

Ohjelmistotuotanto

Testaus 2

1

Rakenteellinen testaus (white box)

- Rakenteellinen testaus perustuu ohjelman rakenteen hyväksikäyttöön - tieto ja kontrollivuoesityksiin
 - tietovuo (data flow) - tiedon kulku
 - kontrollivuo (control flow) - kontrollin kulku
- Kontrollin kulkua voidaan kuvata perinteisellä vuokaaviolla (flowchart)

©Harri Laine 2

Kontrollin kulku

- vuokaavio
 - 1-1 vastaavuus ohjelmakoodin kanssa
 - jokainen lause näkyy kaaviossa
 - liian yksityiskohtainen testauksen kannalta
- vuoverkko (flowgraph)
 - abstraktio vuokaaviosta
 - kontrollin haarautumis- ja yhtymiskohdat esitetään verkon solmuina (node)
 - peräkkäiset suoritettavat lauseet yhdistyvät yhdeksi prosessiksi

©Harri Laine 3

Ohjelmasta vuoverkoksi

```

procedure sort begin
  while records_remain begin
    read record;
    if record.field1 = 0 then begin
      process record;
      store in buffer;
      increment counter;
    end else begin
      if record.field2=0 then begin
        reset counter;
      end else begin
        process record;
        store in file;
      end;
    end;
  end while;
end;

```

©Harri Laine 4

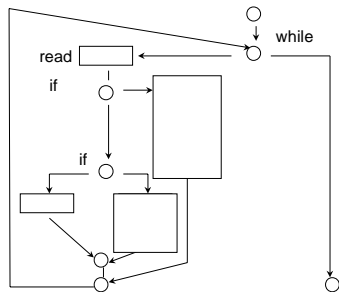
Ohjelmaa vastaava vuokaavio

©Harri Laine 5

Yhdistetään peräkkäiset lauseet

©Harri Laine 6

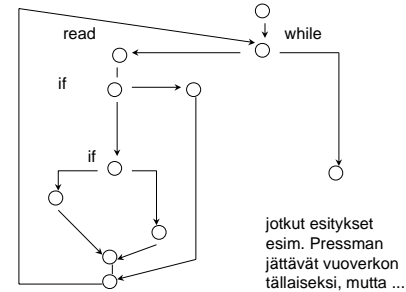
Korvataan ehdot haarautumasolmuilla



©Harri Laine

7

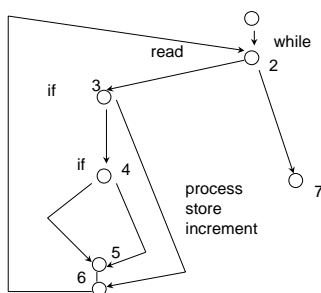
Korvataan lausesekvenssit prosessisolmuilla



©Harri Laine

8

Jatketaan poistamalla prosessisolmut ja sijoitetaan toiminnallisuus särmiin



©Harri Laine

9

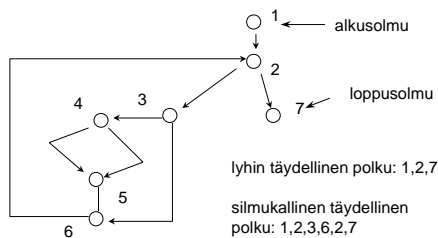
Vuoverkon käsitteitä

- vuoverkon polku (path), reitti:
 - jono linkkien yhdistämiä solmuja
 - sama solmu voi esiintyä polulla useaan kertaan (silmukat)
- segmentti:
 - polun osajono
- segmentin pituus:
 - segmentin solmujen lukumäärä
- täydellinen polku (complete path):
 - polku, joka alkaa vuoverkon alkusolmusta ja päättyy sen loppusolmuun
 - alkusolmu: ei sisääntulevia linkkejä (esimerkissä 1)
 - loppusolmu: ei ulosmeneviä linkkejä (esimerkissä 7)

©Harri Laine

10

Vuoverkon käsitteitä



©Harri Laine

11

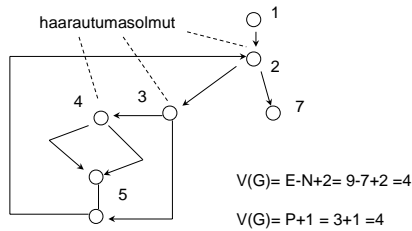
Vuoverkon käsitteitä

- McCaben kompleksisuusmitta (cyclomatic complexity), merkitään $V(G)$, verkolle G
 - ohjelman vuoverkon alueiden määrä
 - $V(G) = E - N + 2$, missä E =särmien määrä ja N =solmujen määrä
 - $V(G) = P + 1$, missä P on verkon haarautumasolmujen lukumäärä
 - käytetään mittaamaan ohjelman monimutkaisuutta
 - käytetään laskettaessa tarvittavien testien lukumäärää

©Harri Laine

12

Vuoverkon käsitteitä



©Harri Laine

13

Polkutestaus

- Polkutestauksessa on tarkoitus valita testiaineistot siten, että ne aiheuttavat tiettyjen vuoverkon polkujen suorituksen
- On määritelty erilaisia kattavuusmittoja polkutestaukselle:
 - polkukattavuus (path coverage)
 - käydään läpi kaikki mahdolliset täydelliset polut (polku/ testiäjo)
 - usein mahdotonta koska polkuja on liian paljon

©Harri Laine

14

Polkutestaus

- lausekattavuus (statement coverage)
 - käydään jokaisessa ohjelman lauseessa, eli jokaisessa vuoverkon solmussa ja prosessisärmässä, vähintään kerran testiäjojen yhteydessä
- haara- eli päätöskattavuus (branch/decision coverage)
 - käydään ohjelman jokaisen ehdon kaikki arvot (kaikki vuoverkon särmät) läpi vähintään kerran testiäjojen aikana
 - McCaben kompleksisuusmitta antaa tällöin tarvittavien testiäjojen vähimmäismäärän
 - yleisesti tavoiteltu kattavuus

©Harri Laine

15

Polkutestaus

- ehtokattavuus (condition coverage)
 - ohjelman jokaisen päätöksen kaikkien osaehtojen on saatava kaikki arvonsa testiäjojen aikana
- moniehtokattavuus (multiple condition coverage)
 - testaus on suoritettava ohjelman jokaisen päätöksen osaehtojen kaikilla arvoyhdistelmillä
- silmukkatestaus (loop testing)
 - suoritetaan ohjelman toistolauseita (vuoverkon silmukallisia polkuja) useampaan kertaan saman testiäjojen aikana (koska virheet kasautuvat silmukoihin)
 - 0 kertaa, 1 kerta, tyypillinen määrä, maksimimäärä, maksimi+1 kertaa
 - tarkentaa haarakattavaa testauksia

©Harri Laine

16

Polkutestaus

- täydellistä kattavuutta voi olla mahdoton saavuttaa

```
if a<100 then begin
  b:=a;
```

```
  ....
  .... ei sijoituksia b:lle
```

```
  if b>100 then begin ..... end;
end;
```

saavuttamaton kohta

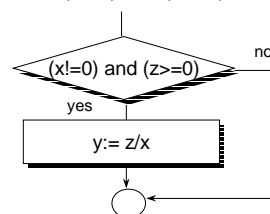
©Harri Laine

17

Polkutestaus

- Esimerkki:

```
if (x!=0) and (z>=0) then y:= z/x; endif;
```



Lausekattavuus:
 x=1, z=0: **yes**
 1 testiäjo

Haarakattavuus:
 x=1, z=0: **yes**
 x=1, z=-1: **no**
 2 testiäjoa

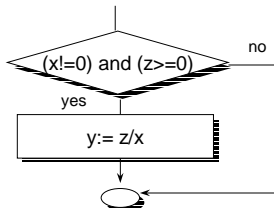
©Harri Laine

18

Polkutestaus

Esimerkki:

if (x!=0) and (z>=0) then y:= z/x; endif;



Ehtokattavuus:

x=0: no, z=0: yes: **no**
 x=1: yes, z=-1: no: **no**
 2 testiajoa
 mutta yes-haara jää testaamatta

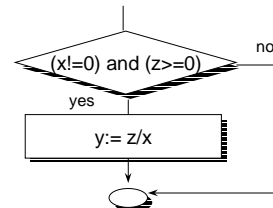
©Harri Laine

19

Polkutestaus

Esimerkki:

if (x!=0) and (z>=0) then y:= z/x; endif;



Moniehtokattavuus:

x=0: no, z=0: yes: **no**
 x=0: no, z=-1: no: **no**
 x=1: yes, z=0: yes: **yes**
 x=1: yes, z=-1: no: **no**
 4 testiajoa

©Harri Laine

20

Tietovuotestaus

- Tietovuotestauksessa testiaineiston laadinta perustetaan tietorakenteiden (muuttujien) tilamuutoksiin (asetus, käyttö laskennassa, käyttö päätöstilanteissa, 'nollaus')
- Suhteutetaan tilamuutokset vuoverkkoon
- Erilaisia aineiston laatimisstrategioita
 - kaikki asetukset (kunkin muuttujan jokainen asetus testattava vähintään kerran)
 - kaikki päätöskäytöt (kunkin asetuksen kaikki käytöt päätöstilanteissa testattava, ellei tällaisia ole on testattava vähintään yksi laskentakäyttö)
 - kaikki käytöt (kunkin asetuksen kaikki käyttötilanteet testattava)

©Harri Laine

21

Oliopohjainen testaus

- Perintä ja polymorfismi tuovat mukanaan testausongelmia (mutta poistavat myös joitakin)
 - voi olla vaikea hahmottaa minkä luokan palvelu varsinaisesti suoritetaan
 - ohjelmoijalla on voinut olla väärä käsitys siitä miten palvelu kehittyi luokkahierarkiassa
 - palvelun valinta perustuu polymorfismiin, jolloin virheelliseen valintaehtoon liittyviä ongelmia ei esiinny

©Harri Laine

22

Oliopohjainen testaus

- Yksikkötestaus vastaa luokan testaus
 - metodien satunnaiset suoritusjärjestykset
 - elinkaaritestausta - tilakaavion pohjalta
 - ositus : testataan vain osaa luokasta
 - olion tilaa muuttavat & olion tilan säilyttävät metodit
 - attribuuttiperustainen ositus
 - toimintojen kategorisointiin pohjaustuva jaottelu (alustus, laskenta, kysely, ..., lopetus)
- Yhteistyön testaus suorituspolku (thread) testauksena

©Harri Laine

23

Oliopohjainen testaus

- Käyttötapapohjainen testaus soveltuu validointitestaukseen ja suorituspolkutestaukseen
- Käyttötapatestauksella voidaan testata myös käyttöliittymää
- Skenaariot (esimerkkitapaukset) käyttötavan ilmentymiä - toimintasarja
 - tyypilliset tapaukset
 - poikkeustilanteet

©Harri Laine

24

Virheenjäljitys

- Onnistunut testi johtaa tarpeeseen jäljittää virhe
- Varsin huonosti systematisoitu alue
 - virhe voi olla kaukana ulkoisesta ilmenemispaikastaan
 - virheen korjaaminen voi tilapäisesti estää muiden virheiden ilmenemisen
 - virheen syy saattaa olla käyttäjässä
 - virheellistä toimintaa voi olla vaikea toistaa
 - virhe voi johtua ohjelmointikielen tai laitteistoarkkitehtuurin erityispiirteistä

©Harri Laine

25

Virheenjäljitys

- Virheenjäljitys on ongelmanratkaisutilanne, johon auttaa
 - kokemus
 - näkökulman vaihto
- Jäljitysmenetelmiä
 - raaka työ
 - työkalujen mekaaninen käyttö:
 - muistivedokset (dump)
 - suoritusjäljitin (trace, debugger)
 - suuri määrä tulosteita
 - vaikea löytää tarvittavaa tietoa

©Harri Laine

26

Virheenjäljitys

- peruutus
 - lähdetään havaitusta häiriöstä ja peruutetaan mahdollisia toimintapolkuja pitkin
 - tietovuovipaloijaa käyttämällä (slicer, esim .HyperSoft)
 - ohjelma näyttää virhekohtaan johtavat tietovuot ja mahdollistaa niiden seuraamisen
- syiden eliminointi
 - induktio:
 - kerää virheeseen liittyvät tiedot
 - hypoteesi virheen syystä
 - hypoteesin testaus
 - deduktio:
 - kerää mahdolliset syyt
 - eliminoi syitä havaintojen perusteella

©Harri Laine

27

Korjaus

- Ennen löydetyn virheen korjausta, olisi syytä pohtia:
 - Esiintyykö samantyyppinen virhe myös muualla ohjelmassa?
 - jos esiintyy, niin korjataan
 - Aiheuttaako muutos mahdollisesti uusia virheitä ohjelmaan?
 - jäljitetään sivuvaikutukset
 - Mitä olisi pitänyt tehdä toisin, ettei virhettä olisi syntynyt
 - vältetään jatkossa, oppiva tuotantoprosessi

©Harri Laine

28