Ohjelmistotestausprosessi: Helmet tarkennettu hakutoiminto

Elli Uutela

2022 Laurea

Laurea-ammattikorkeakoulu

Ohjelmistotestausprosessi: Helmet tarkennettu hakutoiminto

Elli Uutela

Tietojenkäsittely

Opinnäytetyö

Kesäkuu, 2022

Laurea-ammattikorkeakoulu **Tiivistelmä**

Tietojenkäsittelyn koulutusohjelma

Tradenomi (AMK)

Elli Uutela

Ohjelmistotestausprosessi: Helmet tarkennettu hakutoiminto

Vuosi 2022 Sivumäärä 37

Tämän opinnäytetyön tarkoituksena oli suorittaa Helmet.fi -sivuston tarkennettuun hakutoimintoon kohdistuva ohjelmistotestausprosessi, keskittyen haun toiminnallisuuteen ja käytettävyyteen. Ohjelmistotestausprosessia varten laadittiin testaussuunnitelma ja testitapauksia. Testitapaukset suoritettiin, ja niiden tulosten pohjalta kerättiin tietoa tarkennetun hakutoiminnon toiminnallisuudesta. Tämän lisäksi tulosten analysointivaiheessa kirjattiin testausprosessin aikana hakutoimintoon kohdistuneet käytettävyyteen liittyneet havainnot. Työllä ei ollut toimeksiantajaa ja sen päätavoitteena oli kehittää tekijän omaa osaamista ohjelmistotestauksen suhteen ja toimia taidonnäytteenä. Toissijaisena tavoitteena voidaan nähdä Helmetin tarkennetun haun toiminnallisuuden ja käytettävyyden arviointi, sekä kehitysehdotuksien antaminen testauksen tulosten perusteella.

Työn tietoperusta keskittyy ohjelmistotestaukseen ja pyrkii antamaan hyvän kokonaiskuvan ohjelmistotestausprosessin eri vaiheista. Ohjelmistotestausprosessin yksittäisistä vaiheista eniten huomiota kiinnitettiin testauksen suunnitteluun, jonka teoriaa hyödynnettiin Helmetin tarkennetun haun testausta ja testitapauksia suunnitellessa.

Opinnäytetyön kaksi keskeistä testaustyyppiä ovat tutkiva testaus ja toiminnallinen testaus, joiden lisäksi osa testitapauksista seuraa mustalaatikkotestauksen ekvivalenssijako -menetelmän periaatteita. Nämä menetelmät eivät vaadi lähdekoodin tuntemusta, jonka vuoksi ne valikoituivat opinnäytetyöhön.

Tarkennetun haun testausprosessin suurimpana haasteena oli tarkennetun haun alkuperäisten vaatimusten tuntemattomuus, joka teki varsinaisten virheiden löytämisen testitapausten avulla haasteelliseksi, koska se miten tarkennetun haun ominaisuuksien oli tarkoitus tarkalleen toimia ei ollut työn tekijän tiedossa. Vaikka yksi testitapauksista epäonnistui ja löysi virheen, ilman haun alkuperäisiä vaatimuksia ei ollut varmaa tietoa siitä oliko tämä virhe oikeasti tarkoituksenmukainen.

Asiasanat: ohjelmistotestaus, ohjelmistotestausprosessi, testauksen suunnittelu, testitapaus

Laurea University of Applied Sciences **Abstract**

Degree Programme in Business Information Technology

Bachelor's Degree

Elli Uutela

Software testing of the Helmet advanced search function

Year 2022 Pages 37

The aim of this thesis was to complete a software testing process on Helmet.fi webpage's advanced search function focusing on its functionality and usability. A testing plan and multiple test cases were planned for the testing process. The test cases were executed and information about the functionality of the advanced search function collected based on the results. In addition, information on the usability of the search function during the testing process was gathered in the result analysis section. The thesis was not commissioned, so the main purpose was to improve the author's own competence regarding software testing and act as a show of competence on the topic. Evaluation of Helmet's advanced search function's functionality and usability and suggestions of possible development ideas based on the results of testing can be seen as the secondary objective of the thesis.

The theoretical framework of the thesis focuses on software testing and aims to give provide an overall picture about the different stages of software testing. Out of the stages of software testing, this thesis focuses mostly on the planning stage of the testing which is utilized when planning the testing and test cases for the Helmet advanced search.

The two main test methods utilized with the test case groups were explorative testing and functional testing, which were, in some test cases, accompanied by elements of the equivalence distribution black box testing method. These testing methods were not dependant on the source code being known which was why they were picked for this specific thesis.

The biggest challenge of the advanced search function testing process was that the original requirements set for the advanced search function were unknown. This made finding actual defects within the test cases challenging, since the author of this thesis did not have any information on how the features of the advanced search were precisely supposed to work. Although the test cases found one defect, it could not be classified as a real fault or as an intended functionality without the original requirements.

Keywords: software testing, software testing process, test planning, test case

Sisällysluettelo

1 Johdanto

Tämän opinnäytetyön tavoitteena oli suunnitella ja suorittaa testitapauksia Helmet-sivuston tarkennetussa haussa ja kuvata ohjelmistotestausprosessin kulku vaihe vaiheelta työn teoriaosaan nojaten. Ennen testitapauksien laatimista ja testauksen suunnittelemista tarkennettuun hakuun ja sen ominaisuuksiin, sekä haun suodattimiin tutustuttiin tutkivan testauksen avulla, jonka pohjalta päätettiin, millainen lähestymistapa testaukseen ottettiin ja miten testausta lähdettiin suunnittelemaan.

Helmet-sivusto on pääkaupunkiseudun kirjastojen yhteinen verkkosivu, jonka kautta käyttäjät voivat muun muassa varata kirjoja, lainata e-kirjoja, nähdä kirjastojen aukioloajat ja selata kirjastojen aineistoa. Tämä opinnäytetyö keskittyy aineiston selauksen työkalun, tarkennetun haun, toiminnallisuuden testaamiseen. Opinnäytetyössä hyödynnetään mustalaatikko-testausmenetelmiä, sillä ne soveltuvat hyvin testausprojekteihin, joissa ohjelmistokoodi ei ole testaajan käytettävissä ja testaus kohdistuu ohjelman syötteisiin. Opinnäytetyö ei ole toimeksianto Helmet-kirjastoilta eikä sen kirjoittajalla ole minkäänlaista yhteyttä Helmettiin satunnaista asiakkuutta lukuun ottamatta.

Opinnäytetyössä käytetään tutkivan, toiminnallisen ja jo mainitun mustalaatikkotestauksen testausmenetelmiä. Tutkivaa testausta hyödynnetään itse tarkennettuun hakuun ja sen ominaisuuksiin ja toiminnallisuuksiin tutustuessa. Varsinaiset testitapaukset ovat toiminnallista testausta, jotka keskittyvät testaamaan tarkennetun haun ennalta määrättyjä ominaisuuksia, jotka ovat tässä työssä operaattorit ja vuosilukukentät. Vuosilukukenttien yhteydessä hyödynnetään erityisesti mustalaatikkotestauksen testausmenetelmiin kuuluvan ekvivalenssijaon periaatteita.

Opinnäytetyön ensisijainen tavoite oli kehittää itse tekijänsä osaamista ohjelmistotestaajana ja toimia taidonnäytteenä. Lisäksi sen tarkoitus on toimia hyvänä esimerkkinä testausprosessin suunnittelusta ja toteuttamisesta. Työn toissijainen tavoite on antaa mahdollisia kehitysehdotuksia Helmetin tarkennettuun hakuun, erityisesti sen toiminnallisuuteen ja käytettävyyteen, testitapausten tulosten perusteella.

2 Työn lähtökohdat

2.1 Työn tavoitteet

Opinnäytetyön ensisijainen tavoite on kehittää tekijänsä omaa osaamista ohjelmistotestaajana ja toimia taidonnäytteenä. Konkreettisemmin työn tavoitteena on perehtyä Helmet-sivuston aineistonhakutyökaluun, laatia testaussuunnitelma ja testata sivun toiminnallisuutta sekä käytettävyyttä testitapauksien avulla, ja raportoida tulokset esittäen lopuksi mahdollisesti löytyneet kehitysehdotukset. Valmiin opinnäytetyön tulisi toimia esimerkkinä ohjelmistotestausprosessista ja sen eri vaiheista, jakaa tietoa Helmet- sivuston tarkennetun haun toiminnallisuudesta ja käytettävyydestä, sekä esittää mahdollisia kehitysehdotuksia.

2.2 Testauksen rajaus

Opinnäytetyössä keskitytään suomenkielisen Helmet-verkkosivuston ”tarkennettu haku” -hakutyökaluun, ja testitapaukset eivät täten kohdistu sivuston muihin hakutoimintoihin ”pikahakuun” ja ”perinteiseen aineistohakuun”. Testitapaukset keskittyvät tarkennetun haun perustoiminnallisuuksien varmistamiseen, ja yhden haun päätoiminnallisuuden testaukseen kohdistettiin enintään kymmenen testitapausta.

2.3 Keskeiset käsitteet

Ohjelmistotestaus	Prosessi, jossa ohjelmaa ja sen komponentteja testataan kohdistamalla ohjelmaan testejä virheiden löytämiseksi.
Virhe	Ihmisen tekemä erehdys, joka voi aiheuttaa ohjelmistokoodissa tai ohjelmistotuotteen dokumentaatiossa vian.
Vika	Tilanne, jossa ohjelma ei enää vastaa sille alun perin asetettuja vaatimuksia ja kriteereitä. Vika voi johtaa häiriöön.
Häiriö	Tilanne, jossa ohjelma ei toimi halutulla tavalla.
Testitapaus	Yksittäinen dokumentoitu testi, jonka tarkoitus on varmistaa jokin ennalta määritetty ohjelman toiminnallisuus.

Toiminnallinen testaus	Testausmenetelmä, joka keskittyy testaamaan testattavan ohjelman käytössä suorittamia toimintoja, eli sitä mitä ohjelman tulisi tehdä, kun se toimii kuten sen on tarkoitus toimia.
Vaatimusmääritelmä	Dokumentti, jossa kuvataan miten ohjelmistotuotteen ja sen komponenttien tulisi toimia.
Tutkiva testaus	Testausmenetelmä, jossa testaaja suunnittelee ja suorittaa testitapauksia tutustuen samalla testattavaan tuotteeseen.
Käytettävyystestaus	Testausmenetelmä, jossa testaajat keskittyvät havainnoimaan miten ohjelmistotuotteen käyttäjät käyttävät tuotetta suorittamalla ennalta määriteltyjä testitapauksia.
Mustalaatikkotestaus	Testausmenetelmä, jossa testattavan ohjelman sisäinen rakenne ja algoritmit ovat testaajalle täysin tuntemattomat, jonka vuoksi testauksessa keskitytään ohjelman syötteisiin.
Testaussuunnitelma	Projektikohtainen dokumentti, johon on kirjattu vähintään mitä ohjelmasta testataan, missä vaiheessa projektia testausta tehdään ja mitä testausmenetelmiä projektissa käytetään.

3 Ohjelmistotestaus

Ohjelmistotestaus on prosessi, jossa ohjelmaa ja/tai sen komponentteja testataan kohdistamalla ohjelmaan erilaisia testejä virheiden löytämiseksi. Ohjelmistotestauksen tarkoituksena on varmistaa, että testattava ohjelma toimii kuten sen on tarkoitus toimia ja vastaa sille asetettuja vaatimuksia. Tähän pyritään löytämällä ja korjaamalla ohjelmasta virheitä, mikä vaikuttaa positiivisesti ohjelman laatuun ja/tai luotettavuuteen. (Myers, Sandler & Badgett 2011, 2 & 6.) Ohjelmistotestauksen termi on vuosien saatossa määritelty useaan otteeseen korostaen testauksen eri puolia ja edellä esitetty Myersin määritelmä, johon tässä työssä on nojattu, on vain yksi monista. Hetzelin määritelmän mukaan ohjelmistotestaukseksi voidaan kutsua kaikkea toimintaa, joka keskittyy arvioimaan ohjelman ominaisuuksia, ja toimii täten ohjelmiston laadun mittarina. IEEE 829 testidokumentaation

määritelmässä ohjelmistotestauksen määritellään olevan ohjelmistotuotteen analysointiprosessi, jonka määränä on löytää eroja ohjelman nykyisen tilan ja sille asetettujen vaatimusten välillä, sekä arvioida ohjelman ominaisuuksia. (Nayyar 2019, 32.)

Virheellä tarkoitetaan testauksen yhteydessä ihmisen tekemää erehdystä, joka voi aiheuttaa ohjelmistokoodissa tai ohjelmistotuotteen dokumentaatiossa vian. Nämä ihmisen tekemät erehdykset voivat olla esimerkiksi näppäilyvirheitä koodissa tai heikosta kommunikaatiosta johtuneita väärinkäsityksiä vaatimusmäärittelyssä, jossa määritetään miten valmiin ohjelman tulisi toimia. Jos virhe johtaa tilanteeseen, jossa ohjelma ei enää vastaa sille alun perin asetettuja vaatimuksia ja kriteereitä, kyseessä on vika. Vika voi edelleen johtaa häiriöön, jolloin ohjelma ei enää toimi halutulla tavalla. (Chenmuturi 2010, 42.)

Ohjelmistotestaus sovittuu osaksi suurempaan ohjelmistotuotantoprosessin kokonaisuutta ja etenee rinnakkain ohjelmistotuotantoprojektin vaiheiden kanssa. Ohjelmistotestaustaukseen lukeutuvia toimia on mahdollista suorittaa ohjelmistotuotteen koko elinkaaren ajan, tuotekehityksen alusta aina sen ylläpitovaiheeseen saakka. Testaus ohjelmistotuotteen kehitysvaiheen aikana päättyy aikaisintaan silloin kun ohjelmiston testattavat komponentit suorittavat niiltä vaaditut toiminnot ja täyttävät asetut vaatimukset, eikä tuotteesta enää löydy merkittäviä virheitä laadittujen testitapausten myötä. Tämä ei kuitenkaan välttämättä tarkoita testausprosessin päättämistä, sillä kuten edellä mainittiin, testausta voidaan jatkaa vielä ohjelmistotuotteen ollessa tuotannossa esimerkiksi ylläpitotestauksen avulla. (Kasurinen 2013, 13.)

Testausprosessi on sisällöltään varsin laaja ja sen tarkalle rakenteelle ja lukuisille eri vaiheille ei ole yksittäistä tarkkaa mallipohjaa tai ohjekirjaa. Tämä johtuu ohjelmistotestauksen vahvasta projektikohtaisuudesta. Yleisellä tasolla testaus voidaan kuitenkin jakaa karkeasti seitsemään osaan: testauksen suunnitteluun, testauksen seurantaan ja hallintaan, testianalyysin, testien suunnitteluun, testien valmisteluun, testien suoritukseen, sekä testauksen päättämiseen. (ISTQB 2018, 17.)

Ohjelmistotestauksen tyyppi ja sen suorittamisajankohta projektissa on riippuvainen itse projektista, sen tyypistä ja tavoitteista, sekä projektissa käytettävästä vaihejakomallista, joka määrittää projektin vaiheet ja niiden järjestyksen. Vaihejakomallit jaetaan tavallisesti kolmeen pääkategoriaan: sekvenssimalleihin, iteratiivisiin malleihin ja inkrementaalisiin malleihin. Näiden mallien lisäksi on olemassa monia muita malleja, kuten nopean kehityksen malli (RAD) ja erilaisia ketteriä menetelmiä. (Homès 2012, 43–44.)

Sekvenssimalleissa projekti jaetaan useisiin ennalta määrättyihin vaiheisiin ja siirtyminen seuraavaan vaiheeseen tapahtuu, kun edellisen vaiheen tavoitteet on saavutettu. Esimerkiksi vesiputosmalli, v-malli ja w-malli ovat sekvenssimalleja. Sekvenssimallia hyödyntävässä projektissa testaus tapahtuu yleensä vasta tuotteen ohjelmistokehityksen jälkeen myöhään

projektissa. Iteratiivisissa malleissa, kuten spiraalimalli, projekti etenee kehityssykleissä, jossa projektin vaiheita toistetaan, kunnes vaadittu laatutaso saavutetaan. Toisin kuin sekvenssimalleissa iteratiivisissa malleissa testausta tehdään jo ohjelmistokehityksen aikana. Inkrementaaliset mallit ovat edellä mainittujen mallien sekoitus. Kuten sekvenssimalleissa, kun vaiheen tavoite on saavutettu, inkrementaalisessa mallissa siirrytään seuraavaan vaiheeseen. Mutta kun viimeinen vaihe on suoritettu, aloitetaan uusi sykli, jonka alussa tuotteeseen lisätään komponentteja ja ominaisuuksia. Inkrementaalisissa malleissa testaus tapahtuu tavallisesti jokaisen syklin lopussa. (Homès 2012, 44–52.)

Testauksen yleisten kaikenkattavien tavoitteiden lisäksi ohjelmistotestauksella on myös tarkempia projektikohtaisia tavoitteita. Näiden tavoitteiden täyttämiseksi on kehittynyt lukuisia erilaisia testaustyyppejä. Toiminnallinen testaus keskittyy nimensä mukaisesti testaamaan testattavan ohjelman käytössä suorittamia toimintoja, eli sitä mitä ohjelman tulisi tehdä, kun se toimii kuten sen on tarkoitus toimia, joka on määritetty vaatimusmäärityksessä. Toiminnallinen testaus arvioi kuinka hyvin testattava ohjelma vastaa sille asetettuja toiminnallisia vaatimuksia, ohjelman valmisastetta, soveltuvuutta, sekä oikeellisuutta. Toiminnallisen testauksen vastakohta tunnetaan nimellä ei-toiminnallinen testaus. Ei-toiminnallinen testaus keskittyy arvioimaan toiminnallisuuden sijaan ohjelman ominaisuuksia, kuten sen tietoturvaa, käytettävyyttä ja suorituskykyä, eli sitä, kuinka hyvin ohjelma toimii kvalitatiivisesti. (ISTQB 2018, 35.)

3.1 Testaustoiminnan hyviä käytäntöjä

Testaustoiminnan laajempaa kokonaisuutta on kannattavaa lähteä avaamaan alalla hyvin tunnettujen sertifikaattien ja mallien avulla. Seuraavissa alaluvuissa esitellään IEEE SWEBOK-malli, joka keskittyy kuvaamaan testaustoimintaan kuuluvaa sisältöä, sekä ISTQB (International Software Testing Qualifications Board)-sertifikaatin testaukselle asettamat seitsemän perusperiaatetta. (Kasurinen 2013, 75.) Ennen IEEE SWEBOK-malliin ja ISTQB-sertifikaattiin perehtymistä esitetään kuitenkin muutama niiden ulkopuolelle putoava testaukseen liittyvä periaate.

Kaiken ohjelmistotuotteeseen kohdistuvan testauksen tulisi aina suorittaa kolmannen osapuolen riippumaton ammattitaho, sillä se lisää tehokkaan ja sujuvan testausprosessin mahdollisuutta. Ohjelmistokoodaajien ei myöskään ikinä pitäisi testata omaa koodiaan, vaan jättää se kolmannelle osapuolelle. Kannattavaan ohjelmistotestaukseen kuuluvien testien tulisi ensisijaisesti keskittyä täyttämään asiakkaan ohjelmistotuotteelle asettamat vaatimukset. Ennen testauksen aloittamista tulisi laatia mahdollisimman yksityiskohtainen suunnitelma ohjelmalle tehtävästä testauksesta ja aloittaa suunnitelmaan perustuva testaus mahdollisimman pian ohjelmistokehitysprosessissa, sillä se tutkitusti vähentää löydettyjen virheiden määrää ohjelmistokehitysprosessin myöhemmissä vaiheissa. (Nayyar 2019, 35.)

Testien tulisi ohjelmistotestauksen alussa olla yksittäisiä tai kohdistua ohjelman yksittäisiin komponentteihin ja testauksen edetessä kasvaa testaamaan komponenttien yhteistoimintaa, sekä lopulta hyväksymistestaukseen eli varmistamaan ohjelman kokonaistoimintaa. Tämänkin jälkeen testaus jatkuu ohjelman ollessa tuotannossa, sillä Pareto-periaatteen mukaan 80% virheistä löydetään ohjelmistotestausprosessissa ja 20% virheistä joudutaan jäljittämään varsinaisen prosessin ulkopuolella. (Nayyar 2019, 35.)

3.1.1 ISTQB-sertifikaatti: testauksen seitsemän perusperiaatetta

ISTQB on kerännyt sertifikaatteihinsa ohjelmistotestaukseen liittyvää ydintietoa ja käytännön vinkkejä. Esimerkiksi ISTQB:n Core Foundation-sertifikaatin esittämät testauksen seitsemän perusperiaatetta sisältävät hyviä neuvoja jokaiselle testauksen parissa työskentelevälle. (ISTQB 2018, 15.)

Ensimmäinen ISTQB:n sertifikaatin esittämistä perusperiaatteista liittyy suoraan ohjelmistotestauksen määritelmään. Testaus voi osoittaa vikojen olemassaolon ohjelmistotuotteessa, mutta ei koskaan niiden poissaoloa. Se että ohjelmasta ei löydy vikoja ei ole todiste tuotteen virheettömyydestä, koska ohjelmassa voi olla edelleen löytämättömiä vikoja. Testaus voi ainoastaan vähentää uusien vikojen löytämisen todennäköisyyttä. (ISTQB 2018, 15.)

Sertifikaatin toinen esittämä perusperiaate on muistutus siitä, että täydellinen testaus on mahdotonta ja resurssisyistä siihen ei kannata edes pyrkiä. Testauksen suunnitteluvaiheessa on tärkeää määrittää, mitä ohjelmistotuotteessa on tärkeintä testata ja mihin testaus olisi kannattavinta kohdentaa, priorisoinnin, riskianalyysin ja testaustekniikoiden avulla. (ISTQB 2018, 15.)

Kolmas perusperiaate liittyy testauksen resursseihin. Aikainen testaus säästää aikaa ja rahaa. Vikojen löytäminen mahdollisimman aikaisin ohjelmistotuotantoprojektissa säästää resursseja, koska projektin loppupuolella löydetyt viat ovat kalliimpia korjata. Tämän vuoksi testaus on kannattavaa aloittaa mahdollisimman aikaisin ohjelmistotuotantoprojektissa, kuten esimerkiksi iteratiivisia malleja hyödyntävissä projekteissa tehdään. (ISTQB 2018, 15.)

ISTQB:n neljäs perusperiaate on, että viat kasaantuvat. Ohjelmistotuotteen viat keskittyvät yleensä pieneen osaan sen komponenteista. Tämä tekee vikakeskittymien ennustamisesta ja potentiaalisesti ongelmallisten komponenttien tunnistamisesta riskianalyysin avulla erityisen tärkeää, koska tällöin näihin osiin voidaan kohdentaa testausta. (ISTQB 2018, 15.)

Viides periaate koskee hyönteismyrkkyparadoksia. Hyönteismyrkkyparadoksi viittaa siihen että, kun samoja testejä toistetaan tarpeeksi kauan, ne eivät lopulta enää löydä uusia vikoja. Tämän vuoksi testausprosessin aikana on tärkeää laatia uusia testitapauksia ja muutettava

vanhoja, sekä muuttaa testiaineistoa. Hyönteismyrkkyparadoksia ei voi kuitenkaan soveltaa kaikkeen testaukseen, sillä esimerkiksi automatisoidun regressiotestauksen yhteydessä vikojen vähäisyys on merkki onnistumisesta. (ISTQB 2018, 15.)

Sertifikaatin kuudes perusperiaate on, että testaus on tilannesidonnaista. Erilaisia ohjelmistotuotteita testataan eri tavoin ja testaus on usein hyvin projektikohtaista. (ISTQB 2018, 15.)

Seitsemäs ja viimeinen ISTQB:n perusperiaate palaa takaisin ensimmäiseen periaatteeseen ja lähestyy virheettömyyden harhaluuloa toisesta näkökulmasta. Jos rakennettu ohjelma on käyttökelvoton, vikojen löytämisellä ja korjaamisella, sekä tavoitteiden täyttämisellä ei ole minkäänlaista vaikutusta. Ohjelmasta käyttökelvottoman voi tehdä esimerkiksi se, ettei sitä yksinkertaisesti ole suunniteltu oikein, ettei se saavuta sille asetettuja odotuksia tai täytä käyttäjän tarpeita. (ISTQB 2018, 15.)

3.1.2 IEEE SWEBOK-malli

IEEE SWEBOK (Software Engineering Body of Knowledge) -malli jakaa testaustoiminnan kuuteen pääkategoriaan sekä 19 alakategoriaan ja antaa hyvän käsityksen siitä mitä kaikkea testaukseen kuuluu. (Kasurinen 2013, 37.)

SWEBOK-mallin mukaan testauksen kuuteen pääkategoriaa kuuluvat testauksen perusasiat, testaustasot, testausmenetelmät, testauksen mittarit, testausprosessi ja testaustyökalut. Perusasioihin lukeutuvat asiat kuten testaukseen liittyvä terminologia, testauksen yhteys ohjelmistokehitykseen ja laadunhallintaan, sekä testitapausten valintaperusteet. Testaustasojen osio kattaa testauksen tavoitteet ja testitavat, sekä varsinaiset testauksen kohteet. Testausmenetelmät keskittyvät testauksen tekemisen lähtökohtien määrittelyyn ja testitapausten erilaisiin lähestymistapoihin. Testauksen mittarit osio perehtyy testattavan ohjelman laadun, testitapausten ja testaustyön arviointiin. Testiprosessi kattaa testauksen työvaiheet, testauksen lopetusehdot ja muut testauksen hallinnointiin liittyvät asiat. Nimensä mukaisesti testauksen työkalut -kategoria keskittyy testauksessa käytettäviin työkaluihin, esimerkiksi testitapausgeneraattoreihin, virheenjäljitystyökaluihin ja erilaisiin testausta tukeviin tukityökaluihin. (Kasurinen 2013, 37–38.)

3.2 Testausmenetelmät

3.2.1 Tutkiva testaus

Tutkiva testaus on testausmenetelmä, joka sekoittaa testattavasta ohjelmasta oppimisen, testien suunnittelun ja testien suorittamisen. Tutkivassa testauksessa testaaja suunnittelee ja suorittaa testitapauksia tutustuen samalla testattavaan tuotteeseen. Tämä testausmenetelmä

eroaa muista menetelmistä erityisesti vapaamuotoisuutensa vuoksi, sillä se ei vaadi pitkää suunnitteluprosessia tai tarkkaa testaussuunnitelmaa. (Copeland 2003, 202.)

Tutkivan testauksen testausprosessit ovat tapauskohtaisia, mutta niihin sisältyy yleensä ainakin joitain seuraavaksi esitetyistä vaiheista. Prosessi alkaa testaajan rakentamalla itselleen käsityksen siitä, miten ohjelma käyttäytyy toimiessaan ”oikein”. Seuraavaksi testaaja keksii testitapauksia, jotka todistaisivat, että ohjelma ei toimi kuten sen on tarkoitus toimia ja toteuttaa testitapaukset. Kun testitapaukset on suoritettu, testaaja vertaa omia odotuksiaan saatuihin tuloksiin. Tämän jälkeen voidaan palata takaisin prosessin alkuun ja toistaa prosessia, kunnes törmätään eroon odotetun tuloksen ja saadun tuloksen välillä. Toisenlaisessa prosessissa ohjelmaa saatetaan vain tutkia testauksen avulla ennen kuin testaaja luo oman käsityksensä siitä, miten ohjelman olisi määrä toimia. (Copeland 2003, 202–205.)

Koska tutkivassa testauksessa suoritettavista testitapauksista ei ole ennalta määritettyä listaa, testaajan edellinen testi toimii usein suunnannäyttäjänä seuraavalle testille ja testien päämotivaationa saattaa olla lisäinformaation kerääminen testattavasta ohjelmistotuotteesta. Myös testaajan oma käsitys ohjelman mahdollisista riskikohdista tai riskianalyysi voi antaa tutkivalle testaukselle tarkempaa suuntaa. (Copeland 2003, 202–205.)

3.2.2 Käytettävyystestaus

Käytettävyystestaus on testausmenetelmä, jossa testaajat keskittyvät havainnoimaan miten tuotteen todelliset käyttäjät käyttävät ohjelmistotuotetta suorittamalla ennalta määriteltyjä tehtäviä eli testitapauksia. Käytettävyystestausta voidaan tehdä sekä ohjelmistokehitysprosessin aikana ongelmien löytämiseen ja korjaamiseen, että ohjelmistotuotteen valmistumisen jälkeen tuotteella asetettujen vaatimusten toteutumisen varmistamiseen. Itse käytettävyyden termillä tarkoitetaan ohjelmistotestauksen yhteydessä käyttäjän kokemusta tuotteen helppokäyttöisyydestä ja käytön tehokkuudesta, jota käytetään mittaamaan käyttäjätyytyväisyyttä. (Barnum 2010, 13–19.)

Käytettävyystestauksen kannalta on tärkeää tunnistaa tuotteen todellinen käyttäjäryhmä ja luoda sen perusteella käyttäjäprofiili, joka vastaa tuotteen oletettua peruskäyttäjää ja jonka kaltaisia henkilöitä voidaan hakea osallistumaan käytettävyystestaukseen. Isommissa käytettävyystestausprojekteissa voidaan luoda myös useampia käyttäjäprofiileita, jotta tuotteen todellisesta käyttäjäryhmästä saadaan parempi käsitys. Käytettävyystestauksen aikana valitut testikäyttäjät suorittavat etukäteen määritettyjä testitapauksia, jotka keskittyvät tietyn tehtävän suorittamiseen ja/tai tietyn tavoitteen saavuttamiseen ohjelmassa, ohjelmistotestaajien havainnoidessa. Testikäyttäjiä voidaan esimerkiksi ohjeistaa kertomaan ajatteluprosessinsa ääneen testitapauksen aikana, mikä helpottaa havainnointiprosessia. Ohjelmistotuotteen käytettävyystestausprosessi toistetaan

ihanteellisesti useaan otteeseen, jotta ensimmäisellä testauskierroksella tehdyt muutokset voidaan myös testata. (Barnum 2010, 13–19.)

3.2.3 Mustalaatikkotestaus

Mustalaatikkotestauksessa testattavan ohjelman sisäinen rakenne ja algoritmit ovat testaajalle täysin tuntemattomat. Tämän vuoksi mustalaatikkotestauksessa ei keskitytä itse ohjelmaan ja sen sisältöön, vaan sille annettaviin syötteisiin löytäen tilanteita, joissa ohjelma ei toimi sille määritetyllä tavalla. Mustalaatikkotestauksen vastakohta on lasilaatikkotestaus, jossa testattavan ohjelman sisäinen rakenne ja algoritmit ovat näkyvissä testaajalle. Lasilaatikkotestauksessa keskitytään siihen mitä ohjelma sisäisesti tekee syötteen saatuaan. Tavoitteena on varmistaa, että halutut osat ohjelman rakenteessa aktivoituvat, kun ohjelma suoritetaan. (Myers, Sandler & Badgett 2011, 8–10.) Käytännössä mustalaatikkotestauksessa ohjelmalle annetaan siis syöte, jonka ohjelma käsittelee testaajan näkemättä, ja lopulta palauttaa tuloksen. Kun taas lasilaatikkotestauksessa ohjelmalle annetaan syöte, nähdään mitä ohjelma sisäisesti tekee syötteellä, kunnes ohjelma palauttaa jonkinlaisen tuloksen. (Kasurinen 2013, 52–53.)

Mustalaatikkotestauksen testaustekniikoihin kuuluvat ekvivalenssiositus, raja-arvoanalyysi, päätöstaulutestaus, tilasiirtymätestaus ja käyttötapaustestaus. Näitä tekniikoita voidaan hyödyntää toiminnallisessa ja ei-toiminnallisessa testauksessa. (ISTQB 2018, 50–53.)

Ekvivalenssiositus jakaa ohjelman syötteet ja muut tietoelementit ekvivalenssiluokkiin, jossa ohjelman oletetaan käsittelevän samaan ryhmään kuuluvat tiedot samalla tavalla. Tekniikan tavoitteena on ryhmittää syöte joko kelvollisiin, ohjelman oletettavasti hyväksymiin arvoihin, tai epäkelvollisiin, ohjelman oletettavasti hylkäämiin arvoihin. Huomaa että myös yksittäinen arvo voi olla ekvivalenssiluokka. Testausvaiheessa jokaisesta luokasta on tarkoitus testata ainakin yhtä arvoa mahdollisimman korkean kattavuuden saavuttamiseksi. (ISTQB 2018, 51–52.)

Raja-arvoanalyysi liittyy vahvasti ekvivalenssijakoon, sillä se keskittyy ekvivalenssiluokan pienimpään ja suurimpaan arvoon, eli kahden eri ekvivalenssiluokan vierekkäisiin arvoihin. Raja-arvoanalyysissa keskitytään näihin arvoihin, koska on huomattu, että virheitä ilmenee useimmin ekvivalenssiluokkien rajoilla kuin niiden keskellä, jokin luokka saattaa olla esimerkiksi laajempi kuin on kuviteltu, kunnes raja-arvoanalyysi paljastaa sen. (ISTQB 2018, 52.)

Päätöstaulutestaus keskittyy eri syötteiden yhdistelmien tuottamiin erilaisiin lopputuloksiin. Päätöstaulutestauksessa rakennetaan taulukko, jossa ehdot (esim. syötteet) ja toimenpiteet (esim. tulokset) muodostavat taulukon rivit ja jokainen sarake kuvaa ehtoyhdistelmää. (ISTQB 2018, 52–53.)

Tilasiirtymätestauksessa hyödynnetään usein tilasiirtymäkaaviota kuvaamaan ohjelmiston tilat, kuinka se saapuu ja poistuu tilasta, sekä liikkuu tilojen välillä. Tavoitteena on selvittää vaikuttaako ohjelman nykytila tai sen aikaisempi tapahtumahistoria ohjelman toimintaan, joka sopii hyvin esimerkiksi valikoihin liittyvään testaukseen. (ISTQB 2018, 53.)

3.3 Testauksen suunnittelu

Testauksen suunnittelun keskeisimmäksi käsitteeksi nousee usein testaussuunnitelma. Testaussuunnitelma on projektikohtainen dokumentti, johon on tavallisesti kirjattu vähintään mitä ohjelmasta testataan, missä vaiheessa projektia testausta tehdään ja mitä testausmenetelmiä käytetään. Testaussuunnitelmat ovat projektikohtaisia eikä niille ole yksittäistä kaiken kattavaa mallipohjaa. Suunnitelman laatimiseen vaikuttaa muun muassa projektin tyyppi ja valittu vaihejakomalli. Organisaatioissa testaussuunnitelma rakennetaan usein organisaation testausstrategian ja testauspolitiikan pohjalta. (Kasurinen 2013, 90.)

Testausstrategia on organisaatiokohtainen ohjedokumentti siitä, miten testaustoimintaa tulisi tehdä kyseisessä organisaatiossa ja organisaatiolla voi olla useita testausstrategioita erityyppisille projekteille. Testausstrategiaa laatiessa on tulisi huomioida myös organisaation muut dokumentit ja käytännöt, jotta testausstrategia on yhdenmukainen niiden kanssa. Näihin dokumentteihin voivat kuulua esimerkiksi organisaation tietoturvakäytännöt, laatukäytännöt, projektinhallintakäytännöt ja aiempi testaustyö, kuten aikaisemmat testausstrategiat ja -suunnitelmat. (Kasurinen 2013, 87–88.)

Konkreettisesti testausstrategia on usean sivun dokumentti, johon sisältyvät tavallisesti ainakin seuraavaksi mainitut asiat. Kuvaus testauksessa hyödynnettävistä työkaluista, joihin kuuluvat testausympäristö, testausvälineet ja mahdollinen testausautomaatio. Strategiassa määritetään myös organisaatiossa käytettävät testaustekniikat, testivaiheet ja testauksen toimintamallit. Dokumenttiin sisältyy yleensä tarkka kuvaus testaushenkilöstöstä, johon kuuluvat muun muassa henkilöstön tehtävänkuvaukset ja vastuualueet. Testausstrategia määrittää myös sen, miten testitapauksia suunnitellaan, valitaan ja priorisoidaan, sekä riskienhallintaan liittyvät toimenpiteet. Myös erilaiset laatuvaatimukset, kuten testauksen aloitus- ja lopetusehdot, laadunvalvonta ja -varmennus, sekä noudatettavat protokollat ja standardit, päätetään testausstrategiassa. (Kasurinen 2013, 87–88.)

Testausstrategian parina toimii organisaatioissa usein toinen dokumentti, jota kutsutaan testauspolitiikaksi. Testauspolitiikka on organisaation johdon laatima dokumentti, jossa tarkoitus on kuvata testauksen vastuuhenkilöt organisaatiossa ja testaustoiminnan tavoitteet. Toisin kuin testausstrategiassa, testauspolitiikan laatijoilla ei tarvitse olla testauksen ammattilaisia, sillä testauspolitiikka määrittää enemmänkin organisaation suhdetta testaukseen. (Kasurinen 2013, 86.)

3.3.1 Testaussuunnitelma

Vaikka testaussuunnitelmalle ei ole yhtä kaikenkattavaa pohjaa, erilaiset ohjelmistotestausstandardit toimivat hyvinä suunnanantajina testaussuunnitelman perussisällöstä. Seuraavissa esitellään ISO/IEC 29119 -standardin ja SPACE DIRT -menetelmän testaussuunnitelmien sisältöä. (Kasurinen 2013, 91.)

ISO/IEC 29119-standardi nostaa testausorganisaation keskiöön neljä dokumenttia. Näistä kaksi ensimmäistä ovat aiemmin mainitut testauspolitiikka ja testausstrategia, joiden tarkoitus on määrittää, miten testausta lähestytään organisaatiotasolla. Testauspolitiikka ja testausstrategia ovat molemmat organisaation johdon luomia dokumentteja, jotka toimivat suunnanantajana muulle organisaatiolle ja testauksen suunnittelijoille. Standardin neljäs esittämä dokumentti on testausraportti, joiden kautta testaustiimi jakaa tietoa projektin etenemisestä organisaation johdolle ja toimia kommunikaatiokanavana. Projektin lopussa testausraporteista voidaan koota loppuraportti, joka antaa kattavan kokonaiskuvan projektin etenemisestä sen koko ajalta. (Kasurinen 2013, 81–82.)

Testaussuunnitelma on ISO/IEC 29119-standardissa testauksen kolmas keskeinen dokumentti. Testaussuunnitelma on projektikohtainen dokumentti, jonka tarkoitus on sovittaa organisaation testausstrategian ja -politiikan yksittäisen projektin tarpeisiin. (Kasurinen 2013, 82.)

Tavallisesti aivan ensimmäiseksi ISO/IEC 29119-standariin pohjautuva testaussuunnitelma antaa kuvauksen itse projektista. Tähän kuvaukseen sisältyy koko projektin aikataulu, mitä projektissa tehdään ja muuta yleistä tietoa projektista. Itse testaustoiminnan tarkka aikataulu voidaan kirjata erikseen projektin aikataulusta. Projektin lisäksi myös itse testattavasta ohjelmistotuotteesta kirjoitetaan yleinen kuvaus, jossa kerrotaan tuotteen komponenteista ja niiden mahdollisista yhteyksistä toisiinsa, sekä mitä komponenttien ja koko ohjelman on tarkoitus tehdä. Lisäksi kirjataan mitä projektissa on tarkoitus testata ja mitä komponentteja tuotteesta testataan, sekä mihin rajoitteisiin ja ongelmiin testauksessa voidaan törmätä. Tuotteelle tehdään myös riskikartoitus, johon listataan tuotteeseen kohdistuvat merkittävät riskit ja ehdotukset siitä, miten ne voidaan välttää. Testaussuunnitelmassa tulisi hyödyntää organisaation testausstrategiaa ja määrittää sen avulla missä, milloin, miten ja kenen olisi tarkoitus testata tuotetta. Testaussuunnitelmaan on myös hyvä kertoa testaustiimin jäsenistä, kuka vastaa mistäkin testaamisen osasta ja siitä mitä taustatietoja testaajilla tulisi olla jo ennen testauksen aloittamista. (Kasurinen 2013, 91.)

Toisin kuin ISO/IEC 29119-standardin testaussuunnitelma, SPACE DIRT -menetelmä soveltuu pienemmille organisaatioille, joilla ei välttämättä ole käytössä testausstrategian ja -politiikan kaltaisia dokumentteja. Näiden dokumenttien puutteen vuoksi, verrattuna ISO/IEC 29119-

standardiin SPACE DIRT -menetelmän testaussuunnitelma onkin yksinkertaisempi laatia. (Kasurinen 2013, 91.)

SPACE DIRT on lyhenne, joka muodostuu menetelmän testaussuunnitelman osioiden englanninkielisistä alkukirjaimista. Laajuus (Scope), rajaa sen mitä on osia tuotteesta tarkoitus testata. Ihmiset (People), kattaa testauksen aikataulun, testaajien vastuut ja koulutusvaatimukset. Lähestymistapa (Approach), määrittää mitä testausmenetelmiä käytetään missäkin vaiheessa. Kriteerit (Criteria), nimensä mukaisesti kattaa testauksen aloitus-, lopetus-, keskeytys- ja jatkamiskriteerit. Ympäristö (Environment), tarkentaa sitä millaisen testausympäristön testaus vaatii. Tuotokset (Deliverables), kuvaa mitä testausprosessin tuottaa kehitysprosessien käyttöön. Satunnaiset (Incidentals), määrittelee testaukseen liittyvät poikkeavuudet sekä sen kuka saa muokata testaussuunnitelmaa. Riskit (Risks), listaa mahdolliset riskit ja miten niitä voidaan ehkäistä. Tehtävät (Tasks), kertoo testausprosessiin sisältyvät testauksen tehtävät. (Kasurinen 2013, 92.)

3.3.2 Testitapausten suunnittelu

Kun projektin testaussuunnitelma on valmis, voidaan siirtyä testitapauksien laatimiseen. Testitapaus on yksittäinen dokumentoitu testi, jonka tarkoitus on varmistaa jokin ennalta määritetty ohjelman toiminnallisuus. Testitapaus rakentuu ohjelmalle annettavista syötteistä, testitapauksen esiehdoista ja testitapauksen odotetusta tuloksesta. Testitapauksen onnistuminen määräytyy sen mukaan vastaako testitapauksen odotettu tulos tulosta, joka testitapauksesta saadaan sen suorittamisen jälkeen. Projektin testitapaukset ja niiden tulokset kirjataan testitapausdokumenttiin. Testitapausdokumentissa testeille annetaan tavallisesti tunnistusnumero, kuvaus testitapauksesta ja sen vaiheista, testitapauksen odotettu tulos, testitapauksen varsinainen tulos, sekä se onnistuiko testitapaus. (Chenmuturi 2010, 146–151.)

Testitapaukset voidaan jakaa kahteen tyyppiin niiden suoritustavan perusteella: itsenäisiin testitapauksiin, sekä sarjassa suoritettaviin testitapauksiin. Sarjassa suoritettavat testitapaukset ovat tavallisesti lyhyitä ja yksinkertaisia testitapauksia, jotka pohjautuvat sarjassa jo aiemmin suoritettuihin testitapauksiin. Tämä tekee sarjassa suoritettavista testitapauksista riippuvaisia toisistaan, jos yksi testitapaus keskellä sarjaa epäonnistuu niin myös kaikki sitä seuraavat testit voivat olla virheellisiä. Itsenäiset testitapaukset eivät tarvitse muita testitauksia, jotta ne voitaisiin suorittaa. Verrattuna sarjassa suoritettaviin testitapauksiin itsenäiset testitapaukset ovat kooltaan suurempia ja monimutkaisempia, sillä ne eivät ole riippuvaisia muista testitapauksista ja pitävät sisällään kaiken yksittäisen testitapauksen tarvitseman informaation. (Copeland 2003, 5–8.)

Se mihin ohjelmistotuotteen komponentteihin ja osiin testitapauksia suunnitellaan, määritetään riskianalyysin avulla. Riskianalyysin tavoitteena on löytää ohjelmistotuotteesta

riskejä tai uhkia, jotka vaikuttavat ohjelman toimintaan, sen kannattavuuteen ja loppupeleissä tuotteen valmistumiseen. Riskianalyysissa löydetyt merkittävimmät ja eniten ohjelman toimintaan haitallisesti vaikuttavat riskit, toimivat tavallisesti hyvänä paikkana aloittaa testitapauksien laatiminen. (Kasurinen 2013, 96.)

Testitapauksien valinta jatkuu riskianalyysin jälkeen testitapausten varsinaisiin valintakriteereihin. Monissa projekteissa testitapaukset valitaan joko suunnitelmalähtöisesti tai riskilähtöisesti, mutta valintakriteeriksi voi nousta myös tuotteelle aiemmin määritetyt laatuvaatimukset tai testitapauksia voidaan yksinkertaisesti lähteä laatimaan tuotteen tärkeimmiksi priorisoiduita komponenteista vähemmän tärkeisiin. Suunnitelmalähtöisten testitapausten valintavan tavoitteena on saavuttaa ohjelmalle määritetyt laatuvaatimukset mahdollisimman kustannustehokkaasti. Riskilähtöisten testitapausten valintatavan tavoitteena on varmistaa, että ohjelman tärkeimmät toiminnot toimivat ja että suurimmat ongelmat saadaan poistettua hyödyntäen rajallisia resursseja mahdollisimman tehokkaasti. Käytännössä organisaatio kuitenkin valitsee testitapauksensa harvoin pelkän suunnitelma- tai riskilähtöisen valintamenetelmän perusteella vaan ottaa myös vaikutteita toisesta menetelmästä. (Kasurinen 2013, 96–98.)

4 Helmet tarkennettu haku

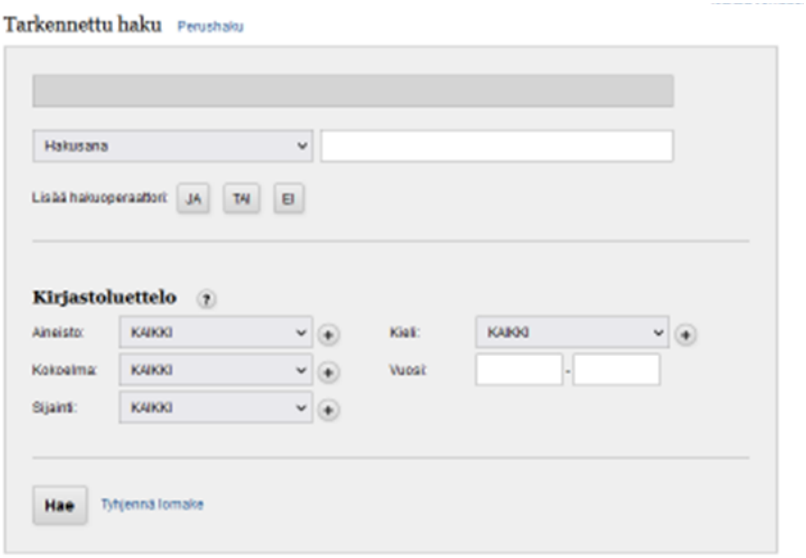
Tämän opinnäytetyön testauksen kohteena on Helmet -verkkosivun (www.helmet.fi) tarkennettu hakutoiminto. Helmet on pääkaupunkiseudun eli Helsingin, Espoon, Vantaan ja Kauniaisten yleisten kirjastojen verkosto, jotka jakavat yhteensä yli 3,2 miljoonaa teosta. (Helmet 2022a.) Helmetin verkkosivusto on näiden kirjastojen yhteinen verkkopalvelu. Verkkokirjasto sisältää yleistä tietoa Helmet-kirjastoista, asiakkaan tiedot kirjautumistoiminnon takana, aineistoesittelyitä, kokoelmaluettelun ja tämän työn kannalta oleellisen tiedonhaun. Helmetin sivuilla on kolme hakutoimintoa: pikahaku, perinteinen aineistonhaku ja tarkennettu haku, joihin tässä työssä keskitytään. (Helmet 2022b.) Opinnäytetyötä ei tehty minkäänlaisessa yhteistyössä Helmetin kanssa, eikä opinnäytetyön kirjoittajan ollut mahdollista nähdä sivuston lähdekoodia, sen toimintojen alkuperäisiä vaatimusmäärittelyitä ja muita samankaltaisia seikkoja.

Tarkennettuun hakuun tutustuttiin aluksi tutkivan testauksen keinojen avulla. Testaajalla oli alustava käsitys siitä, miten tarkennetun haun eri toimintojen oli tarkoitus toimia, ja sitä, miten haku oikeasti toimi lähdettiin selvittämään yksinkertaisten kirjaamattomien testien avulla. Suoritetut testit olivat melko yksinkertaisia ja keskittyivät selvittämään mitä tarkennetun haun eri ominaisuudet, kuten suodattimet ja hakusanat tarkalleen tekivät, sekä hakukenttien hyväksymiin syötteisiin. Testien päätarkoitus oli siis kerätä lisää informaatiota

tarkennetun haun toiminnasta ja testit valittiin sen pohjalta, mistä haun osasta tarvittiin vielä lisäinformaatiota.

Verkkokirjaston tarkennettu haku -toiminto jakautuu kahdelle sivulle: aloitussivulle, jossa alustava haku suoritetaan ja hakutulossivulle, jossa hakutuloksia voidaan edelleen rajata suodattimien avulla. Tämän opinnäytetyön testitapaukset keskittyvät pääasiassa aloitussivun esittämiin tarkennetun haun ominaisuuksiin, jonka vuoksi termiä tarkennettu haku käytetään työssä kuvaamaan pääasiassa pelkkää aloitussivua.

Kuten alla olevasta kuviosta 1: Tarkennetun haun aloitussivu, voidaan havaita, tarkennetun haun rakenne voidaan jakaa kahteen osaan. Tarkennetun haun ylempi puolikas sisältää hakusanat ja niihin liitettävät hakuoperaattorit, alempi puolikas on erilaisia haun rajoittimia sisältävä kirjastoluettelo.



Kuvio 1: Tarkennetun haun aloitussivu

Haun oletusnäkymässä tarkennetulla haulla on yksi käytettävä hakukenttä, johon kirjoitettava arvo on oletusarvoisesti tyypiltään hakusana. Tätä voidaan muuttaa hakukentän vasemmalla puolella olevasta alasvetovalikosta, joka määrittää aina vieraiseen hakukenttään syötetyn arvon tyypin. Alasvetovalikon antamat vaihtoehdot hakukentän arvon tyypiksi ovat: hakusana, teoksen nimi, aihe ja tekijä. Yhdestä alasvetovalikosta ei voi samaan aikaan valita useampaa vaihtoehtoa, hakukentän arvo ei voi siis samaan aikaan olla tyypiltään tekijä ja aihe. Tästä eteenpäin alasvetovalikon ja hakukentän yhdistelmää kutsutaan tässä työssä hakuehdoksi.

Kun alasvetovalikon tyypiksi asetetaan hakusana, tarkennettu haku kohdistuu kaikkiin niihin teoksiin joiden ”Teoksen tiedot”-osiossa hakusana esiintyy. Verkkokirjastossa teoksen tietoihin sisältyy muun muassa teoksen nimi, julkaisutiedot, asiasanat, tekijä ja muuta yleistä

tietoa aineistosta. Myös aihetyyppinen hakuarvo etsii tuloksia ”Teoksen tiedot”-osiosta, mutta keskittyy löytämään vastaavia tuloksia sen asiasanoista. Tekijä ja nimi -tyypin arvot hakevat nimensä mukaisesti hakutuloksia, joissa syötetty arvo vastaa teoksen tekijän nimeä tai teoksen nimeä. Alasvetovalikon eri vaihtoehtojen toiminnan perusteella vaikuttaa siltä, että hakusanatyyppinen haku sisältää myös kaikki muut hakutyyppivaihtoehdot, mikä tekee siitä kattavimman hakutyypin.

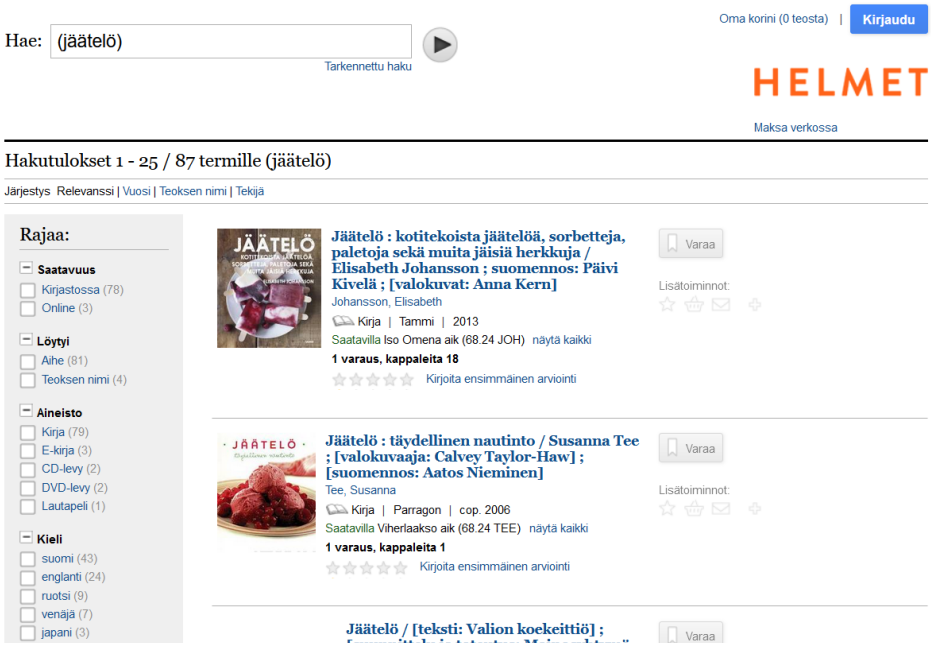
Tarkennettu haku voi olla vain yksi hakuehto, mutta alustavissa kokeiluissa ilmeni, että hakuehtoja on mahdollista lisätä yhteen hakuun ainakin yli sata. Hakuehtoja lisätään painamalla yhtä tarkennetun haun tarjoamista hakuoperaattoripainikkeista, niitä voidaan poistaa poistamalla koko hakuehto hakuehdon oikealla puolella olevasta x-painikkeesta tai sivun alalaidan ”Tyhjennä lomake” -vaihtoehdosta, hakuehdon hakuoperaattoria ei voi vaihtaa sen luomisen jälkeen. Hakuoperaattorit ovat Boolean arvot JA, TAI ja EI. Jokaiseen tarkennetun haun hakuehtoon liittyy hakuoperaattori, joka lukee hakuehdon vasemmalla puolella. Vaikka ensimmäisen hakuehdon edessä ei ole näkyvää hakuoperaattoria, se toimii kuten JA-operaattorin omaavat hakuehdot eli hakee vain kyseistä hakuehtoa vastaavia tuloksia. EI-hakuoperaattori poistaa hakutulossivulta kaikki hakuehtoa vastaavat tulokset. TAI-operaattori eroaa kahdesta muusta hakuoperaattorista hiukan, sillä yhdistää kaksi hakuehtoa ja hakee tuloksia, jotka vastaavat vähintään toista hakuehdoista.

Pienimmillään tarkennettu haku voidaan kirjoittamalla hakusanaksi yksi kirjain ja klikkaamalla hae-painiketta, esimerkiksi a-kirjaimelle löytyy huimat 144767 hakutulosta. Myös numeroille ja erikoismerkeille löytyy hakutuloksia, mutta välimerkeistä tulee hakutulossivulla ilmoitus ”Ei hakutuloksia”. Jos haku suoritetaan kaikkien hakuehtojen hakukenttien ollessa täysin tyhjä, tarkennettu haku palauttaa tekstin ”Anna vähintään yksi hakusana.”, eikä hakua suoriteta. Jos haku suoritetaan, kun ensimmäisen hakuehdon hakukenttä on tyhjä, tarkennettu haku siirtää sitä seuraavan hakuehdon ensimmäiseksi hakuehdoksi. Tällä tavalla TAI- ja EI-hakuoperaattorit on mahdollista saada hetkellisesti haun ensimmäiseksi hakuehdoksi. Mutta kun hakutulossivulta palataan takaisin tarkennettuun hakuun, ensimmäinen ei-tyhjä hakuehto muuttuu automaattisesti näkymättömän JA-operaattorin omaavaksi oletushakuehdoksi.

Tarkennetun haun kirjastoluettelo puolikas mahdollistaa tarkennetun haun rajaamisen aineiston, kokoelman, teoksen fyysisen sijainnin, teoksen kielen ja sen julkaisuvuoden niiden alasvetovalikon vaihtoehtojen ja julkaisuvuoden syötekenttien mukaan. Aineistolla tarkoitetaan rajauksen yhteydessä sitä, minkä tyyppinen teos on. Alasvetovalikkoon on listattu 24 eri aineistotyyppiä, joihin kuuluvat muun muassa kirja, e-kirja, CD-levy, lautapeli ja videokasetti. Kokoelma-alasvetovalikosta voidaan valita mihin suurempaan kokoelmaan haun kohde kuuluu, onko se esimerkiksi kaunokirja, osa Espoon kirjastojen kokoelmaa tai varastokokoelmaa. Sijainti määrittää teoksen fyysisen sijainnin eli kirjaston, kirjastoauton tai

jopa tarkan osaston kirjastossa. Kieli nimensä mukaisesti määrittää haetun aineiston kielen. Vuosiluku kenttiin voi syöttää sen miltä aikaväliltä hakutuloksia haetaan, teoksen julkaisuvuoden perusteella. Molempia näistä vuosilukukentistä ei ole pakko täyttää, ensimmäinen niistä määrittää mistä vuodesta alkaen hakutuloksia haetaan ja toinen mihin vuoteen saakka tuloksia haetaan.

Hakua voidaan rajata edelleen hakutulossivulla, mikä on nähtävillä alla kuviossa 2, jossa haku on tehty sanalla ”jäätelö”. Hakutulossivu esittää yhdellä sivulla enintään 25 tarkennetun haun löytämää hakutulosta ja jatkaa niitä tarvittaessa seuraaville sivuille. Hakutulossivujen välillä liikkuminen on mahdollista edellinen ja seuraava -painikkeiden avulla, tai siirtymällä enintään kaksi sivua eteen tai taakse päin klikkaamalla hakutulossivun numeroa.



Kuvio 2: Tarkennetun haun hakutulossivu hakusanalle ”jäätelö”

Hakutulokset voidaan järjestää sivun ylälaidassa relevanssin, vuoden, teoksen nimen ja tekijän mukaan, oletusarvoisesti ne on järjestetty relevanssin mukaan. Se miten sivu määrittää hakutulosten relevanssin on mahdoton tietää ilman sivun koodin tai sen määrittävän dokumentaation näkemistä, mutta järjestämismenetelmät ovat suhteellisen selkeiltä. Vuoden mukaan järjestyvät tulokset esitetään julkaisujärjestyksestä uusimmasta vanhimpaan. Teoksen nimi asettaa tulokset aakkosjärjestykseen teoksen nimen mukaisesti ja tekijä tekee saman kirjailijan nimen perusteella.

Hakutulossivulla haun rajaaminen tapahtuu sen vasemman laidan ”Rajaa”-sarakkeessa. Rajaa-sarake antaa vaihtoehdon rajata tuloksia saatavuuden, aineiston, kielen, sijainnin, julkaisuajan, kokoelman, genren, aiheen ja paikan mukaan. Työn kirjoittamisen aikana sarakkeeseen lisättiin myös uusi ”löytyi” rajaustapa, joka toimii käytännössä silloin, kun

hakuehdon tyyppi oli tarkennetussa haussa hakusana. Löytyi antaa mahdollisuuden rajata hakua hakusanasta teoksen aiheeseen, nimeen tai tekijään, eli hakuehdon muihin tyyppeihin tarkennetussa haussa. Jokaisella rajaustavalla on oma osittain avonainen alasvetovalikkonsa, joissa on alustavissa näkyvissä enintään viisi rajausvaihtoehtoa, muut rajausvaihtoehdot saadaan näkyviin painamalla alasvetovalikon alapuolella sijaitsevaa lisää -painiketta. Useimmilla rajaustavoilla rajausvaihtoehdot on järjestetty alasvetovalikkoon niin, että rajausvaihtoehdot, jotka kohdistuvat suurimpaan määrään hakutuloksia on listattu valikon huipulle. Poikkeuksena tästä on sijainti, joka on listattu aakkosjärjestyksessä, sekä kokoelma, jonka listauskriteeri ei ole helposti pääteltävissä. Edellä mainituista rajaustavoista saatavuus, genre ja paikka, jotka eivät olleet jo aiemmin näkyvillä tarkennetun haun aloitussivulla.

Saatavuus määrittää onko aineisto juuri tällä hetkellä saatavilla ja lainattavissa kirjastossa tai verkossa. Genren käsite vaikuttaa kattavan hakutulosten rajauksen yhteydessä useita aineiston ominaisuuksia ja sillä on päällekkäisyyksiä monen muun rajaustavan kanssa. Genre voi kattaa esimerkiksi aineiston tyypin, kokoelman ja aiheen, sekä aineiston varsinaisen kirjallisuuden genren. Myös paikka on hieman epämääräinen rajaustermi, se saattaa määrittää esimerkiksi tekijän kotimaan tai teoksen tarinan sijainnin.

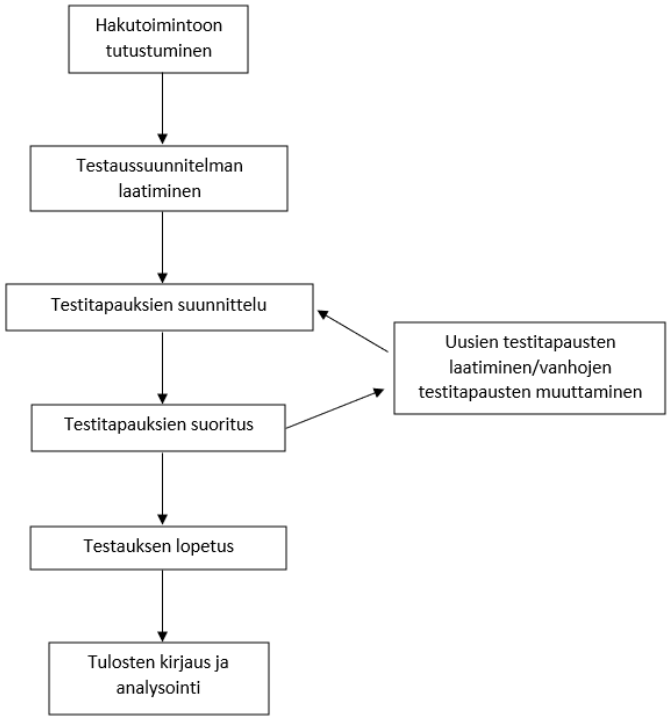
5 Testaussuunnitelma

Opinnäytetyön testaussuunnitelma suunniteltiin soveltamalla ISO/IEC 29119-standartissa esitettyjä testaussuunnitelman sisältöjä. Tässä luvussa esitetty testaussuunnitelma on hyvin vapaamuotoinen ja yksinkertaistettu versio ISO/IEC 29119-standardin mallista, joka keskittyy pääasiassa kuvaamaan testausprosessin eri vaiheita, sillä mallin kuvaama laaja testaussuunnitelma oli turhan tarkka ja kattava yhden henkilön suorittamalle pienelle testausprojektille. Erillistä testaussuunnitelmadokumenttia ei luotu.

Testaussuunnitelman ensimmäinen vaihe on tutustua testauksen kohteeseen eli tässä tapauksessa Helmet-sivuston tarkennettu haku -toimintoon, joka esitettiin aiemmassa alakappaleessa. Tutustumisvaiheeseen käytettiin kaksi päivää. Nyt kun testauksen kohde on tuttu ja sen komponentit ja niiden toiminnot on selitetty, voidaan siirtyä eteenpäin ja keskittyä muuhun testaussuunnitelman sisältöön. Tämän jälkeen siirrytään testitapauksien yksityiskohtaiseen suunnitteluun, suorittamiseen ja tarpeen vaatiessa muuttamiseen. Tätä sykliä jatketaan, kunnes testitapauksiin ei enää tarvitse tehdä muutoksia, jolloin testaus päätetään. Testaussuunnitelman viimeinen vaiheena on testauksen tulosten kirjaus ja analysointi.

Helmetin tarkennetun haun alustava testaussuunnitelma laadittiin yhden päivän aikana ja sitä täydennettiin ja tarkennettiin testausprosessin edetessä, jonka päätyttyä se kirjoitettiin vielä

viimeisen kerran puhtaaksi ja helpommin ymmärrettävään muotoon. Kuvio 3: Testaussuunnitalma, kuvaa Helmetin tarkennetun haun testaussuunnitelman päävaiheita.



Kuvio 3: Testaussuunnitelma

Opinnäytetyön testitapaukset on tarkoitus suorittaa verkkokirjaston suomenkielisessä versiossa, Firefox-selaimessa ja Windows-käyttöjärjestelmän omaavalla tietokoneella. Testitapaukset suoritetaan kahdessa ryhmässä, kahtena eri päivänä ja toistetaan kahteen kertaa testaajasta riippuvien virheiden, kuten näppäilyvirheiden välttämiseksi. Kaikki testitapaukset suoritetaan vielä kolmannen kerran juuri ennen kuin ne kirjataan opinnäytetyöhön.

Kaikki opinnäytetyön testitapaukset kirjataan Excel-dokumenttiin. Testitapausdokumentissa jokaiselle testitapaukselle annetaan tunnistenumero, testitapaus ja sen vaiheet selitetään, tehdään hypoteesi testitapauksen oletetusta tuloksesta, kirjataan testitapauksen oikea tulos sen suorituksen jälkeen ja määritetään, onnistuiko testitapaus. Testitapauksia voidaan muuttaa testauksen aikana, jotta ne soveltuisivat paremmin tarkennetun haun toiminnallisuuden ja käytettävyyden testaamiseen tai jos testitapauksen odotettu tulos ei vastaa sen oikeaa tulosta.

Testitapauksista keskittyvät verkkokirjaston tarkennetun haun perustoiminnallisuuksien testaamiseen, ja niiden on tarkoitus tuottaa kvalitatiivista eli laadullista tietoa tarkennetun haun toiminnallisuudesta ja käytettävyydestä. Koska tarkennetun haun alkuperäiset

toiminnallisuus- ja käytettävyysvaatimukset eivät ole testaajan käytettävissä, tämän työn esittämät väitteet tarkennetun haun toiminnallisuudesta ja käytettävyydestä ovat vain oletuksia ja myöhemmin tuloksissa ehdotuksia siitä, miten niitä voitaisiin kehittää asiantuntija-analyysin myötä. Opinnäytetyö ei kerää kvantitatiivista eli määrällistä tietoa, sillä kaikki työn tekijä on ainoa testitapaukset suorittava henkilö, työn aikarajoitteiden vuoksi testitapauksille ei ollut mahdollista luoda testikäyttäjien ryhmää.

Testaus lopetetaan, kun kaikki testitapaukset on suoritettu vähintään kolme kertaa, eikä testitapauksiin enää tarvitse tehdä muutoksia. Molempien testitapausryhmien testitapauksien laatimiseen ja suorittamiseen käytettiin lopulta yhteensä noin viikko.

Viimeiseksi testaussuunnitelmaa seurattaessa siirrytään kirjaamaan testitapausten tuloksia testitapausdokumentista opinnäytetyöhön ja analysoimaan testauksen tuloksia. Tähän aikaa oli varattu kaksi viikkoa.

6 Testitapaukset

Helmetin tarkennettuun hakuun kohdistuvat testitapaukset on jaettu kahteen testitapausryhmään, joista ensimmäinen keskittyy hakuoperaattorien toimintaan ja toinen vuosilukukenttiin. Testitapausryhmät suoritettiin eri päivinä, eivätkä ne ole millään asteella riippuvaisia toisistaan, vaan ovat toisistaan täysin itsenäisiä. Alustaviin kuuteentoista testitapaukseen lisättiin yksi uusi testitapaus, joka lisättiin testitapausryhmään kaksi epäonnistuneen testitapauksen vuoksi. Tämä testitapaus oli muunnelma epäonnistuneesta testitapauksesta, jonka avulla selvitettiin, mikä aiemmassa testitapauksessa tarkalleen aiheutti virheen.

Testitapauksien valitsemiseen hyödynnettiin tutkivan testauksen aikana kerättyä tietoa tarkennetusta hausta. Tutkiva testaus onnistui avaamaan hyvin tarkennetun haun tärkeimpiä ydinominaisuuksia, joka mahdollisti testitapausten priorisoinnin ja kohdistamisen haun tärkeimpiin ominaisuuksiin, eli ydinhakuun ja sen operaattoreihin, mihin erityisesti testitapausryhmä yksi keskittyi. Testitapausryhmä kahden vuosilukukenttiin kohdistuvat testitapaukset suunniteltiin suorana jatkona tutkivan testauksen aikana suoritetuille testeille, joihin keskityttiin tällöin vain hyvin ylimalkaisesti, sillä kenttiin ei ollut kannattavaa perehtyä kattavasti tutkivan testauksen aikana, koska niiden tarkempi testaaminen ekvivalenssijaon avulla olisi vaatinut tarkempaa testitapausten suunnittelemista jo tutkivan testauksen aikana. Vuosilukukenttien tarkempi testaus vaati testitapausdokumenttia, johon kirjata testin vaiheet ja tulokset.

6.1 Testitapausryhmä yksi

Ensimmäisen testiryhmän testitapaukset keskittyvät testaamaan tarkennetun haun hakuoperaattorien toimintaa, syötekentän arvo on näissä testitapauksissa aina ”hakusana”. Selkeyden vuoksi hakusanoja kutsutaan nimillä hakusana1 ja hakusana2. Kuten aiemmin mainittu, hakuoperaattorit Helmetin tarkennetussa haussa ovat JA, TAI ja EI. Yksittäisessä haussa voi tarvittaessa käyttää ainakin yli sataa testioperaattoria, mutta seuraavat testitapaukset käyttävät enintään kolmea hakuoperaattoria yhdessä haussa. Helmetin tarkennetussa haussa ensimmäiseen syötekenttään liitettyä hakuoperaattoria ei ole mahdollista muuttaa, se hakee aina syötettä vastaavia tuloksia eli toimii kuten JA-operaattori.

Ensimmäisen testitapausryhmän testitapaukset ovat pääasiassa sarjassa suoritettavia testitapauksia, sillä niiden onnistuminen määräytyy useassa testitapauksessa jo aiemmin suoritettujen testitapausten tuloksista, jos aikaisemmissa testitapauksissa on siis tapahtunut virheitä, jotka ovat jääneet testaajalta huomaamatta, niistä riippuvat testitapaukset eivät enää ole valideja.

Testitapauksen yksi tarkoituksena toimia pohjana tuleville testitapauksille ja testata tarkennetun haun minimaalisinta perustoiminnallisuutta, eli hakukenttään syötetään yksi hakusana ja haku suoritetaan. Testitapauksen odotettu tulos on, että haku löytää hakusanan sisältäviä tuloksia. Jatkossa tässä testitapauksessa käytettyä hakusanaa kutsutaan nimellä hakusana1. Kuvio 4: Testitapaus 1, havainnollistaa kuinka testitapaus 1 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
1	1) Ensimmäiseen hakukenttään syötetään hakusana "kala". 2) Suoritetaan haku.	Haku löytää tuloksia, jotka sisältävät sanan "kala".	-	Testitapausta ei suoritettu

Kuvio 4: Testitapaus 1

Toinen testitapaus toimii jatkona testitapaukselle yksi. Ensimmäiseen hakukenttään syötetään testitapauksessa yksi käytetty hakusana1, hakuun lisätään JA-operaattori, jonka hakukenttään syötetään myös hakusana1. Oletettu tulos on, että hakutulosten määrä ja järjestys on sama kuin testitapauksessa yksi, sillä hakutulosten määrän ei pitäisi muuttua hakusanan ollessa sama, eli hakusana1, JA-operaattorin yhteydessä. Kuvio 5: Testitapaus 2, havainnollistaa kuinka testitapaus 2 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
2	1) Ensimmäiseen hakukenttään syötetään hakusana "kala". 2) Lisätään JA-hakuoperaattori. 3) JA-hakuoperaattorin hakukentän arvoksi syötetään hakusana "kala". 4) Suoritetaan haku.	Haku löytää tuloksia, jotka sisältävät sanan "kala" ja "kala". Hakutuloksien määrä ja järjestys on sama kuin testitapauksessa 1.	-	Testitapausta ei suoritettu

Kuvio 5: Testitapaus 2

Kolmas testitapaus etenee pitkälti kuten toinen testitapaus, ensimmäiseen hakukenttään syötetään jälleen hakusana1, ja lisätään JA-operaattori. Tässä testitapauksessa JA-operaattorin hakukenttään syötetään kuitenkin eri hakusana, tästä eteenpäin hakusana2, kuin testitapauksen ensimmäisessä hakuehdossa. Nyt hakutulosten tulisi sisältää sekä ensimmäiseen että toiseen hakukenttään syötetyt hakusanat. Voidaan myös olettaa, että hakutulosten määrä on vähäisempi kuin aiemmissa testitapauksissa, koska aiemmin hakutulosten tuli sisältää vain hakusana1. Kuvio 6: Testitapaus 3, havainnollistaa kuinka testitapaus 3 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
3	1) Ensimmäiseen hakukenttään syötetään hakusana "kala". 2) Lisätään JA-hakuoperaattori. 3) JA-hakuoperaattorin hakukentän arvoksi syötetään hakusana "lintu". 4) Suoritetaan haku.	Haku löytää tuloksia, jotka sisältävät sanan "kala" ja "lintu". Hakutuloksien määrä on vähäisempi kuin testitapauksessa 1.	-	Testitapausta ei suoritettu

Kuvio 6: Testitapaus 3

Myös neljäs testitapaus on toisen testitapauksen kaltainen. Ensimmäiseen hakukenttään syötetään hakusana1, lisätään TAI-hakuoperaattori, jonka hakukenttään syötetään jälleen hakusana1. Tämän testitapauksen pitäisi löytää saman määrän hakutuloksia kuin ensimmäisen testitapauksen, sillä se hakee pelkästään yhtä hakusanaa vastaavia tuloksia. Kuvio 7: Testitapaus 4, havainnollistaa kuinka testitapaus 4 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
4	1) Ensimmäiseen hakukenttään syötetään hakusana "kala". 2) Lisätään TAI-hakuoperaattori. 3) TAI-hakuoperaattorin hakukentän arvoksi syötetään hakusana "kala". 4) Suoritetaan haku.	Haku löytää tuloksia, jotka sisältävät sanan "kala". Hakutuloksien määrä ja järjestys on sama kuin testitapauksessa 1.	-	Testitapausta ei suoritettu

Kuvio 7: Testitapaus 4

Viidennessä testitapauksessa ensimmäiseen hakukenttään syötetään hakusana1, lisätään TAI-operaattori, jonka hakukenttään syötetään hakusana2. Testitapauksen tulisi löytää hakutuloksia, joissa hakusana1 tai hakusana2 esiintyy, hakutuloksien määrän tulisi myös olla

suurempi kuin ensimmäisessä testitapauksessa, koska hakutulos voi sisältää molemmat tai vain toisen hakusanoista. Kuvio 8: Testitapaus 5, havainnollistaa kuinka testitapaus 5 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
5	1) Ensimmäiseen hakukenttään syötetään hakusana "kala". 2) Lisätään TAI-hakuoperaattori. 3) TAI-hakuoperaattorin hakukentän arvoksi syötetään hakusana "lintu". 4) Suoritetaan haku.	Haku löytää tuloksia, jotka sisältävät sanan "kala" tai "lintu". Hakutuloksien määrä on suurempi kuin testitapauksessa 1.	-	Testitapausta ei suoritettu

Kuvio 8: Testitapaus 5

Testitapaus kuusi on tässä testitapausryhmässä ensimmäinen instanssi, jossa käytetään EI-hakuoperaattoria. Kuten aikaisemmissa testitapauksissa ensimmäiseen hakukenttään syötetään hakusana1, seuraavaksi lisätään EI-operaattori, jonka hakukenttään syötetään myös hakusana1. Tämän testitapauksen ei pitäisi löytää yhtään hakutulosta, sillä EI-operaattori poistaa kaikki hakusana1:n esiintymät hakutuloksista. Kuvio 9: Testitapaus 6, havainnollistaa kuinka testitapaus 6 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
6	1) Ensimmäiseen hakukenttään syötetään hakusana "kala". 2) Lisätään EI-hakuoperaattori. 3) EI-hakuoperaattorin hakukentän arvoksi syötetään hakusana "kala". 4) Suoritetaan haku.	Haku ei löydä tuloksia, koska ensimmäinen hakuehto hakee sanaa "kala", mutta toinen hakuehto poistaa sanaa "kala" vastaavat tulokset.	-	Testitapausta ei suoritettu

Kuvio 9: Testitapaus 6

Seitsemännessä testitapauksessa ensimmäiseen hakukenttään syötetään hakusana1, lisätään EI-operaattori, jonka hakukenttään syötetään hakusana2. Testitapauksen hakutulosten tulisi siis palauttaa hakutuloksia, joissa hakusana1 esiintyy ja joihin hakusana2 ei sisälly. Lisäksi hakutulosten määrän tulisi vastata testitapausten 2 ja 3 erotusta. Kuvio 10: Testitapaus 7, havainnollistaa kuinka testitapaus 7 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
7	1) Ensimmäiseen hakukenttään syötetään hakusana "kala". 2) Lisätään EI-hakuoperaattori. 3) EI-hakuoperaattorin hakukentän arvoksi syötetään hakusana "lintu". 4) Suoritetaan haku.	Haku löytää tuloksia, jotka sisältävät sanan "kala" mutta eivät sanaa "lintu". Hakutulosten määrä on testitapausten 2 ja 3 erotus, eli 411 hakutulosta.	-	Testitapausta ei suoritettu

Kuvio 10: Testitapaus 7

Testitapaus kahdeksan on ensimmäisen testitapausryhmän viimeinen testitapaus. Testitapauksen ensimmäiset vaiheet ovat samat kuin kuudennen testitapauksen, ensimmäiseen hakukenttään syötetään hakusana1, lisätään EI-operaattori, jonka hakukenttään syötetään hakusana1. Tämän jälkeen lisätään JA-operaattori, jonka hakukentän arvoksi asetetaan jälleen hakusana1. Oletuksena on, että tällä haulla ei löydy yhtäkään hakutulosta, sillä EI-operaattorin tulisi kumota molemmat JA-operaattorit, kun hakusana on niissä kaikissa sama. Kuvio 11: Testitapaus 8, havainnollistaa kuinka testitapaus 8 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
8	1) Suoritetaan testitapauksen 7 vaiheet 1, 2 ja 3. 2) Lisätään JA-hakuoperaattori. 3) Uuden JA-hakuoperaattorin hakukentän arvoksi syötetään hakusana "kala". 4) Suoritetaan haku.	Haku ei löydä tuloksia, koska EI-hakuoperaattori sanalle "kala" kattaa kaikki hakuehdot.	-	Testitapausta ei suoritettu

Kuvio 11: Testitapaus 8

6.2 Testitapausryhmä kaksi

Toisen testitapausryhmän testitapaukset keskittyvät testaamaan tarkennetun haun vuosilukukenttiä, jotka rajaavat hakua teoksen julkaisuvuoden mukaan. Kaikkien tämän ryhmän testitapausten hakusanaksi on asetettu aiemmissa testitapauksissa käytetty hakusana1 ja käytössä on vain haun ensimmäinen hakukenttä, sillä niillä ei ole näissä testitapauksissa suurta merkitystä, tämän vuoksi niitä ei myöskään erikseen mainita testitapauksia kuvatessa. Testitapausten numerot jatkuvat siitä mihin ne edellisessä testitapausryhmässä jäivät.

Testausdokumentin yhdeksännessä testitapauksessa vasemmanpuoleiseen vuosilukukenttään syötetään vuosiluku 2010. Vasemmanpuoleisen kentän tulisi määrittää sen vuosiluvun, milloin teos on aikaisintaan julkaistu eli odotettavasti hakutulosten pitäisi olla vuodelta 2010 ja sitä seuraavilta vuosilta. Kuvio 12: Testitapaus 9, havainnollistaa kuinka testitapaus 9 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
9	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo 2010.	Hakutulokset ovat vuodelta 2010 ja siitä eteen päin.	-	Testitapausta ei suoritettu

Kuvio 12: Testitapaus 9

Testitapauksessa kymmenen on testitapauksen yhdeksän peilikuva. Oikeanpuoleiseen vuosilukukenttään syötetään vuosiluku 2010. Oikeanpuoleisen kentän tulisi määrittää sen

vuosiluvun, milloin teos on viimeistään julkaistu eli odotettavasti hakutulosten pitäisi olla vuodelta 2010 ja sitä ennen. Kuvio 13: Testitapaus 10, havainnollistaa kuinka testitapaus 10 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
10	1) Asetetaan hakusanaksi "kala". 2) Syötetään oikean puoleiseen vuosilukukenttään arvo 2010.	Hakutulokset ovat vuodelta 2010 ja sitä ennen.	-	Testitapausta ei suoritettu

Kuvio 13: Testitapaus 10

Yhdennessätoista testitapauksessa vasemmanpuoleiseen vuosilukukenttään syötetään arvo 2010 ja oikeanpuoleiseen kenttään sama 2010. Oletuksena on, että tällä vuosivälillä hakutulokset ovat pelkästään vuodelta 2010. Kuvio 14: Testitapaus 11, havainnollistaa kuinka testitapaus 11 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
11	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo 2010. 3) Syötetään oikean puoleiseen vuosilukukenttään arvo 2010.	Hakutolukset ovat pelkästään vuodelta 2010.	-	Testitapausta ei suoritettu

Kuvio 14: Testitapaus 11

Kahdestoista testitapaus on lähes sama kuin yhdestoista, mutta sen vuosilukuväli on suurempi. Vasemmanpuoleiseen vuosilukukenttään syötetään arvo 2010 ja oikeanpuoleiseen kenttään arvo 2020. Tämän haun tulosten pitäisi olla vuodelta 2010 ja siitä eteen päin vuoteen 2020 saakka, myös vuoden 2020 pitäisi kuulua tuloksiin. Kuvio 15: Testitapaus 12, havainnollistaa kuinka testitapaus 12 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
12	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo 2010. 3) Syötetään oikean puoleiseen vuosilukukenttään arvo 2020.	Hakutolukset ovat vuodelta 2010 ja siitä eteen päin vuoteen 2020 asti (vuosi 2020 kuuluu tuloksiin).	-	Testitapausta ei suoritettu

Kuvio 15: Testitapaus 12

Testitapauksessa kolmetoista vasemmanpuoleisen vuosilukukentän luvuksi asetetaan 2010 ja oikean 2000. On odotettavissa, että tälle haulle ei löydy hakutuloksia, koska teosta ei voi olla

julkaistu vuoden 2010 jälkeen mutta samalla ennen vuotta 2000. Kuvio 16: Testitapaus 13, havainnollistaa kuinka testitapaus 13 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
13	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo 2010. 3) Syötetään oikean puoleiseen vuosilukukenttään arvo 2000.	Ei hakutuloksia, koska teosta ei voi olla julkaistu vuoden 2010 jälkeen mutta samalla ennen vuotta 2000.	-	Testitapausta ei suoritettu

Kuvio 16: Testitapaus 13

Neljännessätoista testitapauksessa vasemmanpuoleisen vuosilukukentän arvoksi annetaan 2023, joka on tämän työn kirjoituksen aikana vielä useita kuukausia tulevaisuudessa. Tämän vuoksi oletuksena on, että tällä haulla ei saada tuloksia. Kuvio 17: Testitapaus 14, havainnollistaa kuinka testitapaus 14 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
14	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo 2023.	Ei hakutuloksia, sillä teoksen julkaisuvuosi oli testauksen suoritusaikana vielä tulevaisuudessa.	-	Testitapausta ei suoritettu

Kuvio 17: Testitapaus 14

Viidennessätoista testitapauksessa vasemmanpuoleisen vuosilukukentän arvoksi annetaan 2022 eli työn tämän työn kirjoitusvuosi. Oletuksena on, että hakutulokset ovat tältä kuluneelta vuodelta. Kuvio 18: Testitapaus 15, havainnollistaa kuinka testitapaus 15 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
15	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo 2022.	Hakutulokset kuluneelta vuodelta (2022).	-	Testitapausta ei suoritettu

Kuvio 18: Testitapaus 15

Kuudestoista testitapaus on ainoa tämän ryhmän testitapauksista, jossa käytetään miinusmerkkiä kuvamaan vuosilukua ennen ajanlaskun alkua. Vasempaan vuosilukukenttään syötetään arvo -20 ja oikeaan 2000. Tämän haun pitäisi palauttaa hakutuloksia vuodesta 20 eaa. vuoteen 2000 saakka, myös vuosi 2000 kuuluu mukaan hakutuloksiin. Kuvio 19: Testitapaus 16, havainnollistaa kuinka testitapaus 3 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
16	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo -20. 3) Syötetään oikean puoleiseen vuosilukukenttään arvo 2000.	Hakutulokset vuodesta 20 eaa. vuoteen 2000 asti (vuodet -20 eaa. ja 2000 kuuluvat tuloksiin).	-	Testitapausta ei suoritettu

Kuvio 19: Testitapaus 16

Tämän testitapausryhmän viimeinen testitapaus on testitapaus seitsemäntoista, joka on hyvin samankaltainen kuin sitä edeltävä testitapaus. Vasemman vuosilukukentän arvoksi asetetaan 0 ja oikean arvoksi 2010. Oletuksena on, että hakutulokset ovat vuodelta 0 ja siitä eteen päin vuoteen 2010 asti (vuosi 2010 kuuluu tuloksiin). Testitapaus seitsemäntoista suunniteltiin muita testitapauksia huomattavasti myöhemmin, testitapaus kuudentoista epäonnistuttua. Kuvio 20: Testitapaus 17, havainnollistaa kuinka testitapaus 17 on kirjattu testitapausdokumenttiin.

Testi numero	Kuvaus testitapauksesta	Testitapauksen odotettu tulos	Tulos	Status
17	1) Asetetaan hakusanaksi "kala". 2) Syötetään vasemman puoleiseen vuosilukukenttään arvo -20. 3) Syötetään oikean puoleiseen vuosilukukenttään arvo 2010.	Hakutulokset ovat vuodelta 0 ja siitä eteen päin vuoteen 2010 asti (vuosi 2010 kuuluu tuloksiin).	-	Testitapausta ei suoritettu

Kuvio 20: Testitapaus 17

7 Testitapausten tulokset ja jatkokehitysehdotukset

Kaikki suunnitellut testitapaukset suoritettiin, mutta testitapausryhmään kaksi kuuluneen testitapauksen kuusitoista tulos ei vastannut sille asetettua odotettua tulosta ja täten epäonnistui. Tämä ei kuitenkaan tarkoita, että testitapaus kuusitoista olisi itsessään epäonnistunut, sillä testitapauksien tarkoitus on varmistaa ohjelman tietyn osan toiminnallisuus, johon virheiden löytäminen ja korjaaminen ohjelmasta lukeutuu.

Testitapaus kuusitoista kohdistui tarkennetun haun alemman puolikkaan vuosilukukenttiin. Testitapauksessa vasemmanpuoleiseen vuosilukukenttään syötettiin arvo -20 ja oikeanpuoleiseen kenttään arvo 2000. Testitapauksen oletuksena oli, että tarkennettu haku löytää tuloksia vuodesta 20 ennen ajanlaskun alkua vuoteen 2000. Käytännössä testitapaus ei ole realistinen, sillä olisi absurdia ajatella, että Verkkokirjastossa tai missään julkisessa kirjastossa olisi näin vanhoja teoksia, mutta siitä huolimatta haun pitäisi löytää hakutuloksia, jotka sijoittuvat hakuajanjakson loppupuolelle. Kun testitapaus suoritettiin, se ei kuitenkaan löytänyt yhtään hakutulosta. Muutaman testiarvon syöttämisen jälkeen näyttää siltä, että

testitapaus epäonnistui vasemmanpuoleiseen vuosilukukenttään syötetyn miinusmerkin vuoksi, sillä vaikuttaa siltä, että vuosilukukentät eivät hyväksy välimerkkejä. Välimerkkien hylkääminen on toistuva ilmiö tarkennetun haun kaikissa syötekentissä, sillä samaan ilmiöön törmättiin työssä aiemmin päähakukenttiin tutustuessa.

Kaikki ensimmäisen testitapausryhmän testitapaukset onnistuivat virheettömästi ja varmistivat tarkennetun haun perusominaisuuksien, haun ja operaattorien, toiminnan. Tällä testitapausryhmälle odotettujen tulosten asettaminen oli selkeästi helpompaa kuin toiselle testitapausryhmälle sillä tarkennetun haun käyttämät operaattorit olivat Boolean -operaattoreita, joiden toimintatavat ovat universaalisti tiedossa kaikille, jotka ovat ohjelmoineet aiemmin. Toiminnallisuuden suhteen testitapausryhmä yksi toimi suoritettujen testitapausten kannalta täydellisesti, mutta sen käytettävyydestä nousi testausprosessin aikana muutama huomio.

Useita peräkkäisiä tarkennettuja hakuja tehdessä operaattorien muuttaminen tuntui työläämmältä kuin sen tarvitsisi olla. Operaattori lisätään yhtä kolmesta operaattori -painikkeesta, JA, EI ja TAI, painamalla, mutta kun operaattori on lisätty sitä ei ole enää mahdollista muuttaa. Ainoa tapa, jolla hakuoperaattori, ja sivuvaikutuksena koko hakuehto, voidaan poistaa, on hakuehdon oikealla puolella olevasta x-painikkeesta tai sivun alalaidan ”Tyhjennä lomake” -vaihtoehdosta. Jos hakuoperaattori halutaan siis vaihtaa esimerkiksi JA-operaattorista TAI-operaattoriksi, koko hakuehto pitää poistaa, eli hakusanatyyppi on valittava uudestaan ja hakukenttä on täytettävä uudestaan. Tämä vaatii käyttäjältä huomattavasti enemmän työtä kuin hakuoperaattorin suoraan vaihtaminen ja hakusanatyypin ja hakukentän syötteen säilyttäminen.

Toinen merkittävä kehitysehdotus tarkennetun haun käytettävyyttä koskien on sen hakusanatyyppien selittäminen tarkennetun haun aloitussivulla. Teoksen nimi ja kirjailija ovat selkeitä hakusanatyyppejä, mutta millaisia tuloksia hakusanatyyppiset ja aihetyyppiset hakusanat tarkalleen etsivät? Jotta käyttäjät voisivat hyödyntää kaikkia eri hakusanatyyppejä, tulossivulla esiintyviä suodattimia ja tarkennetun haun muita ominaisuuksia, heidän tulee tietää mitä ne tarkalleen tekevät. Verkkokirjastosta löytyy kyllä sivu, joka antaa erinäisiä hakuohjeita ja selittää esimerkiksi operaattorien toimintaa käyttäjälle, mutta tarkennetun haun sivulla ei ole mitään viittausta kyseisen sivun olemassaoloon. Linkki tälle hakusanaohjeita-sivulle tarkennetun haun yhteydessä antaisi käyttäjille mahdollisuuden oppia lisää tarkennetun haun toiminnasta ja mahdollistaa näin tarkennetun haun tehokkaamman käytön. Vaihtoehtoisesti tarkennetun haun aloitussivulle voisi lisätä kysymysmerkki -painikkeita, esimerkiksi kirjastoluettelon yhteyteen, hakusanatyyppien ja suodattimien viereen, jotka antavat klikatessa lyhyen selityksen kyseisestä haun ominaisuudesta.

Useita tarkennettuja hakuja peräkkäin suorittaessa myös hakutulossivulta siirtyminen takaisin tarkennetun haun aloitussivulle nousi tärkeäksi ominaisuudeksi. Hakutulossivulla on hyperlinkillä varustettu Tarkennettu haku -teksti, joka vie takaisin tarkennettuun hakuun säilyttäen aiemmat hakuehdot. Pelkkä Tarkennettu haku -teksti ei kuitenkaan kerro käyttäjälle onko hän palaamassa edelliseen tekemäänsä hakuun vai aloittamassa täysin uuden haun, mikä saattaa johtaa siihen, että käyttäjä alkaa etsimään hakutulossivulta jonkinlaista ”palaa” -painiketta. Tarkennettu haku -tekstin täsmentäminen esimerkiksi muotoon ”Palaa tarkennettuun hakuun” tai yksinkertaisesti ”Edellinen”, saattaisi helpottaa ja selkeyttää tarkennettua hakua ensikertaa kokeilevien käyttäjien käyttökokemusta.

8 Yhteenveto

Suoritettujen testitapausten pohjalta vaikuttaa siltä, että Helmetin tarkennettu haun perustoiminnallisuudet toimivat varsin hyvin, mikä olikin odotettavissa jo tuotannossa olevalta tuotteelta. Testitapausten ja tutkivan testauksen pohjalta löytyi kuitenkin muutama tarkennetun haun käytettävyyteen liittyvä kehitysehdotus, jotka saattaisivat parantaa haun käyttökokemusta.

Kaiken kaikkiaan Helmetin Verkkokirjaston tarkennetun haun toiminnallinen testaus oli haastavaa ilman alkuperäisiä vaatimusmäärityksiä ja pääsyä lähdekoodiin. Toiminnallisen testauksen tarkoitus on testata ohjelman jotain tiettyä toiminnallisuutta, kun käsitystä siitä miten tämän toiminnallisuuden on tarkoitus toimia ei ole, on vaikea tehdä väitteitä testauksen tuloksista. Epäonnistunut testitapaus saattaakin toimia alkuperäisen vaatimusmäärityksen mukaan juuri kuin sen olisi tarkoitus toimia tai päinvastoin, mutta koska vaatimusmääritys ei ole testaajan tiedossa, nämä virheet saattavat jäädä huomioimatta tai resursseja voidaan käyttää sellaisten virheiden korjaamiseen, jotka eivät todellisuudessaan ole virheitä. Testaaja, joka ei tiedä alkuperäisestä vaatimusmäärityksestä mitään, voi vain tehdä oletuksia siitä, miten toiminnallisuuksien on tarkoitus toimia, ja jos nämä oletukset ovat virheellisiä myös niihin liittyvät testitapaukset ovat mahdollisesti virheellisiä. Toiminnallinen testaus ilman alkuperäistä vaatimusmäärítelmää muistuttaa läheisesti hyvin suunniteltua tutkivaa testausta.

Toiminnallinen testaus ilman alkuperäistä vaatimusmäärítelmää voi olla joissain tapauksissa täysin merkityksetöntä. Mutta yksinkertaisissa ohjelmissa, missä ohjelman suunnitellut toiminnallisuudet on mahdollista arvata, sitä voidaan hyödyntää perustoiminnallisuuksien ja esimerkiksi syötekenttien toimivuuden testaamiseen. Monimutkaisemmissa ohjelmissa toiminnallista testausta ei kuitenkaan tulisi käyttää, kun vaatimusmäärítykset eivät ole tiedossa, sillä virheellisten oletettujen toiminnallisuuksien määrä on suurempi.

Tämän opinnäytetyön toiminnallinen testaus oli testaajan mielestä pääasiassa onnistunutta, muutamaa ongelmakohtaa lukuun ottamatta. Alkuperäisten vaatimusmääritelmien tunteminen olisi kuitenkin selkeyttänyt testitapausten laatimista ja nopeuttanut tutkivaa testausta, sillä oletuksien pohjalta työskentely on hitaampaa.

9 Oman oppimisen arviointi

Ohjelmistotestausprosessin suorittaminen Helmetin tarkennetulla haulla oli hyvin opettavainen projekti, joka selkeytti kirjoittajan omaa, aiemmin pääasiassa teoriasta ja pienistä projekteista muodostunutta pirstaleista käsitystä ohjelmistotestauksen kokonaisuudesta. Opinnäytetyön alussa teoriaosuuden kirjoittaminen antoi mahdollisuuden koota kirjoittajan oman aiemman tietämyksen ohjelmistotestauksesta, ja lähteä laajentamaan ja syventämään teoreettista osaamista keskittyen erityisesti tulevien testitapausten kannalta konkreettisiin aiheisiin.

Teoriaosuuden yksi haaste oli sen kohdentaminen testauksen keskeisimpiin termeihin ja testausosuuden kannalta oleellisiin aiheisiin, sillä testaus on käsitteenä hyvin laaja ja jos aihealuetta ei rajata tarpeeksi kirjoitetun tekstin määrä nousee helposti yli opinnäytetyön suositeltujen rajojen. Se että työn alussa ei vielä ollut tarkkaa näkemystä millaisia testitapauksia oli määrä rakentaa, johti siihen, että teoriaosuuteen jouduttiin myöhemmin palaamaan uudestaan ja täydentämään. Tämä olisi voitu yleisen paremman suunnittelun lisäksi välttää sillä, että tutkiva testaus tarkennettuun hakuun olisi suoritettu heti työn alkumetreillä, joka olisi antanut ylipäätänsä paremman käsityksen hakukonetyökaluun kohdistuvista testausmahdollisuuksista. Kaiken kaikkiaan teoriaosuus oli mielestäni suhteellisen hyvin onnistunut opinnäytetyölle asetettujen tavoitteiden kannalta ja kokosi hyvin opintojaksoilla läpikäydyn informaation syventäen sitä työn keskeisimpien kokonaisuuksien osalta.

Varsinaisen testausosuuden tekemisessä ei ollut merkittäviä haasteita, mutta tästä huolimatta jokaisen läpilukukerran osuudesta löytyi jotain uutta kehitettävää. Ensimmäisen testausosuus version valmistuttua huomasin osiossa teorian ja käytännön yhteyden puutteen, mitä pyrin korjaamaan mahdollisimman paljon seuraavissa versioissa. Testitapausten laatiminen ja suorittaminen oli mielenkiintoista ja mielestäni kannattava kokemus käytännössä, ja sai minut ymmärtämään miksi testitapausdokumentti, ja testitapausten yksityiskohtainen kuvailu ovat tärkeitä.

Kaiken kaikkiaan olen tyytyväinen opinnäytetyöhöni ja siihen mitä opin prosessin aikana. Työn kirjoittaminen oli hyvä kokemus ja sen kirjoittaminen toimi muistutuksena siitä, että vaikka

aihealueen teoria olisi kuinka tuttu, teorian sovittaminen käytäntöön on aina oma haasteensa ja jotkin asiat voi oppia täysin vasta kun niitä soveltaa käytännössä.

Lähteet

Sähköiset

Barum, C. 2010. Usability Testing Essentials : Ready, Set... Test!. E-kirja. Elsevier Science & Technology.

Chenmuturi, M. 2010. Mastering Software Quality Assurance: Best Practices, Tools and Techniques for Software Developers. E-kirja. J. Ross Publishing Inc.

Copeland, L. 2003. Practitioner's Guide to Software Test Design. E-kirja. Artech House.

Helmet 2022a. Mikä Helmet on? Viitattu 9.5.2022. https://www.helmet.fi/fi-FI/Info/Mika_Helmet_on

Helmet 2022b. Tietoa sivustosta. Viitattu 9.5.2022. https://www.helmet.fi/fi-FI/Info/Tietoa_sivustosta

Homès, B. 2012. Fundamentals of Software Testing. E-kirja. John Wiley & Sons, Incorporated.

International Software Testing Qualifications Board (ISTQB) 2018. Perustason sertifikaattisisältö. Viitattu 14.4.2022. https://tivia-jasenyhdistykset.fi/fistb-testi/wp-content/uploads/sites/30/2020/12/CTFL-2018-Sertifikaattisisalto-20181010-1-Valmis.pdf

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. E-kirja. Jyväskylä: Docendo.

Myers, G., Sandler, C., Badgett, T. 2011. The Art of Software Testing. E-kirja. John Wiley & Sons, Incorporated

Nayyar, A. 2019. Instant Approach to Software Testing. E-kirja. BPB Publications.

Kuviot