Ohjelmistotuotantoprojekti

Ryhmä Muppett

TESTAUSDOKUMENTTI

Helsinki 5.8.2008

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Ohjelmistotuotantoprojekti, kesä 2008

Projekti:

Muutos- ja korjauspyyntöjen priorisointityökalu

Asiakas:

Oodi-konsortio/ Sampo Lehtinen

Ryhmä:

Arto Chydenius
Laura Haverinen
Merja Lindén
Topi Musto
Laura Ojala
Toni Sormunen

Ohjaaja:

Marko Lehtimäki

Dokumentin versiohistoria

Versio	Päiväys	Muutokset	Muuttaja
0.0	09.06.2008	Pohja	LH
0.1	31.07.2008	Sisältö	LH
0.2	02.08.2008	Korjauksia	LO
1.0	05.08.2008	Viimeistely	LO, TS

Sisältö

1 Johdanto

1.1 Dokumentin tarkoitus

Tässä dokumentissa kuvataan Helsingin yliopiston tietojenkäsittelytieteen laitoksen ohjelmistotuotantoprojektikurssin kesän 2008 Muppett-ryhmän testaussuunnitelma ja -aikataulu. Testaussuunnitelma käsittelee projektissa käytettäviä yksikkö-, integrointi- ja järjestelmätestausmenetelmiä, ja sen tarkoituksena on antaa ohjeet testauksen toteuttamiseen.

1.2 Dokumentin rakenne

Luvussa 2 esitellään testauksen periaatteet ja siinä käytettävät ohjelmat. Luku 3 käsittelee tarkemmin yksikkötestausta, ja luku 4 tarkastelee integrointitestausta. Luvussa 5 läpikäydään järjestelmätestausta. Luku 6 käsittelee testauksen aikataulua ja luku 7 testitapauksia.

1.3 Kuvaus tuotteesta

Muppett-järjestelmä on muutos- ja korjauspyyntöjen priorisointijärjestelmä. Järjestelmä tarjoaa eri yliopistoille mahdollisuuden ilmaista mielipiteensä esitettävistä muutoksista Oodi-konsortion yliopistoille tarjoamiin järjestelmiin. Ylläpitäjälle järjestelmä tarjoaa tavan kerätä eri yliopistojen mielipiteet muutosten tärkeydestä erilaisissa äänestyksissä.

1.4 Sanasto

JUnit

Java-ohjelmien yksikkötestaukseen tarkoitettu testauskehys. Sen avulla voidaan tehdä toistettavia yksikkötestauksia.

Lasilaatikkotestaus (White-box testing)

Testit perustuvat ohjelman koodiin ja rakenteeseen. Tarkoituksena on varmistaa, että kaikki ohjelman yksittäisten metodien haaroista tulisi käytyä läpi.

Lausekattavuus

Lausekattavuus mittaa sitä, kuinka suuren osan ohjelmakoodin lauseista yksikkötestaus kattaa.

Mustalaatikkotestaus (Black-box testing)

Mustalaatikkotestauksessa testataan, että järjestelmä vastaa sen toiminnallisia ja laadullisia vaatimuksia. Testaaja testaa, toimiiko ohjelma eri syötteillä odotetulla tavalla.

Rajapinta

Eri yksiköt tarjoavat palveluita muille yksiköille rajapintojen kautta.

Regressiotestaus

Regressiotestauksella varmistetaan, että uusi koodi tai lisätty ominaisuus ei riko jo testattua koodia.

Testipaketti (Test suite)

Testitapausten joukko, joilla on yleensä yhteinen tavoite.

Tynkä

Tynkää käytetään testauksessa korvaamaan tarvittavia ohjelman osia. Tynkä toteuttaa testin vaatiman minimitoiminnallisuuden.

1.5 Käytetyt ohjelmointikielet, suositukset ja niiden testauspiirteet

Java

Järjestelmän toteutuksessa käytetään Java-ohjelmointikielen versiota 1.6. Javan piirteitä, joita testaamisessa pitää huomioida, ovat tilariippuvainen käytös, tiedon piilotus, perintä, polymorfismi ja dynaaminen sidonta, abstraktit luokat ja rajapintaluokat, poikkeusten käsittely ja rinnakkaisuus.

XHTML 1.0

Järjestelmän XHTML-sivujen toteutus noudattaa XHTML 1.0 suositusta. XHTML:n suosituksen mukainen käyttö varmistetaan käyttämällä W3C:n XHTML-validaattoria [W3C08].

CSS 2.1

Järjestelmän CSS tyylimäärittelyn toteutuksessa käytetään CSS 2.1 suositusta. CSS-tyylimääritysten suosituksen mukainen käyttö varmistetaan käyttämällä CSS-validaattoria [CSS08].

SQL

Järjestelmän SQL-kyselyjen oikeellisuus varmistetaan testiservletillä. Testiservletti ei ole osa Muppett-järjestelmää, vaan sitä käytetään ainoastaan tulosteiden tarkistamiseen. Jos tulosteet palauttavat oletetun joukon, ovat kyselyt oikeellisia.

2 Testaaminen

Testaamisen tavoitteena on varmistaa, että tuotettava järjestelmä vastaa sen määrityksiä (verifointi) ja että se tuottaa asiakkaan siltä odottamat palvelut (validointi).

Testausta tehdään järjestelmän kehittämisen eri vaiheissa ja se toimii järjestelmän kehittämisen apuvälineenä.

2.1 Testaaminen ja raportointi

Jokaisesta yksikkö- ja integrointitestauksen testattavasta kokonaisuudesta kirjoitetaan raportti, jossa ilmenee testauksen tilanne, ja sitä päivitetään testaamisen edetessä. Raporttiin kirjataan testattavat asiat, hyväksymiskriteerit, ja onko testattava kokonaisuus vielä läpäissyt hyväksymiskriteereitä, eli onko sitä jo hyväksytty. Lisäksi raportissa pitää ilmetä testauksen tekijä ja testauspäivämäärät.

2.2 Testausohjelmat

Yksikkö- ja integrointitestaus tehdään käyttäen JUnit-ohjelmaa. Testien rakenteellista kattavuutta mitataan eclEmma:lla, joka mittaa JUnit-testien lausekattavuutta. Testeissä tarvittavia muita osia varten käytetään pääasiassa JMock:lla tehtyjä vastineita tai tynkätoteutusta.

Järjestelmätestauksessa käytetään Seleniumia, joka nauhoittaa käyttäjän selainikkunassa tekemät valinnat ja syötteet. Selenium automatisoi käyttötapausten testaamisen valituilla syötteillä.

3 Yksikkötestaus

Yksikkötestauksen tarkoitus on varmistaa, että yksikkö tuottaa ne palvelut, mitä sen on määritelty tuottavan. Lisäksi yksikkötestauksella pyritään löytämään virheitä toteutuksesta.

Yksikkö tarkoittaa yleensä luokkaa, mutta se voi koostua myös useammasta luokasta. Testattavaan yksikköön kuuluvat luokat päätetään toteutuksen yhteydessä. Jokaista yksikköä kohden tehdään oma testipaketti, joka testaa yksikön tuottamat palvelut.

3.1 Lähestymistapa

Toteutusvaiheessa testattaville yksiköille tehdään testausraportti. Testausraportti-malli on dokumentin liitteenä 1. Testausraportissa määritellään:

1. Testaussuunnitelma, jossa määritellään:
    - Testausaikataulu
    - Mistä luokista testattava yksikkö muodostuu.
    - Testattavat toiminnot
2. Syötteet ja niiden odotetut tulokset
3. Testauksessa löydetyt virheet
4. Testauksien tekijät ja testauspäivämäärät
5. Hyväksymiskriteerit
6. Hyväksytty/ei-hyväksytty

Järjestelmän toteuttamiseen käytetään toimintosuuntautunutta toteutustapaa. Toimintosuuntautuneessa toteutustavassa luokkien toteutusjärjestys määräytyy käyttötapausten prioriteettien perusteella. Tämä tarkoittaa sitä, että luokkia ei välttämättä toteuteta kerralla loppuun asti, vaan luokasta toteutetaan vain toteutettavan toiminnon vaatimat osat. Tästä johtuen samaa luokkaa saattaa sekä kirjoittaa että testata useampikin ryhmän jäsen.

Yksikkötestien tekemisestä vastaa pääsääntöisesti koodin kirjoittaja. Yksikkötestauksen testausraportteja päivitetään toteutuksen edetessä, ja niistä tulee käydä selkeästi ilmi, mitkä osat on jo testattu.

Regressiotestaus tehdään aina kun luokan implementointi muuttuu. Sen tarkoitus on varmistaa, että uusi koodi tai lisätty ominaisuus ei ole rikkonut jo testattua koodia. Regressiotestaus toteutetaan suorittamalla tarvittavat yksikkötestit uudestaan, ja tarvittaessa muuttamalla aikaisempaa yksikkötestiä vastaamaan tehtyä muutosta.

Testattavan yksikön testaussuunnitelmaa tehdessä huomioidaan se, minkälaisessa roolissa se on järjestelmässä, ja minkälaisen riskin se tuottaa järjestelmälle.

Keskeisessä roolissa olevien ja suuren riskin tuottavien yksiköiden testaaminen toteutetaan tarkemmin, ja ne pyritään saamaan testattavaksi mahdollisimman aikaisessa vaiheessa. Tällaisen yksikön testaamisessa voidaan esimerkiksi kirjoittaa testitapaukset ennen koodia, ja niiden kirjoittajana voi toimia eri henkilö kuin itse yksikön koodin kirjoittajana. Sellaisiin yksikön luokkiin, joilla on tietty tila, voidaan lisäksi kirjoittaa luokkainvariantti, joka tarkastaa luokan tilan jokaisen sitä muuttavan operaation jälkeen.

3.2 Toiminnallisuuden testaus

Toiminnallisuuden testaamisella (Black-box testing) varmistetaan, että testattavan yksikön tuottamat palvelut vastaavat sen määrittelyä. Testattavaa yksikköä varten kirjoitetaan joukko JUnit-testejä, jotka muodostavat yksikön testipaketin.

Jos testattavan yksikön toiminnallisuus riippuu jostakin toisesta yksiköstä, voidaan toinen yksikkö korvata käyttämällä JMockilla toteutettua vastinetta tai tynkää.

3.3 Rakenteellisuuden testaus

Rakenteellisuuden testaaminen (White-box testing) tarkoittaa ohjelmakoodin läpikäyntiä, ja sitä testataan samoin kuin toiminnallisuutta.

3.4 Valittu kattavuus

Yksikkötestauksen kattavuutta mitataan pääasiassa lausekattavuudella. Lausekattavuus mittaa sitä, kuinka suuren osan ohjelmakoodin lauseista yksikkötestaus kattaa. Pienissä ja järjestelmän keskeisissä yksikkötestattavissa kokonaisuuksissa pyritään 100% lausekattavuuteen, mutta tilanteissa, joissa testien kirjoittamiseen kuluu turhan paljon aikaa sen tuottamaan hyötyyn nähden, voi lausekattavuus jäädä pienemmäksi. Tällaisessa tilanteessa asia perustellaan yksikön testausraporttiin. Kokonaisuudessaan toteutettavan järjestelmän yksikkötestauksen lausekattavuuden pitää olla 80%.

3.5 Testisyötteiden valinta

Testisyötteinä käytettäviä arvoja valitaan testattavan kohteen syötteiden eri arvoalueilta ja arvoalueiden rajoilta. Raja-arvoista pyritään löytämään sellaisia rajatapauksia, jotka saattavat tuottaa ongelmia.

3.6 Hyväksymiskriteerit

Yksikkötestit kirjoitetaan siten, että testin tulos on joko hyväksytty tai hylätty. Testattava yksikkö hyväksytään, kun sen testipaketin sisältämät testit on suoritettu hyväksytysti.

4 Integrointitestaus

Integraatiotestauksessa testataan integroitavien yksiköiden palveluiden yhteistyötä. Testaamisessa yksiköitä käytetään niiden tarjoamien rajapintojen kautta.

Integraatiotestauksessa ei pitäisi enää löytyä virheitä yksikön tarjoamasta rajapinnasta, vaan testaamisessa keskitytään varmistamaan, että testattavien yksiköiden yhteistyö toimii oikein.

Ennen toiminnon tai kokonaisuuden integraatiotestausta varmistetaan, että siihen kuuluvien yksiköiden yksikkötestaus on toteutettu ainakin niiden palveluiden osalta, jotka

kuuluvat integroitavaan kokonaisuuteen. Lisäksi suoritetaan palveluiden rajapintojen katselmointi, jolla pyritään löytämään rajapinnoista korkean tason puutteita ja virheitä.

4.1 Lähestymistapa

Integraatiotestaamisessa tehdään sekä rakenteellista että toimintosuuntautunutta testaamista.

Rakenteellisessa testaamisessa rakennetaan ensin kokonaisuuteen kuuluvat osat, ja sitten osat testataan yhdessä. Osia voidaan liittää toisiinsa ylhäältä alas (top-down) - tai alhaalta ylös (bottom-up) -menetelmällä tai kaksisuuntaisella (sandwich) menetelmällä, joka on yhdistelmä molemmista. Ylhäältä alas -menetelmässä rakennetaan ensin ohjelman runko. Runkoon integroidaan yksikkötestattuja osia yksi kerrallaan. Alhaalta ylös -menetelmässä yksikkötestattuja osia integroidaan suoraan toisiinsa, jolloin saadaan aina hiukan isompia yksiköitä, kunnes tuote on valmis.

Toimintosuuntautuneessa testaamisessa testataan järjestelmän toimintoja. Järjestelmän rakenne ei vaikuta testattavaan kokonaisuuteen, vaan testattavia osia liitetään toisiinsa toimintojen mukaan. Toiminnot voivat olla joko käyttäjän toimintoja tai järjestelmän sisäisiä toimintoja.

Testaamisessa käytetään kriittisen moduulin strategiaa. Tämä tarkoittaa sitä, että järjestelmän tärkeimmät toiminnot ja järjestelmän toiminnan kannalta kriittiset osat testataan tarkemmin. Käyttäjän toiminnot on priorisoitu vaatimusdokumentissa, ja järjestelmän sisäiset toiminnot ja osat arvioidaan suunnitteluvaiheessa.

Vähemmän kriittisissä, alhaisemman prioriteetin omaavissa käyttäjän toiminnoissa käytetään rakenteellista testaustapaa. Lähestymistapana voidaan käyttää ylhäältä alas – tai alhaalta ylös -menetelmää tai kaksisuuntaista menetelmää. Näistä valitaan testaukseen parhaiten soveltuva vaihtoehto.

4.2 Rajapintojen testaus

Kutakin integroitavaa kokonaisuutta kohden kirjoitetaan testipaketti, joka testaa kokonaisuuteen kuuluvien rajapintojen vastaavuuden niiden määrittelyyn. Testattavasta kokonaisuudesta kirjoitetaan testausraportti, jossa on määritelty, mitkä rajapinnat kuuluvat testattavaan kokonaisuuteen.

Integrointitestauksessa käytetään samoja menetelmiä kuin yksikkötestauksessa.

4.3 Valittu kattavuus

Testien pitää kattaa kaikkien integroitavien osien rajapinnoista ne palvelut, jotka kuuluvat testattavaan kokonaisuuteen.

4.4 Testisyötteiden valinta

Testisyötteiden arvoalueet valitaan rajapintojen suunnittelun perusteella, ja niistä pyritään löytämään sellaisia raja-arvoja, jotka saattavat aiheuttaa ongelmia.

4.5 Hyväksymiskriteerit

Integraatiotestit kirjoitetaan siten, että testin tulos on joko hyväksytty tai hylätty. Integroitava kokonaisuus hyväksytään, kun kaikki sen sisältämät testit on suoritettu hyväksytysti.

5 Järjestelmätestaus

Järjestelmätestauksessa koko toteutettu järjestelmä testataan kokonaisuutena, ja sen tarkoitus on testata, vastaako toteutettu järjestelmä vaatimusmäärittelyä.

Järjestelmätestaus tehdään järjestämällä testaustilaisuus, johon jokainen projektiryhmän jäsenen osallistuu. Tilaisuudessa testataan kaikki järjestelmätestaukseen valitut käyttötapaukset ja käydään läpi vaatimusmäärittelyn ei-toiminnalliset vaatimukset, laatuvaatimukset ja rajoitteet.

Testaustilaisuudesta tehdään testausraportti, josta ilmenee testatut vaatimukset ja tes-

tien tulokset. Vaatimusmäärittelyn ei-toiminnallisista vaatimuksista, laatuvaatimuksista ja rajoitteista kirjataan raporttiin niiden validointi tai perusteet, jos vaatimusta ei voi validoida.

5.1 Lähestymistapa

Toiminnallisten vaatimusten testaaminen tehdään testaamalla järjestelmän toimintaa käyttöliittymän kautta. Testaamisessa käytetään testaamista varten määriteltyjä käyttötapauksia.

5.2 Vaatimusten testaus

Järjestelmätestauksessa toiminnalliset vaatimukset testataan muodostamalla vaatimusmäärittelydokumentin käyttötapauksista laajennettuja käyttötapaustestejä ja käyttämällä järjestelmää niiden avulla.

5.3 Valittu kattavuus

Vaatimusmäärittelyn käyttötapauksista valitaan testattavaksi ne käyttötapaukset, joiden toiminnallisuus on mukana toteutetussa järjestelmässä.

Toteutusrajoitteista validoidaan XHTML 1.0 suosituksen mukainen käyttövaatimus. Sen testaus toteutetaan validoimalla W3C:n XHTML-validaattorilla [W3C08] kaikki järjestelmän XHTML-sivut. Järjestelmän tyylimäärittelyissä käytetään CSS 2.1 versiota ja myös sen suosituksen mukainen käyttö validoidaan käyttämällä W3C:n validaattoria [CSS08].

5.4 Testisyötteiden valinta

Laajennettujen käyttötapaustestien syötteiden arvoina käytetään yksikkö- ja integraatiotestauksessa käytettyjä arvoja. Arvoista valitaan järjestelmätestaukseen sellaiset arvot, jotka parhaiten soveltuvat testaukseen.

5.5 Hyväksymiskriteerit

Järjestelmätestaus on hyväksytty, kun kaikki valitut testauskäyttötapaukset on testattu hyväksytysti.

6 Testausaikataulu

- Testaussuunnitelma valmistuu 5.8.
- Yksikkö- ja integrointitestausta tehdään toteutuksen kanssa samaan aikaan. Testattavien yksiköiden ja toimintojen integraatiotestauksen aikataulu määritellään niiden suunnittelun yhteydessä.
- Järjestelmätestaustilaisuus pidetään torstaina 21.8.
- Järjestelmätestaus valmistuu 23.8.

7 Lähteet

CSS08 http://jigsaw.w3.org/css-validator/ [3.8.2008]

SQL08 http://fi.wikipedia.org/wiki/SQL [24.7.2008]

W3C08 http://validator.w3.org/ [3.8.2008]

Liitteet

Liite 1. Testausraportti-malli, yksikkötestausraportti.txt

Liite 2. Järjestelmätestauksen testitapaukset.odt