KARELIA-AMMATTIKORKEAKOULU
Tietotekniikan koulutusohjelma

Heikki Majoinen

KEYAQUA-VERKKOTIETOJÄRJESTELMÄN TESTAUS

Opinnäytetyö
Toukokuu 2015

OPINNÄYTETYÖ
Toukokuu 2015
Tietotekniikan koulutusohjelma

Karjalankatu 3
80200 JOENSUU
+358 50 260 6800

Tekijä
Heikki Majoinen

Nimike
KeyAqua-verkkotietojärjestelmän testaus

Toimeksiantaja
Keypro Oy

Tiivistelmä

Tämän opinnäytetyön tavoitteena oli perehtyä ohjelmistotestauksen eri osa-alueisiin sekä selvittää, millaisilla menetelmillä ohjelmistotestausta voidaan suorittaa. Tulosten pohjalta suunniteltiin ja toteutettiin KeyAqua -verkkotietojärjestelmän järjestelmätestaus. Työn toimeksiantajana toimi Keypro Oy, joka on paikka- ja verkkotietojärjestelmiin erikoistunut ohjelmistoyritys.

Teoriaosuudessa käsitellään ohjelmistotestauksen roolia ohjelmistotuotannossa. Lisäksi esitellään tasoja, jolla testausta suoritetaan, sekä eri menetelmiä testauksen toteuttamiseksi.

Opinnäytetyössä saavutettiin sille asetetut tavoitteet. KeyAqua -järjestelmän testaus saatiin suoritettua onnistuneesti testaussuunnitelman mukaisesti. Lisäksi saatiin määriteltyä tärkeimmät jatkokehityskohteet ja määriteltyä prosessi virheiden korjaamista varten. Teoriaosuudesta on apua myös muiden tuotteiden testausta suunniteltaessa.

Kieli
suomi

Sivuja 33
Liitteitä 1
Liitesivuja 1

Asiasanat
ohjelmistotuotanto, ohjelmistotestaus, järjestelmätestaus

THESIS
May 2015
Degree Programme in Information Technology
Karjalankatu 3
FI 80200 JOENSUU
FINLAND
+358 50 260 6800

Author
Heikki Majoinen

Title
Testing of KeyAqua Network Inventory System

Commissioned by
Keypro Oy

Abstract

The purpose of this thesis was to study different sections of software testing and various methods of performing the testing. Based on the results of this study, system testing of KeyAqua network inventory system was planned and carried out. This thesis was commissioned by Keypro Oy, which is a software company specialized in geographic information systems and network information systems.

The theory section covers the role of software testing as a part of software engineering. The section also introduces different levels of testing and different techniques that can be used.

The objectives set for this thesis were achieved. The testing of KeyAqua system was successfully made according to a test plan. In addition, the most important needs of further development were identified and a process for fixing errors was defined. The theory section will also help to plan the testing of other products.

Language
Finnish

Pages 33
Appendices 1
Pages of appendices 1

Keywords
software engineering, software testing, system testing

Sisältö

1 Johdanto 6
2 Ohjelmistotestaus 6
3 Testaus osana ohjelmistoprojektia 9
3.1 Vesiputousmalli 9
3.2 V-malli 10
3.3 RUP 11
4 Testaustasot 12
4.1 Yksikkötestaus 12
4.2 Integrointitestaus 13
4.2.1 Ylhäältä alaspäin -integrointitestaus 14
4.2.2 Alhaalta ylöspäin -integrointitestaus 14
4.3 Järjestelmätestaus 14
4.4 Hyväksymistestaus 15
5 Testausmenetelmät 15
5.1 Musta laatikko -testaus 16
5.2 Lasilaatikkotestaus 17
5.3 Harmaa laatikko -testaus 18
5.4 Regressiotestaus 18
5.5 Savutestaus 18
5.6 Rasitustestaus 19
5.7 Alfa- ja betatestaus 19
5.8 Käytettävyystestaus 20
6 Testaustyökalut 21
6.1 Vikatietokannat 21
6.2 Dokumenttipohjat 22
6.3 Testausautomaation työkalut 22
7 KeyAqua-verkkotietojärjestelmän testaus 22
7.1 Keypro Oy 23
7.2 KeyAqua-verkkotietojärjestelmä 23
7.3 Testaussuunnitelma 24
7.4 Testitapaukset 25
7.5 Testauksen suorittaminen 26
7.6 Testausraportti 26
7.7 Virheiden kirjaaminen 27
7.8 Testauksen tulokset 28

7.9 Asiakasympäristöjen testaus 29
7.10 Jatkokehitys 30
8 Pohdinta 31
Lähteet 33

Liitteet

Liite 1 Esimerkki testitapauksesta

1 Johdanto

Ohjelmistotestaus on tärkeä osa ohjelmistotuotantoa. Testauksen tärkein tehtävä on varmistaa, että kehitettävä järjestelmä vastaa määrittelyä ja että järjestelmä myös toimii oikein. Ohjelmistotestaus vaatii paljon aikaa ja resursseja, joten sen vuoksi se on suunniteltava huolellisesti.

Suunnitelmallisen testauksen avulla pyritään löytämään mahdolliset ohjelmavirheet mahdollisimman aikaisessa vaiheessa, jotta niiden korjaaminen olisi mahdollisimman helppoa. Jokainen asiakasympäristöön tai julkaistuun tuotteeseen päätynyt virhe aiheuttaa tyytymättömyyttä loppukäyttäjissä, ja sillä voi olla suuri merkitys ohjelmistoyrityksen imagoon.

Tämän opinnäytetyön tavoitteena oli perehtyä ohjelmistotestauksen perusperiaatteisiin, sekä selvittää, millä erilaisilla menetelmillä ohjelmistotestausta voidaan suorittaa. Tämän selvityksen pohjalta tehtävänä oli suunnitella ja toteuttaa KeyAqua-verkkotietojärjestelmän järjestelmätestaus. Testauksen suunnittelu aloitettiin testaussuunnitelman luonnilla, jossa kuvattiin miten testaus tullaan toteuttamaan. Testitapaukset suoritettiin testaussuunnitelman mukaisesti joiden tulosten pohjalta muodostettiin testausraportti. Testauksen suorituksen yhteydessä määriteltiin myös prosessi, jolla voitiin testauksen aikana löydetyt virheet ohjata sovelluskehitykseen. Lisäksi suunniteltiin tapaa, jolla asiakasympäristöt tulisi testata.

2 Ohjelmistotestaus

Ohjelmistotestaus on tärkeä osa ohjelmistoprojektin laadunvalvontaa. Ohjelmistotestauksen tärkeimpänä tehtävänä on varmistua, että kehitettävä ohjelma tai järjestelmä vastaa määrittelyä, ja että ohjelman toiminnot toimivat oikein. Testauksen voi ajatella olevan suunnitelmallista virheiden etsintää ohjelmaa tai sen

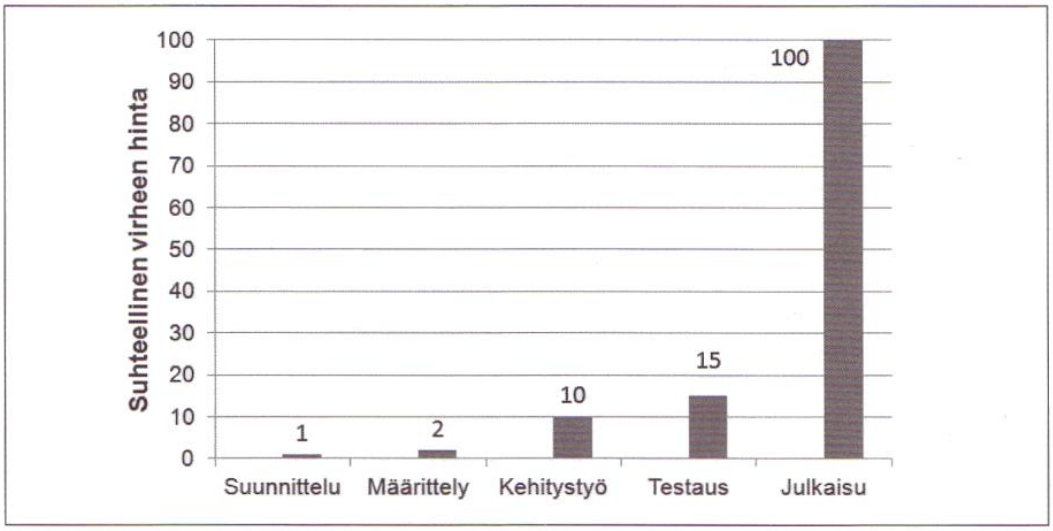
osaa suorittamalla. Virheiden etsimisen ohella on testauksen yhteydessä varmistettava, että ohjelma toimii niin kuin on suunniteltu. [1, s. 437; 2, s. 205.]

Koska nykyisin kehitettävät ohjelmistot ja järjestelmät voivat olla hyvin monimutkaisia ja laajoja kokonaisuuksia, voivat testaajan työtehtävät vaihdella ohjelmistoyritysten välillä merkittävästi. Perinteinen mielikuva testaajasta on henkilö, joka suorittaa mekaanisesti suuren määrän toimintoja ja käskyjä testauksen kohteena olevassa järjestelmässä. Nykyään testaajan ammatti voi kuitenkin sisältää hyvin paljon erilaisia tehtäviä. Testaaja voi esimerkiksi haastatella koekäyttäjiä, suunnitella käytettävää dokumentaatiota, valita testauksessa käytettäviä työkaluja tai jopa kehittää testisimulaatioita ja ohjelmakoodia, jolla testausta voidaan suorittaa. [3, s. 10.]

Ohjelmistotestauksessa suunnitelmallisuus on hyvin tärkeää. Koska testaus vie monesti paljon resursseja ja aikaa, täytyy huolehtia, että testitapaukset on suunniteltu siten, että niillä saadaan tehokkaasti tarkastettua ohjelmiston toiminnoista mahdollisimman paljon. Täydellinen testaaminen on yleensä mahdotonta. Jos ohjelman kaikki toiminnot haluttaisiin testata kaikilla mahdollisilla lukuarvoilla, tulisi syötteitä niin valtavasti, että niiden läpikäyminen veisi vuosikausia. Muutamien tuntien huolellinen, suunniteltu testitapausten suorittaminen tuottaa monesti paremmat tulokset kuin päiväkausien epäjärjestelmällinen kokeilu. [2, s. 210; 3, s. 19–20.]

Ohjelmistotestauksessa virhe tarkoittaa ohjelman poikkeamaa sen määrittelystä. Testauksen tukena tulee siis aina olla tekninen dokumentaatio, jossa kuvataan ohjelmiston tai järjestelmän toiminta. Monesti tulee vastaan tilanne, jossa testaaja tai käyttäjä kokee ohjelman toiminnallisuuden virheelliseksi, mutta itse asiassa ohjelma toimiikin täysin määrittelyn mukaisesti. Tällöin kyseessä onkin ominaisuus eikä vika. Testauksen suunnittelun ja testitapausten luonnin helpottamiseksi onkin tärkeää, että ohjelman tekninen dokumentaatio on helposti saatavilla ja ymmärrettävissä. Mikäli ohjelmiston määrityksiin tulee muutoksia, on kaikkien projektiryhmään kuuluvien henkilöiden oltava tietoisia asiasta. [1, s. 439–442; 2, s. 205–206.]

Ohjelmistoprojektien suurimpina ongelmina ovat usein liian pitkälle venyneet työvaiheet sekä aikataulut, joista ei voida mitenkään pitää kiinni. Tällöin on vaarana, että resurssien säästämiseksi ja aikataulun nopeuttamiseksi karsitaan loppupään toiminnoista, kuten testauksesta. Tämä voi kuitenkin aiheuttaa yllättäviä kustannuksia. Jos ohjelmistosta havaitaan virheitä projektin myöhemmissä vaiheissa, tulee virheiden korjaus huomattavasti kalliimmaksi kuin projektin alkuvaiheissa. Kuvassa 1 on esitelty virheen suhteellista hintaa ohjelmistoprojektin edetessä. Esimerkiksi valmiista tuotteesta löytyneen virheen korjaaminen maksaa kymmenen kertaa enemmän kuin virheen korjaaminen tuotteen ollessa vielä kehitysvaiheessa. [3, s. 16–18.]



Kuva 1. Virheen suhteellinen hinta ohjelmistoprojektin eri työvaiheissa. [3, s. 18.]

Esimerkiksi peliteollisuudessa tämä on yleinen ongelma. Monesti uutuuspelille on asetettu julkaisupäivä, josta ei yksinkertaisesti voida tinkiä. Markkinatilanne voi sanella aikataulun. Tuote on saatava kauppoihin esimerkiksi joulun aikaan tai juuri ennen kuin kilpailija on julkaisemassa omaa tuotettaan. Näin ollen tuotteita päätyy markkinoille keskeneräisinä ja niihin joudutaan julkaisemaan virheitä korjaavia päivityksiä.

Testauksen merkitys riippuu myös kehitettävän ohjelmiston tai järjestelmän luonteesta. Kun suunnitellaan elintärkeitä järjestelmiä, kuten esimerkiksi lennonjohdon valvontajärjestelmää tai ydinvoimalan reaktorin ohjausjärjestelmää, ei

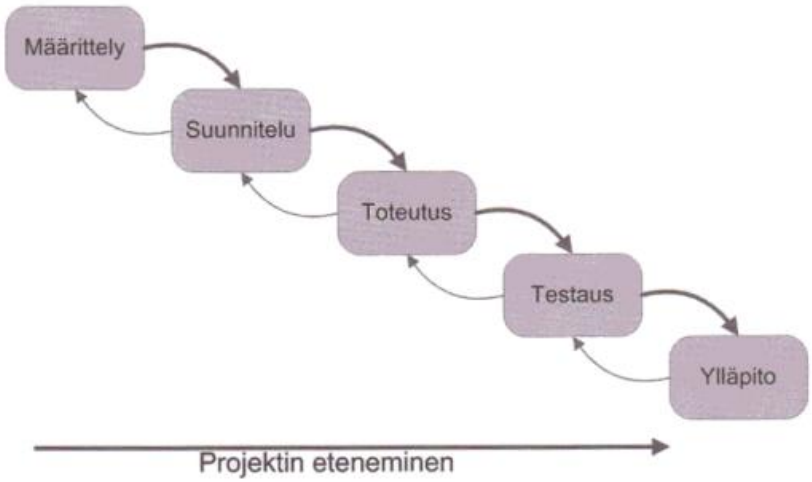
virheisiin yksinkertaisesti ole varaa. Tällaisissa projekteissa voivat testauksen kustannukset olla moninkertaiset verrattuna projektin muihin kuluihin. [1, s. 437.]

3 Testaus osana ohjelmistoprojektia

Ohjelmistotestaus käsitetään yleensä yhdeksi osaksi ohjelmistoprojektia. Perinteisessä vesiputousmallissa testaus on oma erillinen työvaiheensa, kun taas uudemmissa malleissa testausta suoritetaan koko ohjelmistoprojektin ajan. [3, s. 12–15.]

3.1 Vesiputousmalli

Ohjelmistoprojektin vesiputousmallissa projekti jaetaan vaiheisiin. Ohjelmistoprojekti aloitetaan ohjelmiston määrityksellä, jossa kerätään tietoa ohjelmiston tarpeista. Kerätyistä tiedoista muodostetaan projektisuunnitelma, jonka pohjalta projektia ruvetaan toteuttamaan. Kun toiminnallinen kokonaisuus on muodostettu, aloitetaan testausvaihe. Testauksen tavoitteena on varmistaa, että ohjelmisto täyttää kaikki sille asetetut vaatimukset. Kun testaus on saatu toteutettua onnistuneesti, siirtyy ohjelma ylläpitoon (kuva 2). [3, s. 12–13.]

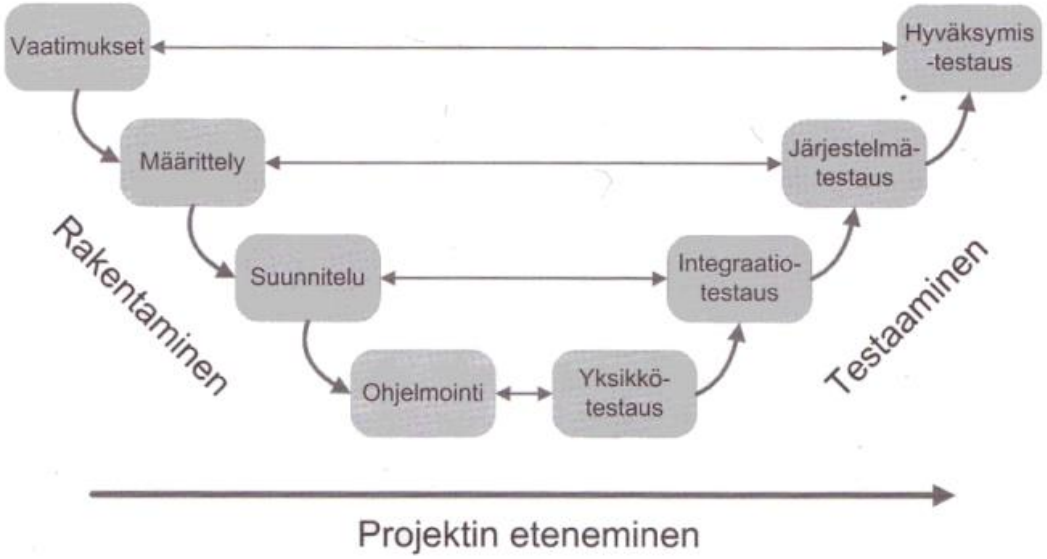


Kuva 2. Ohjelmistoprojektin vesiputousmalli [3, s. 23.]

Vesiputousmalli tarjoaa mahdollisuuden palata aiempaan työvaiheeseen. Kuitenkin tavoitteena on, että projektin vaiheissa palataan takaisin mahdollisimman vähän. Tämä voi aiheuttaa ongelmia etenkin suurissa kehityshankkeissa. Useita kuukausia kestävän projektin aikana voi tulla suuria muutoksia järjestelmän vaatimuksiin. Tällöin täytyy palata takaisin suunnittelu- tai jopa määrittelyvaiheeseen. Tämä monesti aiheuttaa myös muutoksia aikatauluihin, jolloin resurssien käytön suunnittelu vaikeutuu. [3, s. 22–23.]

3.2 V-malli

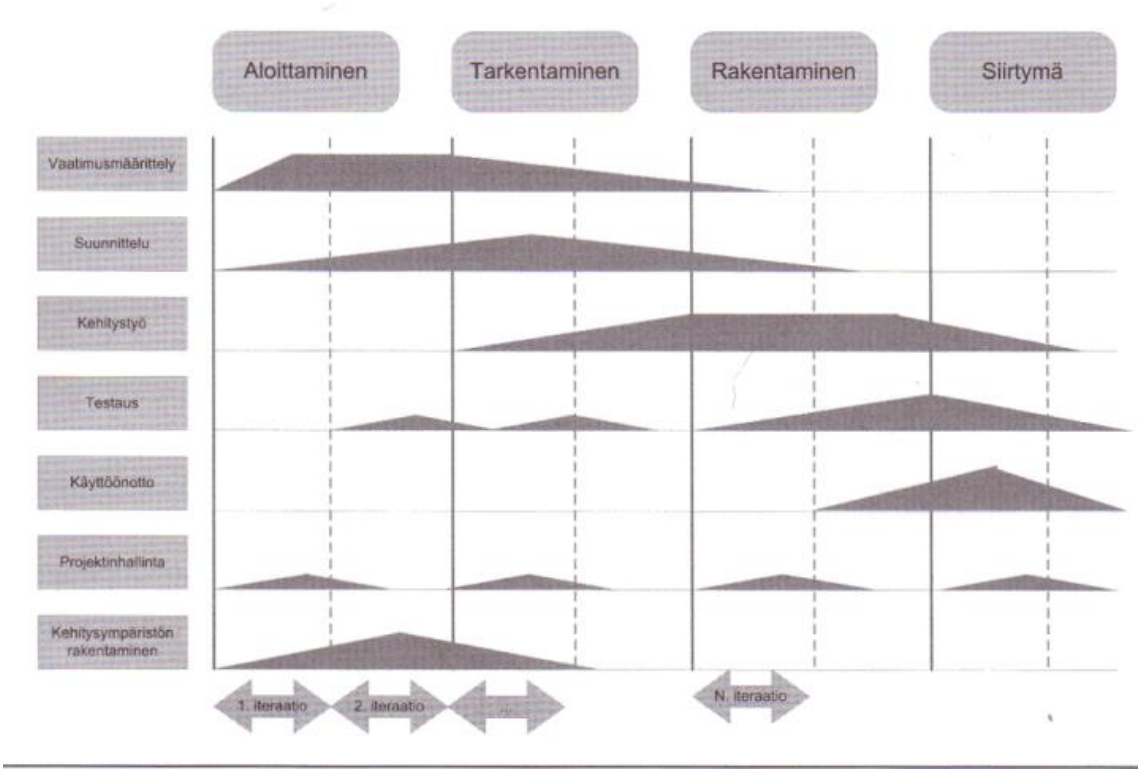
V-mallissa perinteisen vesiputousmallin ongelmia on pyritty ratkaisemaan jakamalla jokaiselle rakentamisen vaiheelle oma testaustaso kuvan 3 mukaisesti. Testaus aloitetaan jo toteutusvaiheen aikana valmistuneille komponenteille yksikkötestien muodossa. Kun yksittäisistä komponenteista käydään muodostamaan järjestelmää, testataan moduulien keskinäistä toimintaa integraatiotesteillä. Lopullista ohjelmakokonaisuutta testataan järjestelmä- ja hyväksymistestauksen avulla. [3, s. 14.]



Kuva 3. V-malli [3, s. 14.]

3.3 RUP

Rational Softwaren 1990-luvulla kehittämä Rational Unified Process -malli pyrkii ratkaisemaan ongelmia, joita suunnittelu- ja määrittelyvaiheiden muutokset aiheuttavat ohjelmistoprojektissa. RUP-mallissa kehitysprosessi on jaettu neljään päävaiheeseen: aloittamiseen, tarkentamiseen, rakentamiseen sekä siirtymään. Näiden päävaiheiden sisällä ohjelmaa kehitetään pienemmissä vaiheissa. Testauksen suunnittelu aloitetaan jo ohjelmiston määrittelyn yhteydessä ja testaus aloitetaan heti kun toiminnallisuutta on testattavana. Testausta suoritetaan jokaisessa päävaihessa, mutta testauksen määrä vaihtelee vaiheesta riippuen (kuva 4). Jokaisen iteraation jälkeen tulokset tarkastetaan ja testataan ennen kuin siirrytään seuraavaan iteraatioon. Tarpeen vaatiessa suunnitteluun voi tulla muutoksia, joten hyvä dokumentointi on tällöin välttämätöntä. RUP-malli sopii erityisesti laajoihin kehityshankkeisiin, joiden toteutus vaatii kuukausia tai jopa vuosia. [2, s. 42–43; 3, s. 24–26.]



Kuva 4. RUP-malli [3, s. 25.]

4 Testaustasot

Aiemmin esitelty testauksen V-malli jakaa järjestelmän rakentamisen ja testauksen vaiheisiin. Näiden vaiheiden sisällä testausta tehdään eri tasoilla. Yksikkötestaus keskittyy yksittäiseen komponenttiin tai moduuliin, integrointitestauksessa puolestaan tarkastellaan komponenttien tai komponenttiryhmien välistä toimintaa ja järjestelmätestauksessa tarkastellaan järjestelmää kokonaisuutena. Lopuksi suoritetaan kohdeympäristössä hyväksymistestaus, jonka jälkeen ohjelmisto siirtyy ylläpitovaiheeseen. Tässä luvussa esitellään tarkemmin testauksen eri tasot. [1, s. 481–482; 3, s. 50–51.]

4.1 Yksikkötestaus

Yksikkötestauksessa tarkasteltavana kohteena on järjestelmän yksittäinen moduuli, funktio tai komponentti. Yksikkötestaus ajatellaan yleensä seuraavaksi

askeleeksi ohjelman osan kehittämisen jälkeen. Yksikkötestauksen suorittaa useimmiten komponentin tai toiminnon kehittäjä. Testitapausten suunnittelussa pohjana toimii tekninen määrittely. [1, s. 487.]

Yksikkötestauksen tavoitteena on löytää komponenteista virheet mahdollisimman aikaisessa vaiheessa. Mikäli testauksen kohteena oleva komponentti sisältää virheitä, pystyy sovelluskehittäjä korjaamaan viat ennen kuin komponentista tulee osa laajempaa kokonaisuutta. [3, s. 51.]

Yksikkötestauksessa usein testauksen kohteena oleva yksittäinen komponentti ei pysty itsenäiseen toimintaan. Testauksen tueksi voidaan kehittää testiajureita sekä testimoduuleita simuloimaan muiden järjestelmien osien toimintaa. Jossain tapauksissa testattavan moduulin toimintaa ei voida kuitenkaan simuloida riittävän hyvin ajureiden ja testikomponenttien avulla. Tällöin kattava testaus voidaan suorittaa vasta testauksen myöhemmissä tasoissa, kuten integrointitestauksessa. [1, s. 487; 3, s. 52.]

4.2 Integrointitestaus

Integrointitestauksessa varmistetaan, että yksikkötestauksessa testatut komponentit toimivat oikein yhtenä kokonaisuutena. Integrointitestauksessa yhdistetään yksikkötestattuja komponentteja järjestelmän määrittelyn mukaisesti. [1, s. 488.]

Myös integrointitestauksessa voidaan joutua rakentamaan testiajureita simuloimaan vielä puuttuvien komponenttien toimintaa. Integrointitestauksessa testitapaukset laaditaan yleensä testaamaan komponenttien välistä toimintaa, kuten tiedon siirtymistä moduulilta toiselle. [1, s. 488; 3, s. 54.]

Integrointitestauksen perusajatelmana on ohjelman rakentaminen ja testaaminen pienissä osissa. Tällöin virhe on helpompi havaita ja korjata kuin kokonaisessa järjestelmässä. Kun järjestelmään lisätään osia, samalla varmistetaan, että kokonaisuus toimii edelleen oikein. Järjestelmää voidaan lähteä tarkaste-

lemaan matalimman tason moduuleista tai vaihtoehtoisesti järjestelmärakenteen huipulta. [1, s. 488; 3, s. 54.]

4.2.1 Ylhäältä alaspäin -integrointitestaus

Ylhäältä alaspäin -integrointitestauksessa korkean tason ohjelmakomponentit testataan ja yhdistetään ensimmäiseksi. Tämä menetelmä antaa hyvän yleiskuvan testattavasta järjestelmästä ja paljastaa nopeasti puutteet toiminnallisuudessa. Koska alhaisen tason moduulit testataan viimeisenä, voidaan niitä joutua korvaamaan testiajureilla. Tämä voi hankaloittaa ja hidastaa testauksen toteuttamista. [3, s. 55; 5.]

4.2.2 Alhaalta ylöspäin -integrointitestaus

Alhaalta ylöspäin -integrointitestauksessa testaus aloitetaan matalimman tason komponenteista, joista siirrytään järjestelmähierarkiassa ylöspäin. Matalan tason komponenteista muodostetaan osajoukkoja, jotka yhdistetään ja testataan. Koska moduulien toiminnallisuus on testauksen aikana hyödynnettävissä, testiajureiden tarve vähenee huomattavasti. Vastaavasti voidaan joutua kirjoittamaan ohjelmalohkoja hoitamaan komponenttien välistä tiedonsiirtoa. Tällä menetelmällä havaitaan matalan tason virheet, jotka muuten voisivat jäädä huomaamatta. [3, s. 55; 5.]

4.3 Järjestelmätestaus

Järjestelmätestauksen tarkoituksena on testata kokonaista järjestelmää ja sen toimintaa verrattuna järjestelmän tai ohjelmiston toiminnalliseen määrittelyyn. Järjestelmätestauksessa todetaan kaikki järjestelmän komponentit, rajapinnat ja mahdolliset ulkoiset liittymät toimiviksi. Järjestelmätestauksessa huomioidaan myös ei-toiminnallisia ominaisuuksia, esimerkkinä kuormitus-, käytettävyys- ja

suorituskykytestit. Järjestelmätestauksen suorittajina tulisi olla muita kuin järjestelmän kehittäjiä. [7; 8, s. 290.]

Mikäli kehitettävä järjestelmä tai tuote on sellainen, josta julkaistaan asiakkaille useita versioita, testauksen merkitys ja määrä kasvaa merkittävästi. On varmistettava, että uudet kehitetyt ominaisuudet eivät vaikuta aiemmin kehitettyyn toiminnallisuuteen. Uusien versioiden testaaminen vaatii paljon resursseja, sillä uusien kehitettyjen ominaisuuksien myötä testitapausten määrä kasvaa, joten sen vuoksi uusien versioiden testausta tulisi voida automatisoida, esimerkiksi erilaisten työkalujen avulla, mahdollisimman paljon. [8, s. 290.]

4.4 Hyväksymistestaus

Hyväksymistestauksen tarkoituksena on lopullisesti varmistaa, että järjestelmä tai ohjelma toimii oikein ja vastaa teknistä määrittelyä. Hyväksymistestaus on usein suunniteltu sarja erilaisia testejä, jossa loppukäyttäjät testaavat järjestelmää sen lopullisessa kohdeympäristössä. Järjestelmän luonteesta riippuen hyväksymistestaus voi kestää viikkoja tai jopa kuukausia. [1, s. 497.]

Hyväksymistestauksen voi siis ajatella olevan testauksen viimeinen vaihe, jossa järjestelmä tarkastetaan lopullisesti. Asiakkaan ja toimittajan sopimuksessa voi olla määritelty, että hyväksymistestauksen jälkeen ohjelmisto siirtyy lakiteknisesti asiakkaan omistukseen. [3, s. 57.]

5 Testausmenetelmät

Aiemmassa luvussa esiteltyjen testaustasojen sisällä testausta voidaan suorittaa eri tavoin. Tässä luvussa on esitelty erilaisia testausmenetelmiä, joita voidaan soveltaa esimerkiksi testitapausten suunnittelussa.

5.1 Musta laatikko -testaus

Musta laatikko -testauksessa järjestelmä ajatellaan läpinäkymättömäksi mustaksi laatikoksi. Järjestelmän sisäisestä toiminnasta ei tiedetä mitään, vaan tarkastellaan ohjelman käyttäytymistä annettuihin syötteisiin (kuva 5). Testien tavoitteena on löytää tilanteet, joissa ohjelma toimii teknisen määrittelyn vastaisesti. [4, s. 2.]

Kuva 5. Musta laatikko -testaus [3, s. 66.]

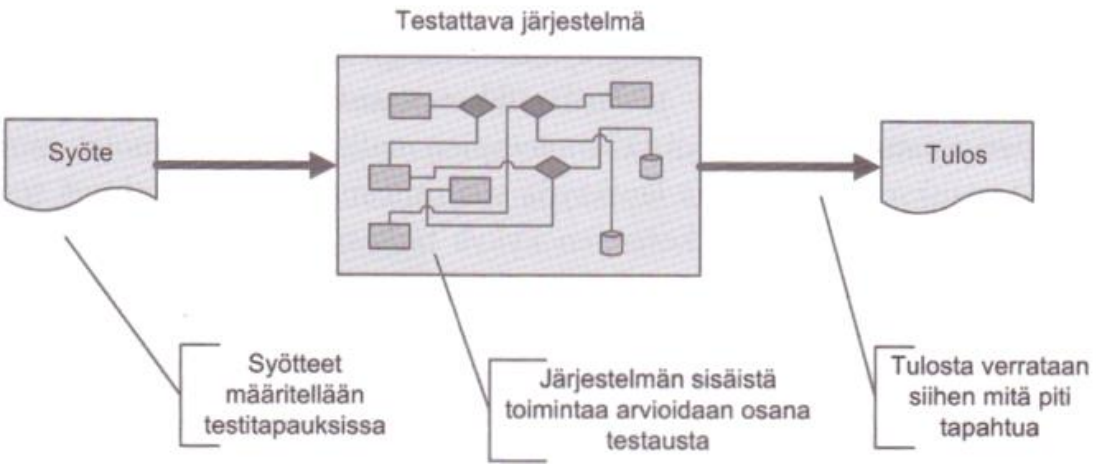
Musta laatikko -testauksen etuja on se, että sitä voidaan käyttää käytännössä missä tahansa testauksen työvaiheessa. Lisäksi se on yksinkertainen toteuttaa, koska testien suorittajalla ei tarvitse olla tuntemusta esimerkiksi käytetyistä ohjelmointikielistä. Testaajan tehtävänä on ainoastaan varmistaa, että ohjelma käyttäytyy oikein annetuilla syötteillä sekä raportoida mahdolliset virhetilanteet ohjelmiston kehittäjille. Musta laatikko -testeillä voidaan myös helposti tarkastella järjestelmän toimivuutta virheellisillä syötteillä antamalla esimerkiksi arvoasteikon ulkopuolelle meneviä lukuarvoja tai syötteitä, jotka ovat tietotyypiltään vääriä. [3, s. 66–67.]

Musta laatikko -testauksessa on kuitenkin myös huonoja puolia. Mikäli järjestelmästä ei ole selkeää kuvausta, voi kattavien testitapausten laatiminen olla vaikeaa. Lisäksi ohjelmakoodia ei hyödynnetä paikannettaessa ohjelman osia, jotka todennäköisimmin aiheuttavat virheitä. [6, s. 3.]

5.2 Lasilaatikkotestaus

Lasilaatikkotestauksessa tarkastellaan järjestelmän sisäistä toimintaa testauksen aikana (kuva 6). Tämä menetelmä on kattavampi kuin musta laatikko -testaus, sillä testaaja näkee myös mitä ohjelman sisällä tapahtuu pelkästään tulosten tarkastelun sijasta. Onnistuneiden testien kohdalla voidaan tarkastaa lähdekooditasolla että ohjelma tosiaan toimii oikein, ja ettei tulos ollut vain sattuma. [3, s. 67.]

Lasilaatikkotestausta voidaan käyttää yksikkö-, integrointi- ja järjestelmätestausvaiheissa. Lasilaatikkotestauksessa testaajan täytyy lukea ohjelman lähdekoodia ja selvittää mitkä osat koodista voivat aiheuttaa virheitä. [4, s. 2.]



Kuva 6. Lasilaatikkotestaus. [3, s. 67.]

Lasilaatikkotestauksessa on myös omat haasteensa. Pelkästään testaajalta vaaditaan huomattavasti enemmän kuin musta laatikko -testauksessa, sillä pelkän järjestelmän toimintaperiaatteen tuntemisen lisäksi testaajan pitää osata lukea myös ohjelmakoodia. Lisäksi lasilaatikkotestauksessa ei voida havaita ongelmia, jotka johtuvat puuttuvista tai virheellisistä ominaisuuksista. Tästä johtuen lasilaatikkotestaus ei voi yksinään toimia laadunvarmistus- tai hyväksymistestauksen pohjana. [3, s. 67–68.]

5.3 Harmaa laatikko -testaus

Harmaa laatikko -testaus on yhdistelmä musta laatikko- ja lasilaatikkotestausta. Sitä voidaan soveltaa tapauksissa, joissa järjestelmän kaikkien komponenttien lähdekoodiin ei päästä käsiksi lasilaatikkomenetelmän vaatimalla tasolla. Tällaisia ovat monesti esimerkiksi verkkopalvelut. Tällöin paikalliset komponentit ovat testattavissa ohjelmakooditasolle asti ja vastaavasti ulkoiset palvelut ja rajapinnat voidaan testata vain musta laatikko -testauksella. [3, s. 68.]

Harmaa laatikko -testauksen edut ovat yhdistelmä musta laatikko- ja lasilaatikkotestauksen etuja. Menetelmällä voidaan laatia kattavia testitapauksia, joissa tarkastellaan myös järjestelmän sisäistä toimintaa. Järjestelmää ei kuitenkaan pystytä testaamaan täydellisesti, koska kaikkea ohjelmakoodia ei ole saatavilla. Lisäksi esimerkiksi rajapintojen puutteelliset kuvaukset voivat aiheuttaa ongelmia. [6, s. 9.]

5.4 Regressiotestaus

Regressiotestauksella tarkoitetaan uudelleentestausta, jota tarvitaan kun ohjelmiston tai komponentin osaa on muutettu. Regressiotestauksen avulla varmistetaan että muutokset eivät ole vaikuttaneet järjestelmän muihin osiin, ja että järjestelmä toimii edelleen oikein. Regressiotestauksen määrä voi kasvaa ohjelmistoprojektin aikana hyvinkin suureksi, joten sitä on mahdollisuuksien mukaan pyrittävä automatisoimaan. Lisäksi myös regressiotestaus kannattaa suunnitella hyvin. Testitapaukset kannattaa kohdistaa muuttuneisiin komponentteihin sekä toimintoihin, jotka todennäköisimmin voivat sisältää virheitä tehtyjen muutosten johdosta. [1, s. 491–492; 3, s. 68–69.]

5.5 Savutestaus

Savutestauksella tarkoitetaan ohjelmiston perustestausta, jolla varmistetaan että testattavan ohjelmiston perusasiat toimivat oikein. Savutestauksessa ei ole

tarkoituksenmukaista käydä koko ohjelmistoa läpi vaan varmistaa, että ohjelmasta ei löydy suuria virheitä ja että ohjelmisto on tarpeeksi toimiva tarkemman testauksen aloittamiseksi. Lisäksi savutestien yhteydessä löydetään monesti yksinkertaisia mutta ohjelmiston toiminnan kannalta keskeisiä virheitä. [1, s. 492–493; 3, s. 72–73.]

5.6 Rasitustestaus

Rasitustestauksen tehtävä on selvittää miten järjestelmä selviää tilanteesta, jossa sitä kuormitetaan normaalia suuremmalla tietomäärällä. Esimerkiksi lähdedatan määrää voidaan kasvattaa huomattavasti normaalia suuremmaksi ja tarkastella, miten järjestelmä toimii kyseisessä tilanteessa. Lisäksi rasitustestauksen tehtävänä on etsiä kohtia joissa tiedonsiirto voi aiheuttaa ongelmia, kuten esimerkiksi palvelimien väliset yhteydet. [1, s. 498; 3, s. 71.]

Johtuen testisyötteiden määrästä rasitustestaus on käytännössä aina automatisoitu. Esimerkiksi internetissä olevan palvelun toimintaa suurilla käyttäjämäärillä voidaan testata luomalla virtuaalikäyttäjiä, jotka simuloivat normaalia toimintaa. [3, s. 71–72.]

5.7 Alfa- ja betatestaus

Alfa- ja betatestaus suoritetaan yleensä tuotteille, joita myydään itsenäisinä ohjelmistotuotteina monille asiakkaille. Mikäli kehitettävä järjestelmä on räätälöity yhden asiakkaan tai organisaation käyttöön, voidaan järjestelmän toimivuus ja vastaavuus tekniseen määrittelyyn todeta hyväksymistestien avulla. Mutta jos loppukäyttäjiä on suuri joukko, on mahdotonta suorittaa hyväksymistestausta jokaisen käyttäjän kohdalla. [1, s. 496.]

Alfatestaus toteutetaan yleensä järjestelmän kehittäjän tiloissa. Testien suorittajina toimii loppukäyttäjiä ja ohjelmiston kehittäjät seuraavat testien etenemistä. Alfatestauksen tulosten perusteella järjestelmän toimintalogiikkaan, käyttöliitty-

mään tai yksittäisiin toimintoihin voi tulla suuriakin muutoksia. Alfatestauksen tavoitteena on todeta, että järjestelmä toimii riittävän hyvin jotta se voi siirtyä betatestaukseen. [3, s. 73.]

Betatestauksessa loppukäyttäjät testaavat ohjelmistoa sen lopullisessa kohdeympäristössä. Toisin kuin alfatestauksessa, ohjelmiston kehittäjät eivät ole läsnä testauksessa eivätkä pääse vaikuttamaan testaukseen. Betatestauksen tulosten perusteella voidaan tehdä muutoksia ja korjauksia ohjelmaan, jotka julkaistaan asiakkaille seuraavissa päivityksissä. Peliyhtiöt tarjoavat usein mahdollisuuden hakea erilliseen betatestausryhmään, joka pääsee testaamaan uutuuspeliä ennen sen varsinaista ilmestymistä. Lisäksi betatestaus tarjoaa mahdollisuuden testata esimerkiksi palvelinkapasiteetin riittävyyttä. [1, s. 496; 3, s. 73.]

5.8 Käytettävyystestaus

Käytettävyystestauksella tarkoitetaan testausta, jonka tarkoituksena on todeta järjestelmän käyttöliittymän toimivuus ja soveltuvuus siihen käyttötarkoitukseen johon järjestelmää ollaan kehittämässä. Käytettävyystestaus voidaan suorittaa esimerkiksi käyttötestien ja niitä seuraavien haastattelujen muodossa. Lisäksi voidaan hyödyntää asiantuntijoita varmistamaan, että käyttöliittymä on suunniteltu oikein. [3, s. 70.]

Käytettävyystestauksen suorittamiseen on olemassa myös erilaisia apuvälineitä. Testausta voidaan esimerkiksi valvoa erilaisten monitorien ja valvontaohjelmien avulla. Erilaisten ohjelmien avulla voidaan tallentaa kaikki käyttäjän järjestelmälle lähetetyt syötteet sekä seurata käyttäjän toimintaa tämän suorittaessa testitapauksia. Lisäksi testauksen yhteydessä voidaan nauhoittaa videokameroiden avulla itse koehenkilöä. Esimerkiksi henkilön kasvojen ilmeitä ja eleitä tulkitsemalla voidaan todeta miten käyttäjä reagoi vaikkapa odottamattomiin tapahtumiin. Käytettävyystestauksen merkitys voi vaihdella hieman riippuen siitä, minkälaista järjestelmää ollaan kehittämässä. Esimerkiksi peli- ja hyötyohjelmateollisuudessa käytettävyydellä on suuri merkitys, sillä jos henkilö kokee

ohjelman käyttöliittymän puutteelliseksi ja käytön vaikeaksi, voi hän helposti alkaa etsimään toista vaihtoehtoa. Käytettävyystestien tulokset voivat muuttaa myös esimerkiksi pelin sääntöjä tai muuta toteutusta. [3, s. 70–71.]

Käytettävyystestauksen tulokset voivat usein yllättää täysin järjestelmän kehittäjät. Loppukäyttäjät näkevät varmasti monia asioita eri tavalla kuin sovelluskehittäjät. Ominaisuudet voivat olla esimerkiksi liian vaikeakäyttöisiä, tai tärkeitä toimintoja voi olla käyttöliittymässä väärin sijoitettuna. Käytettävyystestaus onkin hyvä tapa löytää käyttöliittymän puutteet sekä virheelliset määritykset. Siksi käytettävyystestausta tulisi suorittaa jo käyttöliittymän prototyypeillä. [8, s. 291.]

6 Testaustyökalut

Kuten edellä on todettu, ohjelmistotestaus koostuu monesta eri testausmenetelmästä ja työvaiheesta. Näiden vaiheiden suorituksessa voidaan käyttää erilaisia työkaluja testauksen helpottamiseksi. Työkalujen avulla voidaan esimerkiksi suorittaa tiettyjä testejä automaattisesti. Lisäksi erilaisten analysaattoreiden avulla voidaan tarkastella järjestelmän komponentteja sekä lähdekoodia. Testaustyökalujen avulla voidaan myös helpottaa ja yhtenäistää esimerkiksi testauksen aikana tapahtuvaa dokumentointia ja raportointia. [3, s. 86–88.]

6.1 Vikatietokannat

Vikatietokanta on tärkeä osa ohjelmistokehitysprosessia. Vikatietokannasta löytyy ohjelmasta testauksen aikana löydetyt virheet sekä tehdyt korjaukset. Vikatietokannan avulla kaikki ohjelmiston kehitykseen liittyvät henkilöt voivat seurata testauksen etenemistä, sekä tarkastella löydettyjä virheitä. Erilaisia vikatietokantaratkaisuja löytyy niin suurilta ohjelmistotaloilta kuin avoimeen lähdekoodiinkin perustuvana. Monesti ratkaisuista löytyy myös muita ominaisuuksia joita voidaan käyttää kehitystyön apuna. [3, s. 87.]

6.2 Dokumenttipohjat

Testaus sisältää monia työvaiheita, joiden keskeisenä osana on dokumentaatio. Näin ollen testauksen suunnittelun yhteydessä on laadittava dokumenttipohjat, joiden avulla voidaan yhtenäistää ja helpottaa dokumentointia. Käyttämällä valmiita pohjia testaajat voivat käyttää aikansa ohjelman testien suorittamiseen ja vikojen raportointiin, eikä dokumentoinnin muotoilemiseen. Jos testiraportit ja pöytäkirjat laaditaan yhteistä mallia käyttäen, voidaan kehittää helpommin työkaluja esimerkiksi automaattista raportointia varten keräämällä tiedot dokumenttipohjassa määritellyistä kentistä. [3, s. 88.]

6.3 Testausautomaation työkalut

Testausautomaation työkalut ovat usein ohjelmistoyrityksen itse rakentamia testaustyökaluja, joilla voidaan testata esimerkiksi järjestelmän rajapintojen toimivuutta. Automaatiota voi hyödyntää myös graafisten käyttöliittymien kanssa muun muassa nauhoittamalla testitapauksia, jotka suoritetaan automaattisesti testattavassa ohjelmassa. Testausautomaatiota tarvitaan eniten regressiotestauksen ja kuormitustestauksen työvaiheissa. [3, s. 86.]

Ohjelmistotyötä voi myös valvoa automatisoidusti. Erilaisten analysaattoreiden avulla voidaan tutkia muun muassa järjestelmän komponentteja ja ohjelmakoodia. Esimerkiksi lähdekoodianalysaattoreiden avulla voidaan valvoa, että ohjelmakoodi vastaa syntaksia ja on kirjoitettu organisaation määrittämien käytäntöjen mukaisesti. [3, s. 87.]

7 KeyAqua-verkkotietojärjestelmän testaus

Tämän opinnäytetön tavoitteena oli erilaisten testausmenetelmien avulla suunnitella ja toteuttaa KeyAqua -verkkotietojärjestelmän järjestelmätestaus. Tässä

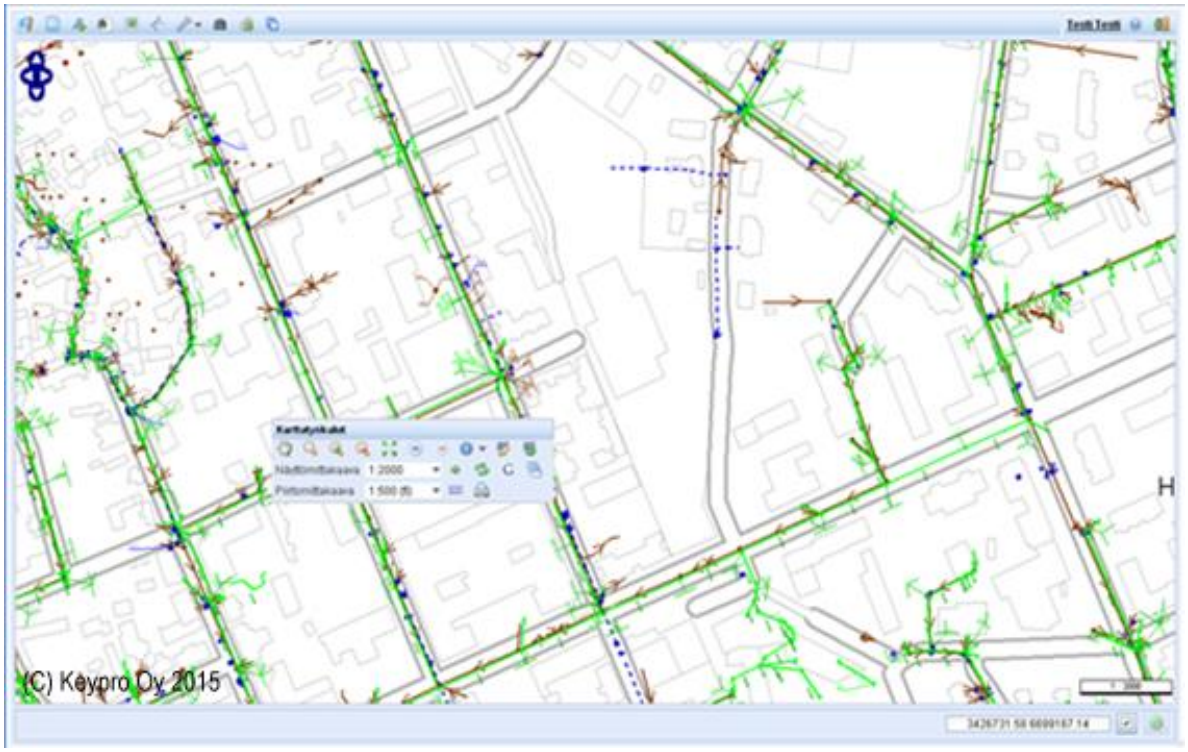
luvussa on esitelty opinnäytetyön toimeksiantaja Keypro Oy, testauksen kohteena ollut KeyAqua -verkkotietojärjestelmä sekä testauksen suorittaminen

7.1 Keypro Oy

Keypro Oy on suomalainen paikka- ja verkkotietojärjestelmiin erikoistunut yritys. Yritys on perustettu vuonna 1995 ja sillä on toimipisteet Joensuussa sekä Vantaalla. Yritys tarjoaa verkkotietojärjestelmien lisäksi erilaisia palveluita, kuten maastomittausta, suunnittelupalveluita sekä verkostojen dokumentointia. Yrityksen asiakaskuntaan kuuluu yli 150 toimijaa, muun muassa vesilaitoksia, teleoperaattoreita sekä kuntia.

7.2 KeyAqua-verkkotietojärjestelmä

KeyAqua on Keypro Oy:n kehittämä selainpohjainen verkkotietojärjestelmä vesi- ja viemäriverkon hallintaan. Ohjelmistoa voidaan käyttää verkoston dokumentointiin, suunnitteluun ja ylläpitoon. Järjestelmä sisältää myös työkalut asiakastietojen hallintaan sekä vesilaitostapahtumien dokumentointiin.



Kuva 7. KeyAqua-verkkotietojärjestelmä.

Valtaosa asiakkaista käyttää ohjelmistoa Keypron ylläpitämästä palvelinympäristöstä SAAS -palveluna (Software as a Service). Näin ollen loppukäyttäjän ei tarvitse asentaa mitään erillisiä ohjelmistoja, vaan käyttö tapahtuu internetselaimen kautta. Järjestelmä on mahdollista hankkia myös lisenssitoimituksena, jolloin ohjelmiston tarvitsemat ohjelmistomoduulit asennetaan asiakkaan palvelinympäristöön.

KeyAqua -ohjelmisto on käytössä yli viidelläkymmennellä vesihuoltoalan toimijalla, joiden koko vaihtelee suurista kunnallisista vesilaitoksista pieniin, muutaman kymmenen osakkaan vesiosuuskuntiin.

7.3 Testaussuunnitelma

KeyAqua -ohjelmiston testaus aloitettiin laatimalla testaussuunnitelma. Testaussuunnitelmassa kuvattiin aikataulu ja tarvittavat resurssit. Lisäksi suunnitelmassa kuvattiin käytettävä testiympäristö. Testauksen suorittamiseen valittiin kaksi henkilöä, joilla kummallakin on vuosien kokemus verkkotietojärjestelmistä,

mutta ei juurikaan tuntemusta KeyAqua -ohjelmistosta. Näin ollen ohjelmiston testauksen yhteydessä on mahdollista testata myös käyttöohjeen soveltuvuus. Testaussuunnitelman laadinnan yhteydessä määriteltiin testattavat ominaisuudet sekä toiminnallisuudet, joita ei tässä vaiheessa testata. Ennen testitapauksien laadintaa määritettiin lomakepohja, jota käytetään testitapausten kirjaamiseen.

Järjestelmätestauksessa KeyAqua –ohjelmistoa testattiin ensimmäisen kerran kokonaisena järjestelmänä. Sovelluskehittäjät suorittivat kehittämilleen toiminnoille ja moduuleille yksikkötestauksen, jotka suoritettiin lasilaatikkomenetelmällä. Lisäksi sovelluskehittäjät tekivät eräänlaisen savutestauksen omassa kehitysympäristössään, eli he tarkastivat että esimerkiksi kehitetty lomake käynnistyy oikein, tai että ohjelma yleensäkään käynnistyy ilman virheitä.

Testaussuunnitelmassa kuvattiin myös työasemat ja ohjelmistoversiot joilla ohjelmaa käytetään. Koska KeyAqua -asiakassopimuksissa on määritelty, että ohjelmisto toimii sekä Internet Explorer- että Mozilla Firefox -selaimella, suoritettiin kaikki testit käyttäen molempia tuettuja selaimia.

Testaussuunnitelmassa määriteltiin myös testauksen lopetuskriteerit. Testaus voidaan katsoa suoritetuksi, kun järjestelmästä ei löydy vakavia virheitä ja testattavat toiminnot toimivat teknisen määrittelyn mukaisesti. Mikäli ensimmäisellä testauskierroksella löytyy vakavia virheitä, jotka estävät ohjelman käytön, tulee virheiden korjauksen jälkeen järjestää uusi testauskierros.

7.4 Testitapaukset

Testitapaukset suunniteltiin siten, että niiden avulla saadaan mahdollisimman kattavasti käytyä ohjelman toiminnallisuus läpi. Testitapaus yleensä käsittää yhden lomakkeen toiminnallisuuden tai läheisesti toisiinsa liittyvät toiminnot. Testitapaus laadittiin käyttämällä musta laatikko -menetelmää, eli testauksen yhteydessä ei huomioitu järjestelmän sisäistä toimintaa.

Jokaiselle testitapaukselle määritettiin mahdolliset alkuehdot. Näitä voi olla esimerkiksi aikaisemman testitapauksen onnistuminen tai se, että työasemalta löytyy testitapauksessa tarvittavat ohjelmat. Testitapauksen eteneminen kuvattiin askelin, joista jokaisessa kerrottiin toiminta sekä odotettu lopputulos. Esimerkki testitapauksesta on kuvattuna liitteessä 1.

Testitapauksiin sisältyi myös tilanteita, joissa järjestelmälle annetaan virheellisiä syötteitä. Tällaisia olivat muun muassa kohteiden luominen niin, että kaikki lomakkeen pakolliset arvot eivät ole täytetty. Tällainen tilanne toistuu helposti myös loppukäyttäjällä. Testauksessa varmistettiin, että virhetilanteissa ilmoitukset ovat selkeitä ja antavat käyttäjälle tiedon, miksi toiminto epäonnistui.

7.5 Testauksen suorittaminen

Testaus suoritettiin testisuunnitelman mukaisesti kolmena eri tapahtumana. Testit suoritettiin neuvotteluhuoneessa, jossa oli kahden testaajan lisäksi testauksen suunnittelija. Ennen testitapausten aloittamista kirjattiin käytetyistä tietokoneista tekniset tiedot sekä käytetty selain ja sen versio.

Testaajat suorittivat testitapaukset järjestyksessä ja testitapauksessa kerrotun etenemisen mukaisesti. Aina yhden testitapauksen askeleen suorituksen jälkeen kirjattiin tulos testitapauslomakkeelle. Ensimmäisellä kerralla testaajat suorittivat testitapaukset käyttäen ohjekirjaa. Testaajien tuli löytää ohjekirjan avulla testitapauksessa esitellyt toiminnot, sekä oikea tapa näiden käyttämiseen. Mahdolliset puutteet käyttöohjeessa raportoitiin tuotepäällikölle.

7.6 Testausraportti

Jokaisen testaustapahtuman yhteydessä laadittiin testauspöytäkirja. Pöytäkirja on Excel-työkirja, jonka ensimmäinen sivu on koostesivu, josta käy ilmi testattava versio, testaajan nimi sekä testausajankohta. Sivulta löytyy myös taulukko, johon on numeroitu kaikki testausohjelmaan liittyvät testitapaukset. Nämä testi-

tapaukset löytyvät pöytäkirjasta omilta välilehdiltään. Koostesivulle merkitään testitapauksen tulos. Erilaisia tulosvaihtoehtoja oli

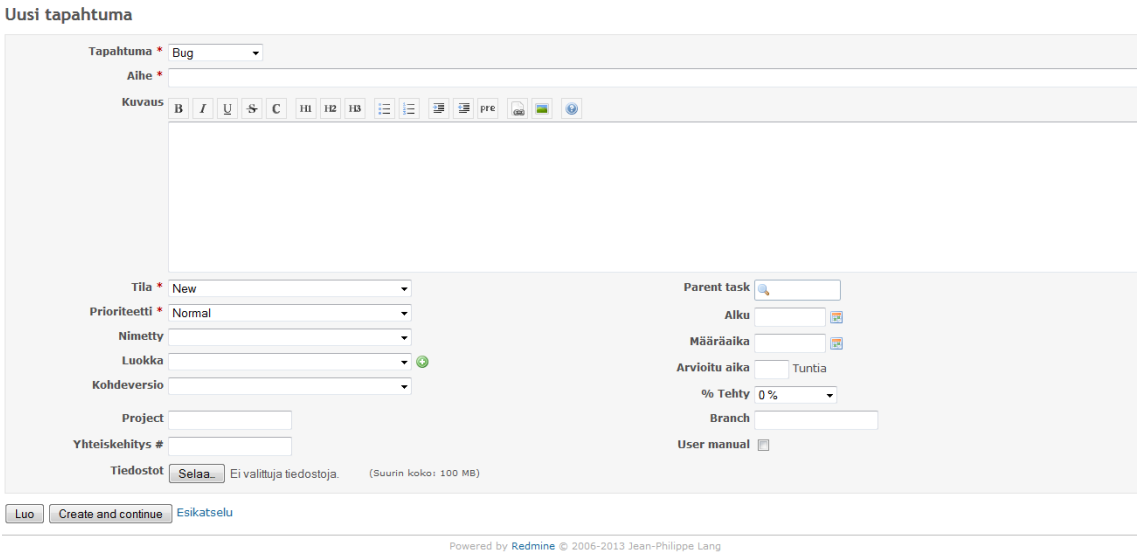
- OK, tarkoittaen että testitapaus on suoritettu ilman virheitä
- NOK, tarkoittaen että testitapauksen suorituksen aikana löytyi virheitä. Virheistä kirjoitettiin kuvaus kommenttikenttään
- tyhjä, tarkoittaen että testitapausta ei voitu suorittaa.

Testauksen jälkeen laadittiin testaussuunnitelmaa, pöytäkirjoja ja muistiinpanoja hyödyntäen testausraportti. Testausraportissa esiteltiin löydetyt virheet sekä arvio niiden vakavuudesta. Testausraportissa huomioitiin myös testaajien mahdolliset reaktiot virheilmoituksiin.

7.7 Virheiden kirjaaminen

Löydetyt virheet kirjattiin tiketteinä Redmine -projektinhallintajärjestelmään. Tiketit osoitettiin tuotepäällikölle, joka ohjasi niitä edelleen sovelluskehittäjille. Redmine -järjestelmässä myös testaajien oli helppo seurata virheenkorjauksen etenemistä, sillä järjestelmä lähettää sähköpostin tiketin seuraajille kun sen tilaa päivitetään. Tiketin kirjaamisen yhteydessä testaaja määritteli virheelle prioriteetin testaussuunnitelman mukaisesti, mutta lopullisen virheiden priorisoinnin teki kuitenkin tuotepäällikkö. Käytetyt prioriteettiasteet olivat

- matala (low)
- normaali (normal)
- korkea (high)
- kiireellinen (urgent).



Kuva 8. Redmine -projektinhallintajärjestelmä

7.8 Testauksen tulokset

Testauksessa löydetyistä virheistä osa oli niin vakavia, että ne estivät testitapauksen jatkamisen kokonaan ja virhetilanne korjaantui vain käynnistämällä järjestelmä uudestaan. Lisäksi kolme testitapausta jäi tässä vaiheessa suorittamatta, koska ohjelmamoduulit olivat vielä keskeneräisiä. Ennen kuin testattavana ollut versio voitiin ottaa asiakasympäristöissä käyttöön, täytyi löydetyt virheet olla korjattuina ja testaus saatua suoritetuksi onnistuneesti kaikille testitapauksille. Näin ollen ensimmäisen testauskierroksen tuloksista voitiin päätellä, että toinen testauskierros oli järjestettävä heti kun virheet olivat korjattu.

Toisella testikierroksella kaikki toiminnot olivat jo testattavissa, ja ensimmäisellä kierroksella havaitut virheet oli korjattu. Näin ollen toisen testauskierroksen jälkeen voitiin todeta, että testaussuunnitelmassa määritellyt lopettamiskriteerit oli saavutettu ja ohjelmaversio oli valmis julkaisua varten.

Testauksen yhteydessä löydettiin myös puutteita käyttöohjeista. Muutamia toimintoja oli kuvattu käyttöohjeessa hieman epäselvästi. Myös käyttöohjeiden korjausehdotukset ohjattiin tuotepäällikölle tiketteinä.

7.9 Asiakasympäristöjen testaus

Järjestelmätestauksen tulosten pohjalta määriteltiin myös käytännöt, joilla varsinaiset asiakasympäristöt tultaisiin testaamaan. Asiakasympäristöille tehdään käyttöönoton yhteydessä kattava testaus yhden testaajan toimesta. Asiakasympäristön testauksesta tehdään samanlainen pöytäkirja kuin versiotestauksestakin, ja se voidaan tarvittaessa esitellä myös asiakkaalle. Testausprosessi aloitetaan luomalla ensin testaajalle osoitettu testaustiketti kyseessä olevan asiakasprojektin alle. Tiketin luonnista vastaa toimitusprojektin projektipäällikkö. Tikettiin kirjataan myös määräaika, jolloin testaus on oltava valmis. Testaustiketissä mainitaan myös ominaisuudet joita ei testata.

Jokainen virhe kirjataan Redmine -projektinhallintajärjestelmään asiakasprojektin alle. Kirjatut virheet osoitetaan järjestelmän toimittajalle. Mikäli kyseessä on kuitenkin sovellusvirhe, järjestelmän asennuksesta vastaava siirtää tiketin Key-Aqua -pääprojektiin. Tiketti on mahdollista liittää myös toiseen, jo olemassa olevaan tikettiin. Tällä tavalla saadaan eritellyksi asiakasympäristössä oleva virhe, josta on jo sovelluskehityksessä tiketti olemassa. Mikäli virhe johtuu puolestaan ongelmasta palvelinarkkitehtuurissa, toimittaja siirtää tiketin ylläpidolle.

Samalla tavoin kuin järjestelmätestauksesta löytyneiden ohjelmavirheiden kohdalla, virheen korjaaja eli tässä tapauksessa toimittaja, vaihtaa korjauksen jälkeen tiketin tilan ratkaistuksi. Tämän jälkeen testaaja suorittaa kyseisen testitapauksen uudestaan, ja jos virhe on korjattu ja ominaisuus todettu toimivaksi, tiketti suljetaan. Mikäli ongelma jatkuu, palauttaa testaaja tiketin takaisin toimituksesta vastaavalle. Kun kaikki kyseiseen asennukseen liittyvät tiketit on korjattu, katsotaan asiakasympäristön testauksen olevan valmis.

Asiakasympäristöjen testaukseen liittyy olennaisena osana myös rajapintojen yli käytettävät ulkoiset palvelut. Monessa kaupungissa on mahdollista lukea käytettäviä kartta-aineistoja esimerkiksi WMS (Web Map Service) -rajapinnan kautta. Lisäksi esimerkiksi asiakasrekisteri on mahdollista lukea ulkoisesta tietolähteestä. Käytössä olevista rajapinnoista on kuvaus Redmine -projektissa ja näiden rajapintojen toimivuus todetaan testauksen yhteydessä.

7.10 Jatkokehitys

Versiotestaukseen on panostettava nykyistä enemmän. Mikäli asiakasympäristöistä löytyy versiopäivitysten jälkeen paljon virheitä, voi tämä pahimmassa tapauksessa ruuhkauttaa toimitusorganisaation toimintaa siten, että uusien asiakasprojektien aikataulut voivat siirtyä virheenkorjausten takia. Lisäksi on mahdollista, että jos virhe johtuu esimerkiksi tietokantaskeemasta puuttuvasta rivistä, joudutaan korjaus tekemään jokaiseen kyseistä versiota käyttävään asennukseen. Pahimmassa tapauksessa liian aikaisin julkaistu versio voi aiheuttaa sen, että sekä sovelluskehitys että toimitusorganisaatio joutuvat keskittymään olemassa olevien ominaisuuksien korjaukseen.

Tulevien versiopäivitysten yhteydessä testitapauksia laajennetaan kattamaan myös uudet ominaisuudet. On myös mietittävä laajuus, jossa asiakasympäristöt testataan versiopäivityksen yhteydessä. Mikäli kaikki asiakasympäristöt käydään laajan, koko ohjelman kattavan testausohjelman mukaisesti, tulee tämä vaatimaan paljon resursseja testaukseen. Voi kuitenkin olla perusteltua käydä ainakin ensimmäiset päivitykset läpi perusteellisesti, koska sitä kautta nähdään todellisten virheiden määrä versiopäivityksessä. Mikäli useassa ympäristössä toistuu samoja virheitä, on esimerkiksi tietokannan päivitysskriptit tarkastettava perusteellisesti.

Testauksen suunnittelun ja suorittamisen yhteydessä ei huomioitu ollenkaan mahdollisuutta käyttää erilaisia testausautomaatiotyökaluja. Tässä on kenties suurin jatkokehityksen kohde tulevaisuudessa. Kuten yllä todettiin, KeyAqua -järjestelmästä julkaistaan uusia versioita, jotka sisältävät uusia ominaisuuksia. Siksi testitapausten määrä tulee kasvamaan tulevaisuudessa. Lisäksi asiakasmäärän odotetaan kasvavan, joten käytännössä joudutaan suorittamaan entistä enemmän samoja testejä useampiin ympäristöihin. Jos näitä testauskertoja saataisiin automatisoiduksi edes osittain, säästäisi se testauksen vaatimia resursseja merkittävästi.

Testauksen suunnittelussa haasteena oli myös järjestelmän määrittelydokumenttien epäselvyys. Kaikissa toiminnoissa ei ollut järjestelmän kuvauksen pe-

rusteella mahdollista yksiselitteisesti tulkita miten toiminnon tulisi toimia. Näin ollen testitapauksen laadinnan yhteydessä jouduttiin tarkentamaan määrittelyä tuotepäällikön kanssa. Kun tulevissa versioissa kehitetään uusia ominaisuuksia, on huolehdittava, että toiminnoista on olemassa selkeä määrittely.

8 Pohdinta

Tämän opinnäytetyön tarkoituksena oli perehtyä ohjelmistotestaukseen sekä selvittää mitä erilaisia osa-alueita ohjelmistotestaukseen kuuluu. Näiden pohjalta suunniteltiin KeyAqua -verkkotietojärjestelmän testaus. Koska testaus on kriittinen osa ohjelmistojen kehitystä, löytyi kirjallista materiaalia aiheesta hyvin. Onnistuin mielestäni saamaan melko kattavan kuvan testauksen eri menetelmistä sekä testauslajeista. Koska tässä vaiheessa ohjelmiston testausta ei käytetty mitään testausautomaatio-ohjelmia, en juurikaan kerännyt näistä tietoa. Tässä voisi olla seuraavan opiskelun ja jatkokehityksen paikka. Toinen hyvä kehityskohde olisi testauksen kattavuuden mittaaminen.

KeyAqua -ohjelmiston testaus onnistui mielestäni hyvin. Testauksessa saatiin kiinni kriittisimmät virheet sekä muutenkin koko prosessia saatiin selkeytettyä. Nyt testaajille, järjestelmän toimittajille sekä sovelluskehittäjille on selkeämpää miten jatkossa virheet dokumentoidaan, miten niiden korjaus järjestetään ja millä tavoin korjausten etenemistä voidaan seurata. Testauksen yhteydessä saatiin myös hahmoteltua miten testausta olisi hyvä kehittää tulevaisuudessa entistä tehokkaammaksi. Hyvänä puolena on myös se, että prosessia voidaan soveltaa helposti myös muiden Keypro Oy:n tuotteiden testausta suunniteltaessa.

Työtä kirjoittaessani opin paljon ohjelmistotestauksesta. Aikaisemmat tiedot rajoittuivat oikeastaan koulun opintojaksoihin, mutta käytännön kokemusta minulla ei aiheesta ollut. Oikeastaan vasta nyt minulle alkaa muodostua käsitys siitä, kuinka iso osa ohjelmistokehitystä testaus on, vaikka jostain syystä testauksen roolia monesti vähätellään. Testauksen merkitys näkyy kuitenkin hyvin tärkeänä osana päivittäistä työtäni, sillä yksi minun työtehtävistäni on KeyAqua -

ohjelmiston käyttäjäkoulutusten pitäminen. On aina ikävää, kun ohjelmasta löytyy virheitä kesken koulutuksen johtuen puutteellisesta testauksesta.

Lähteet

1. Pressman, R. Software Engineering: A Practitioner’s Approach. New York: McGraw-Hill. 2001.

2. Haikala, I. & Mikkonen, T. Ohjelmistotuotannon käytännöt. Helsinki: Talentum. 2011.

3. Kasurinen, J. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo. 2013

4. Khan, M. 2012. A Comparative Study of White Box, Black Box and Grey Box Testing Techniques. 2012. [Viitattu 10.4.2015]. Saatavissa: http://thesai.org/Publications/ViewPaper?Volume=3&Issue=6&Code=IJACSA&SerialNo=3

5. Microsoft Developer Network. Integration testing. 2015. [Viitattu 11.4.2015.] Saatavissa: https://msdn.microsoft.com/en-us/library/aa292128%28v=vs.71%29.aspx

6. Acharya, S. Bridge between Black Box and White Box – Gray Box Testing Technique. 2015. [Viitattu 12.4.2015.] Saatavissa: http://www.ijecse.org/wp-content/uploads/2012/12/Volume-2Number-1PP-175-185.pdf

7. Khannur, A. Khannur's Software Testing Knowledge Center: 4.6 System Testing. 2013. [Viitattu 12.4.2015.] Saatavissa: http://www.khannur.com/sst4.6.htm

8. Haikala, I. & Märijärvi, J. Ohjelmistotuotanto. Helsinki: Talentum. 2004.

Liitteet

Liite 1. Esimerkki testitapauksesta

		Testaaja:				
		Testauspvm:				
		Testattava versio:				
		3. Tulostus				
		Alkuehto: Testattavaan koneeseen on asennettu PDF-selain sekä määritetty tulostin. Tasohallinnassa on ainakin yksi johtolaji sekä taustakartta näkyvissä.				
	Askel	Toiminta	Odotettu tulos	Syötteet	Tulos	Kommentti
	3.1	Siirrytään mittakaavaan 1:1000	Karttanäkymä päivittyy annettuun mittakaavaan	1:1000		
	3.2	Painetaan Tulosta -painiketta	Tulostuslomake avautuu	-		
	3.3	Valitaan paperikooksi A4, suunnaksi pysty ja tarkkuudeksi 300dpi	Lomakkeen arvot muuttuvat vastaaviksi	A4, pysty, 300dpi		
	3.4	Syötetään tulosteen mittakaavaksi 1:400	Mittakaavaksi muuttuu 1:400	1:400		
	3.5	Syötetään Tekijä -kenttään oma nimi	Tekijä -kentästä löytyy arvo			
	3.6	Painetaan Käynnistä tulostus -painiketta ja viedään hiiri kartalle	Hiiren mukana kulkee oranssi suorakaide, joka kertoo alueen joka ollaan tulostamassa	-		
	3.7	Vaihdetaan tulosteen mittakaavaksi 1:600	Suorakaiteen koko muuttuu	-		
	3.8	Vaihdetaan suunnaksi vaaka	Suorakaiteen muoto muuttuu	-		
	3.9	Hyväksytään tulostettava alue	Tulostus käynnistyy: Selain avaa uuden välilehden, ja tuloste avautuu oletus-PDF -selaimeen. Tulosteesta löytyy valittuna olleet johtolajit, sekä otsikkotaulu ja Tekijä -kohdasta testaajan oma nimi	-		
	3.10	Otetaan uusi tuloste vapaasti valituilla arvoilla	Tuloste avautuu syötetyillä arvoilla jossa kohdetasot jotka olivat valittuina	-		