Testausraportti

KotKot

Helsinki 14.12.2008

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti ($9 + 1$ op)

Projektiryhmä

Tuomas Puikkonen
Matti Seise
Paula Mäenpää
Olga Karmanov
Jonne Kohvakka

Asiakas

Heikki Lokki

Johtoryhmä

Sampo Yrjänäinen

Kotisivu

http://www.cs.helsinki.fi/group/kotkot/

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
1.0	14.12.2008	Valmis testausraportti

Sisältö

1 Johdanto **1**

1.1 Dokumentin tarkoitus 1

1.2 Kuvaus tuotteesta 1

1.3 Käytetyt ohjelmointikielet ja niiden testauspiirteet 1

2 Sanasto **1**

3 Yksikkötestaus **4**

3.1 Lähestymistapa 4

3.2 Testattavat kohdat 4

3.3 Hyväksymiskriteerit 5

3.4 package kotkot_mokkula 5

3.4.1 Luokka Mokkula 5

3.4.2 Luokka DALDB 6

3.4.3 Luokka Kaeli 7

3.4.4 PaivitaTiedot 7

4 Integrointitestaus **7**

4.1 Lähestymistapa 8

4.2 Testattavat kohdat 8

4.3 Hyväksymiskriteerit 9

4.3.1 Vanhojen pesätietojen lisääminen tiedostosta tietokantaan 9

5 Järjestelmätestaus **9**

5.1 Järjestelmään kirjautuminen 10

5.2 Käyttäjän lisääminen 10

5.3 Nettilomakkeen tallentaminen, museovirkailija 10

5.4 Nettilomakkeen käsittely, museovirkailija 10

5.5 Esitäytettyjen lomakkeiden tulostaminen rengastajalle 10

5.6 Seura-aineiston tulostaminen 11

5.7 Tietokannan hallinta, muoseovirkailija 11

5.8 Testimuotoisen datan siirtäminen tietokantaan 11

5.9 Nettilomakkeen täyttäminen, rengastaja 11

6 Testausaikataulu **11**

1 Johdanto

Testaus on yksi tärkeimmistä ohjelmistokehitysprojektin työvaiheista. Laadukkaalla, oikein ajoitetulla testauksella voidaan varmistaa, että projektin toiminnalliset ja laadulliset vaatimukset saavutetaan.

1.1 Dokumentin tarkoitus

Tämä dokumentti käsittelee KotKot-ohjelmistotuotantoryhmän petolintujen pesien “Haukka“-pesienseurantajärjestelmän testausta. Se määrittelee projektissa käytettävän testausprosessin, käytettävät menetelmät, testauksen kattavuuden ja testauksen raportoinnin. Testausraportin tavoitteena on toimia testausvaiheessa testauksen ohjeistuksena sekä ohjelman mahdollisen jatkokehityksen aikana testauksen toistettavuuden ja suunnittelun apuna. Tarkoitus on, että Haukka-järjestelmä toimii projektin päätyttyä virheettömästi ja toteuttaa vaatimusdokumentissa määritellyt vaatimukset. Valitettavasti, toteutusvaiheen ongelmien takia, järjestelmän kattava testaus jäi toteutumatta ja tässä raportissa kuvatut testausohjeet ja menetelmät jäivät teorettiselle tasolle.

1.2 Kuvaus tuotteesta

Projektin tarkoituksena on suunnitella ja toteuttaa petolintujen pesätarkastusten yhteydessä lomakkeille kerättyjen tietojen tallettamiseen ja käyttöön soveltuva tietokanta sekä käyttöliittymä. Järjestelmä on kuvattu tarkemmin suunnitteludokumentissa. Projekti on jatkoa aikaisempien työryhmien ohjelmistotuotantoprojekteille: Sääksi ja Merikotka-järjestelmille.

Järjestelmä toteutetaan ja testataan sen toteutusympäristössä eli Tomcat servlet -ympäristössä tietojenkäsittelytieteen laitoksen koneella db.cs.helsinki.fi ja järjestelmä on tarkoitettu käytettäväksi Firefox-selaimen kautta (Firefox versio 2.0.0 tai uudempi).

1.3 Käytetyt ohjelmointikielet ja niiden testauspiirteet

Ohjelmointikielenä tuotantoprojektissa käytetään Java-ohjelmointikieltä. Muita toteutuksessa käytettyjä kieliä ovat XHTML, CSS, FreeMarker Template Language, JavaScript ja Oraclen SQL-lauseet. Oliokielenä Java tuo testaamiseen omat haasteensa, jotka liittyvät lähinnä periytymiseen sekä dynaamiseen sidontaan. Ominaisuuksien periytyminen ja polymorfismi voivat hankaloittaa suoraviivaista rakenteellista testaamista. Olioiden väliset yhteydet voivat tuottaa monimutkaisia rakenteita, joiden testaaminen on hankalaa.

XHTML- ja CSS-koodin testauksen apuna on mahdollista käyttää yleisesti saatavilla validaattoreita, jotka tarkistavat koodin syntaksin olevan spesifikaation mukainen.

2 Sanasto

Sanastoon on koottu KotKot-ohjelmistotuotantoprojektissa toteutettavaan ohjelmistoon ja sen testaamiseen liittyviä termejä.

Arvoalueanalyysi

Testitapausjoukon rajaaminen järkeviksi osa-arvoalueiksi, joiden reunoilta valitaan testauksessa käytettävät arvot.

Bottom-up -strategia

Integrointitestauksen strategia jossa integrointi aloitetaan yksiköistä, joita integroidaan yhteen kunnes koko järjestelmä on koottu.

CSS (Cascading Style Sheets)

Tyylimäärittelyt määrittelevät kuinka dokumentti esitetään ruudulla ja tulosteissa. Mahdollistavat värien, fonttien, asemoinnin jne. lisäämisen HTML-dokumentteihin. CSS-määritykset voidaan lisätä suoraan dokumenttiin tai ne voidaan mää-rittää css-tyylitiedostossa.

Freemarker-template

FTL (FreeMarker Template Language) -kielellä luotuja HTML-templaatteja.

Haukka-järjestelmä, järjestelmä

Toteutettava järjestelmä, joka sisältää käyttöliittymän, tietokannan ja näiden välillä olevat toiminnallisuudet.

Haaraumakattavuus

Testausmenetelmä, jolla pyritään käymään läpi testiyksikön jokainen haaraumakohta (esim. if-lauseiden vaihtoehdot).

XHTML (Extensible HyperText Markup Language)

Rakenteellinen merkkauskieli, jolla webin sivut kirjoitetaan. JavaScript on sulautettu XHTML-sivujen sisälle.

Integrointitestaus

Testausvaihe, jossa toimivia yksiköitä liitetään toisiinsa. Rajapintojen testaus.

Java

Projektissa käytettävä ohjelmointikieli.

JavaScript

Selaimessa suoritettava skriptauskieli, jolla voidaan toteuttaa yksinkertaisia dynaamisia elementtejä HTML-dokumentissa.

Java-servlet

Java-kielinen ohjelma web-palvelimella, jolla tuotetaan dynaamisia web-sivuja.

Java-luokka

Java-luokka kuvaa olion rakenteet (attribuutit) ja käyttäytymisen (metodit). Luokat ovat yksikkötestauksen kohteina.

JDBC (Java Database Connectivity)

Ohjelmointirajapinta, joka mahdollistaa pääsyn lähes mihin tahansa tietolähteeseen Java-ohjelmointikielestä.

JUnit

Testikehys Java-luokkien yksikkö- ja integrointitestaukseen.

Järjestelmä

ks. Haukka-järjestelmä

Järjestelmätestaus

Ohjelmistotuotteen (järjestelmän) testaus kokonaisuutena.

Kattavuus

Luku, joka kertoo kuinka hyvin suoritetut testit ovat testanneet testatun yksikön rakennetta.

Kattavuuskriteeri

Testauksen kattavuuden minimiarvo.

Käyttötapaus Käyttötapauksia käytetään toimijan ja järjestelmän välisen vuorovaikutuksen kuvaamiseen. Toimija voi olla henkilö, toinen tietojärjestelmä jne. Käyttötapauksessa kuvataan toimijan tavoite jonkin päämäärän saavuttamiseksi, ja mahdollisimman yksityiskohtaiset tiedot tilanteen taustoista.

Lasilaatikkotestaus (White-box testing)

Lasilaatikkotestausta kutsutaan rakenteelliseksi testausmenetelmäksi. Se perustuu testattavan kohteen rakenteen tuntemiseen. Mahdollisia virhealttiita paikkoja voidaan arvioida tarkastelemalla koodia ja keskittää testaus näihin kohtiin. Testien tuloksia verrataan ja analysoidaan suhteessa odotettuihin tuloksiin.

Lausekattavuus

Lausekattavuudella tarkoitetaan testeissä läpikäytyjen yksikön lauseiden lukumäärä/yksikön kaikkien lauseiden lukumäärä. Lausekattavuutta käytetään yksikkötestauksessa kertomaan paljonko yksikön lauseista on testattu.

Luokkatestaus

ks. yksikkötestaus.

Metodi

Java-luokan sisällä oleva aliohjelma, jota voidaan kutsua itse luokasta tai toisesta Java-luokasta.

Mokkula-komponentti

Kertakäyttöiseksi tarkoitettu tiedon siirtoväline, jolla saadaan olemassa olevat pesätiedot siirretyksi uuteen tietokantaan.

Mustalaatikkotestaus (Black-box testing)

Mustalaatikkotestauksessa testataan testattavan komponentin toiminnallisuutta. Siinä komponentin sisäinen rakenne (esimerkiksi koodi ja tietorakenteet) ei ole näkyvissä. Testaus perustuu syötteiden ja sovelluksen antamien tulosten analysointiin. Saatuja tuloksia verrataan odotettuihin, jolloin voidaan päätellä sovelluksen toimivuus testitapauksessa.

Servlet

ks. Java-servlet.

SQL (Structured Query Language)

Monien tietokantajärjestelmien käyttämä kyselykieli.

Yksikkö

Koodin selkeästi rajautuva kompakti osio, esim. luokka tai metodi. Myös tietokan-nan luonnissa

käytetyt SQL-lauseet muodostavat yksiköitä.

Yksikkötestaus

Yksiköiden testaamista (ks. yksikkö).

3 Yksikkötestaus

Yksikkötestauksessa testauksen kohteena ovat pienimmät loogiset ohjelmistonosat. Projektissa tällaisia ovat esimerkiksi Java-luokat, metodit ja muut toiminnallisuutta sisältävät yksiköt, kuten Freemarket-templatet. Myös pieniä vahvasti toisiinsa sitoutuneita luokkia voidaan testata yksikkötestauksen menetelmin. Tavoitteena on se, että kaikki yksiköt tulee testattua riittävän kattavasti ja todennettua, että ne toteuttavat niiltä vaaditut tehtävät.

Yksikkötestaus, koko sen laajuudessaan, on suoritettu vain Mokkula-luokalle, muiden yksikköiden testaustulokset puuttuvat.

3.1 Lähestymistapa

Yksikkötestauksessa testaustapaukset perustuvat ohjelman koodiin ja sen rakenteisiin. Java-koodi testataan yksikkövaiheessa vähintään 80% lausekattavuudella, ja mahdollisuuksien ja aikataulun puitteissa myös mahdollisimman haaraumakattavasti. Apuvälineenä käytetään JUnit testaussovelluskehitystä. JUnit:in keskeinen ajatus on se, että ohjelmiston jokainen metodi testataan sitä vastaavalla testimetodilla. Yksikkötestaus on sekä luokan kirjoittajan että testausvastaavan vastuulla.

XHTML-, CSS-, FTL- ja SQL-koodi käydään läpi 100% lausekattavuudella.

Tietokanta-ajoissa käytetyt scriptit toteutetaan Javalla, ja testataan kuten muukin Java-koodi. Lisäksi scripteihin sisältyvät SQL-lauseet käydään läpi 100% lausekattavasti.

XHTML-koodin sisällä käytetty JavaScript käydään läpi 100% lausekattavuudella.

3.2 Testattavat kohdat

Kaikki dataa muokkaavat metodit testataan sekä kelvollisilla että kelpaamattomilla syötteillä. Mikäli metodin käyttäytyminen voi muuttua riippuen sen tilasta, tulee testaus toistaa eri tiloissa. Syötteiden optimoimiseen käytetään arvoalueanalyysin periaatteita, jossa testiparametreiksi valitaan metodin ulkoisten tai sisällä olevien rajoitusten lähellä olevat arvot, sekä jokin arvo luokan arvoalueen “keskeltä“. Lisäksi kelpaamattomilla syötteillä testataan, että metodi osaa käsitellä niitä tarkoituksenmukaisella tavalla ja että poikkeusten käsittely on kunnossa. Arvoalueanalyysissa testin syötearvoalue ositetaan osa-arvoalueiksi seuraavasti:

- NULL
- minimiarvo-1
- täsmälleen minimiarvo
- minimiarvon ja maksimiarvon väliltä
- täsmälleen maksimiarvo

- maksimiarvo+1

Taulukoista testataan osa-arvoalueet:

- indeksi negatiivinen, nolla, maksimi, maksimi+1, tyhjä taulukko

Merkkijonoista testataan osa-arvoalueet:

- tyhjä merkkijono, merkkijonon pituus 0.
- merkkijonon pituus 1.
- arvo merkkijonon keskivaiheilta
- merkkijonon pituus maksimi.
- merkkijonon pituus ylittää maksimin.
- erikoismerkit merkkijonoissa.

Lisäksi tulisi varmistaa, että kaikkiin koodiriveihin päästään jostain käsiksi ja poikkeuksien käsittely on kunnossa.

3.3 Hyväksymiskriteerit

Testattu yksikkö hyväksytään yksikkötestausvaiheen lopussa, kun seuraavat vähimmäisvaatimukset on saavutettu:

- Kaikki sen palvelut on onnistuneesti testattu määrätyillä osa-arvoaluilla
- Kaikki tilat on testattu
- Kaikki määritellyt poikkeustilanteet on testattu
- Lausekattavuus on vähintään 80%, yksinkertaisimmissa yksiköissä 100%

Alla esitetään yksikkötestauksen testitapapaukset.

3.4 package kotkot_mokkula

3.4.1 Luokka Mokkula

private static Integer checkInt(String substring)

Tarkoitus	Testataan, että metodi muuttaa String-tyyppisen arvon Integer-arvoksi
Odotettu tulos	Palauttaa merkkijonon sisältö numerona tai null-arvona
Syöte	merkkijono, luku.
Tulos	OK

public static void parsiRivi(String line)

Tarkoitus	Testataan, että metodin avulla rivin sisältö jaetaan pesälomakekenttiin.
Odotettu tulos	Pesälomakkeen muuttujat saavat arvonsa
Syöte	Merkkijonorivi(80 merkkiä), lyhennetty merkkijonorivi
Tulos	OK

public static boolean testaaRivi(String line)

Tarkoitus	Testataan, että metodi tarkistaa lähtötiedoston rivit: ne eivät ole tyhjiä, eivät alkaa “#“, eikä rivin pituus ylittää 80 merkkiä
Odotettu tulos	Jos rivin muoto on oikea, kutsutaan parsiRivit() metodia ja sen jälkeen tehdään vielä tarkastuksia pesälomakkeen muuttujille.
Syöte	Merkkijonorivi(80 merkkiä), lyhennetty merkkijonorivi, tyhjä rivi, “#“ alkava rivi, “väärät“ syötteet vuosikymmennelle (null arvo), koordinaateille, lajin nimelle, pesimistulokselle
Tulos	OK
Huom.	Jos tiedoston ensimmäinen merkki ei ole luku eikä “#“ vuosikymmennen tarkistuksen virheilmoitus ei tulostu; koordinaattien leveys voi olla 00000(?), pesimistuloksen “väärät“ tunnukset menevät läpi;

private static boolean lajiLoytyy(String laji)

Tarkoitus	Testataan, että parametrina annettu lajinimi vastaa etukäteen määriteltyihin
Odotettu tulos	Lajinimi on oikein tai ei
Syöte	Merkkijono, numerot
Tulos	OK

private static void epaonnistuneetRivitLokiin

Tarkoitus	Testataan, että virheelliset rivitiedot viedään lokitiedostoon
Odotettu tulos	Virheelliset rivit on viety lokitiedostoon “virhelliset.txt“
Syöte	Metodin parametreja: rivin numero, rivin sisältö, tiedoston nimi
Tulos	OK

3.4.2 Luokka DALDB

public Connection luoYhteys()

Tarkoitus	Testataan, että yhteyden muodostaminen tietokantaan onnistuu
Odotettu tulos	Jos tietokanta-ajuri puuttuu tai tietokantaserveri ei toimi, saadaan virheilmoitus, muuten yhteys on muodostettu
Syöte	Metodin kutsu
Tulos	OK

public Connection suljeYhteys()

Tarkoitus	Testataan, että yhteyden sulkeminen onnistuu
Odotettu tulos	Yhteys tietokantaan suljettu tai jos jotain meni pieleen, saadaan virheilmoitus
Syöte	Metodin kutsu
Tulos	OK

3.4.3 Luokka Kaeli

public static File showDialogAndGetFile(String path)

Tarkoitus	Testataan, että metodi avaa tiedoston valintaikkunan ja valitsee tiedoston
Odotettu tulos	Valintaikkuna avautuu ja käyttäjä pystyy valitsemaan tiedosto
Syöte	Metodin kutsu
Tulos	OK

3.4.4 PaivitaTiedot

public static boolean tarkistaHeruux()

Tarkoitus	Testataan, että metodi suorittaa pesälomeketiedoille tarkastukset ennen kuin uudet pesä- ja tarkastustiedot lisätään tietokantaan.
Odotettu tulos	Metodi tarkistaa tietokannasta olemassa olevien pesien koordinaatit ja pesä_Id, jos pesää ei ole - lisätään uusi pesä. Tarkastuskäynti-tapauksessa: metodi tarkistaa päivämäärän, pesä-Id ja pesäkoordinaatit, jos kyseessä on uusi tarkastuskäynti, lisätään se tietokantaan
Syöte	Mokkula - luokan pesälomaketiedot (oikeat parametrit sekä kelvottomat)
Tulos	OK

public static void lisaaPesa()

Tarkoitus	Testataan, että uuden pesän lisääminen onnistuu
Odotettu tulos	Tietokantaan on lisätty uusi pesä ja tarkastuskerta
Syöte	Metodin kutsu, Mokkula-luokan tiedot
Tulos	OK

public static void lisaaTarkastuskerta()

Tarkoitus	Testataan, että tarkastuksen tiedot lisätään tietokantaan
Odotettu tulos	Tarkastustiedot ja tarkastusyhteenvetotiedot on lisätty tietokantaan
Syöte	Metodin kutsu, Mokkula-luokan tietojen perusteella
Tulos	OK

public static void avaa()

Tarkoitus	Testataan, että yhteys tietokantaan on avattu
Odotettu tulos	Yhteys tietokantaan on muodostettu tai saadaan virheilmoitus
Syöte	Metodin kutsu
Tulos	OK

Mokkula-paketin Java-luokkien yksikkötestauksessa on käytetty staattisia testeja, lasilaatikkotestausta ja JUnit-kehitystä. JUnit:n käyttö oli vain suuntaa antava, oikeiden testiluokkien virittämiseen aikaa ei riittänyt.

4 Integrointitestaus

Integrointitestauksessa testataan yksiköiden (Java-luokat, Freemarker-templatet) väliset rajapinnat. Integrointitestaukseen luokat etenevät kun ne ovat läpäisseet luokkatestauksen, eli on todettu, että ne toimivat oikein yksinään. Integrointitestauksen tarkoitus on varmistaa komponenttien

toiminta yhteistyössä toistensa kanssa ja paikantaa mahdollisia virheitä komponenttien välisistä rajapinnoista. Integrointitestaus on tyypillisesti black box -testausta, joka ei perustu komponentin sisäiseen rakenteeseen, vaan rajapintojen kautta välittyvien syötteiden ja tulosten analysointiin.

4.1 Lähestymistapa

Integrointitestaus tullaan tekemään ns. bottom-up -strategialla jossa yksikkötestattuja osia integroidaan toisiinsa yksi kerrallaan kunnes kaikki yksiköt on integroitu järjestelmään. Integrointitestauksessa testataan järjestelmän komponenttien toimintaa keskenään, sekä niiden toimintaa tietokannan kanssa. Testauksessa keskitytään yksiköiden rajapintoihin. Testausprosessi etenee seuraavasti:

1. Selvitetään, mitä rajapintojen palveluja integroidut osat vaativat toisiltaan ja tarjoavat toisilleen.
2. Tehdään jokaiselle palvelulle arvoalueanalyysi ja valitaan sen perusteella testisyötteet.
3. Käytetään rajapintaa annetuilla testisyötteillä kutsujan kautta.

Integrointitestauksessa ei pitäisi tulla ilmi muuta kuin rajapintaongelmia, sillä kukin yksikkö on jo testattu erikseen ja siten varmistettu, että ne toimivat oikein. Vaikka kaikki yksiköt toimisivat oikein, voi tulla ongelmia niiden yhteistyössä. Integrointitestauksessa mahdollisesti ilmeneviä ongelmia voivat olla muun muassa:

- kutsuja ymmärtää rajapinnan väärin
- kutsuttava palauttaa väärin tulkitun arvon
- rajapintaa käytetään väärällä tavalla
- kutsuja voi odottaa palvelulta sivuvaikutuksia, jotka eivät toteudu, tai kutsuttava aiheuttaa sivuvaikutuksia, joita kutsuja ei odottanut
- kutsuja voi aiheuttaa poikkeustilanteen, johon ei oltu varauduttu.
- kutsuja ja kutsuttava voivat ymmärtää palvelun syötteiden arvoalueet eri tavoin.

Kahden yksikön integrointitestaus on valmis, kun kaikki yksiköiden välinen yhteistyö on testattu, mukaan lukien virheiden ja poikkeusten testaus. Integrointitestausvaihe päättyy, kun kaikki yksiköt on testatusti integroitu yhteen.

4.2 Testattavat kohdat

Integrointitestaus uusille komponenteille pyritään tekemään heti kun uusi komponentti on valmis, eli se on ohjelmoitu, dokumentoitu ja yksikkötestattu. Testausta varten on määriteltävä mihin kaikkiin järjestelmän yksiköihin testattava yksikkö on suoraan tai välillisesti yhteydessä ja mitä rajapintojen palveluja integroitavat palvelut pyytävät toisiltaan ja tarjoavat toisilleen. Testisyötteillä käytetään testattavaa rajapintaa kutsujan kautta. Kun osien liittyminen toisiinsa rajapintojen

tarjoamien palveluiden kautta on selvitetty, valitaan sopivat testisyötteet ja rajapinnan toimivuus testataan.

Suuri osa toteutusta tulee olemaan Template-tiedostojen toteuttaminen, joiden yhteydet muuhun järjestelmään tulee myös integrointitestata. Järjestelmän tietokantayhteydenottojen kattava integrointitestaaminen voi olla hankalaa, joten tietokantaan lähetettävien hakujen validointiin tulee kiinnittää erityistä huomiota

4.3 Hyväksymiskriteerit

Kun yksiköiden välinen toiminta (niiden rajapinnat) on osoitettu toimiviksi ja kaikki komponentit on integroitu yhteen on integrointitestaus suoritettu. Tällöin voidaan ryhtyä testaamaan koko järjestelmää.

Haukka-järjestelmän integrointitestaus on Mokkulan toiminta tietokannan kanssa. Mokkulan tehtävä on siirtää pesiin ja tarkastuksiin liittyvä tiedot tietokantatauluihin, samalla tarkistaen, että tiedot ovat oikeassa muodossa eikä toistu.

4.3.1 Vanhojen pesätietojen lisääminen tiedostosta tietokantaan

Testin kuvaus	Testi alkaa yrittämällä avata tiedosto,jossa siirrettäväksi tarkoitetut pesätiedot sijaitsevat.
Testin tulos	Jos pesätiedot ovat oikeassa muodossa ja tietokantayhteys on muodostettu - viedään tiedot tietokantaan. Jos tietokantayhteys ei toimi - saadaan virheilmoitus, jos tiedoston rivit eivät vastaa ehtoja - luodaan lokitiedosto ja viedään virheelliset rivit sinne, oikeat rivitiedot aina päättyvät tietokantaan (jos nitä ei ole sielä entuudestaan), jos tiedostoa ei ole olemassa - saadaan virheilmoitus
Havaittujen virheiden kuvaus	“Pesintä onnistui ainakin: lentopoikaisiin“-muuttujan arvo muuttuu kun viedään se tietokantaan - korjattu; Suunnitteludokumentin tietokantataulujen kuvaus: taulu Pesa_lintulaji, attribuutti “lajikoodiID“:n tyypiksi on laitettu NUMBER - tietokannassa se on VARCHAR; taulu Vuosi, attribuutti vuosiluvu:n kuvauksessa on kerrottu, että se on nelinumeroinen vuosiluku - tietokannassa se on yksinumeroinen.
Tulos	OK

5 Järjestelmätestaus

Järjestelmätestaus tehdään integrointitestauksen jälkeen. Järjestelmä testataan kokonaisuutena, johon kuuluvat ohjelmiston lisäksi tietokanta, laitteisto ja järjestelmän kanssa yhteistyössä toimivat ulkoiset ohjelmat kuten selain. Järjestelmä testataan käyttöliittymän kautta. Järjestelmän testaus ja testitapausten valinta perustuu käyttötapauksiin ja käyttäjävaatimuksiin. Kustakin vaatimuksesta kirjataan täyttyykö vaatimus. Jos vaatimus ei täyty, täytyy kirjata miten vaatimuksen täyttymättömyys käy ilmi. Kirjataan myös mikäli jotain vaatimusta ei voida havaita tai testata. Testitapaukset kirjoitetaan siten, että niiden avulla voi yksiselitteisesti määritellä täyttääkö järjestelmä annetun

vaatimuksen vai ei, eli voiko määritelty operaatiota tehdä järjestelmällä vai ei. Testitapauksessa käsitellään yleensä vain yksi tapa suorittaa operaatio. Jos saman operaation voi tehdä useammalla tavalla, on vaihtoehtoiset tavat testattu ainoastaan, mikäli niistä on testausdokumentissa maininta.

Järjestelmätestauksen tavoite on selvittää:

- Tekeekö järjestelmä ne toiminnot, jotka on vaadittu?
- Vastaako järjestelmä sille asettettuja laadullisia vaatimuksia?
- Toimiiko järjestelmä suunnitellussa ympäristössä?
- Pysyykö järjestelmä pystyssä ja onko se vikasietoinen?

Vaikka järjestelmätestausvaiheeseen ei koskaan päästy, alla on mahdolliset järjestelmätestauksen testaustapaukset:

5.1 Järjestelmään kirjautuminen

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.2 Käyttäjän lisääminen

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.3 Nettilomakkeen tallentaminen, museovirkailija

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.4 Nettilomakkeen käsittely, museovirkailija

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.5 Esitäytettyjen lomakkeiden tulostaminen rengastajalle

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.6 Seura-aineiston tulostaminen

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.7 Tietokannan hallinta, muoseovirkailija

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.8 Testimuotoisen datan siirtäminen tietokantaan

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

5.9 Nettilomakkeen täyttäminen, rengastaja

Testin kuvaus	xxx
Testin tulos	xxx
Havaittujen virheiden kuvaus	xxx

6 Testausaikataulu

Yksikkötestausta tehdään käytännössä koko toteutuksen ajan. Integrointitestaus aloitetaan kun yksikkötestauksessa on hyväksytty Järjestelmätestaus voidaan aloittaa integrointitestauksen valmistuttua, järjestelmätestauksesta saatetaan joutua palaamaan vielä integrointitestaukseen. Hyväksymistestauksen suorittaa asiakas valmiille ohjelmistolle.

Testauksessa mahdollisesti löytyvät virheet ja puutteet, joita ei kyetä korjaamaan projektin kuluessa dokumentoidaan ylläpitodokumenttiin. Tähän dokumentointiin kuuluu ainakin virhetilanteen kuvaus, virheen vakavuuden arviointi sekä ohjeet virhetilanteen välttämiseen ja siitä toipumiseen.