Testaussuunnitelma

Ohjelmistotuotantoprojekti Nero

Helsinki 5.11.2004

Ohjelmistotuotantoprojekti

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

581260 Ohjelmistotuotantoprojekti (6 ov)

Projektiryhmä

Johannes Kuusela
Jyrki Muukkonen
Teemu Sjöblom
Ville Sundberg
Osma Suominen
Timi Tuohenmaa

Asiakas

Reijo Siven
Juhani Haavisto

Johtoryhmä

Juha Taina
Mikko Olin

Kotisivu

`http://www.cs.helsinki.fi/group/nero/`

Versiohistoria

Versio	Päiväys	Tehdyt muutokset	Kirjoittaja
0.1	17.10.2004	Ensimmäinen luonnos	Jyrki Muukkonen
0.2	24.10.2004	Tarkennuksia ja korjauksia	Jyrki Muukkonen
0.3	27.10.2004	DbUnit ja EMMA	Jyrki Muukkonen
0.4	28.10.2004	Viittet muihin dokumentteihin	Jyrki Muukkonen
0.5	30.10.2004	Järjestelmätestauksen tarkennuksia	Jyrki Muukkonen
0.6	31.10.2004	Rasitustestaus	Jyrki Muukkonen

Sisältö

1 Johdanto

Tässä dokumentissa esitellään ohjelmistotuotantoprojekti Neron testaussuunnitelma. Neroprojekti on Helsingin yliopiston tietojenkäsittelytieteen laitoksen syksyn 2004 ohjelmistotuotantoprojekti, jonka tarkoituksena on tuottaa graafisella käyttöliittymällä varustettu tilanhallintasovellus. Projekti on jatkoa kesällä 2004 tehdylle Rooma-projektille.

Testauksen tavoitteena on varmistaa ohjelmiston virheettömyys. Ohjelmistoa testataan neljällä eri tasolla. Erilliset komponentit testataan yksikkötestein. Komponenttien toteuttamat yhtenäiset rajapinnat testataan integraatiotestauksessa. Rasitustestauksessa testataan ohjelman toiminnallisuus suurella tietomäärällä. Järjestelmätestauksessa käydään läpi kaikki vaatimusdokumentissa[Ner04a] kuvatut käyttötapaukset. Hyväksymistestauksessa asiakas validoi ohjelmiston toiminnallisuuden.

2 Testausstrategia

Koska projektin aikataulu on tiukka, pyritään testejä automatisoimaan mahdollisimman paljon. Yksikkö- ja integraatiotestit voidaan automatisoida kokonaan. Järjestelmätestaus sekä hyväksymistestaus ovat suurilta osin manuaalisia. Yksikkö-, integraatio- ja järjestelmätestauksen lausekattavuus tullaan mittaamaan. Koska Rooma-projektin Java-luokkia ei tulla käyttämään hyväksi, ei myöskään niihin liittyviä testejä voida käyttää sellaisenaan.

2.1 Testiaineisto

Toisin kuin Rooma-projektissa, käytössämme on laaja ja oikeaa dataa sisältävä testitietokanta. Rooma-projektin testaussuunnitelmassa[Roo04a] todettiin, että suoritettavat testit eivät saisi tehdä tietokantaan pysyviä muutoksia. Näin voidaan varmistaa, että tietyt testeissä hyödynnettävät oletukset ovat voimassa testien lähtötilanteissa, ja testien tekemät tietokantamuutokset eivät jää voimaan testien päätyttyä. Rooma-projektissa tämä ei testausraportin[Roo04b] mukaan täysin onnistunut, minkä johdosta osa ohjelmiston virheistä löydettiin liian myöhäisessä vaiheessa.

2.2 Testauksessa käytettävät työkalut

Testien automatisointiin tullaan käyttämään Rooma-projektin tapaan JUnit-testaustyökalua[1]. JUnitin avulla voidaan automatisoida sekä Java-luokkien yksikkötestit että ylemmän tason rajapinnoille suoritettavat integraatiotestit.

Tietokantaa tarvitsevien automatisoitujen testien apuna käytetään JUnitin laajennosta, DbUnitia[2]. DbUnitin avulla tietokanta voidaan asettaa tiettyyn tilaan ennen testejä. Testien jälkeen tietokannan alkuperäinen tila palautetaan. *DbUnit-testejä ei saa ajaa yhtäai-*

[1] `http://www.junit.org/`
[2] `http://dbunit.sourceforge.net/`

kaisesti samalla tietokannalla. Ennen DbUnit-testien ajamista on siis varmistettava, ettei kukaan muu ole tekemässä samaa operaatiota.

Testien kattavuutta mitataan EMMA-työkalulla[3]. EMMAn avulla voidaan tarkistaa kuinka monta prosenttia sen kautta suoritetuista luokista, metodeista, lohkoista sekä lauseista käytiin läpi kyseisellä suorituskerralla. Kaikkien testien yhteisen luokka- ja metodikattavuuden tulee olla 100%, lohko- ja lausekattavuuden puolestaan 90%.

Suuri osa projektissa tuotettavasta uudesta toiminnallisuudesta liittyy käyttöliittymäkomponentteihin, joiden automaattinen testaaminen voi olla vaikeaa. Toinen manuaalista testausta vaativa osa-alue on kopiointiosajärjestelmä. Kopiointiosajärjestelmän oletetaan olevan jo tarpeeksi hyvin testattu Rooma-projektin puitteissa. Kopiointiosajärjestelmän testausta ei kuitenkaan ole erikseen mainittu Rooma-projektin testausraportissa.

2.3 Työnjako

Testausvastaava (Jyrki Muukkonen) luo tarvittavan testiaineiston ja rungon sekä normaaleille JUnit-testeille että DbUnit-testeille.

Yksikkötestien toteuttamisessa ei ole varsinaista tiukkaa työnjakoa. Testejä kirjoitetaan sitä mukaa kuin se vain on mahdollista.

Normaalit JUnit-testit tulee ajaa jokaisen muutoksen jälkeen. Virhetilanteiden raportointi ja korjaaminen on muutoksen tekijän vastuulla.

DbUnit-testien ajamisessa on otettava huomioon, ettei kyseisiä testejä ajeta yhtäaikaisesti. Tämän vuoksi niiden ajaminen on pääasiassa testausvastaavan vastuulla. DbUnit-testit tullaan ajamaan päivittäin tarvittavien testiaineistojen luonnin jälkeen.

Lopullinen järjestelmätestaus suoritetaan yhdessä tiistaina 30.11.2004.

Testausvastaava kerää kaikkien testien tulokset ja kirjaa ne erilliseen testausraporttiin.

3 Testauksen vaiheet

Testaus jakautuu siis neljään eri osa-alueeseen: yksikkötestaus, integraatiotestaus, järjestelmätestaus sekä hyväksymistestaus. Automatisoituja testejä tullaan käyttämään koko ajan ohjelmistoa toteutettaessa. Manuaalisia testausvaiheita käydään läpi mahdollisimman paljon, jotta myös niiden tulokset olisivat tarpeeksi luotettavia. Lopuksi jokainen testausvaihe raportoidaan erillisessä testausraportissa.

3.1 Yksikkötestaus

Jokainen toteutettu Java-luokka testataan yksikkötestein. Yksikkötestit kirjoitetaan mahdollisiman valmiiksi jo luokkien suunnitteluvaiheessa. Näin kyseistä luokkaa voidaan tes-

[3] `http://emma.sourceforge.net/`

tata koko toteutusvaiheen ajan, ja luokka voidaan todeta valmiiksi sen läpäistyä kaikki yksikkötestinsä. Koska yksikkötestit kirjoitetaan suurimmaksi osaksi ennen varsinaista koodausvaihetta, on testaus niin sanottua mustalaatikkotestausta.

Yksikkötesteissä metodit testataan kaikilla kyseisen testattavan metodin parametrien ekvivalenssiluokilla. Ekvivalenssiluokilla tarkoitetaan syöteavaruuden jakamista toiminnallisuuksittain. Ekvivalenssiluokkia voivat olla esimerkiksi liian pieni arvo, pienin mahdollinen arvo, lailliselta väliltä oleva arvo, suurin mahdollinen arvo sekä liian suuri arvo. Oleellisimpia testisyötteitä ovat eri ekvivalenssiluokkien rajoilla sijaitsevat arvot, eli esimerkiksi pienin mahdollinen arvo sekä sen viereiset arvot.

Jokaisen testin koodin yhteyteen kommentoidaan kyseisen testin versiohistoria, jolloin siihen liittyviä muutoksia on helppo tarkastella. Testausraporttiin kirjataan jokaisen testin kuvaus sekä tieto sen onnistumisesta taulukossa 1 esitetyllä tavalla.

Testiluokka	TestEsimerkki
Testimetodi	testEsimerkkiPoistaJotain()
Kuvaus	testEsimerkkiPoistaJotain()-testimetodi yrittää poistaa järjestelmästä jotain. Kutsuu Esimerkki-luokan poistaJotain()-metodia arvoilla x ja y. Tärkeää on kertoa mitä metodeita testataan ja millä syötteillä.
Lausekattavuus	Kuinka monta prosenttia testattavien luokkien *metodeista, lohkoista* sekä *koodiriveistä* testin suorittaminen kattoi.
Läpäissyt testin	Kyllä / Ei

Taulukko 1: Esimerkki yksikkötestin kuvauksesta testausraporttia varten.

3.2 Integraatiotestaus

Myös integraatiotestauksessa hyödynnetään JUnit-työkalua. Yksittäisten luokkien testaamisen sijaan integraatiotesteillä testataan yhtenäisien rajapintojen toiminnallisuutta. Integraatiotesteillä varmistetaan että ohjelmisto toteuttaa sisäisesti kaikki siltä vaaditut toiminnot.

Integraatiotestaus on tämän projektin tapauksessa suurimmaksi osaksi käyttöliittymäkomponttien käyttämien rajapintojen testaamista. Integraatiotestit toteutaan yksikkötestien tapaan mahdollisimman pian, joko heti rajapintamäärittelyn jälkeen tai osittain jopa sen yhteydessä. Integraatiotestit raportoidaan yksikkötestien tavoin taulukossa 1 esitetyllä tavalla.

3.3 Rasitustestaus

Rasitustestauksessa keskitytään ainoastaan tietokannan suorituskykyyn. Ohjelmasta tulee aina olemaan käynnissä yhtäaikaisesti korkeintaan yksi instanssi, joten tietokannan rasitustestit ajetaan ainoastaan yhdellä yhtäaikaisella tietokantayhteydellä. Näinollen rasitus-

testein mitataan ainoastaan tietokannan rivimäärien vaikutusta suorituskykyyn. Rasitustestit toteutetaan DbUnitin avulla. Rasitustesteissä käytetään ainoastaan Neron tietokantaluokan tarjoamaa rajapintaa. Tulokset raportoidaan taulukossa 2 esitetyllä tavalla.

Testiluokka	TestEsimerkki
Testimetodi	testEsimerkkiPoistaJotain()
Kuvaus	Mitä osaa tietokannasta testataan ja miten. Miten tietokanta alustettiin.
Tulokset	Rivien määrä tauluittain tietokannassa ennen testejä ja testien jälkeen. Testin tietokantaoperaatioihin käytetty yhteisaika.
Huomiot	Mahdolliset huomiot ja virheet sekä niihin liittyvät Javan ja Oraclen virheilmoitukset. Esim: `ORA-01483: invalid length for DATE or NUMBER bind variable` metodiJotain3():n kutsukerroilla kun taulussa X rivejä yli N kappaletta.

Taulukko 2: Esimerkki rasitustestin kuvauksesta testausraporttia varten.

3.4 Järjestelmätestaus

Järjestelmätestaus kattaa vaatimusdokumentissa määritellyt käyttötapaukset. Jos kyseiset käyttötapaukset ovat tarpeeksi kattavia, ja ne kyetään käymään läpi lopullisen ohjelmiston avulla, voidaan ohjelmisto todeta vaatimusten mukaiseksi.

Järjestelmätestaus suoritetaan ohjelmiston ollessa valmis, kuitenkin viimeistään 30.11.2004. Vaatimusdokumentissa kuvaillut käyttötapaukset käydään läpi ja mahdolliset ongelmat korjataan. Lopuksi kukin käyttötapaus sekä sen testaustulos dokumentoidaan testausraporttiin taulukon 3 mallin mukaisesti. Myös kunkin käyttötapauksen ongelmatilanteet ja muutokset kirjataan ylös.

Käyttötapaus	Käyttötapaus #1
Kuvaus	Käyttötapauksen kuvaus, muutama rivi tekstiä.
Lausekattavuus	Kuinka monta prosenttia koko ohjelman *luokista, metodeista, lohkoista* sekä *koodiriveistä* käyttötapauksen läpikäynti kattoi.
Tulos	"Käyttötapaus pystyttiin viemään läpi"tai "Käyttötapauksen läpikäynti epäonnistui". Lisäksi kommentteja tärkeimmistä yksityiskohdista.

Taulukko 3: Esimerkki käyttötapauksen läpikäynnin raportoinnista.

3.5 Hyväksymistestaus

Projektin loppuvaiheessa vaatimusdokumentissa määritellyt käyttötapaukset käydään läpi myös asiakkaan kanssa. Mikäli hyväksymistestauksessa ilmenee ongelmia, dokumentoi-

maan ne mahdollisimman hyvin ja korjataan siinä määrin kuin se on aikataulun puolesta vielä mahdollista. Hyväksymistestaus suoritetaan 7.12.2004.

Lähteet

Roo04b Jaatinen, M., Ruotsalainen, J., Räsänen, R. ja Talja, T., *Ohjelmistotuotantoprojekti Rooma, Testausraportti*. Helsingin yliopiston tietojenkäsittelytieteen laitos, 2004. URL `http://www.cs.helsinki.fi/group/rooma/`.

Roo04a Jaatinen, M., Ruotsalainen, J., Räsänen, R. ja Talja, T., *Ohjelmistotuotantoprojekti Rooma, Testaussuunnitelma*. Helsingin yliopiston tietojenkäsittelytieteen laitos, 2004. URL `http://www.cs.helsinki.fi/group/rooma/`.

Ner04a Kuusela, J., Muukkonen, J., Sjöblom, T., Sundberg, V., Suominen, O. ja Tuohenmaa, T., *Ohjelmistotuotantoprojekti Nero, Vaatimusdokumentti*. Helsingin yliopiston tietojenkäsittelytieteen laitos, 2004.