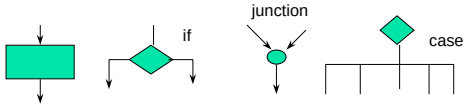
Ohjelmistotuotanto

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)



 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

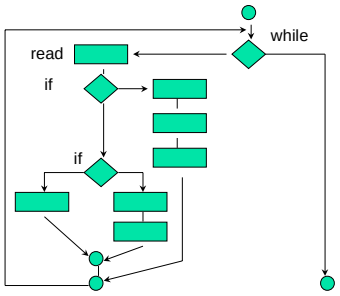
 3

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;
```

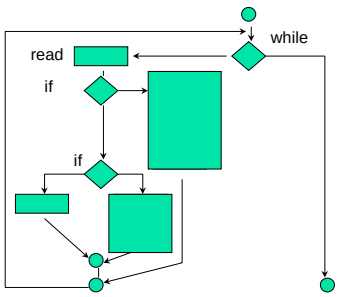
 4

Ohjelmaa vastaava vuokaavio



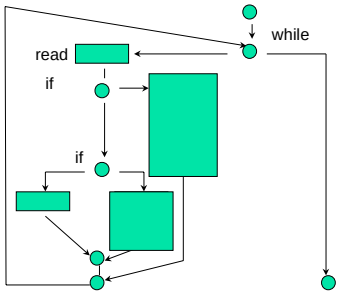
 5

Yhdistetään peräkkäiset lauseet



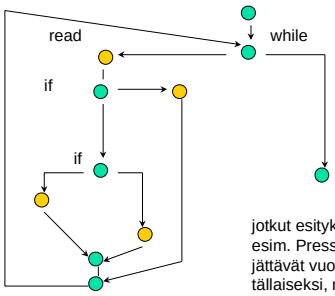
 6

Korvataan ehdot haarautumasolmuilla



 7

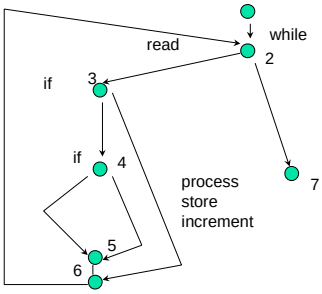
Korvataan lausesekvenssit prosessisolmuilla



jotkut esitykset esim. Pressman jättävät vuoverkon tällaiseksi, mutta ...

 8

Jatketaan poistamalla prosessisolmut ja sijoitetaan toiminnallisuus särmiin



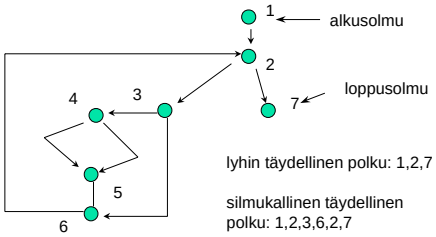
 9

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)

 10

Vuoverkon käsitteitä



lyhin täydellinen polku: 1,2,7

silmukallinen täydellinen polku: 1,2,3,6,2,7

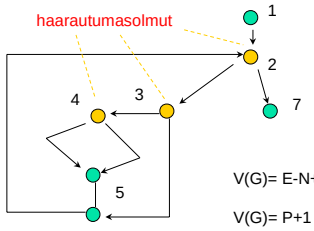
 11

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

 12

Vuoverkon käsitteitä



V(G)= E-N+2= 9-7+2 =4

V(G)= P+1 = 3+1 =4

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/testiajo)
    - usein mahdotonta koska polkuja on liian paljon

Polkutestaus

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

Polkutestaus

- ehtokattavuus (condition coverage)
  - ohjelman jokaisen päätöksen kaikkien osaehtojen on saatava kaikki arvonsa testiajojen 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 testiajon aikana (koska virheet kasautuvat silmukoihin)
  - 0 kertaa, 1 kerta, tyypillinen määrä, maksimimäärä, maksimi+1 kertaa
  - tarkentaa haarakattavaa testausta

Polkutestaus

- täydellistä kattavuutta voi olla mahdoton saavuttaa

```
if a<100 then begin
  b:=a;
  ----
  ----
  ----
  if b>100 then begin  -------  end;
end;
```

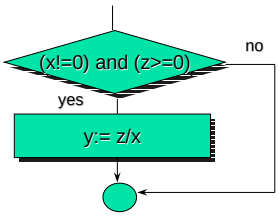
ei sijoituksia b:lle

saavuttamaton kohta

Polkutestaus

- Esimerkki:

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

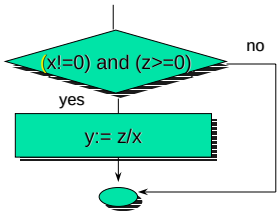


Lausekattavuus:
x=1, z=0: **yes**
1 testiajo

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

Polkutestaus

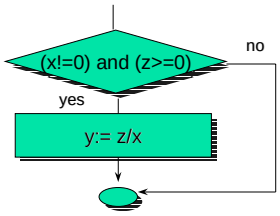
- Esimerkki:
  if (x!=0) and (z>=0) then y:= z/x; endif;



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

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

Tietovuotestaus

- Tietovuotestauksessa testiaineiston laadinta perustetaan tietorakenteiden (muuttujien) tilamuutoksiin (asetus, käyttö laskennassa, käyttö päätöstilanteessa, '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)

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 kehittyy luokkahierarkiasssa
  - palvelun valinta perustuu polymorfismiin, jolloin virheelliseen valintaehtoon liittyviä ongelmia ei esiinny

Oliopohjainen testaus

- Yksikkötestausta vastaa luokan testaus
  - metodien satunnaiset suoritusjärjestykset
  - elinkaaritestaus - 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

Oliopohjainen testaus

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

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

 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

 26

Virheenjäljitys

- peruutus
  - lähdetään havaitusta häiriöstä ja peruutetaan mahdollisia toimintapolkuja pitkin
  - tietovuoviipaloijaa 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

 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

 28