Harjoitustyön testaus

Juha Taina

1. Johdanto

- Ohjelman teko on muutakin kuin koodausta. Oleellinen osa on selvittää, että ohjelma toimii oikein. Tätä sanotaan ohjelman validoinniksi.
- Eräs keino validoida ohjelmaa on testata sitä.
- *Testaus* tarkoittaa, että ohjelmaa suoritetaan tietyillä syötteillä, *testitapauksilla*. Suorituksen jälkeen varmistetaan, että testitapauksella saatu tulos oli oikea.
- Testausta tehdään sekä koodausvaiheessa että ohjelman valmistuttua.

Testauksen kolme piirrettä

- Mikä tahansa ohjelman suoritus ei ole testausta. Testaus on
  - *systemaattista.* Testaus suunnitellaan etukäteen sellaiseksi, että se kattaa *testausstrategian*.
    - Testausstrategia määrittelee, millä tavalla testataan ja milloin on testattu tarpeeksi.
  - *toistettavaa.* Jo tehdyt testit on voitava suorittaa uudestaan ilman että niitä kirjoitetaan uudestaan.
  - *mahdollisimman automaattista*. Testit pitää voida suorittaa automaattisesti. Saatua tulosta pitää voida verrata automaattisesti tiedettyyn oikeaan tulokseen. Testien kirjoitusta ei voida (osata) automatisoida.

2. Koodausvaiheen testaus

- Koodausvaiheessa testataan metodeja. Koodi on näkyvissä, ja sitä käytetään hyväksi testitapauksia suunniteltaessa.
- Tätä testausta sanotaan *lasilaatikkotestaukseksi*, koska ohjelma on eräänlainen läpinäkyvä laatikko.
- Lisäksi on olemassa *mustalaatikkotestausta*, jota käytetään, kun ohjelma on valmis. Siinä ohjelmasta tiedetään syötteet ja oletetut tulokset, mutta ei toiminnan yksityiskohtia. Myöhemmin esiteltävä toiminnallinen testaus on yleensä mustalaatikkotestausta.

Haaraumakattavuus

- Metodien testaukseen on määritelty useita testausstrategioita. Niistä yksinkertaisin hyödyllinen on ns. *haaraumakattavuuden* testausstrategia: lyhyesti haaraumakattavuus.
- Haaraumakattavuudessa suunnitellaan sellaiset testitapaukset, että
  - kaikki koodirivit käydään läpi ainakin kerran,
  - kaikissa ehtolauseissa käydään sekä tosi- että epätosi-vaihtoehdot ja
  - kaikissa silmukoissa käydään sekä nolla kierrosta että ainakin yksi kierros.

Metodiesimerkki

- Esim. olkoon meillä seuraava metodi, joka ilmoittaa toisen asteen yhtälön $ax^2+bx+c = 0$ juurten lukumäärän.

```
public int juuria(int a, int b, int c) {
        int juuria;
        if (b*b-4*a*c > 0) {
                juuria = 2;
        }
       else
        if (b*b-4*a*c == 0) {
                juuria = 1;
        }
        else {
                juuria = 0;
        }
        return juuria;
      }
```

Metodiesimerkki 2

- Edellisessä metodissa on kaksi haaraumaa, joiden molempien tosi- ja epätosivaihtoehdot tulee käydä läpi.
- Tämä saadaan tehtyä esimerkiksi seuraavilla testitapauksilla:
  - a=1, b=3, c=1 (1. tosi)
  - a=1, b=2, c=1 (1. epätosi, 2. tosi)
  - a=2, b=1, c=2 (1. epätosi, 2. epätosi)

Haaraumakattavuuden apu

- Kun koodauksessa toteutetaan haaraumakattavuuden testitapaukset, varmistetaan, että kaikki kirjoitettu koodi tulee käytyä läpi.
- Tämä kuulostaa triviaalilta, mutta se ei ole sitä. Hyvinkin testatuissa ohjelmissa käydyn koodin määrä on usein alle 70%.
- Poikkeustapaukset unohtuvat helpoiten, mutta myös ne pitää testata.

Haaraumakattavuuden heikkouksia

- Valitettavasti pelkkä haaraumakattavuus ei takaa, että ohjelma toimii oikein.
  - Haaraumakattavuus ei voi testata sellaista koodia, joka puuttuu. Esim. edellinen esimerkki oli vajaa, mutta haaraumakattavuuden testitapaukset toimivat kaikki oikein.
  - Haaraumakattavuus ei välttämättä löydä sellaisia virheitä, jotka riippuvat muuttujien arvosta. Esim. a=a/b; voidaan testata arvoilla a=1,b=1 ja kaikki menee oikein, vaikka arvoilla a=1,b=0 tulee nollalla jaon poikkeus.

3. Valmiin ohjelman testaus

- Koska haaraumakattavuus ei yksin riitä, tarvitaan muita testausstrategioita.
- Yleisin käytetty testausstrategia on *toiminnallinen testaus*. Siinä testataan ohjelman toimintaa antamalla sille sopivia syötteitä ja tarkastelemalla saatuja tuloksia.
- Myös metodeita voi ja pitää testata toiminnallisella testauksella.

Syötekombinaatioräjähdys

- Täydellisessä toiminnallisessa testauksessa ohjelma testataan kaikilla syötekombinaatioilla.
- Käytännössä tämä on mahdotonta, sillä syötekombinaatioiden määrä kasvaa tähtitieteellisiin lukuihin.
- Esim. edellisessä esimerkissä kombinaatioiden määrä on noin $4*4*4*10^{9+9+9} = 6,4 * 10^{28}$. Mikään kone ei selviä tällaisesta testitapausten määrästä.

Syötteen ositus

- Testitapausten määrää supistetaan *osittamalla* syötteet. Sen sijaan, että testataan kaikilla syötteillä, valitaan syötteiden osajoukkoja, joista kustakin testattavaksi valitaan vain muutama syöte.
- Ositus tehdään tunnetun toiminnan avulla seuraavasti:
  - Kaikki samalla tavalla käyttäytyvät syötteet kuuluvat samaan osajoukkoon.

Ositusesimerkki

- Esimerkiksi edellä syötteet olivat a, b ja c. Kunkin syötteen syöteavaruus on Javan kokonaislukujen joukko.
- Osituksessa ratkaisee se, miten ohjelman oletetaan käyttäytyvän eri kombinaatioilla. Esimerkissä tämän ratkaisee toisen asteen yhtälön ratkaisukaava:
  - x = (-b +- sqrt(b*b-4*a*c))/2*a.

Ositusesimerkki

- Osituksen kannalta oleellisia osia ovat
  - b*b-4*a*c ja
  - 2*a.
- Lisäksi ositukseen voidaan ottaa laittomat arvot (esimerkiksi merkkijonot), jos niitä voidaan antaa ohjelmalle. Metodin kohdalla laittomat arvot huomataan käännöksessä, mutta ohjelmalla ne huomataan vasta suoritusaikana.

Ositusesimerkki

- Osittamalla b*b-4*a*c saadaan
  - b*b-4*a*c < 0
  - b*b-4*a*c == 0
  - b*b-4*a*c > 0
- Osittamalla 2*a saadaan
  - 2*a < 0
  - 2*a == 0
  - 2*a > 0

Ositusesimerkki

- Näistä osituksista voidaan valita vaikkapa seuraavat testitapaukset:
  - a=2, b=1, c=2 (b*b-4*a*c < 0)
  - a=1, b=2, c=1 (b*b-4*a*c == 0)
  - a=1, b=3, c=1 (b*b-4*a*c > 0)
  - a=-1,b=2, c=1 (2*a < 0)
  - a=0,b=2,c=1 (2*a == 0)
  - a = 1,b=2, c=1 (2*a > 0), vanha testitapaus
  - Nyt huomataan, että testitapaus a=0 antaa väärän tuloksen 2 (pitäisi olla 1).

Yksinkertaisempi ositus

- Edellinen ositus oli aika vaikea, sillä se perustui toisen asteen yhtälön ratkaisukaavaan. Yksinkertaisempi ositus saadaan vaikkapa seuraavalle metodille:

```
public double kaanteisluku(double a) {
    return 1/a;
}
```

- a:n arvot voidaan osittaa yksinkertaisesti:
  - a < 0. Testi a=-1.0 käy.
  - a == 0. Testi a=0.0 käy.
  - a > 0. Testi a=1.0 käy.

Osituksen teko

- Miten hyvä ositus tehdään? Kokemuksen kautta:
  - Normaalista toiminnasta tehdään osajoukko.
  - Kustakin erikoistapauksesta tehdään osajoukko.
  - Jos ohjelmassa tiedetään olevan jokin ehto, ehdon toteuttavista arvoista tehdään osajoukko ja ehdon hylkäävistä arvoista osajoukko.
  - Yleensä luku 0 on yhden alkion osajoukko.
  - Usein luku 1 on yhden alkion osajoukko.
  - Virheellisistä syötteistä tehdään osajoukko.

Vielä osituksesta

- Osituksella voidaan testata myös toimintoja. Tällöin jokainen tietyn toiminnon aktivoiva syöte kuuluu samaan osajoukkoon.
  - Esimerkiksi tekstinkäsittelyjärjestelmässä komento ”Lihavoi teksti” sisältää osajoukkona kaikki lihavointi päällä syötettävät tekstit. Näistä riittä valita testitapauksiksi muutama.
- Yleensä osajoukosta valitaan muutama testitapaus. Näin pyritään varmistamaan, että osajoukkoihin jako meni oikein.

4. Yhteenveto

- Hyvä testaus parantaa ohjelman laatua. Vaikka koodi näyttäisi silmin katsottuna virheettömältä, siellä voi olla yllätyksiä odottamassa.
- Lasilaatikkotestauksella testataan metodeja koodin perusteella.
- Mustalaatikkotestauksella testataan metodeja ja ohjelmaa syötteiden perusteella.
- Molempia tarvitaan.