

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)

© Harri Laine, Jukka Paakki

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

© Harri Laine, Jukka Paakki

3

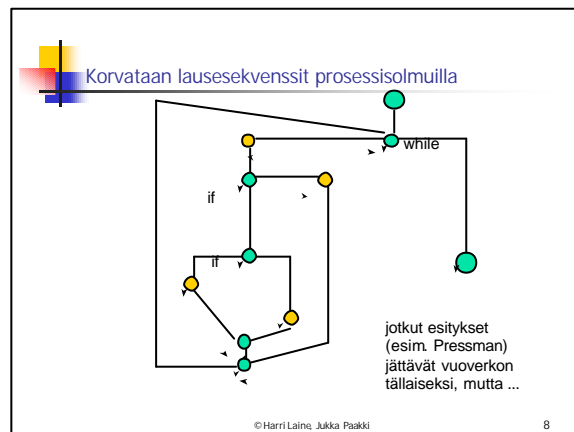
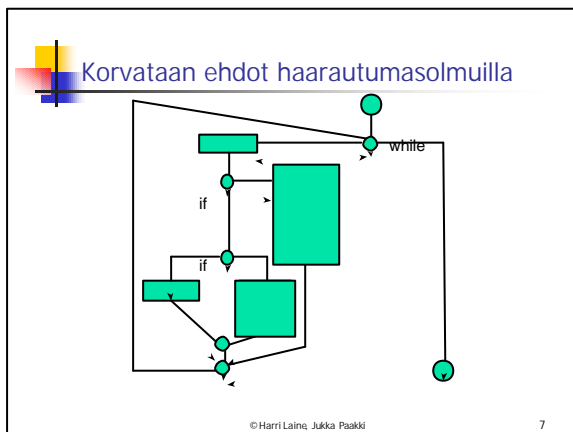
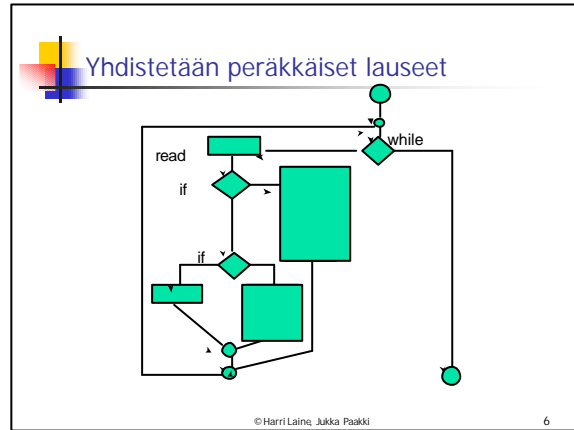
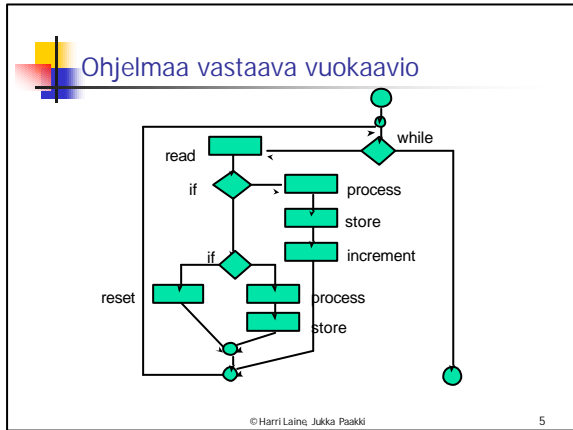
Ohjelmasta vuoverkoksi

```

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, Jukka Paakki

4



Voidaan jatkaa sieventämistä poistamalla prosessisolmut ja sijoittamalla toiminnallisuus särmiin

© Harri Laine, Jukka Paakki 9

Vuoverkon käsitteitä

- vuoverkon polku, reitti (path):
 - jono särmien (linkkien) yhdistämiä solmuja
 - sama solmu voi esiintyä polulla useaan kertaan (silmut)
- 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 linkejä (esimerkissä 1)
 - loppusolmu: ei ulosmeneviä linkejä (esimerkissä 7)

© Harri Laine, Jukka Paakki 10

Vuoverkon käsitteitä

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

© Harri Laine, Jukka Paakki 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)

© Harri Laine, Jukka Paakki 12

Vuoverkon käsitteitä

haarautumasolmut

$V(G) = E - N + 2 = 9 - 7 + 2 = 4$
 $V(G) = P + 1 = 3 + 1 = 4$

© Harri Laine, Jukka Paakki 13

Polkustaus

- Polkustauksessa on tarkoitus valita testitapaukset siten, että ne aiheuttavat tiettyjen vuoverkon polkujen suorituksen
- On määritelty erilaisia kattavuusmittoja polkustaukselle:
 - polkukattavuus (path coverage)
 - käydään läpi kaikki mahdolliset täydelliset polut (polku / testiajo)
 - useimmiten mahdotonta, koska polkuja on liian paljon

© Harri Laine, Jukka Paakki 14

Polkustaus

- 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 sarmät) läpi vähintään kerran testiajojen aikana
 - yleisesti tavoiteltu kattavuus: vahvempi ehto kuin lausekattavuus eli pakottaa testaamaan enemmän

© Harri Laine, Jukka Paakki 15

Polkustaus

- 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 testiajojen aikana (koska virheet kasautuvat silmukoihin)
 - iterointi 0 kertaa (silmukan ohitus), 1 kerran, tyypillinen määrä, maksimimäärä, maksimi + 1 kertaa
 - tarkentaa haarakattavaa testausta

© Harri Laine, Jukka Paakki 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 ("kuollutta koodia")

© Harri Laine, Jukka Paakki 17

Polkutestaus – esimerkki

Ohjelma laskee kokonaislukujen summan syötteenä annettuun ylärajaan saakka:

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

- 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?

© Harri Laine, Jukka Paakki 18

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); }
    
```

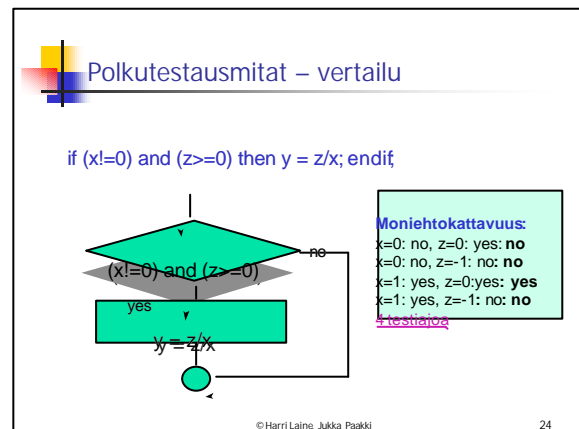
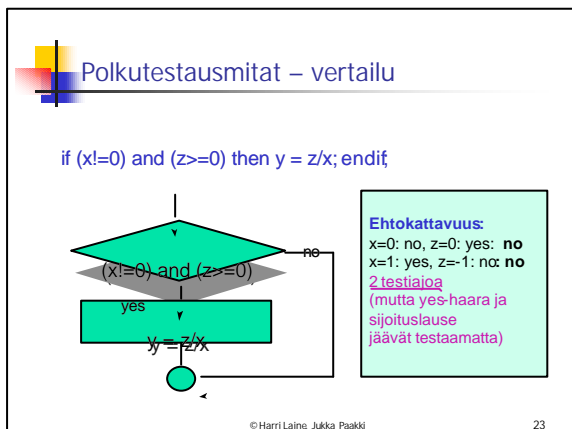
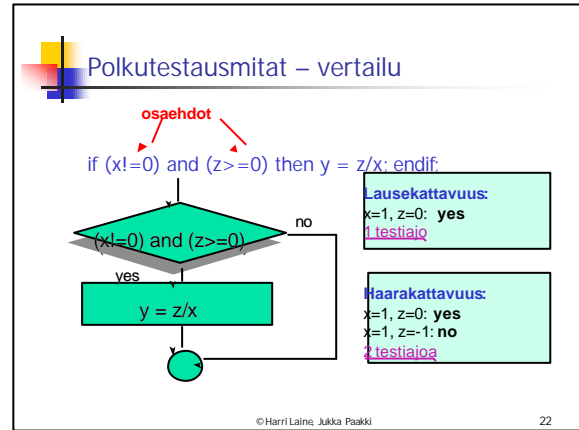
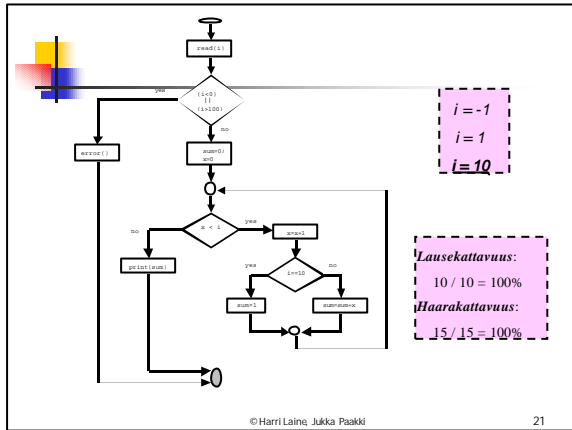
© Harri Laine, Jukka Paakki 19

$i = -1$
 $i = 0$
 $i = 1$
 $i = 50$
 $i = 99$
 $i = 100$
 $i = 101$

Bausekattavuus:
 $9 / 10 = 90\%$
Häarakattavuus:
 $13 / 15 = 87\%$

Ei suoriteta !!

© Harri Laine, Jukka Paakki 20



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)

© Harri Laine, Jukka Paakki 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 kehittyi luokkahierarkiassa
 - palvelun valinta perustuu polymorfismiin, jolloin virheelliseen valintaehtoon (case / switch) liittyviä ongelmia ei esiinny

© Harri Laine, Jukka Paakki 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

© Harri Laine, Jukka Paakki 27

Oliopohjainen testaus

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

© Harri Laine, Jukka Paakki 28

3. Virheenjäljitys

- Onnistunut testi johtaa tarpeeseen jäljittää virhe
- Varsin huonosti systematisoitu alue
 - virhe voi olla kaukana sen ulkoisesta ilmenemispaiasta (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

© Harri Laine, Jukka Paakki 29

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

© Harri Laine, Jukka Paakki 30

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 syyt havaintojen perusteella

© Harri Laine, Jukka Paakki 31

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

© Harri Laine, Jukka Paakki 32