Testaussuunnitelma

Apuri

Helsinki 25.10.2004

Ohjelmistotuotantoprojekti (esimerkki)

HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Kurssi

582429 Ohjelmistotuotantoprojekti (esimerkki) (4 ov)

Projektiryhmä

Raine Kauppinen
Pietu Pohjalainen
Hannu Räisänen
Antti Tevanlinna

Asiakas

Juha Taina

Johtoryhmä

Juha Taina

Kotisivu

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

Versiohistoria

Versio	Päiväys	Tehdyt muutokset
0.7	2.6.2004	Alustava versio

Sisältö

1 Johdanto

Tämä dokumentti on APURI-projektiryhmän testaussuunnitelma. Se määrittelee:

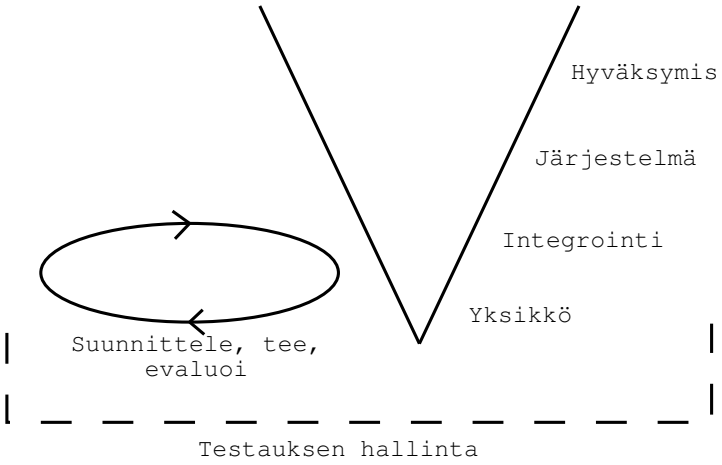
- projektissa käytettävän testausprosessin,
- käytettävät menetelmät,
- testauksen kattavuuden ja
- testauksesta raportoinnin.

Tämä dokumentti ei kerro testauksesta yleisesti, vaan perustiedot oletetaan tunnetuiksi. Yleisesityksiä testauksesta on löydettävissä muun muassa oppikirjoista [Som01, PI00] ja ohjelmistotekniikan kurssien materiaaleista.

Dokumentissa ei toisteta perusasioita testauksesta, koska dokumentin tarkoitus on määritellä projektissa suoritettava testaus ja sen menetelmät. Tämä johtaa myös *määräävään* kirjoitusasuun, joka korostaa dokumentin ohjemaisuutta.

1.1 Testausstrategia ja prosessi

Projektin testausprosessin tärkeimmät osat on esitetty kuvassa 1.



Kuva 1: Testausstrategian tärkeimmät oast.

Projektin tuotos testataan testauksen V-mallin mukaisesti yksikkö-, integrointi-, järjestelmä- ja hyväksymistestausvaiheissa. Lisäksi noudatetaan kevyttä suunnittele, tee ja evaluoi-sykliä testitapausten valmistuksessa. Kaikkien muiden toimintojen ulkopuolella on testauksen hallinnointi, jonka yksi osa on tämä dokumentti. Lisäksi hallinnointiin kuuluu testauksen tarkempi suunnittelu, jota ei erikseen dokumentoida, sekä testauksen toteutumisen tarkkailu.

Testaus suoritetaan mukaille testauksen V-mallia. Luku 2 kuvaa projektin sovelluksen mallista, joka määrää testausprosessin osat eri ohjelmiston abstraktiotasoilla.

1.2 Suunnittele-tee-evaluoi iteraatio

Projektissa sovelletaan suunnittelu-, valmistus- ja evaluointivaiheiden iterointia erityisesti yksikkö- ja integrointitestauksessa. Kun testitapauksia on valmistettu, ne suoritetaan. Suorituksen aikana kerätään kattavuustietoa. Jos kattavuus ei vastaa tavoitteita, palataan suunnittelemaan lisää testitapauksia.

1.3 Testauksen tarkkailu ja muu hallinto

Testauksen vastuuhenkilö (Antti Tevanlinna) tarkkailee testaustavoitteiden täyttymistä sekä muiden määrättyjen kohtien toteutumista. Mikäli testaus ei vastaa suunniteltua, keskustellaan ryhmän kokouksissa epäkohtien syistä ja vaadituista toimenpiteistä.

Lisäksi testauksen vastuuhenkilö hallinnoi testaukseen käytettäviä resursseja ja aikatauluja.

1.4 Työkalut

Testauksessa käytetään hyväksi työkaluja. Yksikkö- ja integraatiotestaus toteutetaan JUnit[1]-, Cactus[2]- ja HttpUnit[3]-kehyksiä hyödyntäen. Testauksen kattavuus tarkistetaan mahdollisuuksien mukaan RITA- ja OptimizeIt[4]-työkalujen avulla. Lisäksi virheideraportin ylläpidossa käytetään emacs-editoria.

Kaikki yksikkö- ja integrointestit automatisoidaan. Järjestelmätestejä automatisoidaan mahdollisuuksien mukaan.

JUnit-, Cactus- ja HttpUnit-työkalujen käyttöohjeet ovat liitteinä.

2 Testaustasot

Tässä luvussa määrätään testausmenetelmät ja kattavuus eri V-mallin tasoilla.

2.1 Yksikkötestaus

Projektissamme pienin testattava yksikkö on metodi osana suurempaa jakamatonta kokonaisuutta - luokkaa. Jokainen ei-triviaali metodi testataan yksikkönä ja sen jälkeen luokka

[1] kotisivut osoitteessa www.junit.org
[2] kotisivut osoitteessa jakarta.apache.org/cactus
[3] kotisivut osoitteessa httpunit.sourceforge.net
[4] kotisivut osoitteessa www.optimizeit.com

testataan yksikkönä.

Jokainen metodi testataan siten, että saavutetaan haaraumakattavuus sekä ehtokattavuus. Testitapaukset luodaan tarkastelemalla ohjelmakoodista löytyviä ehtorakenteita ja luoden kattavuuden saavuttavat testitapaukset. Tämän testauksen yhteydessä tutkitaan metodin palauttamien arvojen oikeellisuus metodin määrityksiä vastaavaksi. Lisäksi voidaan suorittaa laajempaa white-box sekä black-box testausta testaajan harkinnan mukaan. Erityisesti mikäli metodiin liittyy monimutkaisia ehtorakenteita tai muita algoritmejä, on lisätestaus suositeltua.

Metodien testauksen haarauma- ja lausekattavuus tarkastetaan työkaluin. Epätäydellinen kattavuus ei ole hyväksyttävää muuten kuin erikoistapauksissa.

Kun luokan metodit on testattu, muodostetaan luokalle tilasiirtymät kattava testaus. Jokainen luokan tilasiirtymistä on testattava oikeelliseksi. Toisin sanoen, jokaista luokan muuttujaa muuttava operaatio on kutsuttava alkutilassa ja tilan muutos on tarkastettava oikeaksi. Lisäksi jokaisen luokan sisäisen metodikutsun oikeellisuus (luokan sisäinen integrointi) on tarkastettava, jos sitä ei tule tarkastettua metodikohtaisessa testauksessa.

Testattavia luokkia ei määrätä erotettavaksi toisistaan tyngillä. Mikäli tynkiä ei käytetä, on erityisesti keskityttävä testaamaan yhtä luokkaa kerrallaan. Lisäksi yhtäkään luokkaa ei saa jättää yksikkötestaamatta, vaikka se olisi testattu toisen luokan yksikkö- tai integrointitestauksen yhteydessä.

2.2 Integrointitestaus

Projektissamme integrointitestausta suoritetaan heti kun integroitavia luokkia on olemassa. Tämä johtaa siihen, että integrointitestaus ei ole ylhäältä alas eikä alhaalta ylös integrointitestausta. Parhaiten sitä kuvaa määritelmä "jatkuva alhaalta ylös integrointi, jossa käytetään tynkiä korvaamaan puuttuvia osia".

Integrointitestausta suoritetaan luokkajoukoille, jotka muodostavat loogisen kokonaisuuden. Kun tälläinen joukko valmistuu, muodostetaan sen rajapintaluokkaa vastaa testijoukko ekvivalenssijakomenetelmää käyttäen.

Integroidun osan testauksen kutsukattavuus (jokainen kutsu metodista toiseen) tarkastetaan työkaluin. Huomattavaa on että, myös kaikki mahdolliset virtuaalisesti sidotut kutsut on katettava. Epätäydellinen kattavuus ei ole hyväksyttävää muuten kuin erikoistapauksissa.

2.3 Järjestelmätestaus

2.4 Hyväksymistestaus

Hyväksymistestausta suoritetaan alpha- ja beta-vaiheissa.

Alpha-testauksessa asiakas testaa projektin tuotteen ja ilmoittaa lyödetyistä puutteista. Tämän perusteella tehdään päätös tuotteen hyväksymisesta beta-testaukseeen. Jos tuot-

teen laatu ei ole hyväksyttävä, palataan prosessissa korjaukseen, jonka jälkeen aloitetaan alpha-testaus uudestaan.

Beta-testaus suoritetaan ohjelmistotuotantoprojektien yhteydessä. Ohjelmisto annetaan käyttöön valituille ohjelmistotuotantoprojekteille. Beta-testaus on osa ohjelmiston ylläpitoa ja hienosäätöä ohjelmistotuotantoprojektien tarpeita vastaavaksi.

3 Raportointi

Projektissa tuotetaan neljä testaukseen liittyvää dokumenttia: tämä dokumentti, järjestelmätestien kuvaukset, yksikkö- ja integrointitestien kuvaukset sekä virhetiedot muista kuin yksikkö- ja integraatiotesteistä.

Järjestelmätestien kuvaukset kirjataan omaan dokumenttiinsa (nimi?). Jokaisesta testistä kirjataan testin tunniste ja toistamiseen tarvittava informaatio, joka sisältää vähintään alkutilan, testisekvenssin ja tehtävät tarkastukset.

Yksikkö- ja integrointestit dokumentoidaan yksinkertaisesti JavaDoc-työkalun avulla. Niistä kirjataan jokaista testiä kohden testattavat asiat kuten ehdot tai ekvivalenssiluokat.

Virhetiedoista pidetään yllä yksinkertaista raporttia, joka sisältää päiväyksen, testin tunnisteen ja lyhyen kuvauksen virheestä.

4 Virheiden korjaus ja regressiotestaus

Yksikkö- ja integrointitestauksessa löytyneet virheet korjataan ilman erityistä toimintatapaa. Korjauksen jälkeen suoritetaan olemassa olevat automatisoidut testitapaukset uudelleen mahdollisten uusien virheiden havaitsemiseksi. Järjestelmä- ja hyväksymistestauksessa löytyneet virheet analysoidaan ja korjataan suuremmissa erissä, jonka jälkeen suoritetaan yksikkö- ja integraatiotestit sekä tarpeellisiksi katsotut järjestelmätestit.

Lähteet

PI00 Pressman, R. S. ja Ince, D., *Software Engineering: A Practitioner's Approach*. McGraw-Hill, Englanti, 2000.

Som01 Sommerville, I., *Software Engineering*. Addison-Wesley, 2001.

A JUnit-työkalun käyttö

B Cactus-työkalun käyttö

C HttpUnit-työkalun käyttö

D Testauksen automatisointi käännöksen ja pakkauksen yhteydessä

Projektissa käytettävä Ant-työkalu mahdollistaa testauksen automaattisen suorituksen ohjelmiston käännöksen ja pakkaamisen yhteydessä. Työkalun toiminnan määrää käyttäjän tekemä xml-muotoinen tiedosto `build.xml`, joka sisältää työkalun tulkitsemat käskyt.

Ant-työkalun käskyjen tarkemmat tiedot löytää työkalun kotisivulla olevasta manuaalista[5].

Ant-työkalun käskytiedostoon lisätään seuraavat osat.

1. JUnit-testien suoritus:

```
<junit printsummary="yes" haltonfailure="yes">
  <classpath>
    <!--tarvittavat luokat testien suorittamiseksi-->
  </classpath>

  <formatter type="plain"/>

  <!--suoritetaan kaikki 'Test'-merkkijonon nimessään
      sisältävät luokat paitsi Erityiset AllTests-luokat-->
  <batchtest fork="yes" todir="${reports.tests}">
    <fileset dir="${src.tests}">
      <include name="**/*Test*.java"/>
      <exclude name="**/AllTests.java"/>
    </fileset>
  </batchtest>
</junit>
```

2. Cactus-testien suoritus: Cactus testien suoritus on hiukan monimutkaisempaa vaatien war-pakkauksen. Lisätietoja Cactus-työkalun liittämisestä Ant-työkaluun löytyy verkosta [6] [7]. Ant-tiedostoon tulee useita kohtia, joista tässä esitellään vain tär-

[5] http://ant.apache.org/manual/index.html
[6] http://jakarta.apache.org/cactus/integration/ant/index.html
[7] http://jakarta.apache.org/cactus/integration/ant/howto_ant_cactus.html

keimmät osat ilman yksityiskohtia. Nämä on jokaisen ryhmän asetettava käytettävän tiedostorakenteen mukaisiksi. Esimerkkikäskytiedostossa on toimiva Cactus-testi.

```
<!-- tuodaan Cactus käytettäväksi Ant-työkaluun -->
<taskdef resource="cactus.tasks">
  <classpath>
...
  </classpath>
</taskdef>

<!-- muunnetaan war-tiedosto, muuta XXX
     ja YYY vastaamaan tarpeitasi -->
<cactifywar srcfile="XXX.war" destfile="YYY.war"/>

<!-- asennetaan järjestelmä ja suoritetaan testit -->
<cactus warfile="${target.dir}/test.war" fork="yes"
      failureproperty="tests.failed">

...

</cactus>
```

3. HttpUnit-testien suoritus