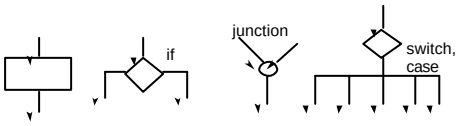
Ohjelmistotuotanto

Ohjelmistojen testaus 2

1

2. 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 (omana laatikkonaan)
  - liian yksityiskohtainen testauksen kannalta
- vuoverkko (flowgraph)
  - abstraktio vuokaaviosta
  - kontrollin haarautumis- ja yhtymiskohdat esitetään verkon solmuina
  - peräkkäiset suoritettavat lauseet yhdistyvät yhdeksi prosessiksi

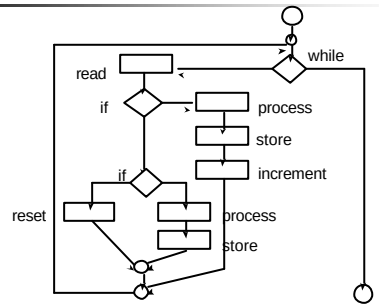
 3

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;
```

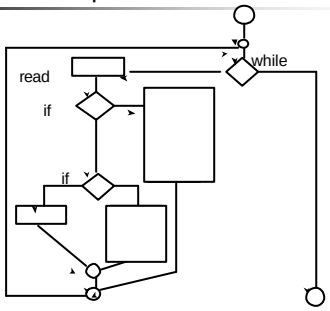
 4

Ohjelmaa vastaava vuokaavio



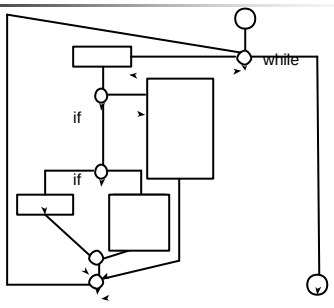
 5

Yhdistetään peräkkäiset lauseet



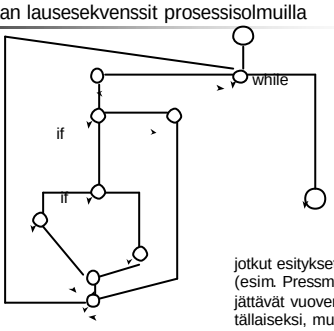
 6

Korvataan ehdot haarautumasolmuilla



 7

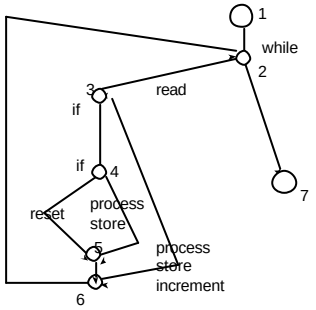
Korvataan lausesekvenssit prosessisolmuilla



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

 8

Voidaan jatkaa sieventämistä poistamalla prosessisolmut ja sijoittamalla toiminnallisuus särmiin



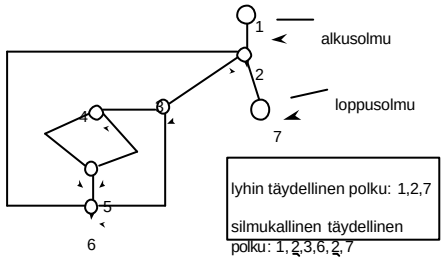
 9

Vuoverkon käsitteitä

- vuoverkon polku, reitti (path):
  - jono särmien (linkkien) yhdistämiä solmuja
  - sama solmu voi esiintyä polulla useaan kertaan (silmukat)
- segmentti:
  - polun osajono
- segmentin (polun) pituus:
  - segmentin (polun) 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



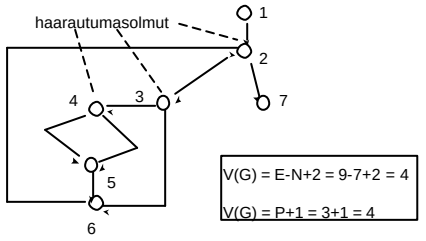
 11

Vuoverkon käsitteitä

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

 12

Vuoverkon käsitteitä



Polkutestaus

- Polkutestauksessa on tarkoitus valita testitapaukset siten, että ne aiheuttavat tiettyjen vuoverkon polkujen suorituksen
- On määritelty erilaisia kattavuusmittoja polkutestaukselle:
  - polkukattavuus (path coverage)
    - käydään läpi kaikki mahdolliset täydelliset polut (polku / testiajo)
    - useimmiten 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
  - yleisesti tavoiteltu kattavuus: vahvempi ehto kuin lausekattavuus eli pakottaa testaamaan enemmän

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)
  - iterointi 0 kertaa (silmukan ohitus), 1 kerran, tyypillinen määrä, maksimimäärä, maksimi +1 kertaa
  - tarkentaa haarakattavaa testausta

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
(“kuollutta koodia”)

Polkutestaus – esimerkki

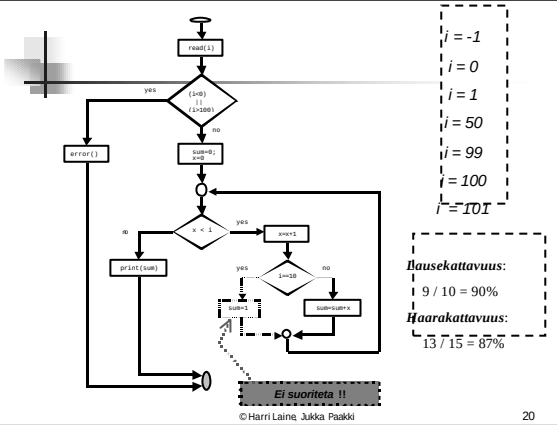
Ohjelma laskee kokonaislukujen summan syötteenä annettuun ylärajaan saakka:

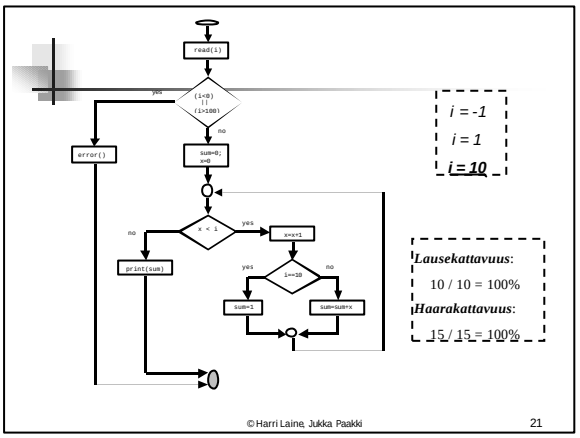
$$sum = 0 + 1 + 2 + \ldots + i, \quad i \in [0, 100]$$

- Mustalaatikko- ja raja-arvotestaus:
  - i = -1, i = 0, i = 1, i = 50, i = 99, i = 100, i = 101
  - ohjelma toimii testiaineistolla oikein
  - kuitenkin: i = 10 $\Rightarrow$ sum = 1
  - miksei testaus havainnut virhettä?

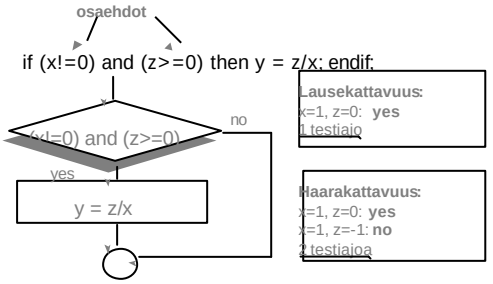
Polkutestaus – esimerkki

```
read(i);
if ((i < 0) || (i > 100))
   error();
else
   { sum=0; x=0;
   while (x < i)
      { x=x+1;
      if (i==10) sum=1; else sum=sum+x; }
   print(sum); }
```

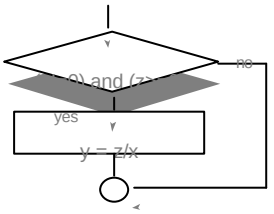


Polkutestausmitat – vertailu



Polkutestausmitat – vertailu

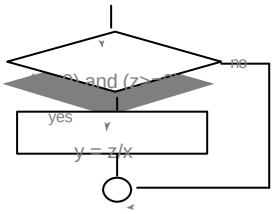
if (x!=0) and (z>=0) then y = z/x; endif;



Ehtokattavuus:
x=0: no, z=0: yes: **no**
x=1: yes, z=-1: no: **no**
2 testiajoa
(mutta yes-haara ja sijoituslause jäävät testaamatta)

Polkutestausmitat – vertailu

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 (arvon asetus, käyttö laskennassa, käyttö päätöstilanteessa, tuhoaminen)
- Tilamuutokset kuvataan (tieto)vuoverkossa
- Erilaisia testiaineiston laatimisstrategioita:
  - kaikki asetukset (kunkin muuttujan jokainen arvoasetus testattava vähintään kerran)
  - kaikki päätöskäytöt (kunkin arvoasetuksen kaikki käytöt päätöstilanteissa testattava; ellei tällaisia ole, on testattava vähintään yksi laskentakäyttö)
  - kaikki käytöt (kunkin arvoasetuksen kaikki mahdolliset käyttötilanteet testattava)

 25

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 luokkahierarkiassa
  - palvelun valinta perustuu polymorfismiin, jolloin virheelliseen valintaehtoon (case / switch) liittyviä ongelmia ei esiinny

 26

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 pohjautuva jaottelu (alustus, laskenta, kysely, ..., lopetus)
- Yhteistyön testaus suorituspolku (thread) -testauksena

 27

Oliopohjainen testaus

- Käyttötapauspohjainen testaus soveltuu validointitestaukseen ja suorituspolkutestaukseen
- Käyttötapaustestauksella voidaan testata myös käyttöliittymää
- Skenaariot (esimerkkitapaukset) käyttötapauksen ilmentyminä - toimintasarja
  - tyypilliset tapaukset
  - poikkeustilanteet

 28

3. Virheenjäljitys

- Onnistunut testi johtaa tarpeeseen jäljittää virhe
- Varsin huonosti systematisoitu alue
  - virhe voi olla kaukana sen ulkoisesta ilmenemispaikasta (so., häiriöstä)
  - virheen korjaaminen voi tilapäisesti estää muiden virheiden ilmenemisen
  - virheen korjaaminen saattaa aiheuttaa uusia virheitä
  - virheellistä toimintaa voi olla vaikea toistaa
  - virhe voi johtua ohjelmointikielen tai laitteistoarkkitehtuurin erityispiirteistä

Virheenjäljitys

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

Virheenjäljitys

- peruutus
  - lähdetään havaitusta häiriöstä ja peruutetaan mahdollisia suorituspolkuja pitkin
  - tietovuoviipaloijat (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

Korjaus

- Ennen löydetyn virheen korjausta, on syytä pohtia:
  - Esiintyykö samantyyppinen virhe myös muualla ohjelmassa?
    - Jos esiintyy, niin korjataan
  - Aiheuttaako muutos mahdollisesti uusia virheitä ohjelmaan?
    - Jäljitetään sivuvaikutukset (esim. viipaloijalla, HyperSoft)
  - Mitä olisi pitänyt tehdä toisin, ettei virhettä olisi syntynyt?
    - Vältetään jatkossa, oppiva tuotantoprosessi

