jamk.fi

Web-sovelluksen testaus ja käytettävyys

Eetu Roponen

Opinnäytetyö
Toukokuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), mediatekniikan koulutusohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Roponen Eetu Pekka Aleksi	Opinnäytetyö, AMK	Toukokuu 2017
	Sivumäärä 42	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi
Web-sovelluksen testaus ja käytettävyys

Tutkinto-ohjelma
Mediatekniikka

Työn ohjaaja(t)
Kari Niemi

Toimeksiantaja(t)
Axiell Finland Oy

Tiivistelmä

Työn tilaajana oli Axiell Finland Oy, joka on osa Axiell Group -konsernia. Axiell Finland Oy on erikoistunut suomalaisten kirjastojen IT-palveluiden tuottamiseen, ja yksi heidän päätuotteistaan on Aurora-kirjastojärjestelmä. Aurora on tarkoitettu ensisijaisesti kirjastohenkilökunnan käytettäväksi, jonka avulla he voivat esimerkiksi varata teoksia, lisätä asiakkaita ja hallinnoida yleisesti kaikki tarvittavia tietoja. Aurora on toteutettu aiemmin Silverlight-tekniikkaan perustuen, mutta sen tuen päättymisen myötä, alettiin Auroraa kehittää uusin tekniikoin.

Opinnäytetyössä testattiin kehityksessä olevaa versiota Aurorasta, niin ohjelmisto- kuin käytettävyystestausta suorittaen. Tavoitteena oli parantaa sovelluksen käytettävyyttä ja yleistä toimivuutta. Testit suoritettiin pääosin vertailemalla kehitysversion toimivuutta ja käytettävyyttä sovelluksen vanhempaan versioon, sekä tarkkailemalla täyttääkö sovellus sille asetetut vaatimukset. Käytetyt testausmenetelmät vaihtelivat riippuen testauskohteesta, mutta usein työn aikana käytettiin mustalaatikko-testausta (black box testing) ja harmaalaatikko-testausta. Työn aikana muutama yksikkö kokeili Auroran kehitysversiota, jonka takia ei koettu tarpeelliseksi tehdä haastatteluun perustuvia käytettävyystestejä.

Tuloksena saatiin kartoitettua sovelluksen yleistä toimivuutta ja käyttöliittymän parannuskohteita. Testien tuloksina saatiin myös tietoja eri näyttökokojen, selainten ja selainversioiden vaikutuksista sovelluksen toimivuuteen. Näiden tietojen avulla saatiin parannettua sovelluksen yleistä käytettävyyttä sekä virheettömyyttä. Tulokset välttävät myös toistamatta samoja virheitä uudestaan jatkokehityksen aikana.

Avainsanat (asiasanat)
ohjelmistotestaus, käytettävyys, käytettävyystestaus, verkkosovellus, käyttöliittymä

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Roponen Eetu	Bachelor's thesis	May 2017
		Language of publication: Finnish
	Number of pages: 42	Permission for web publication: x

Title of publication
Testing and usability of web software

Degree programme
Media Engineering

Supervisor(s)
Niemi Kari

Assigned by
Axiell Finland Oy

Abstract

This project was assigned by Axiell Finland Oy, a part of Axiell Group corporation. Axiell Finland Oy specializes in producing IT -services for Finnish libraries, and one of their main products is Aurora library software. Aurora is designed primarily for library staff. With Aurora, the staff can e.g. reserve books, add customers and control all generally needed info. Aurora is built with Silverlight, however with the browser support expiring, its development was started with fresh technologies and methods.

The thesis focuses on software and usability testing of the development version of Aurora. The main goal was to improve its usability and general functionality. For the most part, the tests were performed by comparing the development version to the old version and making sure Aurora fulfilled its requirements. The testing methods varied depending on the focus of the test, and mainly black box and gray box testing were used. During the thesis project, a few of the library units using the old version of Aurora also tested the development version, which is why no interview-based usability tests were conducted.

As a result, numerous flaws and errors regarding functionality and usability were found. In addition to these test results, information about screen size, browsers and browser versions affecting Aurora's functionality were found. With this kind of information, it was possible to improve the usability and functionality of Aurora. The test results also help avoiding the same problems in the future.

Keywords/tags (subjects)
software testing, usability, usability testing, web software, user interface

Miscellaneous

Sisältö

Kuviot

1 Työn lähtökohdat

1.1 Axiell Finland Oy

Axiell Finland Oy on osa Axiell Group -konsernia, joka kehittää ratkaisuja kirjastoille, kouluille, arkistoille, museoille ja yrityksille 35:ssä eri maassa. Konsernin pääkonttori sijaitsee Ruotsin Lundissa ja konserni työllistää yhteensä noin 290 työntekijää 24 toimistossa (Axiell Finland Oy n.d.).

Axiell Finland Oy on erikoistunut suomalaisten kirjastojen IT-palveluiden tuottamiseen. Yritys työllistää Jyväskylän ja Vantaan toimipisteissä yli 20 henkilöä. Henkilökunta koostuu kirjastoammattilaisista, ohjelmoijista, tuotekehittäjistä ja erilaisista erikoisasiantuntijoista. Jyväskylän toimipiste sijaitsee Lutakon kaupunginosassa.

1.2 Aurora-sovellus

Aurora on alunperin Silverlight-tekniikalla toteutettu selainpohjainen kirjastojärjestelmä, joka on suunniteltu kirjastohenkilökunnan käytettäväksi. Järjestelmän avulla henkilökunta voi esimerkiksi hallita kirjaston toimintaprosesseja, kuten aineistohankintaa, kokoelmahallintaa, asiakaspalvelua, lainauksenvalvontaa ja toiminnan vaikuttavuutta. Auroraa voidaan käyttää myös itsepalvelupisteissä, ja sillä voi hallinnoida niteiden, asiakkaiden, lainojen ja varausten lisäksi myös esimerkiksi muistutuksia, asiakasviestejä ja sähköisiä kirjeitä.

Järjestelmä tukee useita rajapintoja ja on muokattavissa kirjastojen erilaisia muuntuvia käyttötarpeita vastaavaksi. Aurora-järjestelmä on tällä hetkellä käytössä ympäri Suomea erilaisissa kirjastoissa, kuten yleisissä-, oppilaitos- ja erikoiskirjastoissa. (Axiell Finland Oy n.d.)

1.3 Tavoite

Kun tieto Microsoft Silverlight tuen päättymisestä julkistettiin, aloitettiin myös Auroran kehitys eri tekniikoilla. Sovellus pysyy edelleen selainpohjaisena järjestelmänä, mutta se toteutetaan HTML- ja JavaScript-tekniikoita käyttäen.

Tavoitteena oli parantaa sovelluksen käytettävyyttä ja testata sen yleinen toimivuus eri tekniikoilla muutoksen yhteydessä. Testausvaiheessa keskityttiin niin käytettävyyteen kuin sovelluksen yleiseen virheettömyyteen. Tavoitteena oli myös kuitenkin pitää sovelluksen ulkoasu ja käyttöliittymä tarpeeksi lähellä Silverlight-versiota, jotta sitä käyttänyt kirjastohenkilökunta pystyy käyttämään uutta versiota ilman täydellistä sovellukseen perehdyttämistä. Sovellukseen tuli uusia ominaisuuksia ja suurimmat käyttäjälle näkyvät muutokset olivat yksittäisiä toimintoja, joiden tavoitteena oli parantaa käytettävyyttä, sekä mahdollistaa yksityiskohtaisempia toimintojen toteutuksia. Koska suurta uudistusta käyttöliittymään ei tehty, käytettävyyttä pyrittiin parantamaan esimerkiksi nopeuttamalla hakutoimintoja, tekemällä parannuksia käyttöliittymään muuttamatta ikoneita tai termistöä ja kehittämällä ohjelmasta responsiivinen.

2 Käytettävyys

2.1 Käytettävyys yleisesti

Käytettävyydellä ja sen parantamisella tarkoitetaan sitä, että käyttäjän ja laitteen välistä yhteistoimintaa saadaan tehokkaammaksi ja käyttäjää miellyttävämmäksi. Käytettävyys käyttää hyväksi kognitiivisen psykologian sekä ihmisen ja koneen vuorovaikutuksen tutkimusta. (Kuoppala, Parkkinen, Sinkkonen & Vastamäki 2006, 17-20.)

Nielsen on määritellyt käytettävyyden yhdeksi osaksi käyttökelpoisuutta, mutta siihen vaikuttavia tekijöitä sen käytettävyyden lisäksi useita. Käytettävyyden on kuitenkin oltava harkittu ja yleisesti hyvä, jotta tuote olisi käyttökelpoinen. Huono käytettävyys usein ilmeneekin tehokkuuden laskemisena pitkällä aikavälillä, sekä käyttäjän turhautumisena. (Kuoppala ym. 2006, 17-20.)

Nielsenin (2012) mukaan käytettävyys voidaan jakaa seuraaviin viiteen eri alueeseen:

1. Opittavuus (Learnability). Kuinka helposti käyttäjä kykenee suorittamaan yksinkertaisia tehtäviä ensimmäisellä käyttökerrallaan?
2. Tehokkuus (Effiency). Kun käyttäjä on oppinut käyttöliittymän, kuinka nopeasti he pääsevät haluttuun lopputulokseen?
3. Muistettavuus (Memorability). Kun käyttäjät palaavat käyttämään tuotetta tauon jälkeen, kuinka helposti he saavuttavat aiemman tason?

4. Virheettömyys (Errors). Kuinka paljon ja kuinka vakavia virheitä käyttäjät kohtaavat? Kuinka helposti virheistä selvitään eteenpäin?
5. Tyytyväisyys (Satisfaction). Kuinka tyytyväisiä käyttäjät ovat tuotteeseen? Kuinka mukava tuotetta on käyttää?

Muita laatua käsitteleviä osakohtia on käyttökelpoisuus (utility), joka viittaa tuotteen toimivuuteen. Eli tekeekö tuote sen mitä käyttäjä haluaa sen tekevän? Käytettävyys ja käyttökelpoisuus ovat yhtä tärkeitä määriteltäessä yhdessä tuotteen hyödyllisyyttä. Tuotteen käytettävyydellä ei ole väliä, mikäli tuote ei tee sitä mitä käyttäjä haluaa sen tekevän. Samalla tavalla käyttäjä kokee tuotteen käyttökelvottomaksi, mikäli hän ei huonon käytettävyyden takia pääse sillä haluttuun lopputulokseen. Nielsen määrittelee käyttökelpoisuuden tarkoittavan sitä, että tarjoaako tuote halutut ominaisuudet ja toiminnot. Käytettävyyden hän on määritellyt tarkoittavan sitä, kuinka helppo ja mukava ominaisuuksia ja toimintoja on käyttää. (Nielsen 2012.)

2.2 Käytettävyys web-tekniikoissa

Jotta käyttäjä pystyy käyttämään tuotetta, hänen täytyy pystyä havaitsemaan tuotteessa kaikki tehtävän suorittamisen kannalta oleelliset asiat. Käyttäjän täytyy myös pystyä seuraamaan toimenpiteittensä vaikutusta tuotteen tilaan. Mikäli hän ei näe kaikkea mitä kuuluisi, on yleensä syynä joko se, että hänen huomionsa keskittyi käyttöliittymässä väärään asiaan tai että asiat hahmottuvat hänelle joko väärin tai ei lainkaan. (Kuoppala ym. 2006, 155-158.)

Havaitseminen ei kuitenkaan ole pelkästään aisteihin perustuvaa. Jotta käyttäjä pystyy käyttämään tuotetta, hänen täytyy pystyä tunnistamaan asiat ja mieltämään ne joksikin. Ei siis riitä, että asiat ainoastaan löytyvät käyttöliittymästä, vaan käyttäjän täytyy havaita ja tunnistaa ne. Käyttäjä voi nähdä verkkosivuston yläreunassa esimerkiksi paljon käytetyn, yksinkertaistetun kuvan talosta tietämättä, mitä se tarkoittaa. Kun käyttäjä klikkaa ikonia ja ymmärtää sen vievän sivuston "kotisivulle", hän ymmärtää myös sen assosiaation koti-sanaan ja näin muistaa sen todennäköisemmin jatkossa. (Kuoppala ym. 2006, 67-69.)

Aistien lisäksi havainnointiin vaikuttavat ennakkokäsitykset, muistot ja odotukset. Esimerkiksi aiemmin uutisia julkaisevalla verkkosivustolla käynyt käyttäjä tietää

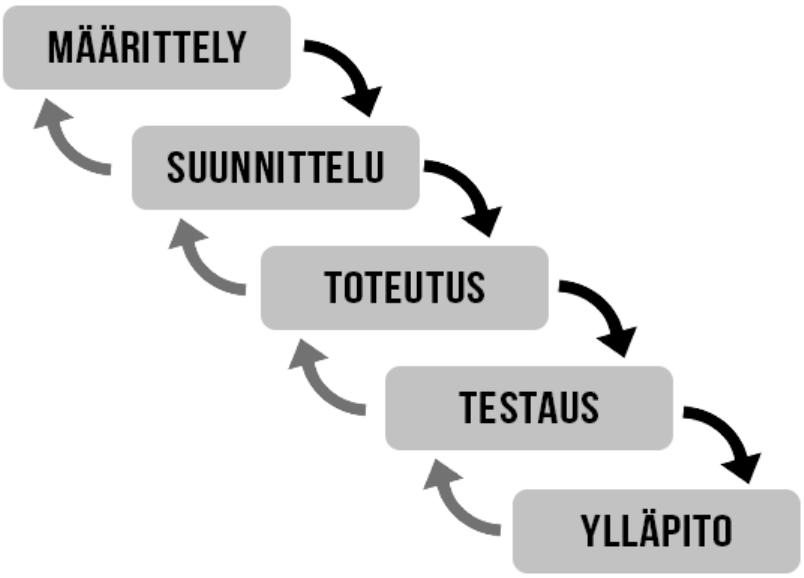
kutakuinkin, mitä odottaa, kun hän menee vieraalle uutissivustolle. Käyttäjä voi esimerkiksi muistaa aiemmista kokemuksista ja sitä kautta odottaa vieraalta sivustolta, että uutiset jaotellaan eri kategorioihin ja että etusivulla on usein tuoreimmat uutiset. (Kuoppala ym. 2006, 67-69.)

Käytettävyys on myös ratkaiseva tekijä käyttäjän tehdessä valintaa kilpailijoiden välillä. Mikäli esimerkiksi kaksi ilmaista verkkosivustoa tarjoavat hänelle sopivat palvelut, valitsee hän todennäköisestimyös jatkossa sen sivuston, jossa on parempi käytettävyys. Tuotteen hyvä käytettävyys saa käyttäjät pysymään tuotteen parissa ja käyttämään sitä uudestaan. (Nielsen 2012.)

3 Testaus

3.1 Ohjelmistotestaus

Ohjelmistotestauksella tarkoitetaan työtä, jota tehdään siksi, että toteutettavasta ohjelmistotuotteesta saadaan toivotun kaltainen ja että kaikki siihen valmiiksi saadut ominaisuudet varmasti toimivat niille tarkoitetulla tavalla. Testaustyötä voidaan myös pitää jatkuvana vertailutyönä: Tehtävänä on tarkastaa, että se, mitä on saatu valmiiksi, vastaa sitä, miten se on suunniteltu, sekä tunnistaa kohdat, jotka eivät vastaa suunniteltua tulosta. Testauksesta on yleisesti mielikuva, että se on vain yksi työvaihe ohjelmistotuotannon perinteisessä vesiputousmallissa. Vesiputousmallissa on perinteisesti viisi eri työvaihetta, jotka ovat esitetty kuviossa 1. (Kasurinen 2013, 13-14.)

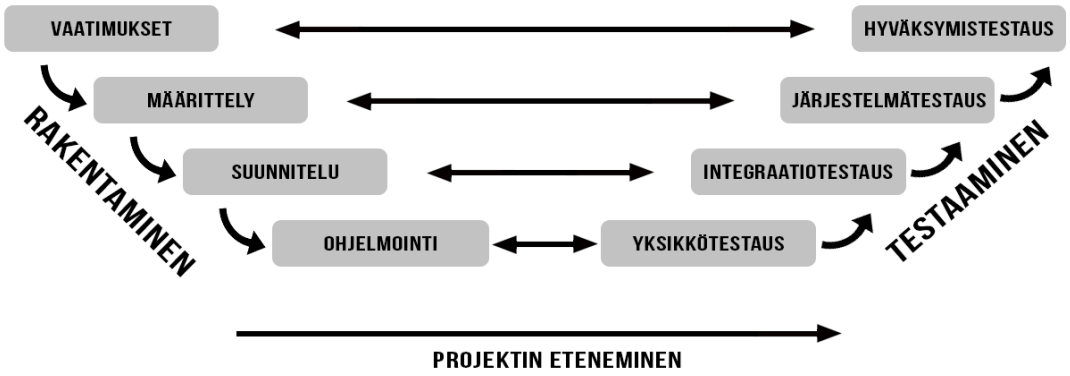


Kuvio 1. Yleisesti käytetty vesiputousmalli ohjelmistotuotannossa

Tavoitteena vesiputousmallissa on, että projekti etenee vaiheittain eikä vaiheen valmistumisen jälkeen jouduta palaamaan kuin korkeintaan yksi taaksepäin. Projekti aloitetaan tekemällä vaatimusmäärittely, jossa kootaan kaikki saatavilla oleva tieto kuten kannattavuuslaskelmat, alustava lista asiakastarpeista sekä käyttäjäkunnasta ja esimerkiksi markkina-analyysi. Näistä vaatimuksista muodostetaan suunnitelma, jonka avulla määritellään tuotteen toiminnalliset vaatimukset, rakenne sekä itse projektisuunnitelma koko kehitysprojektin toteuttamista varten. Tätä suunnitelmaa hyödyntäen ohjelma ja sen vaatima laitteisto toteutetaan ja kootaan. Kaikkien osakokonaisuuksien oltaessa valmiita, pienemmät kokonaisuudet kootaan ja yhdistetään toimivaksi kokonaisuudeksi. Tämän jälkeen aloitetaan varsinaisen kokonaisuuden testaaminen ja virheiden korjaus. Kokonaisuuden ollessa siinä kunnossa että, se ei enää sisällä toimintaan vaikuttavia merkittäviä virheitä tai toteuttaa kaikki siltä odotetut toiminnot, testausvaihe voidaan saattaa päätökseen. Tuote on valmis käyttöönotettavaksi, kun testausvaihe on saatu onnistuneesti loppuun. (Kasurinen 2013, 14.)

Vesiputousmallia pidetään kuitenkin usein nykyisin vanhanaikaisena. Esimerkiksi vaatimusmäärittelyä tehdessä on täydellinen vaatimusten määritteleminen usein etukäteen todella vaikeaa. Siksi nykyisin kehitystyölle nähdään parempana

vaihtoehtona esimerkiksi V-malli (ks. kuvio 2). Kasurinen (2013) esittää teoksessaan Ohjelmistotestauksen käsikirja vaihtoehtoisen ohjelmistotuotannon toimintamallin. V-mallin rakentamispuoli muistuttaa paljon vesiputousmallia, mutta eroavaisuudet näkyvät erityisesti testauksessa. V-mallissa testaus ei ole yksittäinen vaihe, vaan tuotetta testataan jatkuvasti läpi kehityksen. Yksikkötesteillä tarkastetaan ohjelmointityö, integraatiotestauksella suunnitelmien toteutuminen, järjestelmätestauksella määrittelyjen paikkaansapitävyys ja hyväksymistestauksella yleiset vaatimukset. Ohjelman läpäistäessä jokaisen testaustason, se on valmis käyttöönotettavaksi ja siirrettäväksi ylläpitoon.



Kuvio 2. Vaihtoehtoinen ohjelmistotuotannon toimintamalli, V-malli

Yksikkötestauksella varmistetaan, että juuri kehitetty toiminto, muutos tai komponentti toimii ainakin pääosin. Koska yksikkötestit ovat usein lyhyitä ja pieniä kokonaisuuksia, testaajana toimii usein kohteen ohjelmoija. Tällä tavalla pyritään karsimaan isommat virheen jo ennen kuin komponentti liitetään osaksi isompaa kokonaisuutta. Usein kuitenkin ongelmaksi muodostuu se, että yksittäinen komponentti ei pysty yksin toteuttamaan sen tarkoitusta täysin. Komponentit voivat olla riippuvaisia toisistaan, jotta ne toimivat oikein, ja niiden välillä voi olla jatkuvaa vuorovaikutusta. Näissä tapauksissa voidaan joko tehdä testikomponenttejä (mock objects, test studs), joiden tehtävä on simuloida toisen komponentin toimintaa, tai testata riippuvaiset komponentit yhdessä. (Kasurinen 2013, 51-53.)

Integrointitestaus (integration testing) on yksikkötestauksen jälkeinen työvaihe, jossa tuotteen eri osat sovitetaan yhteen ja tavoitteena on saada lopulta koko järjestelmä

toimimaan yhtenä kokonaisuutena. Integrointitestauksessa valmistunut komponentti tai osa-alue yhdistetään toiseen jo testattuun ja toimivaan kokonaisuuteen. Integraatiotestauksessa toteutetaan siten myös laajempia testitapauksia kuin yksikkötestauksessa, mutta ei kuitenkaan koko sovelluksen mittakaavan mukaisia. Tällä tavoin testattuun kokonaisuuteen lisäämällä ja integrointitestausta suorittamalla kokonaisuutta voidaan laajentaa ja pitää se kehityksen aikana ehjänä. (Kasurinen 2013, 54-56.)

Järjestelmätestaus (system testing) on kolmas vaihe perinteisessä V-mallissa, johon siirrytään integrointitestauksen jälkeen. Kun komponentit on koottu yhdeksi toimivaksi kokonaisuudeksi, aloitetaan koko järjestelmän yhtenäinen testaus. Koska projektien tavoitteet, resurssit ja lukuisat muut yksityiskohdat voivat vaihdella, ei järjestelmätestauksella tarkoiteta mitään yksittäistä testaustapaa tai menetelmää, vaan yksinkertaisesti vaihetta, jonka aikana testataan järjestelmä kokonaisuutena. Järjestelmätestauksesta puhuttaessa se usein kuitenkin mielletään työvaiheeksi, jossa tehdään mustalaatikko- ja lasilaatikko-testausta (black box testing, glass box testing/white box testing). Vaiheen tavoitteena on, että testattu järjestelmä toimii kokonaisuutena, ja että se toteuttaa kaikki sille asetetut tavoitteet. Testit suoritetaan usein vielä testausympäristössä, koska virheitä etsitään edelleen yksittäisistä komponenteistä, eivätkä isommatkaan muutokset järjestelmään ole epätavallisia. Kun järjestelmä on todettu toimivaksi kokonaisuudeksi ja yksittäisistä komponenteistä ei löydy virheitä, siirrytään hyväksymistestaukseen. (Kasurinen 2013, 65-68.)

Hyväksymistestaus (acceptance testing) on perinteisen V-mallin testauksen viimeinen työvaihe, jonka tavoitteena on todentaa järjestelmä riittävän korkealaatuiseksi ja kyvykkääksi täyttämään sille annetut vaatimukset. Hyväksymistestauksella usein tarkoitetaan sitä työvaihetta, jossa järjestelmä tarkastetaan virallisesti. Kun asiakas on todennut hyväksymistestauksen onnistuneeksi, usein tässä vaiheessa järjestelmä siirtyy lakiteknisesti asiakkaan haltuun ja se todetaan valmiiksi. (Kasurinen 2013, 13.)

Ketterän ohjelmistokehityksen (agile software development) menetelmistöjen yleistyessä, myös testaajien rooli on muuttunut. Pelkän testauksen lisäksi ketterässä ohjelmistokehityksessä testaajan työnkuva voi sisältää myös yleistä laaduntarkkailua sekä ohjelman analysointia. Ketteriä menetelmiä käyttävissä projekteissa testaajat

ovat usein osa kehitysryhmää, ja ovat siksi aktiivisesti kehittäjien kanssa vuorovaikutuksessa. Näillä keinoilla kehittäjät ja testaajat voivat ennaltaehkäistä virheitä jo ennen niiden implementointia. (Oluwole 2014.)

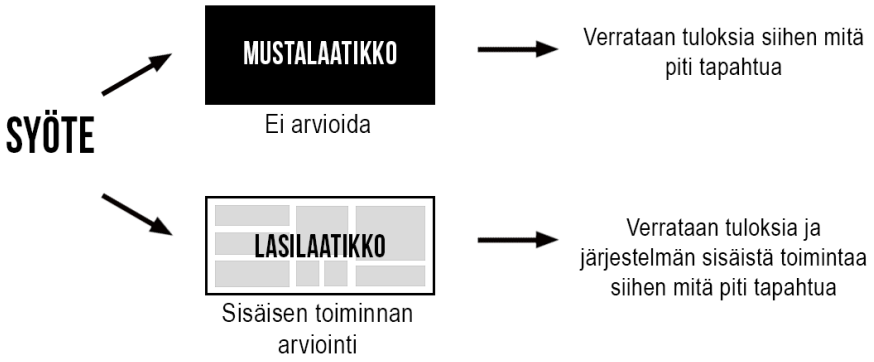
3.2 Testausmenetelmät

Jokaisen testausvaiheen sisällä voidaan vaihtaa käytettäviä testausmenetelmiä, joiden avulla voidaan kohdentaa testaus koskemaan paremmin haluttua kohdetta. Mustalaatikko -testauksella (black box testing) tarkoitetaan perinteisintä testausmenetelmää, joka tapahtuu vertailemalla vaatimuksia ja saatuja tuloksia keskenään. Mustalaatikko-termi on muodostettu siitä, että testaaja ei näe ohjelman sisäistä toimintaa testien aikana, vaan tarkastelee ainoastaan lopputulosta (Black Box Testing: Types and techniques of BBT 2017). Käytännössä testaus tapahtuu niin että, ohjelmalle annetaan syötteitä ja katsotaan, millaisia tuloksia ohjelma saa aikaiseksi (ks. kuvio 3). Näitä tuloksia verrataan toivottuihin tuloksiin ja vaadittuun toimintaan. Mustalaatikko-testeissä ei kiinnitetä huomiota siihen, kuinka ohjelma käsittelee syötteet, tai mitä sen sisällä tapahtuu, vaan tarkastellaan ainoastaan sen antamia tuloksia. Tästä syystä mustalaatikko-testauksella voikin jäädä löytämättä osa virheistä ja parannuskohteista (Beal n.d.). Tätä testausmenetelmää voidaankin käyttää kaikissa testauksen työvaiheissa, jos ohjelmasta on vain toimiva versio. Mustalaatikko-testauksen vahvuuksia on se, että testaajan ei tarvitse perehtyä itse ohjelmointiin tai tietää ohjelman toiminnasta funktiotasolla vaan riittää, että testaaja kykenee käyttämään ohjelmaa. (Kasurinen 2013, 65-66.)

Mustalaatikko-testauksessa on tärkeää, että testaaja todentaa kohteen toimivuuden kaikissa olosuhteissa. Tämä tarkoittaa usein sitä, että testaajan täytyy kokeilla komponentin toimivuus sille toivotuilla syötteillä ja ei-toivotuilla. Esimerkiksi hakutulosten ollessa listattuna useammalle ohjelman sivulle, täytyy testaajan testata sivujen välinen navigointi syöttämällä kenttään positiivinen ja negatiivinen luku, erikoismerkkejä sekä kirjaimia. Koska menetelmässä ei perehdytä ohjelman funktionaaliseen toimintaan, jää tehtäväksi kokeilla hakutulosten selaamista eri arvoilla. Tästä syystä mustalaatikko-testausta automatisoidaankin laajoissa projekteissa toistuvan työn minimoimiseksi. Mikäli ei kuitenkaan käsitellä laajoja

datamääriä tai samankaltaisia testikohteita ei ole useita, ei testauksen automatisointi ole kannattavaa. (Kasurinen 2013, 65-66.)

Lasilaatikkotestauksella (white box testing, glass box testing) tarkoitetaan mustalaatikko-testausta syvällisempää ja tarkempaa menetelmää, jossa perehdytään myös siihen, mitä ohjelman sisällä tapahtuu. Testaaja siis tarkastalee antamiensa syötteiden ja tulosten lisäksi myös lähdekooditasolla ohjelman toimintaa (ks. kuvio 3). Testaaja voi esimerkiksi seurata, mitä ja milloin funktioita kutsutaan, tai tarkastella, vaihtuuko tietokannan sisältämä data oikein. Lasilaatikkotestauksen heikkouksia onkin se, että vaikka sillä saadaankin tarkemmin tietoa kuin mustalaatikko-testauksella, täytyy testaajan ymmärtää vähintään perusteet ohjelmointilogiikasta sekä tuntea ohjelman toiminta tarkasti lähdekoodista saakka. Toisaalta taas lasilaatikkotestausta suorittaessaan testaajat oppivat ohjelman toiminnan läpikotaisesti, jonka avulla on mahdollista löytää muutostarpeita joita mustalaatikko-testauksella ei välttämättä löytyisi. (Kasurinen, 2013, 67-68.)



Kuvio 3. Mustalaatikko- ja lasilaatikko-testaus

Harmaalaatikko-testaus (grey box testing) on yhdistelmä lasilaatikko- ja mustalaatikko-testausta. Harmaalaatikko-testauksen ajatuksena on yhdistää mustalaatikko-testauksen menetelmä verrata vaatimuksia saatuihin tuloksiin ja lasilaatikkotestien kykyä tarkastella järjestelmän sisäpuolta. Tavoitteena on siis yhdistää molempien hyvät puolet. Harmaalaatikko-testaus sopii tilanteisiin, joissa ei joko ole tarvetta tai mahdollisuutta testata ohjelmaa lähdekooditasolla, mutta

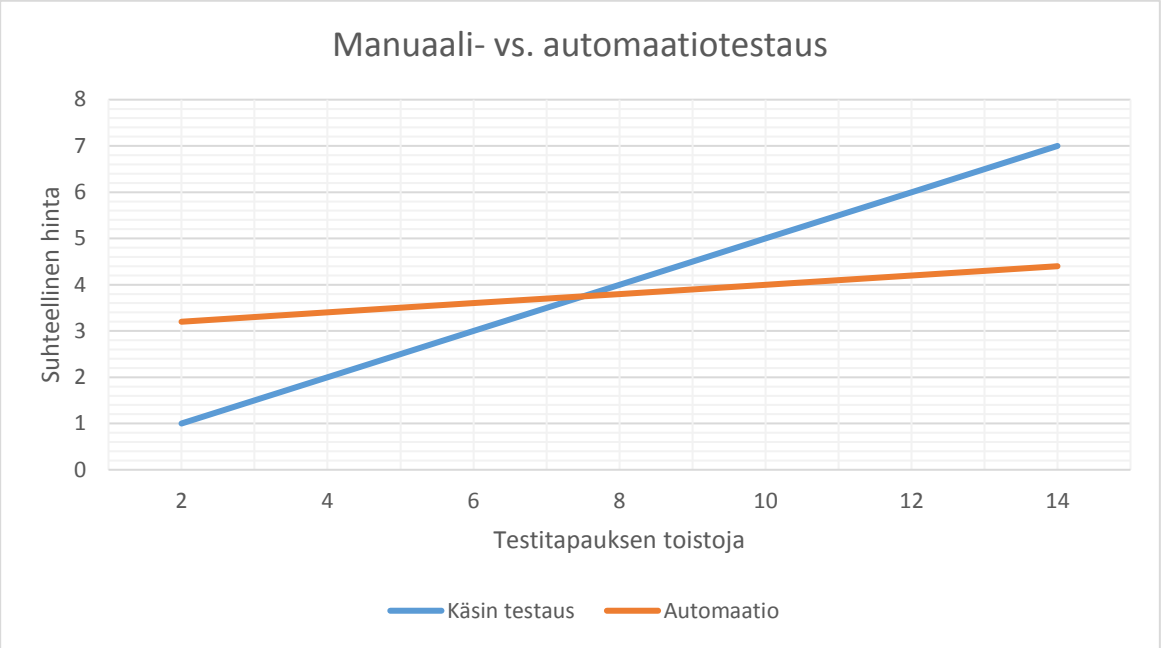
paikallisiin komponentteihin voidaan päästä käsiksi. Käytännössä siis oma järjestelmä tunnetaan ja sitä voidaan tarkastella lasilaatikkona, mutta sen alla toimivat palvelinratkaisut tai relaatiot muihin järjestelmiin testataan mustalaatikko-testauksena. (Kasurinen, 2013, 68.)

Ad hoc -testauksella tarkoitetaan täysin suunnittelematonta ja dokumentoimatonta testausta. Käytännössä se tarkoittaa sitä, että testaaja tarkastaa kaiken toimivan ainakin päällisin puolin. Ad hoc -testauksesta ei kirjata ylös yksityiskohtia tai varsinaisia virheitä, vaan usein kehittäjä vain tarkistaa kohteiden toimivuuden. (Kasurinen 2013, 74-75.)

3.3 Manuaalitestaus vs. automaatiotestaus

Testausautomaatiolla tarkoitetaan testaustoiminnon muotoa, jossa ohjelmaa varten rakennettu työväline testaa osan ohjelman kohteista. Sen tavoitteena on testata usein toistuvat kohteet, ja siten keventää testaajien työmäärää. Täysin satunnaisuuteen perustuvaa testausautomaatiota kutsutaan niin sanotuksi "apinatestaukseksi". Siinä automaatti testaa ohjelman toimivuutta antamalla satunnaisia syötteitä käytössä olevalle järjestelmälle, ja ohjelman mahdollisesti kaatuessa, automaatti tallentaa virheen aiheuttaneen syötesarjan. Testausautomaation käyttökohteena voi olla myös käyttöliittymän tarkastukset. Automaatti voi testata esimerkiksi tekstikenttien toimivuutta kirjoittamalla kenttään testisyötteen. Automaation käyttöönotto ja sen tekeminen yhdelle testitapaukselle on kallimpaa ja hitaampaa kuin yksittäisen testin ajaminen, mutta testin toistaminen on triviaali toimenpide joka ei maksa käytännössä mitään. Tämän vuoksi automaatiotestaus tulee suhteessa halvemmaksi mitä useampia toistuvia kohteita sen avulla testataan (Kasurinen 2013, 76-79.)

Mikäli testi sama testi suoritetaan projektin aikana 4-20 kertaa, tulisi silloin harkita automaatiotestauksen käyttöönottoa. Eli jos testikohteet vaihtuvat usein, eikä ole tarvetta toistuvalle testaamiselle, ei ole taloudellisesti tai ajallisesti kannattavaa ottaa käyttöön automaattia testaamaan kyseisiä kohteita (ks. kuvio 4). (Kasurinen, 2013)



Kuvio 4. Manuaali- ja automaatiotestauksen kustannukset havainnollistettuna

3.4 Käytettävyystestaus

3.4.1 Käytettävyystestaus yleisesti

Käytettävyystestauksella tarkoitetaan sen selvittämistä, kuinka hyvin laitteen käyttäjät pystyvät suorittamaan tehtäviään tuotteella. Käytettävyystestaus ei ole varsinainen vaihe projektissa, vaan sitä voidaan tehdä läpi jokaisen perinteisen V- tai vesiputousmallin vaiheissa. Usein käytettävyyttä voidaankin arvioida jo suunnitteluvaiheessa käyttäen lähes kokonaan staattisia malleja, joissa ei ole muuta kuin käyttöliittymän luonnos. Käytettävyystestausta ei kuitenkaan voida suorittaa loppuun, ennen kuin kaikki toiminnot ja ominaisuudet on kehitetty toimiviksi. Koska käytettävyystestauksessa arvioidaan yksinkertaisesti sovelluksen käytettävyyttä eikä etsitä varsinaisia lähdekoodissa esiintyviä virheitä, voi muutostarpeita löytyä jo testatuista komponenteistakin. Käytettävyystestauksella ei kuitenkaan tarkoiteta pelkästään järjestelmän visuaalista puolta ja sen testausta, vaan yleisesti sitä, kuinka tehokkaasti käyttäjä pääsee haluamaansa lopputulokseen. Esimerkiksi hakutoimintojen pitkä kesto voi turhauttaa käyttäjää ja olla käytettävyystestauksen piirissä oleva kohde, eikä se silti ole varsinainen virhe käyttöliittymässä tai sen suunnitelussa, vaan toteutusvaiheessa parannettava kohde. (Hyysalo 2006, 155-158.)

Käytettävyyttä voidaan arvioida ja testata monilla eri menetelmillä. Asiantuntija-arvioilla tai esimerkiksi testitehtävillä voidaan hakea tietoa siitä, kuinka käyttäjät hahmottavat laitteen toiminnan, aiheuttavatko jotkin laitteen toiminnot tai ominaisuudet virhesuorituksia tai ymmärtävätkö käyttäjät ne toisin kuin on suunniteltu. Testauksen avulla siis kartoitetaan tuotteen muutostarpeita. Käytettävyyttä suunniteltaessa on olemassa paljon erilaisia heuristiikkoja ja metodeja joita voidaan pitää yleisenä ohjesääntönä, mutta tunnetuin todennäköisesti on Jacob Níelsenin kehittämä 10 heuristisen säännön kokoelma. Hyysalo (2006) suomensi kokoelman teoksessa Käyttäjätieto ja käyttäjätutkimuksen menetelmät seuraavasti:

1. Käyttöliittymän tulisi olla mahdollisimman yksinkertainen, selkeä ja sen tulisi tuntua luontevalta käyttää.
2. Käytä käyttäjien kieltä; heille tuttuja termejä ja ilmauksia.
3. Minimoi ulkoa muistettavien asioiden määrä ja auta muistamista käyttöliittymän suunnittelulla.
4. Tee käyttöliittymästä yhdenmukainen ja samoilla periaatteilla toimiva.
5. Anna käyttäjälle riittävää palautetta siitä, mitä hän kulloinkin tekee, mitä hän saattanut laitteen tekemään ja missä tilassa tai moodissa laite kulloinkin on.
6. Merkitse selkeästi miten eri tiloista ja toiminnoista pääsee pois.
7. Luo laitteistoon oikopolkuja nopeuttamaan kokeneitä käyttäjiä.
8. Virhetilanteissa luo selkeät ja käyttäjälle ymmärrettävät virheilmoitukset, jotka auttavat häntä ratkaisemaan tilanteen.
9. Ehkäise virheiden tekemistä laitteen suunnittelulla.
10. Tarjoa riittävä ja selkeä apu ja dokumentaatio.

Edellä mainitulla tai muilla vastaavilla heuristiikoilla ei kuitenkaan voida korvata varsinaista käytettävyystestausta, sillä usein vastaavat ohjesäännöt jäävät liian moniselitteisiksi, jotta niitä voitaisiin käyttää tarpeeksi tehokkaasti projektissa. Yleinen virhe käytettävyystestauksessa on sen tekeminen liian myöhään osana suunnitteluprosessia. Mikäli testit suoritetaan juuri ennen tuotteen julkaisua, ei resurssit todennäköisesti riitä korjaamaan testeissä löydettyjä virheitä ja parannuskohteita, jolloin myös testien merkitys jää vähäiseksi. Tämän takia usein suositaan mallia, jossa tuotetta testataan läpi kehityksen. (McCracken 2016.)

3.4.2 Pikatestit

Pikatestauksella tarkoitetaan testejä, joilla voidaan testata käyttöliittymää ennen sen varsinaista toteutusta. Yleinen tapa toteuttaa pikatestejä on näyttää koekäyttäjälle kuvia suunnitellusta käyttöliittymästä, pyytää häntä kertomaan, kuinka hän lähtisi

suorittamaan testitehtävää. Käyttäjälle kerrotaan alussa lyhyt lähtötilanne ja yksinkertainen tehtävä suoritettavaksi. Lähtötilanne on pyrittävä kertomaan niin, ettei se paljasta tai vihjaa käyttäjälle mitään tehtävän suorittamisen kannalta olennaisia yksityiskohtia. Tämän jälkeen käyttäjälle voidaan näyttää käyttöliittymää kuvastavat kuvat esimerkiksi videoprojektorin tai monitorin kautta. Koekäyttäjää pyydetään kertomaan, kuinka hän lähtisi suorittamaan lähtötilanteen perusteella testitehtävää. Mikäli käyttäjä lähtee suorittamaan tehtävää haluttua reittiä pitkin, vaihdetaan hänelle näytetty kuva ja kysytään, kuinka hän jatkaisi tehtävän suorittamista. Mikäli käyttäjä valitsee jonkin muun kuin toivotun vaihtoehdon, voidaan häneltä kysyä, että miksi hän päätyi siihen ratkaisuun, ja kuinka ohjelmaa voitaisiin parantaa. (Wiio 2004, 218-221.)

3.4.3 Käytettävyystestit

Käytettävyystestaus muistuttaa pikatestausta, mutta se on tarkemmin suunniteltu, toteutettu ja dokumentoitu. Toimivan käytettävyystestin saavuttamiseksi, täytyy laatia testaussuunnitelma, hankkia testikäyttäjät ja analysoida testitulokset. Käytettävyystestauksen vahvuuksia on se, että sen avulla voidaan löytää mahdollisia tulevia virheitä. Käytettävyystestin suorittamiseen ei kuitenkaan tarvita varsinaista virallista laboratoriota, vaan tehokkaan testin voi suorittaa myös tavallisemmassakin ympäristössä. Usein riittää pelkkä huone, jossa on joko ääni- ja videonauhoitus käynnissä testikäyttäjän kanssa. Etätestauksessa riittää se, että testitiimillä on mahdollisuus nähdä käyttäjän navigointi sovelluksessa ja käyttäjä itse. (Usability test n.d.)

Testikäyttäjiä hakiessa, on tärkeää, että he ovat loppukäyttäjän kaltaisia. Tämän takia on tärkeää myös löytää toimivat välineet sopivien testikäyttäjien löytämiseksi. Mikäli tuote on esimerkiksi lenkkeilijöille tarkoitettu mobiilisovellus, voi sopivat testikäyttäjät löytää sosiaalisen median kautta eri liikunta- ja lenkkeilyryhmistä. (McCracken 2016.)

Käytettävyystestausta voidaan tehdä joko paikallisesti (local testing) tai etänä (remote testing). Etätestauksen voi järjestää verkossa toimivien palveluiden kuten esimerkiksi WebExin avulla. Usein etätestauksessa kustannukset pysyvät matalampana, sillä testikäyttäjien mahdolliset matkustuskustannukset ovat yleensä

tuotteen kehittäjien vastuulla. Etätestaus on myös usein kätevämpää testikäyttäjälle, jonka takia etätestaukseen heitä on helpompi löytää. Mikäli testit päätetään suorittaa etänä, on todennäköistä, että löytyy myös sopivammat testikäyttäjät, koska heitä voi hakea laajemmalta alueelta. Paikallisen testauksen vahvuuksia puolestaan on se, että testit on mahdollista suorittaa ilman suurempia teknisiä rajoitteita (esimerkiksi ilman verkkoyhteyttä) ja testikäyttäjää voidaan valvoa tarkemmin, jonka avulla saadaan mahdollisesti tarkempia tuloksia. (McCracken 2016.)

Vaikka testikäyttäjien ottaminen yrityksen tai projektin sisältä olisikin kaikista kätevin, halvin ja nopein tapa suorittaa käytettävyystestit, ei sitä yleisesti suosita. Yrityksen sisäisiä työntekijöitä käytettäessä, voidaan myös pitää keskeneräinen tuote paremmin salassa kilpailijoilta, mutta se taas ei tuo tuotteelle julkista näkyvyyttä. Esimerkiksi peliteollisuudessa ovat kehittäjät ja julkaisijat saaneet peleille valtavaa huomiota siirtämällä pelin beta-testauksen kuluttajille. Tämä toimii kuitenkin vain jos pelille on saatu tarpeeksi laaja kohdeyleisö ja beta-testausvaiheesta saadaan luotua eksklusiivinen ilmiö (Jones, n.d.). Yrityksen tai projektin sisäisiä työntekijöitä ei usein kannata kuitenkaan rekrytoida testikäyttäjiksi, koska heillä voi olla laajaakin aiempaa kokemusta sovelluksesta. Mikäli testikäyttäjä on käyttänyt tai edes nähnyt tuotteen etukäteen, ei hänen testeistä saada yhtä validia tulosta kuin loppukäyttäjien testeistä. Tämä vaikuttaa suoraan käytettävyystestien tuloksiin ja vääristää niitä. Testikäyttäjä on käytettävyystesteissä todennäköisesti myös yrityksen sisäistä työntekijää puoluettomaampi. (Li, 2016.)

4 CASE: Axiell Auroran käytettävyys- ja ohjelmistotestaus

4.1 Testien suunnittelu

Koska Aurora oli toteutettu jo aiemmin eikä käyttöliittymään oltu suunniteltu suuria muutoksia, projektityöskentelyn malleja ja testausmenetelmiä sovellettiin. Suurelle osalle testattavista komponenteista ei ollut tarpeellista tehdä vaatimusmäärittelyä, sillä jo Silverlight-versiossa olevien komponenttien toimivuutta voitiin pitää vaatimuksena Gold-version komponenttien toimivuudelle. Testauskohteena oli myös paljon samoja komponentteja, jotka löytyivät Silverlight-versiosta, mutta niille oli joko tehty pieniä muutoksia tai lisätty ominaisuuksia. Samoista syistä johtuen, ei

testaussuunnitelmia toteutettu kuin laajimmille ja monimutkaisimmille testauskohteille. Näitä kohteita varten oli versiotiedotteessa määritelty niiden toiminnalliset vaatimukset.

Erityisesti laajempia kokonaisuuksia varten, usein luodaan testaussuunnitelma, josta löytyy tärkeimmät seikat, jotka tulee ottaa huomioon tuotetta testattaessa. Testauskohteesta ja käytettävistä menetelmistä riippuen, näitä seikkoja voivat olla esimerkiksi testattava kohde, testaaja, testausjärjestys, tuotteen vaatimustaso ja mahdollisen asiakkaan tarpeet. Näiden lisäksi usein testaussuunnitelmat kattavat myös käytettävät menetelmät ja käytössä olevat resurssit. Mikäli testaus suoritetaan vertailemalla tuotteen uutta versiota vanhempaan versioon, tulee testausolosuhteet säilyttää mahdollisimman samankaltaisina. (Mitchell 2007, 9-11.)

Tämän projektin kohdalla se tarkoitti sitä, että samat toiminnot testattiin samoilla arvoilla samaa tietokantaa käyttäen, ja että näyttökoko pidettiin samana testien läpi. Mikäli vastaavilla seikoilla on mahdollisuus vaikuttaa testituloksiin, mainitaan ne testaussuunnitelmassa, jotta testin voi tarvittaessa toistaa jälkeenpäin. Projektin aikana laadittuihin testaussuunnitelmiin kirjattiin usein myös tyypillinen käyttötapaus testauskohteesta.

Näitä laajempia kokonaisuuksia testattaessa, oli tärkeää kirjata kaikki huomiot ylös, myös silloin mikäli ohjelma toimi vaaditulla tavalla. Niiden avulla kaikkia poikkeavuuksia voitiin verrata jo aiemmin testattuihin toimintoihin, jonka avulla voitiin karsia virheen aiheuttavia tekijöitä pois ja siten lopulta kohdentaa virhe. Tämä tehtiin usein sen takia, että kehittäjät löytävät ja heillä on mahdollisuus tarvittaessa toistaa virhe nopeasti. Testattavan kohteen laajuudesta riippumatta, aina oli tavoitteena löytää mahdollisen virheen aiheuttava tekijä. Mikäli tässä ei onnistuttu, annettiin virheen korjaavalle kehittäjälle niin tarkat ohjeet kuin mahdollista virheen toistamiseksi.

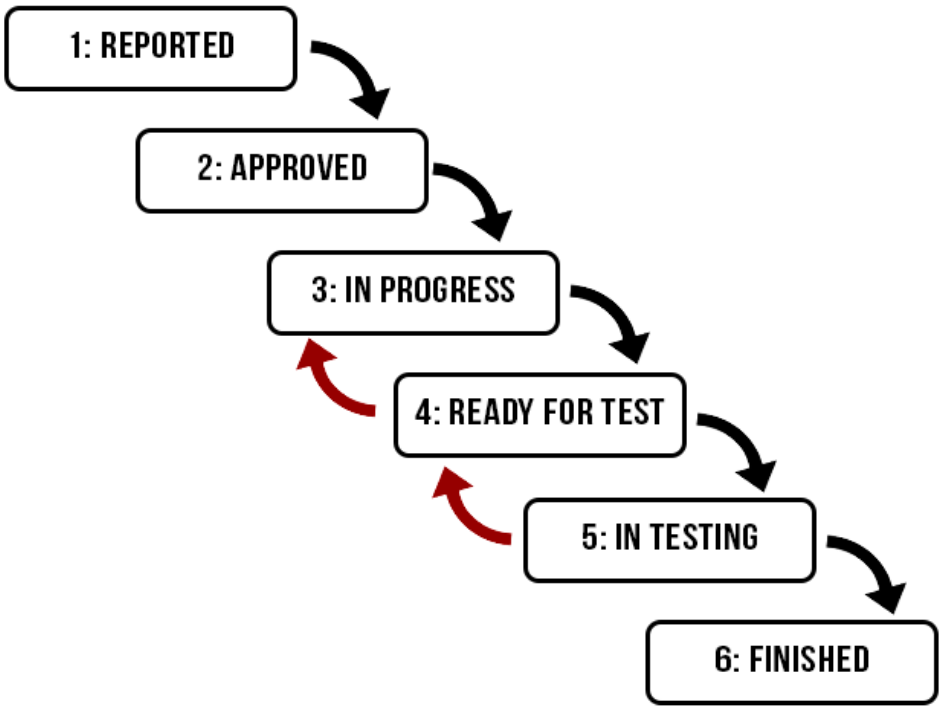
Vastaavat kohteet voitiin testata mustalaatikko-testausmenetelmällä, sillä varsinainen ohjelmointilogiikka oli jo testattu Silverlight-versiossa. Kuitenkin kaikki projektin aikana testatuista komponenteista testattiin joko mustalaatikko- tai harmaalaatikko-testauksella, sillä järjestelmän laajuuden takia ei nähty järkeväksi perehdyttää testaajaa lähdekooditasolle. Yleinen käytäntö projektin aika olikin se,

että vaatimusten asettamisen ja kehittäjien toteutuksen jälkeen komponentti testattiin joko mustalaatikko- tai harmaalaatikko-testauksella, jonka jälkeen se siirrettiin takaisin kehittäjille korjattavaksi tai hyväksyttiin toimivana. Varsinaista ad hoc -testausta ei suoritettu, vaan jokaisesta testattavasta kohteesta kirjattiin ylös huomiot, mutta varsinaista suunnitelmaa tai raporttia ei jokaiselle laadittu.

Järjestelmää testattaessa helmikuussa 2017, oli pienellä osalla Auroraa käyttävistä yksiköistä ollut Auroran kehitysversio testattavana. Tästä syystä ei päädytty tekemään käytettävyystestejä testihenkilöiden ja haastatteluiden avulla, sillä tarvittavat palautteet saatiin jo kehitysversiota kokeilevilta yksiköiltä. Kehitysversiota kokeilevat yksiköt antoivat palautteen suoraan kehittäjille, tai kehityksestä vastaaville henkilöille videopalavereiden kautta. Näiden palautteiden avulla voitiin käytettävyyttä parantaa ensisijaisia käyttäjiä paremmin vastaavaksi. Testaajan tehtäväksi jäikin testata käytettävyyttä ohjelmistotestauksen ohessa, yksittäinen osio tai kokonaisuus kerrallaan.

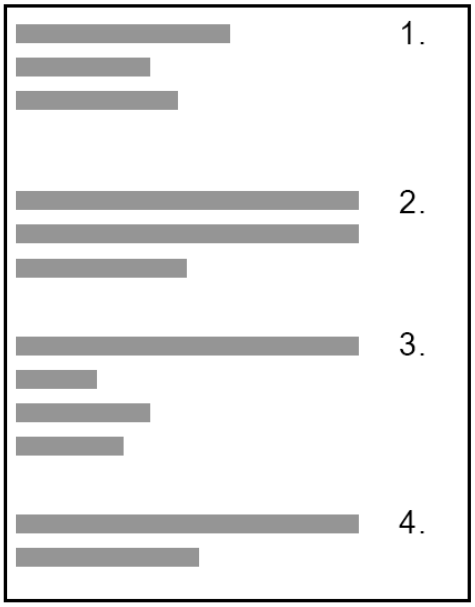
4.2 Testausmenetelmät

Yleinen menetelmä virheiden raportointiin kehittäjien ja testaajien välillä sisälsi kuusi vaihetta (ks. kuvio 5). Ensimmäisessä vaiheessa virheraporttiin kirjattiin tehtävän vaatimukset. Näitä olivat joko jonkin tietyn kohteen kehittäminen, korjaaminen tai testaaminen. Task-tyyppisille kohteille määriteltiin lyhyesti komponentin toimintavaatimukset ja mahdollinen tavoitepäivämäärä jolloin komponentin tulee olla valmis (ks. liite 3).



Kuvio 5. Virheraportin kulku kehittäjän ja testaajan välillä.

Bug-tyyppisille kohteille kirjattiin virheen aiheuttanut alue (esimerkiksi ohjelman sivu), kohde (esimerkiksi tekstikenttä), ohjelman versionumero, käytetty tietokanta, testauksessa käytetty selain ja sen versionumero sekä testauspäivämäärä (ks. kuvio 6). Kuten liitteessä 5 on nähtävissä, virheitä käsitteleviin kohteisiin kirjattiin myös aiheutunut virhe ja vaiheistettu ohje, kuinka virheen pystyy toistamaan. Näihin kohteisiin lisättiin usein myös kuvakaappaus joko virheestä tai virheen aiheuttavasta toimenpiteestä ja komponentista. Kohteisiin tehdyt muutokset kuitattiin usein päivämäärällä ja omilla nimikirjaimilla. Laajemmalle ja yksityiskohtaisemmalle standardille ei ollut tarvetta. Virheiden raportointiin käytettiin Visual Studio -ohjelmiston sisäisiä työkaluja, jonka avulla näkyivät virheraportteihin tehdyt muutokset, niiden tekijät ja muutosten päivämäärä.



Kuvio 6. Virheraportin rakenne

Virheen raportoinnin (1: Reported) ja sen hyväksymisen jälkeen (2: Approved), siirrettiin kohde kehittäjälle korjattavaksi tai kehitettäväksi (3: In Progress). Vaiheen 3 jälkeen kehittäjät kirjasivat virheraporttiin kehitys- tai korjaustoimenpiteensä ja mahdolliset lisähuomiot. Nämä saattoivat olla yleisiä ohjeita testaajalle tai huomioita esimerkiksi muuttuneista vaatimuksista. Tämän jälkeen kehittäjä siirti kohteen testausjonoon (4: Ready for Test).

Kun kohteen testaus oli päätetty aloittaa, siirretiin sen tila testattavaksi (5: In Testing). Viidennessä vaiheessa suoritettiin varsinainen testaus. Tähän mennessä virheraportti oli voinut olla jo aiemmin testaajalla, jolloin testattiin kehittäjien tekemät korjaukset. Mikäli korjausten jälkeen ei alkuperäistä virhettä enää aiheutunut, testattiin kohteen yleinen toimivuus vielä uudestaan. Liite 7 on virheraportti, joka on kiertänyt kehittäjän ja testaajan välillä enemmän kuin yhden kierroksen, ja josta on siten nähtävissä kehittäjien korjaukset ja niiden testaus. Vastaavalla tavalla hoidettiin bug-tyyppiset virheraportit projektin aikana. Mikäli kohde oli kuitenkin ensimmäistä kertaa testaajalla, testattiin se läpikotaisesti.

Ensimmäisellä testauskerralla aloitettiin kohteen testaaminen sen vaatimuksista. Mikäli kohde ei toiminut vaatimusten mukaisella tavalla, dokumentoitiin tarpeelliset

tiedot virheen toistamiseksi. Kohteen testausta jatkettiin virheestä huolimatta, jotta jo ensimmäisellä testauskerralla saataisiin kartoitettua suurin osa virheistä. Virheet löytyivät usein kuitenkin vaatimusten ulkopuolelta. Esimerkiksi jos testikohteena oli teoksen lainaus, saattoi lainaus toimia moitteetta, mutta samalta sivulta löytyi muita virheitä, jotka vaikuttivat lainaukseen. Tällaisissa tapauksissa ei kuitenkaan välttämättä ollut tarvetta raportoida kaikkia virheitä eteenpäin, sillä sivuja ei usein testattu täysinä kokonaisuuksina, ja siten yksi sivun osa tai toiminto saattoi olla kehitysvaiheessa, kun muita toimintoja testattiin. Vastaavissa tapauksissa testaaja varmisti kehittäjiltä, että onko virheen aiheuttaja edelleen kehityksessä, vai raportoidaanko virhe eteenpäin.

Testattaessa HTML input-kenttien toimivuutta varausten lukumäärää asetettaessa, syötettiin kenttään arvoja myös kokonaislukujen ulkopuolelta. Input-kenttien toimivuutta kokeiltiin positiivisilla ja negatiivisilla kokonaisluvuilla, erikoismerkeillä, kirjaimilla ja kaikkien niiden yhdistelmillä. Liite 7 on virheraportti hakutuloksien selaamisesta, jossa aiheutui virhe syötettäessä kenttään kokonaisluvusta poikkeava arvo. Vastaavalla tavalla testattiin kaikki kentät joihin käyttäjä pystyi itse syöttämään arvot. Sovelluksen arkkitehtuurissa, ohjelmointilogiikassa tai lähdekoodissa olevat virheet esiintyivät usein tietynlaisena epäloogisuutena tai huomioimattomuutena. Liite 5 on virheraportti tapauksesta, jossa virhe esiintyi tekemällä asiat tietyssä järjestyksessä, eikä vain syöttämällä ei-toivottuja arvoja (ks. liite 10). Eräänlaisena lähtökohtana tällaisten virheiden etsimiseen oli ajatella, että mitä kehittäjä mahdollisesti ei ollut ottanut huomioon kohdetta kirjoittaessaan. Sivuilta joilla oli valittavissa useita eri kenttiä, esimerkiksi checkbox-elementtejä, dropdown-valikoita tai radio-buttoneita, pyrittiin virheitä etsimään tekemällä asetuksista eri yhdistelmiä. Tällä tavalla asetuksista saattoi pystyä luomaan ristiriidan, ja sitä kautta aiheuttaa ohjelmistossa virheen.

Aurora toimi testattaessa lähes poikkeuksetta samalla tavalla Firefox-, Internet Explorer- ja Chrome-selaimilla. Suurimmat erot liittyivät selainten kykyyn tukea tiettyjä CSS-elementtejä ja suorittaa JavaScriptilla toteutettuja toimintoja. Liite 9 on virhraportti, jota korjattaessa huomattiin, että kohde käyttäytyy eri tavalla Internet Explorer -selaimessa, kuin esimerksi Google Chrome- tai Mozilla Firefox -selaimissa. Kaikkia kohteita ei nähty järkeväksi testata jokaisella suositulla selaimella, vaan usein

pelkästään Internet Explorer- ja Chrome- tai Firefox-selaimet todettiin riittäviksi. Tällaista käytäntöä noudatettiin ennen kaikkea siksi, koska Internet Explorer sisälsi suppeammat tuet joillekin JavaScript-toiminnoille ja CSS-elementeille, jonka takia siinä saattoi esiintyä virheitä joita ei Chromium-pohjaisissa selaimissa esiintyisi.

Jos tässä vaiheessa löydetyt virheet vaikuvat järjestelmän isompaan osa-alueeseen, esimerkiksi useille järjestelmän eri sivuille, tehtiin niistä erilliset virheraportit. Liite 3 on raportti sisäänkirjautumisen kehittämisestä ja sen testaamisesta. Kyseistä kohdetta testattaessa, päätettiin virhe raportoida erikseen (ks. liite 4), koska sisäänkirjautuminen itsessään toimi, mutta se mahdollisti virheen muualla järjestelmässä. Nämä virheraportit linkitettiin testauksessa olevaan kohteeseen ja ne mainittiin testauksen päättyessä. Tällaista käytäntöä noudatettiin usein etenkin task-tyyppisissä kohteissa.

Kun kohde oli saatu onnistuneesti testattua ja se saavutti kaikki sille asetetut vaatimukset, eikä ylimääräisiä ei-toivottuja ominaisuuksia löytynyt, voitiin se merkata valmiiksi (6: Finished). Tässä vaiheessa virheraporttiin kirjattiin lyhyesti testitulokset ja muut mahdolliset huomiot, sekä testivälineet, päivämäärä ja testin suorittajan nimikirjaimet.

Esimerkiksi käyttäjien hallinta -sivun (ks. liite 3) toimintojen testaus edellytti täysiä oikeuksia, jonka takia sitä ei voitu testata tavallisena käyttäjänä. Testaajalla oli pääosin täydet oikeudet, jotta esimerkiksi asiakkaiden lisäys (ks. liite 2: task 13523) ja sisäänkirjautuminen (ks. liittet 3 ja 4) saatiin testattua tehokkaasti. Tavallisen käyttäjän oikeuksia voitiin käyttää kuitenkin esimerkiksi hakutuloksia (ks. liite 7) käsittelevissä kohteissa. Mikäli testauskohde oli tarpeeksi laaja, testattiin pelkkien toimintojen toimivuuden lisäksi kohde myös käyttötapauksen kaltaisesti. Kuten liitteessä 1 on kerrottu, käyttötapauskohtaiset kohteet testattiin käyttötapauksen ”alusta loppuun”. Tarkoittaen sitä, että kohde testataan samassa järjestyksessä kuin missä käyttäjä sitä todennäköisesti käyttää. Tällä tavoin voitiin virheiden lisäksi kartoittaa myös yleisesti käytettävyyttä ja käyttäjäkokemusta haittaavia tekijöitä.

Testatut kohteet ja virheet eivät keskittyneet vain yksittäiseen sovelluksen sivuun tai osa-alueeseen, vaan ne sisälsivät pienempiä kokonaisuuksia eri puolelta järjestelmää. Liitteissä 1 ja 3 mainittujen sivujen lisäksi, muita kokonaisia sivuja alatoimintoineen

testattiin kolme kappaletta (ks. liite 2 bug ID: 13729, bug ID: 13744 ja task ID: 13531). Tässä vaiheessa sovellukseen oli kehitetty noin 20 eri pääsivua alatoimintoineen. Pääsivut sisälsivät omia alasivuja ja toimintoja, mutta ainoastaan pääsivuille oli mahdollista päästä aloitusnäytöltä. Kaikki pääsivuista eivät kuitenkaan olleet lopullisesti valmiita, sillä toimintoja, joita tarvitaan useilla eri sivuilla, ei välttämättä ollut vielä lisätty. Esimerkiksi hakutulosten selaus oli vielä kehitysvaiheessa, vaikka useita hakutuloksia sisältäviä sivuja oli muuten jo testattu. Vastaavat toiminnot lisättiin myöhemmin, jolloin kaikkien sivujen hakutoiminnot testattiin kerralla. Näistä syistä olikin testattaessa tärkeää erottaa, että onko kyseessä virhe vai keskeneräinen kohde. Tällaisissa tapauksissa asia joko tarkistettiin versiotiedotteesta tai varmistettiin suoraan kohteen kehittäjältä, jotta turhia ja virheellisiä virheraportteja laadittaisiin mahdollisimman vähän.

5 Tulokset ja pohdinta

5.1 Tulokset

Liitteessä 2 on listattuna kaikki raportoidut testatut kohteet ja löydetyt virheet. Näitä oli testausvaiheen lopussa yhteensä 48 kappaletta, joista 22 oli onnistuneesti testattu. 6 kappaletta oli testattu joko kerran tai useammin, mutta palautettu kehittäjille korjattavaksi. 18 kappaletta näistä oli uusia löydettyjä virheitä tai parannuskohteita, joita kehittäjät eivät olleet ehtineet vielä korjata. Näistä osa oli "sivutuotteena" löytyneitä virheitä, eivätkä siis välttämättä kuuluneet laajempaan testauskokonaisuuteen. 1 kohde oli testattavana ja 1 oli hylätty sen ollessa duplikaatti. Raportoitujen virheiden lisäksi projektin alkuvaiheessa epävirallisia korjauskehotuksia tai ilmoituksia virheistä lähetettiin kehittäjille 12 kappaletta.

Virheiden vakavuus vaihteli huomattavasti. Osa virheistä saattoi aiheuttaa koko järjestelmän kaatumisen (ks. liite 12), kun taas osa saattoi olla ainoastaan poikkeus järjestelmässä käytettyjen ikonien järjestyksessä (ks. liite 11). Virheiden vakavuutta ei varsinaisesti arvioitu, mutta laajemmat testauskohteet priorisoitiin yksittäisten toimintojen testausta korkeammalle. Tavallista laajemmat testauskohteet sisälsivät usein kokonaisen sivun testauksen kaikkineen toimintoineen (ks. liitteet 1 ja 3). Testatut kohteet sisälsivät myös varsinaisten sivujen ulkopuolisia toimintoja (ks.

liitteet 4 ja 5). Nämä kohteet testattiin tiiviissä yhteistyössä kehittäjien kanssa ja ne vaativat normaalia laajempaa asetusten muuttamista. Tästä syystä myös niiden testaus oli monimutkaisempaa ja enemmän aikaa vaativaa.

Testeissa keskityttiin myös ohjelman käyttäytymiseen eri selaimilla ja näyttöko'oilla. Koska jokaista 28:aa kohdetta testattaessa, oli tehtävänä testata kohteen toimivuus ja käytettävyys, ”sivutuotteena” löytyneiden virheiden tai bugien lukumäärä oli melko korkea. Osa ”sivutuotteena” löytyneistä virheistä olivat useammassa eri kohteessa esiintyviä tai vain sivusivat testauskohdetta, jonka takia niistä päätettiin luoda erillinen ilmoitus lähetettäväksi kehittäjille, eikä liittää niitä alkuperäiseen ilmoitukseen. Näistä 58:sta eri virheestä suurin osa oli sovelluksen toimivuuteen ja toiminnallisuuksiin liittyviä virheitä. Käytettävyystestauksessa löytyneiden virheiden vähyys selittyy sillä, että käyttöliittymään ei tehty suuria muutoksia Silver-versiosta, jonka käyttöliittymä oli testattu jo sen kehityksen aikana. Virheiden vähyys ei myöskään kerro niinkään huonosta testauksesta, vaan pikemminkin hyvin toteutetusta tuotteesta.

Tuloksista ei koitunut suoraa rahallista hyötyä työn tekijälle, eikä välittömästi myöskään työn tilaajalle. Tulokset ovat kuitenkin nähtävissä Auroran käytettävyydessä ja tulevat tuomaan välillisesti rahallista hyötyä tilaajalle pidemmällä aikavälillä.

Tulokset eivät keskittyneet yhteen sovelluksen sivuun, mutta eivät myöskään koko sovellukseen. Laaditut 48 virheraporttia koskivat yhteensä noin 15 laajempaa eri järjestelmän sivua tai kokonaisuutta. Esimerkiksi liitteessä 1 mainittu testauskohde sisälsi noin 50 eri alasivua tai toimintoa, joista useista ei löytynyt lainkaan virheitä. Näistä alasivuista ei siksi myöskään laadittu erillisiä virheraportteja, vaan niiden testaus ja testaustulokset mainittiin liitteen 1 kohteen virheraportissa. Testattuja kohteita oli siis huomattavasti suurempi määrä kuin raportoituja virheitä.

5.2 Resurssit

Opinnäytetyön testausvaiheeseen käytettiin noin 300 työtuntia, joka oli alkuperäisen suunnitelman mukainen määrä. Resurssit eivät kuitenkaan jakautuneet tasan testauskohteiden kanssa. Esimerkiksi liitteessä 1 mainittu testauskohde oli

huomattavasti laajempi kuin liitteessä 6 mainittu kohde. Virheraporteissa ei käsitelty testauskohteeseen käytettyä aikaa, mutta henkilökohtaista tuntikirjanpitoa pidettiin läpi projektin.

5.3 Osaaminen

Opinnäytetyön merkittävin tulos on tekijälleen sen tuoma osaaminen ja kokemus. Eniten osaamista kertyi erityisesti sekä ohjelmistotestauksen, että käytettävyystestauksen teoriatasosta ja niiden toteuttamisesta. Yleinen tietotaso ohjelmistokehityksestä ja etenkin käytettävyydestä kehittyi työn aikana huomattavasti. Oleminen osana ohjelmiston kehittämistä ja projektityöskentelyä, kehitti selvästi yleisiä projektityöskentelyn taitoja, sekä yleistä ymmärrystä ohjelmistokehityksestä. Testien suorittamisen kautta kasvoi tietotaso myös ohjelman teknisestä toteutuksesta. Mitä enemmän ohjelmasta löysi logiikkaan pohjautuvia virheitä, sitä paremmin pystyi ymmärtämään sen toimintalogiikkaa ja etsimään vastaavia virheitä muualta sovelluksesta.

5.4 Pohdinta

Vaikka alkuperäisen suunnitelman mukaan olisi käytettävyystestaus ollut etusijalla, työn aikana ohjelmistotestaus muodostui laajemmaksi kohteeksi. Tämä johtui siitä, että käyttöliittymään ei ollut tehtävissä suuria muutoksia, ja siitä, että ohjelmistotestaukselle oli Aurora-järjestelmän puolesta enemmän tarvetta.

Testien suorittaminen onnistui hyvin läpi koko työn, mutta uuden Aurora-version kehittäminen on projektina itsessään jo niin laaja, että opinnäytetyö ei ole tarpeeksi kattava käsittelemään koko sovelluksen käytettävyyttä ja testausta. Sovelluksen täydelliseen toimivuuden ja käytettävyyden testaamiseen olisi vaadittu huomattavasti enemmän resursseja, jonka takia tulokset jäivät rajatuiksi ja kertovatkin ainoastaan osan sovelluksen testauksesta. Tilaajan toiveena oli kuitenkin myös panostaa dokumentoituun testaukseen, jonka takia läpi koko projektin, kaikki huomiot kirjattiin ylös. Suuri osa näistä dokumenteista oli epävirallisia, ja niitä oli käytetty vain testaajaa auttavana muistiona, mutta opinnäytetyön lopetusvaiheeseen mennessä, oli laadittu myös yleisiä työohjeita, virheraportteja ja

testaussuunnitelmia, joita voidaan jatkossakin käyttää hyödyksi. Testaussuunnitelmia ja virheraportteja voidaan käyttää myös tulevien testien lisäksi myös suunniteluvaiheessa avuksi. Mikäli Auroran HTML5-versioon kehitetään jatkossa laajoja uusia ominaisuuksia, voi aiemmin laadituissa virheraporteissa sekä testaussuunnitelmissa olla otettu huomioon suunnitelun kannalta olennaisia seikkoja. Dokumentteja tarkkailemalla voidaan välttää myös kehitysvaiheessa toistamasta samoja virheitä uudestaan.

Mikäli HTML5-version julkaisun jälkeen sovelluksesta julkaistaan usein uusi versio, voisi automaatiotestaus osoittautua tehokkaaksi tavaksi testata sovelluksen yleinen toimivuus (esimerkiksi tekstikentät, valikot ja painikkeet). Tällä tavoin testaajat vapautuisivat monimutkaisempiin ja käyttäjäkohtaisempien kohteiden testaukseen. Automaatiotestaus nousee kuitenkin arvoonsa vasta pitkällä aikavälillä ja suurillä testimäärillä.

Kehittäjät ja testaajat työskentelivät pääosin samassa toimipisteessä, jonka takia epäselvyydet esimerkiksi virheraporttien osalta voitiin selvittää kasvotusten. Jos kehittäjät ja testaajat kuitenkin hajautuvat enemmän, tai projekti laajenee huomattavasti, voisi standardisoidut virheraportit selkeyttää niin testaajien kuin kehittäjienkin työtä. Suuria epäselvyyksiä ei työn aika ilmennyt, mutta nekin olisivat olleet vältettävissä mikäli virheraportit olisi kirjattu tarkemmin. Esimerkiksi lomakkeisiin pohjautuvat, esitäytetyt virheraportit voisivat mahdollisesti toimia tässä käyttötarkoituksessa.

Auroran käytettävyyttä ja yleistä virheettömyyttä saatiin parannettua, mutta testattavaa olisi riittänyt vielä sadoiksi tunneiksi. Tästä huolimatta, testit itsessään saatiin suoritettua onnistuneesti ja niistä saatiin kerättyä myös jatkokehityksen kannalta arvokasta tietoa.

Lähteet

Axiell Finland Oy. n.d. Axiell Finland Oy:n verkkosivut. Viitattu 8.2.2017. http://www.axiell.fi/axiell_finland/.

Black Box Testing: Types and techniques of BBT. 2017. Artikkeli Software Testing Help-sivustolla. Julkaistu 17.4.2017. Viitattu 17.5.2017. http://www.softwaretestinghelp.com/black-box-testing/.

Beal, V. n.d. Black Box Testing. Viitattu 17.5.2017. http://www.webopedia.com/TERM/B/Black_Box_Testing.html.

Dargis, J. 2016. GOLD bug reports. Julkaistu 28.2.2017. Viitattu 16.5.2017. https://www.utest.com/articles/gold-bug-reports

Hyysalo, S. 2006. Käyttäjätieto ja käyttäjätutkimuksen menetelmät. Helsinki: Edita Prima Oy.

Jones, M. n.d. Why Beta Testing Could Be Your Best Marketing Move Ever. Viitattu 18.5.2017. https://www.coxblue.com/why-beta-testing-could-be-your-best-marketing-move-ever/.

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

Kuoppala, H., Parkkinen, J., Sinkkonen, I. & Vastamäki, R. 2006. Käytettävyyden psykologia. 3. uud. p. Helsinki: Edita Prima Oy.

Li, A. 2016. Employees as Usability-Test Participants. Viitattu 17.5.2017. https://www.nngroup.com/articles/employees-user-test/.

McCracken, C. 2016. How to Conduct Usability Testing from Start to Finish. Viitattu 17.4.2017. http://uxmastery.com/beginners-guide-to-usability-testing/.

Mitchell, P. 2007. A Step-by-Step Guide to Usability Testing. iUniverse, Inc.

Nielsen, J. 1999. Designing web usability. David Dwyer.

Nielsen, J., 2012. Usability 101: Introduction to Usability. Viitattu 8.3.2017. https://www.nngroup.com/articles/usability-101-introduction-to-usability/.

Oluwole, Dele. 2016. The Tester's Role in Agile Is More than "Just Testing". Viitattu 18.5.2017. https://www.scrumalliance.org/community/articles/2014/june/the-tester%E2%80%99s-role-in-agile-is-more-than-just-testi/.

Usability testing. n.d. Artikkeli Usability.gov-sivustolla. Viitattu 18.5.2017. https://www.usability.gov/how-to-and-tools/methods/usability-testing.html/.

Wiio, A. 2004. Käyttäjäystävällisen sovelluksen suunnittelu. Helsinki: Edita Publishing Oy.

Liitteet