksi syvennytään tarkemmin projektin kannalta oleellisiin seikkoihin.

Työn tuloksena syntyi testitapauskirjasto, eli jäsennelty kokoelma testitapauksia. Lisäksi valittiin, asennettiin ja otettiin käyttöön Tarantula -niminen testitapaustenhallintajärjestelmä. Projektin tuloksena Tarantula otettiin käyttöön myös Modultekin toisella tuotekehitystä suorittavalla osastolla, Product Information Systems -osastolla.

IMPROVING TESTING IN A COMPANY'S SOFTWARE DEVELOPMENT

Leppäniemi, Jani
Satakunnan ammattikorkeakoulu, Satakunta University of Applied Sciences
Degree Programme in Business Information Systems
April 2014
Supervisor: Nieminen, Hans
Number of pages: 40
Appendices: 0

Keywords: testing, test case, quality, Modultek

The purpose of this thesis was to improve the testing of the PDM research & development department of Modultek Ltd. The goal was to create a unified test case library and to choose a test case management system that would suit the company's needs. Installation and introduction of the system were also part of the project.

This thesis will describe how the project unfolded and introduce the general theory behind testing. In addition to this, the thesis will take a closer look into subjects that were relevant for the project.

As a result of this project a test case library, an organised collection of test cases, was created. Also a test case management system called Tarantula was installed and introduced. After the project Modultek's other R&D department, Product Information Services, also adapted Tarantula.

SISÄLLYS

1 JOHDANTO

Tämän opinnäytetyön aiheena oli Modultek Oy:n tuotekehitysosaston testauksen kehittäminen. Olin projektin toteutusvaiheessa töissä Modultekin tuotekehitysosastolla määräaikaisessa työsuhteessa. Täten Modultek yrityksenä, sekä sen toiminta ja tuotteet, olivat jo tuttuja.

Projektin tavoitteena oli luoda testitapauskirjasto, sekä valita, asentaa ja ottaa käyttöön testitapaustenhallintajärjestelmä. Testitapauskirjastolla tarkoitetaan tässä yhteydessä organisoitua kokoelmaa testitapauksia. Tämä testauksenkehitysprojekti oli osa isompaa, koko yrityksen kattavaa, Laatu 2013 -projektia.

Testaus aiheena on tavattoman laaja. Tästä syystä työssä tehdään yleiskatsaus testauksen teoriaan ja syvennytään tarkemmin projektin kannalta olennaisiin seikkohin. Testaus on myös kiinnostava ja ajankohtainen aihe. Erityisesti näinä aikoina, kun jatkuvasti etsitään keinoja toiminnan tehostamiseen ja säästöjen löytämiseen, on testauksen rooli ainoastaan kasvanut.

2 PROJEKTIN TAUSTA JA MÄÄRITTELY

2.1 Modultek oy

Modultekon tuotetiedonhallintaan erikoistunut ohjelmistoalanyritys, joka tarjoaa sekä tuote-, että palveluratkaisuja asiakkailleen. Modultekin tunnetuin tuote on Aton PDM –tuotetiedonhallintajärjestelmä, joka on Suomen suosituin PDM-ohjelmisto yli 30% käyttäjäosuudellaan (CAD/CAM-yhdistys 2013). Modultekin muita ratkaisuja ovat mm. SolidPDM ja CodeMaster-tuoteperheen tuotteet.

Modultekin liikevaihto on noin neljä miljoonaa euroa. Yrityksen henkilöstömäärä on noin 40. Yritys jakaantuu kolmeen pääosaan. PDM-osastoon (Product Data Management), PIS-osastoon (Product Information Services) ja OneModultekiin.

PDM-osasto keskittyy Aton-tuotteeseen. Osasto jakaantuu tuotekehitykseen ja asiakasratkaisuihin. Tuotekehitys toteuttaa perustuotteen uusia ominaisuuksia ja korjauspaketteja. Asiakasratkaisut puolestaan tekee asiakaskohtaisia räätälöintejä ja kokonaan asiakaskohtaisia sovelluksia. PIS-osasto keskittyy CodeMaster – tuoteperheeseen. OneModultek pitää sisällään asiakaspalvelun, sisäisen IT:n, sekä markkinointi- ja hallinto-osia.

2.2 Projektin tausta

Modultek Oy:llä oli vuonna 2013 käynnissä koko yrityksen kattava Laatu 2013 -projekti, jonka tarkoituksena oli yrityksen toiminnan laadun parantaminen kaikilla osa-alueilla. Osana Laatu 2013 -projektia, PDM-osaston tuotekehityksessä laatua lähdettiin parantamaan testaustoiminnan kehitysprojektilla.

2.3 Lähtötilanne

Modultek käyttää myös itse PDM-osaston päätuotetta, Aton PDM:ää. Uudet versiot ja korjauspaketit otetaankin käyttöön ensin yrityksen sisällä beta-testausmaisesti. Koska tuote on tuotantokäytössä yrityksessä, saadaan hyvin nopeasti tietoa mahdollisista puutteista. Tätä kautta on kertynyt paljon tietoa siitä, mitkä asiat ovat peruskäytön kannalta oleellisimpia. Toisaalta, on myös saatu tietoa siitä, mitkä asiat vaativat eniten testausta. Kaikki uudet ja muuttuneet ominaisuudet testataan lisäksi kohdennetulla testauksella.

Toimialojen erilaisuuden vuoksi Modultekin sisäinen Atonin käyttö poikkeaa kuitenkin siitä, miten varsinaiset asiakkaat käyttävät Atonia. Modultek on pääasiassa ohjelmisto- ja palveluyritys, kun taas sen asiakkaat pääosin mekaniikka- ja elektroniikkateollisuuden yrityksiä. Ainoastaan asiakkaiden käyttämien ominaisuuksien testaaminen vaatii huomattavan paljon asiantuntijuutta mekaniikka- ja elektroniikkasuunnittelusta. Modultekillä on konsultteja, jotka ovat hyvin perillä

asikkaiden käyttötavoista, ja osaavat täten testata tuotetta oikein. Konsultit ovat kuitenkin henkilöresursseina arvokkaita.

2.4 Projektin määrittely

Projektin tavoite oli kehittää testausta. Tarkoituksena oli ottaa käyttöön järjestelmä, jonka avulla saataisiin kaikki hyöty irti yrityksessä olevasta tietotaidosta. Tavoitteena oli myös tehdä testauksesta helpommin hallittavaa ja henkilöriippumattomampaa.

Toimeksiannon ensimmäinen vaihe oli perehtyä testauksen teoriaan ja etsiä yrityksen käyttöön soveltuvia menetelmiä. Tässä vaiheessa tuli myös perehtyä yrityksen konsulttien avustuksella Atonin Modultekissa vähemmän käytettyihin ominaisuuksiin.

Projektin toinen vaihe oli luoda niin sanottu testitapauskirjasto. Testitapauskirjastolla tarkoitetaan tässä järjestettyä kokoelmaa testitapauksia, joilla tuotteen halutut ominaisuudet saadaan testattua riittävän kattavasti.

Kolmas vaihe oli etsiä yrityksen käyttöön soveltuvia testitapaustenhallintajärjestelmiä ja esitellä sopivimmat vaihtoehdot. Neljännessä vaiheessa valittu järjestelmä tuli asentaa ja ottaa käyttöön.

3 TESTAUS

3.1 Testauksen määritelmä

Ohjelmistotestaus on ohjelman suorittamista tavoitteena löytää virheitä (Myers, 1979, 177). Ohjelmistoissa ei käytännössä ole valmistusvirheitä samalla tavalla kuin fyysisissä tuotteissa. Virheet on tehty jo määrittely-, suunnittelu- tai ohjelmointivaiheessa. Ohjelmisto ei myöskään kulu tai sen toiminta muutu käytössä. Ohjelmistossa esiintyvät virheet ovat siis olleet siinä jo alunperin.

Ohjelmistovirheitä, bugeja, on jokaisessa keskikokoisessa ja sitä isommassa ohjelmassa. Nykyiset ohjelmistot ovat niin laajoja ja usein monimutkaisia, ettei ihminen yksinkertaisesti pysty enää hallitsemaan niitä täydellisesti. Tämän laajuuden vuoksi, myös testaaminen on vaikeaa. Jo yksinkertaisen sovelluksen kaikkien mahdollisten syötteiden testaaminen voi viedä huomattavan kauan. Laajimmissa ohjelmistoissa kaiken testaaminen on siis käytännössä mahdotonta. (Pan, 1999.)

3.2 Testauksen tavoitteet

Ohjelmistotestauksen perimmäiset tavoitteet voidaan jakaa kahteen osaan, laadun varmistukseen sekä verifiointiin ja validointiin (V&V). Laadun varmistuksen tarkoituksena on varmistaa, ettei valmiissa tuotteessa ole puutteita tai virheitä. Nykyisessä tietoyhteiskunnassa lähes kaikki toimii tietokoneavusteisesti, joten kriittisen järjestelmän virhe voi aiheuttaa huomattavaa taloudellista haittaa, tai jopa hengenvaaran. (Pan, 1999). Verifioinnilla pyritään selvittämään tehdäänkö tuotetta oikein (ISTQB:n www-sivut 2014a). Validoinnin tavoitteena on pyrkiä arvioimaan toteuttaako ohjelmisto sille asetetut tavoitteet eli tehdäänkö oikeaa tuotetta (Hetzel, 1988, 280). Laadunvarmistuksen voidaan nähdä olevan ohjelmiston toteuttajan sisäistä toimintaa, kun taas verifiointiin ja varsinkin validointiin tarvitaan yhteistyötä asiakkaan kanssa.

3.3 Ohjelmistokehityksen prosessimallit ja testaus

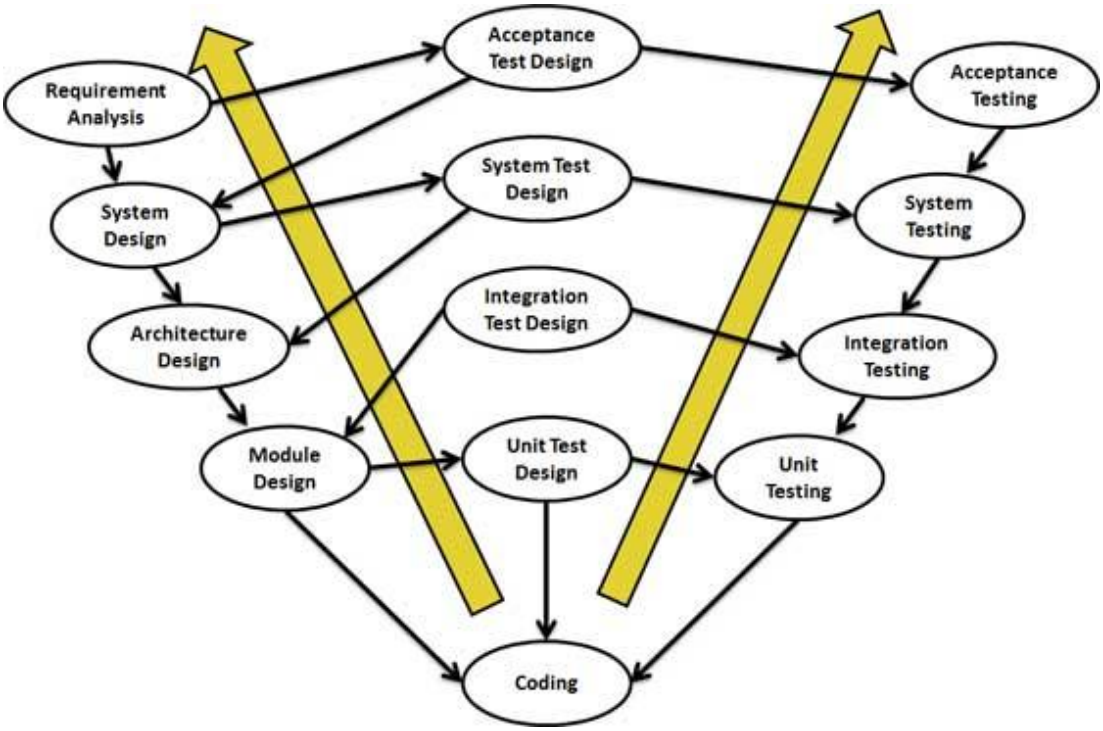
Ohjelmistokehityksessä käytetään useita prosessimalleja. Testaus on olennainen osa niissä kaikissa. Yleisimpiä lienevät vesiputousmalli, V-malli ja niin sanotut ketterät menetelmät.

3.3.1 Vesiputousmalli ja V-malli

Puhtaassa vesiputousmallissa (waterfall model) ohjelmisto kehitetään vaihe kerrallaan, eikä edelliseen vaiheeseen enää palata. Eri esityksissä vaiheet ovat hieman erilaiset, mutta yleensä tunnistettavia vaiheita ovat ainakin määrittely, suunnittelu, toteutus, testaus, käyttöönotto ja ylläpito. Tämä on erittäin kankea lähestymistapa, sillä esimerkiksi määrittelyssä tehty voi tulla vastaan vasta testausvaiheessa. Tällöin sen korjauskustannukset ovat huomattavasti suuremmat, kuin mitä olisivat olleet, jos virhe olisi havaittu aikaisemmassa vaiheessa. Vesiputousmallista on myös jatkokehitettyjä malleja, joissa tapahtuu takaisinkytkentää vaiheiden välillä. (Scacchi, 2001, 5 - 7.) Vesiputousmallissa testaus on siis yksi kehittysprosessin vaiheista.

Vesiputousmallin esitteli Winston Royce jo vuonna 1970. Royce ei käyttänyt termiä vesiputous, mutta kuvasi nykyään vesiputousmallina tunnetun prosessin. Huomionarvoisaa on, että Royce esitti mallin esimerkkinä huonosta prosessista. (Toikkanen, 2004.)

V-malli (V-model) on tavallaan vesiputousmallin jatkokehityksen tulos. V-mallissa ovat mukana samat vaiheet kuin vesiputousmallissakin, mutta siinä jokaista muuta vaihetta vastaa oma testausvaiheensa. (Tutorialspointin www-sivut.)



Kuvio 1. V-malli (Tutorialspointin www-sivut).

3.3.2 Ketterät menetemät

Niin vesiputousmallissa kuin V-mallissakin voivat vaiheet venyä pitkiksi, jolloin virheiden korjaaminen tulee jatkuvasti kalliimmaksi. Niin sanotut ketterät menetelmät (agile patterns) pyrkivät puuttumaan juuri tähän ongelmaan. Ketteriä menetelmiä on useita, muun muassa Scrum, DSDM ja Extreme Programming. Niissä jokaisessa on omat toimintamallinsa ja ominaisuutensa, mutta kaikilla pyritään samaan tavoitteeseen.

Ketterien menetelmien tavoite on pitää kehitys lyhyissä, maksimissaan muutaman viikon pituisissa iteraatiossa. Jokainen iteraatio sisältää kaikkia ohjelmistoprosessin vaiheita, määrittelyä, suunnittelua, toteustusta ja testausta. Kun vaiheet ovat lyhyitä ja toimitaan läheisessä yhteistyössä asiakkaan kanssa, voidaan ongelmat havaita ja korjata jo varhaisessa vaiheessa. Näin saadaan myös ohjattua kehitystä oikeaan suuntaan. (Ambysoftin www-sivut.) Ketterissä menetelmissä testaus on siis aina läsnä.

3.4 Testisykli

Järven ja Mäkilän (2009, 16 - 17) mukaan tyypillinen testisykli voidaan jakaa seuraaviin osiin

- tilanteen analysointi
- testauksen suunnittelu
- testien kehittäminen
- testien suorittaminen
- raportointi
- tulosten analysoiminen.

3.4.1 Tilanteen analysointi

Testaamisen kehittäminen alkaa jo määrittelyvaiheessa. Hyvin tehdystä määrittelystä saadaan selville, mitkä ovat ohjelmiston ydintoimintoja ja miten sen halutaan toimivan. Näin testaus saadaan ohjattua heti oikeaan suuntaan. Kehityksen alkuvaiheessa on myös mahdollisuus valita kyseiseen projektiin parhaiten sopivat testimenetelmät ja -käytännöt.

3.4.2 Testauksen suunnittelu

Suunnittelussa luodaan testaussuunnitelma. Testaussuunnitelmassa määritellään testausorganisaatio ja testauksen tavoitteet. Siinä myös suunnitellaan tarkemmin valittujen menetelmien käyttötavat, testausympäristöt ja testauksen aikataulu.

3.4.3 Testien kehittäminen

Testien kehittämisvaiheessa luodaan testitapaukset käyttäen pohjana vaatimusmäärittelyä ja testaussuunnitelmaa. Tässä vaiheessa myös luodaan testidata, mieluiten yhteistyössä asiakkaan kanssa. Tällöin testidata on mahdollisimman lähellä oikeaa dataa. Testiympäristöt rakennetaan tässä vaiheessa.

3.4.4 Testien suorittaminen

Varsinaisen testaamisen suorittamisessa pyritään käyttämään soveltuvasti eri testausmenetelmiä. Testejä suoritetaan sekä ihmisten tekeminä, että sopivin osin myös automaatiota hyväksi käyttäen.

3.4.5 Raportointi ja tulosten analysointi

Testauksesta syntyy useita raportteja. Testauksen aikaisia väliraportteja ja lopulta myös loppuraportti. Tärkeintä on saada oikea tieto oikeille henkilöille oikeaan aikaan. Sovelluskehittäjää kiinnostavat tietenkin hänen omasta koodistaan löydetyt bugit, kun taas projektipäällikkö on kiinnostunut yleisluontoisemmasta tilastotiedosta. Loppuraportin tuloksien analysoinnilla saadaan arvokasta tietoa siitä, missä vaiheissa syntyy eniten virheitä. Tällöin voidaan seuraavassa vaiheessa tai seuraavissa projekteissa keskittyä näiden kohtien parantamiseen.

3.5 Testaaja

3.5.1 Testausalan tehtävien toimenkuvat

Testauksen parissa työskentelevien henkilöiden työnkuvat voivat poiketa toisistaan paljonkin. Puhtaasti testauksen parissa työskentelevän henkilön tehtävät myös vaihtuvat urakehityksen myötä. Aloitteleva testaaja yleensä ainoastaan suorittaa testejä ja laatii niistä raportteja. Kokemuksen karttuessa painopiste siirtyy testien toteuttamisesta niiden suunnittelemiseen ja uusien testaajien kouluttamiseen. Lopulta pätevä testaaja voi päästä suunnittelemaan kokonaisen projektin testausta ja laatimaan testaussuunnitelmia.

Johtotehtäviin pyrkivät testaajat voivat nousta projektipäälliköiksi tai testausorganisaation johtoon. Näissä tehtävissä hän yleensä johtaa testausryhmää tai koordinoi koko yrityksen testausresursseja. (BCS Certificationsin www-sivut.)

3.5.2 Hyvän testaajan ominaisuuksia

Testaajan tärkein ominaisuus on kiinnostus analysointiin ja testaukseen. Hänen tulee siis pitää siitä, mitä tekee ja olla jatkuvasti valmis sopeutumaan ja oppimaan uusia asioita. Pelkkä innostus ja kiinnostus eivät kuitenkaan riitä pitkälle. Testaajan tulee olla älykäs ja kyetä loogiseen päättelyyn, jotta hän osaa kohdentaa testauksen oikeisiin asioihin ja nähdä syy-seuraussuhteita. Samalla testaajan tulisi kuitenkin olla myös luova ja osata ajatella asioita täysin erilaisista näkökulmista.

Siitä, kuinka hyvä ohjelmoija testaajan tulee olla, ei olla täysin yksimielisiä. Toiset asiantuntijat ovat sitä mieltä, että aivan perustason ohjelmointitaidot riittävät, kun taas toisten mielestä testaajan tulee olla vähintää keskitason ohjelmoija. Parhaimmat testaajat ovat yleensä myös huipputason ohjelmoijia.

Testaajalla tulee olla hyvät projekti- ja ryhmätyötaidot. Testaus on osa ohjelmistoprojektia, joten testaajan tulee ymmärtää oma merkityksensä projektin onnistumisessa. Hänen tulee kyetä priorisoimaan tehtäviään ja kommunikoimaan niin toisten testaajien, ohjelmistokehittäjien kuin projektin johdonkin kanssa. Huipputason testaajan voi jopa nähdä virheiden taakse ja löytää ohjelmiston rakenteesta tai toteutustavasta ongelmakohtia. (McCaffrey, 2008.)

4 TESTIPROSESSI JA TESTAUSMENETELMÄT

4.1 Testiprosessi ja testauksen tasot

Eri testit voidaan jakaa joko ohjelmiston kehittämisen vaiheen mukaan tai niiden laajuuden perusteella. Yleinen jako on:

- yksikkötestaus
- integraatiotestaus
- järjestelmätestaus
- hyväksyttämistestaus.

Jako perustuu siihen, että esimerkiksi yksikkötestausta voidaan suorittaa jo aikaisessa vaiheessa ja siinä testattavat kokonaisuudet ovat kooltaan pienempiä kuin myöhemmissä vaiheissa. Järjestelmätestausta ei tietenkään voida suorittaa, ennen kuin ohjelmisto on kyllin valmis. (SWEBOKin www-sivut 2014.)

4.1.1 Yksikkötestaus

Yksikkötestauksessa (unit testing) testataan ohjelmiston pieniä toiminnallisia osia. Oliopohjaisessa ohjelmoinnissa tämä tarkoittaa yleensä luokka-tasoa. (Binder, 1999 45.) Yksikkötestauksesta tarkemmin luvussa 5.

4.1.2 Integraatiotestaus

Integraatiotestauksessa (integration testing) testataan ohjelmiston eri komponenttien yhteistoimintaa. Käytännössä siis testataan jo toimivaksi todettujen osien välisiä rajapintoja. Integraatiotestausta voidaan lähestyä kahdella tavalla. Alhaalta ylös -tavassa (bottom up) ikään kuin rakennetaan pohjaa korkeamman tason komponenteille testaamalla ensin alimman tason komponentit ja edeten hierakiassa ylöspäin, kunnes koko ohjelmisto on testattu. Toinen vaihtoehto on ylhäältä alas -menetelmä (top down), missä lähdetään korkeimman tason komponenteista ja testataan kaikki mahdolliset haarat, kunnes koko ohjelmisto on testattu. (Beizer, 1990, 21.)

4.1.3 Järjestelmä- ja hyväksyttämistestaus

Järjestelmätestauksessa (system testing) testataan valmiin ohjelmiston toimintaa. Yleensä järjestelmätestaukseen ei siirrytä, ennen kuin integraatiotestaus on suoritettu, joten järjestelmätestauksen pääpaino on yleensä validoinnissa. Järjestelmätestaus on osa ohjelmistotalon sisäistä laadunvarmistusta. (Software Testing Classin www-sivut 2013.)

Hyväksyttämistestauksessa on paljon samoja piirteitä kuin järjestelmätestauksessa. Tavoitteena on saada todettua, että järjestelmä on valmis ja toteuttaa sille asetetut vaatimukset. Asiakas on hyväksyttämistestauksessa mukana. (ISTQB:n www-sivut 2014b.)

4.1.4 Alpha- ja beta-testaus

Alpha-testaus on eräs tapa toteuttaa järjestelmä- tai hyväksyttämistestausta. Alpha-testaus on yleensä ohjelmistotalon sisäistä toimintaa ja jakautuu usein kahteen vaiheeseen. Alpha 1 -vaiheessa testausta suorittavat pääasiassa ohjelmiston kehittäjät kehityksen ohessa. Tavoitteena on löytää bugeja nopeasti. Alpha 2 -vaiheessa testausta suorittaa testaus- tai laadunvarmistushenkilöstö (QA, quality assurance). Alpha-testauksen on tarkoitus simuloida loppukäyttäjän toimintaa mahdollisimman tarkasti. (ISTQB:n www-sivut 2014c.)

Beta-testaus seuraa alpha-testausta. Beta-testivaiheessa ohjelmisto lähestyy jo julkaisuhetkeään ja sen valmiusaste on korkea. Kriittisiä bugeja ei enää tulisi löytyä. Beta-testi on yleensä kohdennettu laajemmalle testiryhmälle kuin alpha-testaus. (CenterCoden www-sivut 2011a.) Joskus suoritetaan jopa niin kutsuttu julkinen beta-testaus, jossa potentiaalisia loppukäyttäjiä pyritään saamaan käyttämään tuotetta. Julkinen beta-testi voi olla joko avoin tai suljettu (open / closed beta). Avoimeen testiin voi osallistua kuka tahansa, suljettuun ainoastaan kutsutut henkilöt. (CenterCoden www-sivut 2011b.)

Sekä alpha- että beta-testausta suoritetaan useimmiten suoraan hyllystä myytäville tuotteille (ISTQB:n www-sivut 2014c).

4.1.5 Ei-toiminnallinen testaus

Ei-toiminnallisessa testauksessa testataan ohjelmiston rakenteellista laatua. Testit ovat yleensä sellaisia, että niiden tulokset voidaan helposti esittää lukumuodossa. Tyypillisiä ei-toiminnallisia testejä ovat vasteaikojen ja kuormituksen keston

testaaminen. Ei toiminnallista testausta sisältyy yleensä kaikkiin edellä esiteltyihin vaiheisiin, mutta hieman eri lähtökohdista. (ISTQB:n www-sivut 2014d.)

Esimerkiksi yksikkötestauksessa voidaan mitata tietyn toiminnon vasteaikaa, kun taas avoimessa beta-testissä voidaan testata järjestelmän toimintaa suurella käyttäjämäärällä.

4.1.6 Regressiotestaus

Ohjelmiston muuttuessa, esimerkiksi päivityksen yhteydessä, voivat muuttuneet komponentit aiheuttaa bugeja myös muuttumattomissa komponenteissa. Regressiotestauksen tarkoitus on saada selville, vaikuttaako ohjelmiston yhteen osaan tehty muutos ohjelmiston muiden osien toimintaan. (Savenkov, 2008, 386.)

4.2 Testaustavat

Lähes kaikki testaus voidaan jakaa kahteen päälähestymistapaan: black box ja white box -testaamiseen.

4.2.1 Black box- ja white box -testaus

Black box -testaus on lähestysmistapa testaukseen, jossa testaaja ei tiedä testattavan ohjelmiston tai sen osan teknistä toteutusta, eikä hänellä ole pääsyä sen lähdekoodiin. Testaus perustuu siis täysin testaajan havainnoimaan käyttäytymiseen. Testaaja antaa ohjelmistolle syötteitä (input) ja saa vastaukseksi tulosteita (output). Testaaja tuntee ohjelmiston määrittelyn, joten hän tietää miten ohjelmiston tulisi reagoida eri tilanteissa.

White box -testauksessa testaajalla on pääsy testattavan kohteen lähdekoodeihin ja hän tuntee sen teknisen rakenteen. Hän voi siis esimerkiksi testata suoraan haluamaansa metodia haluamillaan parametreillä.

Black box -testaus tunnetaan myös nimellä toiminnallinen testaus (functional testing) ja white box -testaus nimellä rakenteellinen testaus (structural testing). (Williams, 2006, 37 - 38.) Huomataan, että nämä kaksi viittaavat suoraan Pressmanin (2005, 388) kahteen laatutekijään, toiminnalliseen ja rakenteelliseen laatuun.

4.2.2 Tutkiva testaus

Tutkiva testaus (exploratory testing) on eräs tapa testata ohjelmistoa. Siinä testaajalla ei ole tarkkaan määriteltyjä testitapauksia, vaan hän luo ne itse testauksen edetessä. Testaaja pyrkii selvittämään, miten ohjelmisto toimii ja löytämään siitä puutteita. Tutkiva testaus on, "yhtäaikaista oppimista, testien suunnittelua ja toteutusta." (Bach, 2002, 2 - 3.) Tutkivaa testausta pidetään usein black box -testauksen osana, mutta Kanerin (2008) mukaan se on kokonaan oma lähestymistapansa, jota voidaan soveltaa mihin tahansa testaukseen. Tärkeintä on testaajan omistautuminen testaukselle.

4.3 Yhteenveto testauksen tasoista ja testausmenetelmistä

Taulukko 1. Yhteenveto testauksen tasoista ja testausmentelmistä (Williams, 2006, 42).

Taso	Kohde	Laajuus	Testaustapa	Kuka testaa?
Yksikkö-testaus	Matala taso Lähdekoodi	Pieni yksikkö, luokka	White box	Kehittäjät
Integraatio-testaus	Matala ja korkea taso	Vähintään useita luokkia	White box, black box	Kehittäjät
Järjestelmä- ja hyväksyttämis-testaus	Vaatimus-määrittely	Koko ohjelmisto	Black box	Testaajat, myöhemmin asiakas
Beta-testaus	Ad hoc	Koko ohjelmisto	Black box	Asiakas, potentiaalinen asiakas
Regressio-testaus	Muutokset	Koko ohjelmisto	White box, black box	Kehittäjät, testaajat

5 YKSIKKÖTESTAUS

5.1 Yksikkötestauksen laajuus ja tavoite

Yksikkötestauksessa tarkastellaan yksittäisiä, pieniä osia ohjelmistosta. Olio-ohjelmoinnissa tarkasteltava kohde on yleensä luokka. Proseduraalisessa ohjelmoinnissa kohde voi olla yksittäinen funktio tai kokonainen komponentti. (Xie, Taneja, Kale & Marinov, 1.)

Yksikkötestaus on yleensä white box -testausta, eli testaajalla on pääsy ohjelmiston lähdekoodiin. Yksikkötestausta voidaan suorittaa tiettyjen komponenttien osalta myös black box -testauksenakin. Yksikkötestauksen suuria etuja on automatisointi. Tällöin kertaallen kirjoitetut testit voidaan ajaa uudelleen useita kertoja, esimerkiksi aina muutosten jälkeen. Tällä tavoin säästetään merkittävästi aikaa.
(Parasoft, 2014, 1 - 4.)

Yksikkötestauksen perimmäinen tarkoitus on varmistaa, että ohjelmiston komponentit toimivat yksinään. Hyvin suoritetun yksikkötestauksen jälkeen siirryttäessä integraatiotestaukseen voidaan siis olettaa komponenttien itsessään toimivan. Tällöin integraatiotestauksessa voidaan keskittyä komponenttien välisen toiminnan testaamiseen. Virheiden löytäminen on myös helpompaa pienemmissä kokonaisuuksissa.

Otetaan esimekiksi tilanne, jossa on kaksi ohjelmistokomponenttia. Nämä komponentit päätetään testata suoraan integraatiotestauksella. Testi päätyy virheeseen. Mahdollisia syitä ovat ainakin:

- Virhe komponentissa 1
- Virhe komponentissa 2
- Virhe molemmissa komponenteissa
- Virhe komponenttien välisessä rajapinnassa
- Virhe testissä

Mikäli komponentit olisi yksikkötestattu, voisi kolme ensimmäistä mahdollisuutta rajata pois. (Microsoft Developer Networkin www-sivut 2013a.)

5.2 Yksikkötestausmenetelmät

Yksinkertaisimmillaan yksikkötestaus on sitä, että ohjelmoija kirjoittaa luokan ja sille testimetodin tai -metodeja. Nämä metodit käyttävät luokan toimintoja tai simuloivat rajapintaa antaen tai vastaanottaen syötteitä. (Microsoft Developer Networkin www-sivut 2013a.) Näin yksittäinen kehittäjä voi jo luoda yksikkötestauksen pohjan. Joissain ohjelmointikielissä on suoraan tuki yksikkötestien kirjoittamiselle.

5.3 Esimerkki ohjelmointikielen tuesta yksikkötestaukselle

Seuraavassa yksinkertainen esimerkki C#-ohjelmointikielen tarjoamasta tuesta yksikkötestaukselle. Esimerkissä luodaan yksinkertainen luokka, joka kuvaa käyttäjän tietoja. Lisäksi luodaan sille yksikkötestiluokka. Esimerkki on toteutettu käyttämällä Visual Studio 2012 Ultimate -kehitystyökalua.

Kayttaja-luokka sisältää merkkijonona käyttäjätunnuksen, totuusarvona aktiivisuuden ja merkkijonolistana roolit. Konstruktorissa luodaan olio oletusarvoilla.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace YksikkotestiEsimerkki.Model
{
    public class Kayttaja
    {
        //ominaisuudet
        public string Kayttajatunnus { get; set; }
        public bool Aktiivinen { get; set; }
        public List<string> Roolit { get; set; }

        //konstruktori
        public Kayttaja()
        {
            Kayttajatunnus = "";
            Aktiivinen = false;
            Roolit = new List<string>();
        }

    }
}
```

Kuvio 2. Kayttaja-luokka.

Testiluokassa luodaan olio ja tarkistetaan, ovatko arvot oikein. Testiluokka on täysin erillinen testattavasta luokasta, joten lopulliseen ohjelmistoon ei jää ylimääräistä ohjelmakoodia.

Visual Studiota käytettäessä testausluokat ovat yleensä omissa projekteissaan kokonaisuuden (solution) sisällä.

```
using System;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using YksikkotestiEsimerkki.Model;
using System.Collections.Generic;

namespace YksikkotestiEsimerkkiTestit
{
    [TestClass]
    public class Kayttajatesti
    {
        //Tarkistaa, ovatko ominaisuudet alustettu oikein kun uusi Kayttaja-luokan ilmentymä luodaan.

        [TestMethod]
        public void TestMethod1()
        {
            Kayttaja uusiKayttaja = new Kayttaja();

            //Assert-luokalla on useita metodeja, joilla voidaan tarkastella erinäisiä ehtoja.
            //Viimeisenä oleva merkkijono on virheilmoitus, joka annetaan mikäli ehto ei toteudu.

            Assert.AreEqual("", uusiKayttaja.Kayttajatunnus, "Pitäisi olla tyhjä merkkijono");
            Assert.IsFalse(uusiKayttaja.Aktiivinen, "Pitäisi olla false");
            Assert.IsNotNull(uusiKayttaja.Roolit, "Ilmentymä pitää luoda");
            Assert.IsInstanceOfType(uusiKayttaja.Roolit, typeof(List<string>), "Tulisi olla List<string>");
        }
    }
}
```

Kuvio 3. Kayttajatesti-luokka.

Kun yksikkötesti on luotu, osaa kehitystyökalu suorittaa testit ja ilmoittaa niiden lopputuloksen. Edellä esitelty esimerkki on hyvin yksinkertainen, mutta antaa kuvan siitä, miten hyvin ohjelmointikieli voi tukea yksikkötestausta. (Microsoft Developer Networkin www-sivut 2013b; NetCellin www-sivut 2012.)

6 TESTITAPAUS

6.1 Testitapauksen määritelmä

Testitapaus (test case) on dokumentoitu kokoelma ohjeita ja syötteitä, joka asettaa lähtökohdat ohjelmiston testaamiselle. Testausta suorittava testaaja suorittaa testausta testitapausten pohjalta. Testitapaukset kertovat, miten ohjelmistoa halutaan testata. Testitapauksen sisältönä on yleensä vähintään suoritettavat askeleet ja niiden odotetut tulokset. Askeleella (step) tarkoitetaan yksittäistä toimintoa, joka tulee suorittaa. Myös muita tietoja voidaan antaa, joko täsmentämään testausta tai helpottamaan testauksen hallintaa ja raportointia. Testitapauksessa tyypillisesti määriteltyjä tietoja ovat:

- testitapauksen tunniste (ID)
- testitapauksen sanallinen kuvaus
- askel tai askeleet
- askelten odotetut tulokset
- käytettävät syötteet ja asetukset
- tieto siitä, suoritettiinko testi onnistuneesti
- testaajan huomiot

Testitapauksen ydin ovat sen askeleet, muut tiedot ovat lähinnä täsmentäviä. (Kaner 2003, 1 - 2.) Testitapauksia voidaan käyttää kaikkiin testausmuotoihin. Täten testitapaukset voivat olla lähtökohdaltaan black box tai white box -testausta. (Parasoft, 2014, 1 - 4.) Testitapausten kokoelmaa kutsutaan nimellä test suite (PC Magazinen www-sivut).

6.2 Testitapauksen kattavuus

Testitapauksen tulisi olla kattava, jolloin testitapausten määrän olisi mahdollisimman pieni. Toisaalta testitapuksen pitäisi olla tarkkaan määritelty. Nämä kaksi tavoitetta ovatkin usein ristiriidassa keskenään. Niinpä testausta suunniteltaessa tulisi valita, kumpaan pyritään: mahdollisimman tarkkoihin vai mahdollisimman kattaviin testitapauksiin. Tähän vaikuttaa myös olennaisesti se, ovatko testien kirjoittajat ja suorittajat eri henkilöitä.

Vapaissa, löyhemmin määritellyissä, testitapauksissa testi voi tulla suoritetuksi eri tavalla eri testikerroilla. Tällöin testin askelten määrä on paljon pienempi kuin tarkkaan määritellyssä testitapauksessa, askelia voi olla jopa vain yksi. Testitapauksen sanallinen kuvaus sen sijaan on merkittävämmässä roolissa. Askelten vähäisyyden vuoksi virheen sattuessa sen kohdentaminen voi olla vaikeampaa. Toisaalta näin saadaan suurempi kattavuus, sillä jokainen testaaja saattaa testata saman tapauksen hieman eri tavalla. (Niittyviita, 2010.)

Tarkkaan määritellyissä testitapauksissa testaus tulee suoritettua aina samalla tavalla. Askeleet ja syötteet on määritelty täsmällisesti ja ohjemaisesti. Tällöin mahdollinen virhe on helppo kohdentaa esimerkiksi toteamalla, että testauksen tulokset poikkeavat odotetuista tuloksista tietyssä kohdassa. Tarkoissa testitapauksissa testaajalla ei välttämättä tarvitse edes olla syvällistä tuntemusta testattavasta ohjelmistosta. Tällöin testaus voidaan jopa ulkoistaa.

6.3 Esimerkki testitapauksesta

Taulukko 2. Esimerkki vapaasta testitapauksesta (ReQtestin www-sivut 2012).

ID	05	
Nimi	Tulostus	
Prioriteetti	Normaali	
Kuvaus	Tiedoston tulostuksen toiminnan varmistaminen.	
Testidata	Käytettävä tunnus ja salasana ovat testi@testi.com ja testi123. Tiedosto on nimeltään Tulostustesti.pdf	
Alkutilanne	Ohjelma on käynnissä, olet tervetuloa-ikkunassa.	
	Kuvaus	Odotettu tulos
Askel 1	Tulosta tiedostosta Tulostustesti.pdf yksi kopio.	Tulostuu yksi sivu, joka on Tulostustesti.pdf:n mukainen.

Taulukko 3. Esimerkki tarkkaan määritellystä testitapauksesta (ReQtestin www-sivut 2012).

ID	05-01	
Nimi	Tulostus 01	
Prioriteetti	Normaali	
Kuvaus	Tiedoston tulostuksen toiminnan varmistaminen.	
Testidata	Käytettävä tunnus ja salasana ovat testi@testi.com ja testi123. Tiedosto on nimeltään Tulostustesti.pdf	
Alkutilanne	Ohjelma on käynnissä, olet tervetuloa-ikkunassa.	
	Kuvaus	Odotettu tulos
Askel 1	Valitse yläpalkista kohta tiedostot.	Tiedostot-ikkuna aukeaa.
Askel 2	Avaa kansio Testit.	Kansio aukeaa.
Askel 3	Klikkaa Tulostustesti.pdf	Rivi värjäytyy siniseksi.
Askel 4	Klikkaa yläpalkista tulosta-painiketta.	Tulostusikkuna aukeaa.
Askel 5	Valitse tulostimeksi Tulostin1 ja kopioiden määräksi 1.	
Askel 6	Paina ok.	Tulostin 1 tulostaa yhden sivun.
Askel 7	Varmista, että tulostettu sivu on Tulostustesti.pdf:n mukainen.	Sivu on Tulostustesti.pdf:n mukainen.

Ensimmäinen tapa ohjaa testausta tutkivan testauksen suuntaan ja hyvä testaaja löytääkin varmasti useita eri tapoja testata kohteena olevaa toimintoa. Tarkasti määritellyistä testitapauksista pitäisi jokaisesta mahdollisesta tavasta tehdä oma testitapauksensa. Vapaammin määritellyissä taasen yksi testitapaus voi pitää sisällään useita tapoja tehdä sama asia. Tässä mallissa testaajan rooli korostuu. Hänen tulee tuntea testauksen kohde niin hyvin, että tietää mahdollisia tapoja olevan useita. Muutoin jokin toiminnallisuus voisi jäädä kokonaan testaamatta.

Jälkimmäinen testitapaus suorittuu samalla tavalla testaajasta ja testikerrasta riippumatta. Virheen sattuessa on helppo raportoida virheen tapahtuneen tietyssä askeleessa. Jälkimmäisessä testitapauksessa voi testissä olla suurtakin hajontaa. Tulostus voi esimerkiksi olla mahdollista toteuttaa myös klikkaamalla haluttua tiedostoa hiiren kakkospainikkeella ja valita avautuvasta context menusta tulostustoiminto. (Niittyviita, 2010; ReQtestin www-sivut 2012.)

6.4 Testitapaustenhallinta

Yksinkertaisimmillaan testitapausten hallintaan riittä yksinkertainen taulukko. Esimerkiksi taulukkolaskentaohjelmalla voi helposti hallita pienen projektin testitapaukset. Laajoissa ohjelmistoissa testitapauksia voi kuitenkin olla todella paljon. Lisäksi testaajia ja kehittäjiä on useita. Tällöin ongelmaksi muodostuu se, kuinka pystytään hallitsemaan niitä kaikkia. Miten voidaan ohjata tietyt tapaukset tietylle testaajalle? Miten saadaan ajankohtainen tieto testauksen tilasta? Missä ohjelmistomoduulissa on eniten virheitä? Mihin vaatimukseen testitapaus liittyy? Miten testien aikataulutus järjestetään? Pelkkä taulukko ei usein enää riitä tällaisessä tilanteessa.

Ongelman ratkaisuun on kehitetty omat työkalunsa, testitapaustenhallintatyökalut (test case management tools). Näillä ohjelmistoilla projektipäälliköt ja testauksen organisoinnista vastuussa olevat henkilöt voivat helposti jakaa työtehtäviä testaajille ja seurata testien edistymistä. Testien kehittäjät saavat testinsä tallennettua järjestelmään ja ne etenevät automaattiseti hyväksyttäviksi. Testaajat saavat tietoonsa, milloin minkäkin testin tulisi olla valmis ja mikä on minkäkin testin prioriteetti. Näin he voivat suunnitella ajankäyttönsä mahdollisimman tehokkaasti.

Työkalun avulla testejä voidaan ryhmitellä. Näin voidaan rakentaa rakenteita joiden avulla voidaan todeta tietyn osa-alueen olevan testattu. Esimerkiksi sisäänkirjautumiseen liittyvät testit voidaan liittää yhteen omaksi ryhmäkseen. Kun nämä kaikki on suoritettu hyväksytysti, voidaan todeta sisäänkirjautumisen toimivan tällä hetkellä. Samalla tavalla voidaan järjestää esimekiksi järjestelmätestaus.

Testitapaukset voidaan liittää järjestelmän vaatimuksiin ja kun nämä kaikki on testattu hyväksytysti voidaan siirtyä eteenpäin. (Kallio, 2005.)

7 PROJEKTIN SUUNNITTELU

7.1 Projektin tehtävät

Projektin tehtävät voitiin jakaa seuraavasti:

- testauksen teoriaan ja nykyaikaisiin menetelmiin perehtyminen
- Aton PDM -tuotteen käyttöön perehtyminen
- testitapauskirjaston luominen
- yrityksen tarpeisiin soveltuvien testauksenhallintajärjestelmien etsiminen ja esittely
- valitun järjestelmän asennus ja käyttöönotto.

7.1.1 Teoriaan perehtyminen

Testauksen teoriaan perehtyminen oli projektin ensimmäinen vaihe. Tämä oli mielestäni erittäin hyvä asia, sillä testaus on aiheena todella laaja. Koulutusohjelmassani ei ollut ollut tarjolla testausopintojaksoa, mutta testausta oltiin toki käsitelty useammankin opintojakson aikana. Kun teoriaan perehtyminen otettiin osaksi projektia, minun oli mahdollista syventyä asiaan tarkemmin. Kun vielä lisäksi oli tiedossa minkälaista järjestelmää ollaan testaamassa, oli perehtyminen helppoa kohdentaa oikeisiin asioihin. Tavoitteena oli testauksen käsitteistön ja periaatteiden opiskelu.

7.1.2 Aton PDM -tuotteeseen perehtyminen

Olin ollut jo työni puolesta kehittämässä Aton PDM:ää. Lisäksi Modultek käyttää Atonia myös omassa toiminnassaan. Täten tuotteen peruskäyttö ja -toiminnot olivat jo entuudestaan tuttuja. Tuotteeseen perehtymisvaiheessa tarkoitus olikin keskittyä ominaisuuksiin, joita Modultekin omassa käytössä ei tarvita, mutta jotka ovat asiakkaille hyvin tärkeitä. Tärkeimmäksi näistä nousi nimikkeiden käsittely.

7.1.3 Testitapauskirjaston luominen

Testitapauskirjaston tavoitteena oli olla järjestetty lajitelma testitapauksia, joilla saadaan Atonin perusominaisuudet testattua. Koska Aton on erittäin laaja järjestelmä, päätettiin tässä projektissa keskittyä nimikkeiden ja dokumenttien hallintaan sekä yleisiin toimintoihin. Yleisiin toimintoihin kuuluvat käytössä vastaan tulevat toiminnot, jotka eivät kuitenkaan ole varsinaisesti tuotteen päätehtävä. Näitä ovat muun muassa sisäänkirjautuminen, salasanan vaihto, kansiorakenteiden hallinta, työalueiden hallinta, erilaiset hakutoiminnot, drag and drop -toiminnallisuus ja vastaavat.

7.1.4 Soveltuvien järjestelmien etsiminen ja esittely

Tässä vaiheessa tavoitteena oli etsiä kolmesta viiteen järjestelmää, jotka sopisivat Modultekin kokoisen yrityksen käyttöön. Alkuvaiheessa järjestelmän valintaan vaikuttavia kriteerejä ei vielä annettu, mutta toiveen esitettiina open source -ratkaisujen suosimista.

7.1.5 Asennus ja käyttöönotto

Valittu järjestelmä tuli valinnan jälkeen asentaa ja ottaa käyttöön. Tähän vaiheeseen kuului palvelimen asentaminen, itse järjestelmän asentaminen, käyttäjien luonti, testitapauskirjaston siirtäminen järjestelmän käytettäväksi ja järjestelmän toiminnan testaus. Lisäksi vaiheeseen kuului järjestelmän käytön esittelyä ja koulustusta.

7.2 Projektin aikataulu ja resurssit

Projektille oli varattu aikaa noin kaksi kuukautta. Olin projektin ainoa henkilöresurssi, joskin muitakin henkilöitä oli mukana kokouksissa ja konsultoimassa ratkaisuja.

Olin itse vastuussa projektin aikatauluttamisesta. Vastaavankaltaista projektia en ollut aikaisemmin suunnitellut, joten ajankäytön suunnittelu oli vaikeaa, mutta opettavaista. Alunperin suunnittelin ajankäytön seuraavasti:

- teoriaan perehtyminen, 1 viikko
- Atoniin perehtyminen, 1 viikko
- testitapauskirjasto, 3 viikkoa
- testauksenhallintajärjestelmien etsiminen ja esitteleminen, 1 viikko
- asennus, käyttöönotto ja testaus, 2 viikkoa.

7.3 Projektin rajaus ja riskit

Projektin päätyttyä tulisi käytössä olla testitapausten hallintajärjestelmä ja testitapauskirjasto. Testitapauksien osalta painopiste on nimikkeisiin ja dokumentteihin liittyvien toimintojen kattamisella. Lisäksi testitapausten tulee kattaa yleistoiminnot. Testitapaukset ovat manuaalisesti testattavia, mutta tulee luoda soveltuvilta osin niin, että mahdollisuus automaattiseen testaamiseen tulevaisuudessa on olemassa.

Projektin riskeiksi arvioitiin lähinnä henkilöresurssien siirtäminen muihin tehtäviin. Projektille varattiin aikaa noin kaksi kuukautta. Järjestelmä otettaisiin todelliseen käyttöön vasta noin kuuden kuukauden kuluttua projektin alusta.

8 PROJEKTIN TOTEUTUS

8.1 Perehtymisvaihe

Käytännössä perehtyminen alkoi testauksen teoriaa käsittelevään kirjallisuuteen tutustumalla. Alkuvaiheessa luin pääasiassa yleisteoksia, jotka kuvailivat testauksen eri vaiheita ja menetelmiä. Haastattelin myös alalla pitkään olleita kollegoja. Heiltä sai arvokasta näkemystä niin ICT-alasta ja testauksesta yleisesti, kuin myös tarkemmin juuri Modultekin tuotteiden testauksesta. Kun olin mielestäni saanut riittävät perustiedot, siirryin myös uudenlaisiin lähteisiin. Näitä olivat muun muassa useat testausaiheiset blogit, joissa pohdittiin esimerkiksi testitapausten määrää ja laajuutta.

Perehtymisvaiheeseen kuului myös Atonin toimintaa perehtyminen. Atonin perustoiminnat olivat minulle jo entuudestaan tuttuja, joten perehtyminen oli helppo kohdentaa oikeisiin asioihin. Yksi näistä oli yksi testauksen painopisteiksi suunnittelussa valituista kohdista, nimikkeiden käsittely. Nimikkeiden käsittelyyn (luomiseen ja muokkaamiseen) sain perehdystystä Modultekin konsulteilta, jotka tuntevat mekaniikka- ja elektroniikkasuunnittelua.

Kuvio 4. Aton PDM 6.1 -pääikkuna.

Lisäksi järjestin palaverin Modultekin asiakaspalvelun kanssa. Tässä palaverissa he toivat esiin asioita, joihin heidän mielestään tulisi kiinnittää huomiota testauksessa. Nämä asiat liittyivät pääosin yleisiin toimintoihin, kuten Atonin pääikkunassa (kuvio 4) tehtäviin hakuihin.

8.2 Testitapauskirjaston luominen

Koska testitapaustenhallintajärjestelmää ei vielä ollut käytössä, aloitin testitapauskirjaston luomisen taulukkolaskentaohjelmalla. Aloitin työn listaamalla Atonin eri toimintoja. Tässä apunani olivat Atonin tekniset kuvaukset. Lisäksi kävin järjestelmällisesti läpi eri näyttöjä ja niissä olevia toimintoja. Näin sain listattua suuren määrän toimintoja.

Seuraavaksi lähdin suunnittelemaan testitapauksia. Tässä vaiheessa pidimme palaverin tuotekehitysosaston kanssa ja päädyimme testitapauksien tarkkuudessa niin sanottuihin vapaisiin (eli ei tarkasti määriteltyihin) testitapauksiin. Tähän ratkaisuun päädyttiin, sillä testaajina tulee toimimaan Modultekin työntekijöitä kaikilta osastoilta. Koska Atonia käytetään monin tavoin, ja useiden toimintojen suorittamiseenkin on useita tapoja, saadaan vapailla testitapauksilla näin hyvinkin suuri kattavuus. Testitapausten luonnissa oli lisäksi perusperiaatteena saada mahdollisimman iso osa listatuista toiminnoista testattua mahdollisimman pienellä määrällä testitapauksia. Täten päätettiin yksinkertaisia testitapauksia yhdistää. Painopisteinä olivat suunnitellusti dokumentti, nimike ja yleiset toiminnot. Kirjoitin testitapauksia myös muihin Atonin osa-alueisiin, mutta painopisteiksi valituista alueista tehtiin enemmän testitapauksia.

Testitapausten kirjoittamisvaiheessa perehtymiseen käytetty aika maksoi itseään takaisin. Vastaani tuli usein tilanteita, joissa en heti osannut tehdä tietyn toiminnon testaamiseksi hyvää testitapausta. Näissä tilanteissa saattoi tulla mieleen, että olin lukenut vastaavan kaltaisesta ongelmasta aikaisemmin esimerkiksi jostain blogista. Tätä kautta perehtymisestä oli todella hyötyä. Tietyissä tilanteissa kysyin myös

Modultekin muulta henkilöstöltä mielipiteitä ja apua jonkin toiminnallisuuden ymmärtämiseen. Testitapauksia syntyi lopulta kymmeniä.

Taulukossa 4 on esimerkki eräästä testitapauskirjaston testitapauksesta. Esimerkissä on yhdistetty kaksi yksinkertaista testitapausta yhdeksi, koska tavoitteena oli mahdollisimman pieni testitapausten kokonaismäärä. Testitapaus on kirjoitettu vapaaksi testitapauseksi, joskin testattavan kohteen luonteesta johtuen se on melko tarkkaan määritelty. Sisäänkirjautumiseen ei ole useita eri tapoja.

Taulukko 4. Esimerkki testitapauksesta, jossa yhdistyy kaksi tapausta

ID	G001	
Name	Login	
Priority	Normal	
Objective	Verify login functionality.	
Test data	Use Aton 6.2 beta client with beta credentials. Server: ...	
Pre-conditions	Aton 6.2 beta is installed.	
	Instrution	Result
Step 1	Start Aton.	Splash screen appears, login screen appears.
Step 2	Choose server.	Server can be chosen.
Step 3	Attempt to log in with incorrect username/password combination.	Error is displayed.
Step 4	Log in with correct credentials.	Log in is successful, main window opens.

Taulukossa 5 on esimerkki melko tyypillisestä testitapauksesta. Atonissa voi objektin, esimerkiksi dokumentin varata itselleen, jolloin sen muokkaaminen on mahdollista. Tässä testitapauksessa ei oteta kantaa siihen, miten dokumenttia muokataan. Tämä testitapaus on esimerkki tilanteesta, jonkalaisia kohtasin usein. Testitapauksen askelista osa on samoja kuin joissain toisissa testitapauksissa, mutta jotkin askeleet poikkeavat.

Kyseisessä tapauksessa vaihtuva kohta on askel numero kolme. Dokumentin vapautuksessa on neljä vaihtoehtoa, new document (luo muokatusta dokumentista kokonaan uuden dokumentin, säilyttäen vanhan ennallaan), new revision (luo uuden revision), update current (päivittää nykyisen) ja discard changes (hylkää muutokset). Kaikki neljä vaihtoehtoa tulisi tietenkin testata. Täysin vapaassa testitapauksessa viimeinen askel olisi voinut vain ohjeistaa vapauttamaan dokumentin. Todettiin kuitenkin olevan tässä tilanteessa järkevämpää tuoda esiin kaikki neljä vaihtoehtoa. Ratkaisin ongelman niin, että testitapaukset ryhmiteltiin yhteen. Tällöin testaaja huomaa, että kyseessä on samaan tilanteeseen liittyvät, mutta kuitenkin eri testitapaukset.

Taulukko 5. Esimerkki tyypillisestä testitapauksesta

ID	D003-3	
Name	Release document, update current	
Priority	Normal	
Objective	Verify release functionality	
Test data	Use Aton 6.2 beta client with beta credentials. Server: ...	
Pre-conditions	Have access to a document	
	Instrution	Result
Step 1	Reserve a document.	Document becomes reserved for you. Lock icon appears.
Step 2	Edit document.	
Step 3	Release document, choose update current.	Confirmation dialog opens. The changes update to the document. Document becomes unreserved. Lock icon disappears.

8.3 Soveltuvien testitapausten hallintajärjestelmien etsiminen ja valitseminen

Modultekin tarpeisiin soveltuvan testitapausten hallintajärjestelmän etsiminen alkoi yleistilanteen kartoittamisella. Testauksen tehostamiseen suunniteltuja työkaluja on huomattavan paljon.

Ensimmäisenä ideana mieleeni tuli Microsoftin Test Manager. Järjestelmä on erittäin monipuolinen ja intergroituu Visual Studioon. Tämä nousi myös ongelmaksi, sillä kaikilla testausta mahdollisesti suorittavilla ei ole Visual Studiota, joten Test Manager jäi pois laskuista.

Modultek oli tämän projektin kanssa samaan aikaan korvaamassa erästä vanhaa järjestelmää JIRA Agile -projektinhallintatyökalulla. Sain ehdotuksen selvittää mahdollisuutta käyttää JIRA Agilea testitapaustenhallintaan. Selvisi, että olisi mahdollista luoda esimerkiksi uusia issue typejä ja sub-taskeja, joilla voitaisiin hallita testitapauksia. Ongelmaksi muodostui kuitenkin JIRA-taskien kertaluontoisuus. Kun task on suoritettu, se on suoritettu. Tässä tavassa pitäisi siis testitapauksetkin luoda joka kerta uudestaan. Tämä olisi ollut toteutettavissa kopioimalla testitapauksia, mutta tämän todettiintuottavan turhaa työtä.

JIRAan liittyen tutkein siihen saatavilla olevien plug-inien käyttömahdollisuutta. Esimerkiksi Zephyr-niminen testauksenhallintatyökalu oli esillä, mutta todettiin liian kalliiksi.

Olin jo projektin alkuvaiheessa törmännyt Tarantula -nimiseen avoimen lähdekoodin työkaluun. Tarkemmalla tutkimisella se vaikutti Modultekin käyttöön erittäin soveltuvalta. Se mahdollistaa useiden testitapauskirjastojen luonnin, testien määräämisen tietyille henkilöille ja testien tulosten ja tilanteen seurannan. Lisäksi se on avoimen lähdekoodin ohjelmisto, mikä oli yksi Modultekin toiveista. Tarantula on saatavilla myös kaupallisena SAAS-ratkaisuna.

Tarantula valittiin käyttöönotettavaksi yksimielisesti esitykseni perusteella. Alkuperäinen tavoite oli hankkia testitapaustenhallintajärjestelmä PDM-osaston käyttöön. Esittelytilaisuudessa mukana olleet PIS-osaston edustajat olivat kuitenkin myös vakuuttuneita Tarantulasta. Yrityksessä päätettiinkin ottaa Tarantula käyttöön molemmille tuotekehitystä tekeville osastoille.

8.4 Järjestelmän asentaminen

Suoritin järjestelmän asentamisen yhteistyössä Modultekin IT-osaston kanssa. Tarantula vaatii web-palvelimen alustakseen. Asennukseen kuului siis palvelimen ja itse Tarantulan asennus.

Palvelimen käyttöjärjestelmäksi valittiin CentOS 6.4 Linuxin 32-bittinen versio, kuten asennusohjeissa suositellaan (Testia Tarantulan www-sivut, 2014). Palvelin asennettiin virtuaalisena Modultekin VMware -ympäristöön. Resursseiksi annettiin 1 vCPU, 4 GB keskusmuistia ja 40 GB kiintolevytilaa. Palvelimen asennus sujui mutkattomasti.

Ennen Tarantulan asennusta palvelimesta otettiin snapshot sekä asennettiin käyttöjärjestelmän uusimmat päivitykset ja VMware Tools -paketti. Lisäksi asennettiin VNC Server etätyöpöytäyhteyksiä varten. Tarantulan asennus suoritettiin asennusohjeen mukaisesti. Asennuksen valmistuttua ei Tarantula kuitenkaan käynistynyt, vaan antoi virheilmoituksen väärästä Ruby on Rails -versiosta. Pitkän tutkimisen jälkeen vika paikannettiin. Asennuksessa komennon "rvm use 1.9.3" jälkeen tuli virheilmoitus "1.9.3-p448 is not installed."

Tässä tilanteessa palattiin snapshottiin ja aloitettiin Tarantulan asennus alusta. Virheilmoitus tuli jälleen, mutta tällä kertaa siihen oli varauduttu. Virheilmoituksen jälkeen oikea versio asentui komennolla "rvm install ruby-1.9.3-p448" minkä jälkeen asennus jatkui ohjeiden mukaan "rvm use 1.9.3" -komennolla. Asennus oli erittäin mielenkiintoinen ja opettavainen prosessi, sillä en ollut aikaisemmin tehnyt vastaavan laajuista asennusta Linux-ympäristössä.

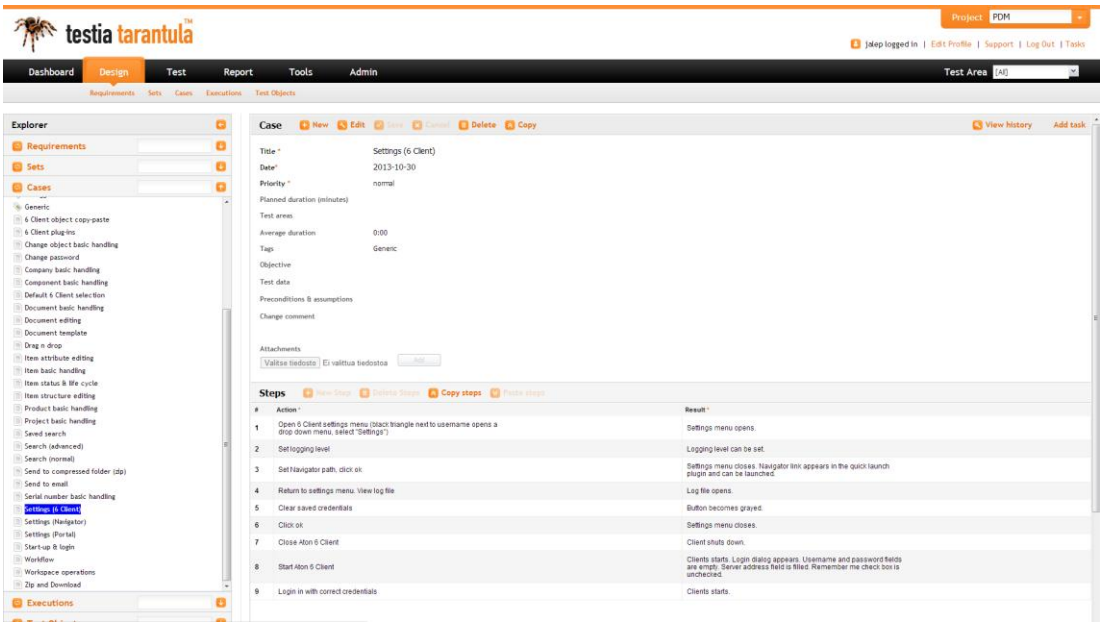
8.5 Järjestelmän käyttöönotto ja testaus

Kun järjestelmä oli asennettu, testasin sen toimivuutta luomalla testiympäristön ja testikäyttäjät Tarantulaan. Testasin testien luotia, muokkausta, poistamista, määräämistä ja järjestelemistä, sekä yleisiä toimintoja. Koska ensimmäinen

varsinainen Tarantulalla toteutettava testaus oli edessä vasta yli kahden kuukauden päästä. Päätettiin Tarantula ottaa testikäyttöön laajemmalla henkilömäärällä.

Tässä vaiheessa sain ohjeet luoda kaikille Modultekin työntekijöille käyttäjätunnukset ja asettaa heille oikeat roolit. Käyttäjätunnusten luominen onnistui helposti taulukkolaskentaohjelmaa hyväksikäyttäen.

Tämän jälkeen loin Tarantulaan omat alueensa PDM- ja PIS-osastoille, sekä Tarantulan itsensä testaamiseen ja kouluttamiseen tarkoitetun alueen. Koska oma työni keskittyi PDM-osastolle, päätettiin, että PIS-osasto ottaa omalta osaltaan Tarantulan käytön omaan hallintaansa. Pidimme palaverin jossa oli mukana lisäkseni PDM- ja PIS-osastoilta osastokohtaisten alueiden admineiksi valitut henkilöt, sekä IT-osastolta koko järjestelmän valvojaksi valittu henkilö. Tässä sovimme tietyistä yhteisistä käytännöistä.



Kuvio 5. Testitapausten luominen ja muokkaaminen Tarantulassa.

Seuraavaksi siirsin kirjoittamani testitapaukset Tarantulaan. En löytänyt työhön helposti toteutettavaa automaatiota, joten suoritin työn käsin. Tarantulan käyttöliittymä on yksinkertainen, mutta kattava. Kuviossa 5 näkyy vasemmalla lista jo luoduista testitapauksista ja oikealla suuren osan näytöstä vievät valittuna olevan testitapauksen tiedot.

Tämän jälkeen järjestin vielä demonstraation Tarantulan toiminnasta. Järjestelmä todettiin helppokäyttöiseksi, joten varsinaista koulutustilaisuutta ei järjestetty tässä vaiheessa.

9 YHTEENVETO

9.1 Tavoitteiden täyttyminen

Projektin tavoitteena oli luoda testitapaukset Atonin testaamiseen, sekä asentaa ja ottaa käyttöön järjestelmä testauksen hallintaan. Kun Tarantula palvelimineen oli asennettu ja testitapauskirjasto siirretty Tarantulaan, todettiin tavoitteiden täyttyneen ja projektin päättyneen. Projektille oli varattu aikaa noin kaksi kuukautta ja projekti valmistui vajaassa kahdeksassa viikossa. Projekti valmistui siis aikataulussaan.

9.2 Merkitys yritykselle

Jokainen yritys pyrkii tietenkin parhaaseen mahdolliseen tulokseen. Projektin tuloksena syntynyt testitapauskirjasto ja käyttöönotettu Tarantula-testitapaustenhallintajärjestelmä pyrkivät tehostamaan yrityksen testausta ja parantamaan laatua. Mikäli laatua saadaan parannettua, näkyy se varmasti myös yrityksen tuloksessa.

Modultek pitää tarkkaa kirjanpitoa asiakaspalveluun tulleista tukipyynnöistä ja niiden laadusta. Tällä tavoin tulevaisuudessa saadaan selville, ovatko tuotevirheistä johtuvat tukipyynnöt ja virheraportit vähentyneet Tarantulan ja testitapauskirjaston käyttöönoton jälkeen. Ensimmäinen Tarantulan ja uuden testitapauskirjaston avulla testattava Atonin versio on Aton 6.2. Joka tapauksessa, jo projektin toteuttaminen herätti yrityksessä lisää keskustelua laadusta ja testauksesta. Projekti sai myös kiitosta jo entuudestaan testauksesta kiinnostuneiden kollegojen keskuudessa.

Tarantula järjestelmänä tulee pysymään käytössä toivottavasti mahdollisimman pitkään. Ainakin tällä hetkellä se toteuttaa yrityksen tarpeet hyvin. Testitapauskirjaston testitapauksiin tulee varmasti muutoksia ja lisäyksiä ajan ja uusien Atonin versioiden myötä. Suurin osa testitapauksista on kuitenkin pyritty vapaiden testitapausten tapaan kirjoittamaan mahdollisimman kattaviksi. Täten testitapaustenkin eliniän voi odottaa olevan pitkä.

9.3 Itsearvio

Projekti poikkesi huomattavasti aikaisemmista projekteista joissa olin ollut mukana. Olin pääasiassa ollut ohjelmistokehitysprojekteissa, joten tämän kaltainen projekti oli uutta ja kiinnostavaa. Projekti sujui mielestäni hyvin. Ongelmatilanteita tuli vastaan, mutta niistä pääsin ylitse kollegojen avustuksella. Vaikka pysyinkin aikataulussa, aikamääräarviot eivät osuneet kohdilleen. Asennus vei paljon vähemmän aikaa kuin mitä olin varannut, perehtyminen ja testitapausten kirjoittaminen puolestaan enemmän. Toisaalta, eri vaiheet eivät olleet niin selkeästi rajattuja kuin oli suunniteltu. Esimerkiksi perehtymisvaiheeseen palattiin uudelleen ja uudelleen projektin aikana. Lisäksi vaiheet menivät hieman limittäin.

9.4 Loppusanat

Projekti oli hyvin antoisa ja toi vaihtelua normaalille ohjelmistokehitystyölle. Projekti antoi hienon mahdollisuuden tutustua testauksen teoriaan, joka on todella laaja aihe. Se herätti myös kiinnostusta testaamiseen aivan omana alanaan ICT-alan sisällä. Projekti toi myös esiin useita ICT-alan puolia. Ohjelmiston testaus kulkee aina käsikädessä ohjelmoinnin kanssa. Projektiin kuului myös palvelinasennus, web-palvelun asennus ja hallinointi sekä tietysti projektinhallinnallisia asioita. Opin tästä projektista paljon ja siitä on varmasti hyötyä tulevaisuudessa.

Työtä tehdessäni huomasin myös, että testauksessa monet asiat ovat muuttuneet, mutta monet ovat myös pysyneet ennallaan aikojen saatossa. Jo 1970-luvulla tuotiin esiin epäkohtia, joiden kanssa yhä painitaan.

LÄHTEET

Ambysoftin www-sivut 2014. Viitattu 13.2.2014. http://www.ambysoft.com/essays/agileTesting.html

Bach, J. 2002. Exploratory Testing Explained. Viitattu 13.2.2014. http://www.satisfice.com/articles/et-article.pdf

BCS Certificationsin www-sivut 2014. Viitattu 17.2.2014. http://certifications.bcs.org/upload/pdf/swt-careerpath.pdf

Beizer, B. 1990. Software Testing Techniques (Second Edition). Van Nostrand Reinhold.

Binder, R. 1999. Testing Object-Oriented Systems: Objects, Patterns and Tools. Addison-Wesley Professional.

CAD/CAM -yhdistys ry. Valokynä-lehti 2/2013.

CenterCoden www-sivut. 2011a. Viitattu 11.2.2014. http://www.centercode.com/blog/2011/01/alpha-vs-beta-testing/

CenterCoden www-sivut. 2011b. Viitattu 11.2.2014. http://www.centercode.com/beta/tests/public/

Hetzel, W. 1988. The Complete Guide to Software Testing, 2nd Edition. Wellesley.

ISTQB:n www-sivut. 2014a. Viitattu 7.2.2014. http://istqbexamcertification.com/what-is-verification-in-software-testing-or-what-is-software-verification/

ISTQB:n www-sivut. 2014b. Viitattu 11.2.2014. http://istqbexamcertification.com/what-is-acceptance-testing/

ISTQB:n www-sivut. 2014c. Viitattu 11.2.2014. http://istqbexamcertification.com/what-is-alpha-testing/

ISTQB:n www-sivut. 2014d. Viitattu 13.2.2014. http://istqbexamcertification.com/what-is-non-functional-testing-testing-of-software-product-characteristics/

Järvi & Mäkelä 2009. Ohjelmistotestaus. Turun yliopisto. Viitattu 10.2.2014. http://www.cs.utu.fi/opinnot/kurssit/Salo/kevat2011/johdanto_testaukseen.pdf

Kallio, S. 2005. Testaushallinan välineen valinta ja käyttöönotto. Viitattu 20.2.2014. http://www.pcuf.fi/sytyke/lehti/kirj/st20051/ST051-28A.pdf

Kaner, C. 2003. What Is a Good Test Case? Florida Institute of Techonlogy. Viitattu 20.2.2014. http://www.kaner.com/pdfs/GoodTest.pdf

Kaner, C. 2008. A Tutorial in Exploratory Testing. Florida Institute of Techonolgy. Viitattu 13.2.2014. http://www.kaner.com/pdfs/QAIExploring.pdf

McCaffrey, J. 2008. What Makes a Good Software Tester? Viitattu 17.2.2014. http://msdn.microsoft.com/en-us/magazine/dd252951.aspx

Microsoft Developer Networkin www-sivut a. 2013. Viitattu 18.2.2014. http://msdn.microsoft.com/en-us/library/aa292197%28v=vs.71%29.aspx

Microsoft Developer Networkin www-sivut b. 2013. Viitattu 18.2.2014. http://msdn.microsoft.com/en-us/library/hh694602.aspx

Myers, G. 1979. The Art of Software Testing. Wiley.

NetCellin www-sivut 2012. Viitattu 18.2.2014. http://www.netcel.com/Resources/Insights/Blog/Unit-Testing-C-Pt1-Basics/

Niittyviita, A. 2010. Näytä minulle hyvä testitapaus. http://ohjelmistotestaus.fi/2010/09/nayta-minulle-hyva-testitapaus/

Paakki, J. 2012. Ohjelmistoprosessit ja ohjelmistojen laatu. Helsingin yliopisto, tietojenkäsittelytieteen laitos. Viitattu 7.2.2014. http://www.cs.helsinki.fi/u/paakki/Laatu-12-Luentokalvot-4.pdf

Pan, J. 1999. Software Testing. Carnegie Mellon University. Viitattu 7.2.2014. http://www.ece.cmu.edu/~koopman/des_s99/sw_testing/

Parasoft 2014. Unit Testing Best Practises. Viitattu 18.2.2014. http://www.parasoft.com/printables/unittestting.pdf?path=/products/article.jsp

PC Magazinen verkkosivut. Viitattu 20.2.2014. http://www.pcmag.com/encyclopedia/term/52775/test-suite

Pressman, S. 2005. Software Engineering: A Practitioner's Approach. Sixth International Edition.

ReQtestin www-sivut. 2012. Viitattu 20.2.2014. http://www.reqtest.com/blog/learn-how-to-write-effective-test-cases/

Savenkov, R. 2008. How to Become a Software Tester. Roman Savenkov Consulting.

Scacchi, W. 2001. Process Models in Software Engineering. Viitattu 13.2.2014. http://www.ics.uci.edu/~wscacchi/Papers/SE-Encyc/Process-Models-SE-Encyc.pdf

Software Testing Classin www-sivut. 2013. Viitattu 11.2.2014. http://www.softwaretestingclass.com/system-testing-what-why-how/

SWEBOKin www-sivut. 2014. Viitattu 11.2.2014. http://www.computer.org/portal/web/swebok/html/ch5#Ref2.1

Testia Tarantulan www-sivut 2014. Viitattu 27.3.2014. http://www.testiatarantula.com/

Toikkanen, T. 2005. Don't draw diagrams of wrong practices - or: Why people still believe in the waterfall model. Viitattu 13.2.2014. http://tarmo.fi/blog/2005/09/dont-draw-diagrams-of-wrong-practices-or-why-people-still-believe-in-the-waterfall-model/

Tutorialspointin www-sivut. 2014. Viitattu 13.2.2014. http://www.tutorialspoint.com /sdlc/sdlc_v_model.htm

Williams, L. 2006. Viitattu 13.2.2014. Testing Overview and Black-Box Testing Techniques. http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf

Xie, T., Taneja, K., Kale, S. & Marinov, D. Towards a Framework for Differential Unit Testing of Object-Oriented Programs. Viitattu 18.2.2014. http://people.engr.ncsu.edu/txie/publications/ast07-diffut.pdf