Ohjelmistotuotanto

Ohjelmistosuunnittelu 1

1

Suunnittelu

- Suunnittelu:
  - asiakaslähtöisten vaatimusten muuntaminen teknisiä mahdollisuuksia tehokkaasti hyödyntäväksi ratkaisuksi
  - luova, asiantuntemusta vaativa prosessi
- Pääkohteet:
  - ohjelmiston yleisrakenne ja toimintaperiaatteet
  - tietorakenteet (ulkoiset, sisäiset)
  - algoritmit, toiminnot
  - käyttöliittymä

 2

Suunnittelu

- Suunnittelun tuloksena on tekninen malli järjestelmästä, joka aiotaan tehdä
- Kaksi hallinnollista kokonaisuutta: yleissuunnittelu ja yksityiskohtainen suunnittelu
- yleissuunnittelu

  (product design / system design / preliminary design)
  - järjestelmän kokonaisrakenne
  - yhteiset tietorakenteet
  - jako pääkomponentteihin (alijärjestelmät, moduulit)
  - pääkomponenttien välisen kommunikoinnin periaatteet
  - ohjelmistoarkkitehtuuri

 3

Suunnittelu

- yksityiskohtainen suunnittelu

  (detailed design / program design / module design)
  - kunkin komponentin sisäinen rakenne
  - algoritmit
  - rakenneosien yhteistyö
- Suunnitteludokumentti syntyy usein vastaavasti kahdessa vaiheessa:
  - yleissuunnitelma (arkkitehtuuri)
  - komponenttisuunnitelmat

 4

Suunnittelu

- *Tietosuunnittelu* (data design) muuntaa tietosisältömallin tietorakenteiksi
  - ulkoiset tietorakenteet (tietokantasuunnnittelu)
    - tietokantakaaviot
  - sisäiset tietorakenteet
    - luokkakaaviot
- *Arkkitehtuurisuunnittelu* (architecture design) määrittelee ohjelmiston korkean tason komponentit ja niiden väliset suhteet
  - luokkakaaviot, olioiden yhteistyökaaviot, arkkitehtuurimallit, kehykset

 5

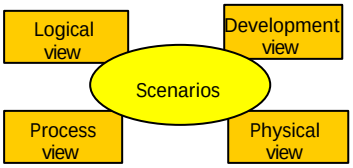
Suunnittelu

- *Liittymäsuunnittelu* (interface design) määrittelee, miten ohjelmiston ulkoiset liittymät toimivat
  - käyttöliittymä
  - laitteistoliittymät
  - tietoliikenne
- *Toimintosuunnittelu* (procedural design) kuvaa yksityiskohtaisesti toiminnallisuden liittämisen arkkitehtuurisuunnitelmassa tunnistettuihin komponentteihin

 6

Arkkitehtuuri - ohjelmiston yleisrakenne

- P. Kruchten: The 4+1 View Model of Architecture, IEEE Software, Nov 1995, pp 42-50.
- Arkkitehtuuria on on syytä tarkastella eri näkökulmista



 7

Arkkitehtuuri - ohjelmiston yleisrakenne

- Logical view (looginen rakenne, sisältömalli)
  - esittää järjestelmän loogiset pääosat
    - alijärjestelmät, tärkeimmät luokat ja niiden väliset yhteydet
    - staattiset rakenteet, liittymät
    - yhteistoiminta osien välillä
  - tavoitteena on kuvan välittäminen sisällöstä kokonaisuutena
  - kuvaustekniikat: luokkakaavio, oliokaavio, yhteistyökaaviot, tietovirtakaaviot

 8

Arkkitehtuuri - ohjelmiston yleisrakenne

- Process view (toiminnan organisointi)
  - esittää kontrollin kulun järjestelmässä
  - kuvaa prosessit, säikeet, rinnakkaisuuden
  - voidaan suhteuttaa sisältömalliin esimerkiksi määrittelemällä osajärjestelminä
  - samasta osajärjestelmästä (prosessirakenteesta) voi olla useita instansseja (prosesseja)
  - osajärjestelmien kytkennät ja yhteistyö voidaan kuvata esim. (luokkakaaviolla, oliokaavioilla,) yhteistyö- ja sekvenssikaavioilla, tietovirtakaavioilla

 9

Arkkitehtuuri - ohjelmiston yleisrakenne

- Development view (ohjelmarakenne, toteutusnäkökulma)
  - ohjelmakomponenttien paketointi ja jako tiedostoihin
  - ohjelmistotekninen rakenne
  - minkälaisiksi kokonaisuuksiksi ohjelmisto pilkotaan?
    - kirjastot
    - komponentit
    - kehykset
    - millä perusteilla kootaan (kerrokset,...)
  - osien yhteenliittyminen voidaan kuvata luokka-, olio- ja komponenttikaavioilla tai moduulirakenteena
  - UML:ssä myös pakkauskaavio (package diagram)

 10

Arkkitehtuuri - ohjelmiston yleisrakenne

- Physical view (laitteistosijoittelu, hajautus)
  - ohjelmiston sijoittelu verkkoon ja laitteistoon
  - ohjelmiston ja prosessien sijoittelu
  - UML:ssä tarjolla erillinen kaavio (deployment diagram)
  - Sijoiteltavia:
    - prosessit - prosessisijoittelu
    - ohjelmat - ohjelmasijoittelu
      - esim. WWW-sovelluksissa komponentti (esim. Java-sovelma) voi olla sijoitettu palvelimelle, mutta sitä suoritetaan työasemassa

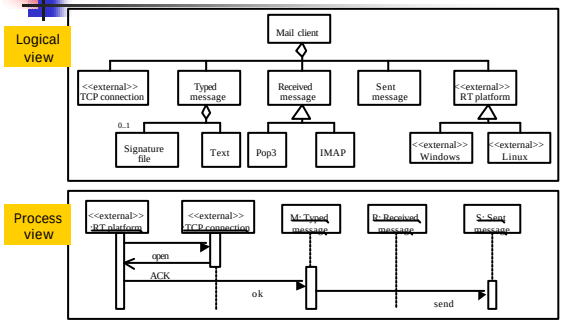
 11

Arkkitehtuuri - ohjelmiston yleisrakenne

- Scenarios (skenaariot, käyttönäkökulma)
  - tyypillisen käytön esimerkkitapauksia
    - kytkennät erityisesti sisältömalliin ja prosessimalliin
    - yhteistyö- ja sekvenssikaaviot keskeisiä (UML: collaboration diagram, sequence diagram)

 12

Ohjelmistoarkkitehtuuri: UML



Suunnittelu: osiin jakamista ja osien kytkemistä

- Suunnittelussa kokonaisuus jaetaan osiin
- Jakamisella pyritään saavuttamaan:
  - toiminnallinen itsenäisyys
    - helpompi tehdä, testata ja ylläpitää
  - työnjako
    - ohjelmistoprojektissa
    - järjestelmän toiminnassa
  - käsiteltävyys
    - sopiva tarkkuustaso
    - sopiva laajuus

Suunnittelu: osiin jakamista ja osien kytkemistä

- Jakamisessa ei ole yksikäsitteistä "parasta" ratkaisua
  - jako perustuu johonkin näkökulmaan, jonka mukaan tietyt palvelut ja tiedot kuuluvat samaan kokonaisuuteen
  - jakoperusteena voi olla:
    - käyttäjä, toiminto / tehtävä, laitteisto, toimipiste
    - toiminnan kohde, ajoitus, jne.
  - jako on yleensä monitasoinen, näkökulma voi vaihdella tasoittain ja olla eri osajärjestelmissä erilainen (vrt. “4+1 View Model”)

Kerrostus

- Suunnittelun ositusperiaatteet:
  - kerrostus (layering)
  - pilkkominen (partitioning)
- Kerrostuksen perustana abstraktiotasot:
  - toiminta-abstraktiot
    - abstrakti kone -ajattelu
    - ylemmän kerroksen palvelut tuotetaan alemman kerroksen palveluiden avulla (vrt. tietoliikenteen OSI-malli)
    - kerrokset perustuvat eri käsitteistöihin
    - ylin kerros vastaa järjestelmän päätoimintoja ja käyttöliittymää, alin kerros usein fyysistä laitteistoa

Kerrostus

- suljetussa kerrosarkkitehtuurissa vain välittömästi alemman tason palvelut käytettävissä
  - parempi ylläpidettävyys
- avoimessa kerrosarkkitehtuurissa kaikkien alempien tasojen palvelut käytettävissä
  - huonompi ylläpidettävyys
  - parempi joustavuus
  - esimerkki kerrostuksesta:
    3. Käyttöliittymäkerros
    2. Looginen toiminta –kerros
    1. Käyttöjärjestelmäkerros
- laitteiston/järjestelmän eristäminen rajapinnan taakse
  - JDBC API kätkee tietokannan hallinnan

Kerrostus

Alempi kerros tarjoaa palveluja ylemmän kerroksen käyttöön, ylempi kerros kutsuu alempaa

Kerrostus

- Tieto-abstraktiot
  - tietokantojen peruskäsitteitä
  - näkymät - looginen taso - fyysinen taso
  - abstraktit tietotyypit - oliot
  - tietokannan piilotus esim EJB-komponenttien taakse

 19

Pilkkominen

- **Pilkkominen** = toiminnallinen tai tiedollinen osiinjako, ilmenee puhtaimmillaan asteittain tarkentuvassa suunnittelussa (top down)
  - asteittain voi tarkentaa sekä toimintaa että tietoa
  - esim. JSD, JSP (Jackson Structured Design, Jackson Structured Programming)

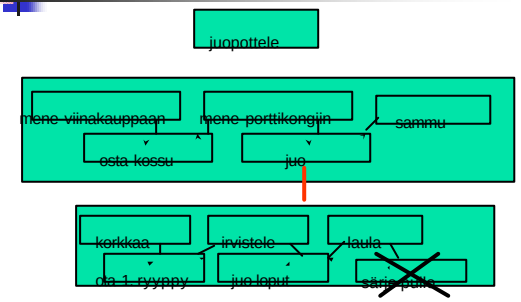
 20

Pilkkominen

1. Selvitä ongelman olennaiset osat
2. Hahmota ainakin yksi mahdollinen ratkaisu
   - mielellään useita ratkaisuja
   - simuloi keskeiset tilanteet
   - valitse yksinkertaisin ratkaisu, ellei ole erityisiä syitä valita toisin
3. Kuvaa järjestelmä hierarkkisesti tarkentaen
   - ylhäältä alaspäin (suurempi kokonaisuus → pienempi kokonaisuus)
   - kuvausmenetelmä sovelluksen mukaan
4. Muodosta vastaava rakennekaavio
5. Kuvaa kunkin tarkennustason oliot

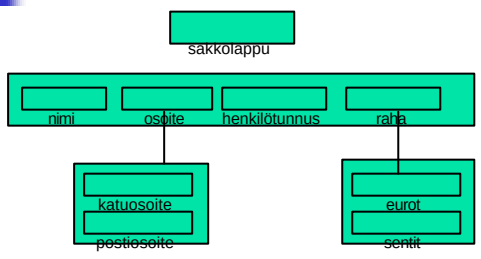
 21

Pilkkominen: toiminta



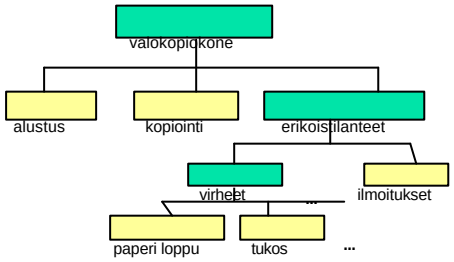
 22

Pilkkominen: data



 23

Pilkkominen: toiminnallinen hierarkia



 24

Yleiset suunnitteluperiaatteet

- Suunnitteluprosessin pitää olla laajakatseista. Hyvä suunnittelija miettii myös vaihtoehtoisia toteutustapoja.
- Olemassaolevia komponentteja tulee käyttää hyväksi suunnittelussa.
- Suunnittelussa pitää mahdollisuuksien rajoissa matkia ratkaistavan ongelma-alueen rakennetta.
- Suunnitelman pitää olla yhtenäinen ja hyvin integroitu. Yhtenäisyys tarkoittaa, että suunnitelma näyttää yhden ihmisen tuotteelta. Integrointi tarkoittaa, että komponenttien rajapinnat on suunniteltu huolellisesti.

 25

Yleiset suunnitteluperiaatteet

- Suunnittelman pitää tukea mahdollisia muutoksia.
- Myös poikkeustilanteisiin pitää varautua. Mahdollisen toipumisen pitää olla käyttäjäystävällistä. Jos toipuminen ei ole mahdollista, ohjelmiston tulee päättyä selkeästi ja antaa informatiivinen virheilmoitus.
- Suunnittelu- ja toteutusvaiheet tulee erottaa selkeästi toisistaan. Suunnitelma on abstraktio toteutuksesta.
- Suunnitelman laadusta täytyy pitää kiinni jo suunnittelun aikana eikä pelkästään korjata suunnitelmaa lopussa.

 26

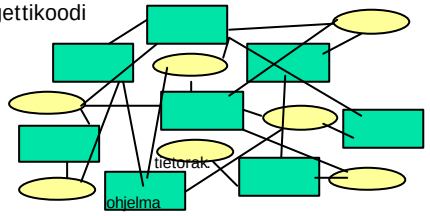
Yleiset suunnitteluperiaatteet

- Suunnitelmaa pitää arvioida suunnittelun aikana. Näin vähennetään suunnittelussa esiintyviä virheitä. Esim. formaalit tekniset tarkastukset sopivat erittäin hyvin tähän tarkoitukseen. Ylemmän abstraktiotason puutteet ja virheet pitää käsitellä ennen alemmalle abstraktiotasolle siirtymistä.
- Yksinkertaisuus on valttia jatkolle.

 27

Ohjelmarakenteita

spagettikoodi

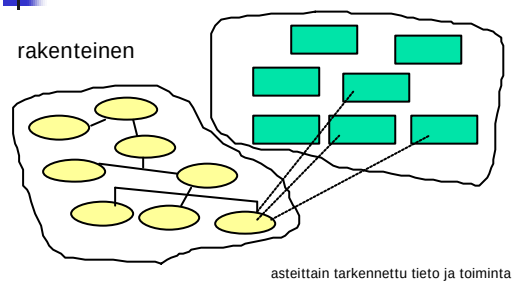


ohjelmointikikkoja, data ja koodi jäsentymätöntä

 28

Ohjelmarakenteita

rakenteinen

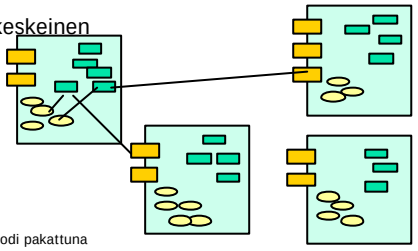


asteittain tarkennettu tieto ja toiminta
rakenteinen ohjelmointi

 29

Ohjelmarakenteita

oliokeskeinen



data ja koodi pakattuna
rajapintojen taakse

 30

Ohjelmarakenteita

Oliokeskeinen
- modulaarisuus
- abstraktit tietotyypit
  - datan ja koodin pakkaaminen
- tiedon piilottaminen, rajapinnat
- olio-ohjelmointikielet
- olioperustaiset suunnittelumenetelmät (esim. UML ja Rational Unified Process (RUP))
- raviolikoodi?

 31

Hyvä ohjelmistorakenne

Yleisiä ominaisuuksia, joihin on syytä kiinnittää huomiota:
- modulaarisuus (modularity)
- kiinteys (cohesion)
- kytkentä (coupling)
- tiedon kätkeminen (information hiding)
- muunneltavuus (modifiability)
- ylläpidettävyys (maintainability)

 32

Hyvä ohjelmistorakenne – modulaarisuus

- Modulaarisuus: ohjelmisto on jaettu erillisiin nimettyihin (usein erikseen käännettäviin) osiin eli moduuleihin
  - tietorakenteita
    - yleensä yksityisiä, käyttö operaatioiden kautta
    - moduuli voi tarjota myös jaetun tietorakenteen
  - operaatioita
    - julkisia tai yksityisiä
  - liittymä
    - julkiset operaatiot, joita muut moduulit voivat kutsua

 33

Hyvä ohjelmistorakenne – modulaarisuus

Modulaarisuus:

moduulin koko

Effort(p+q) > Effort(p) + Effort(q)

- Moduulien määrän kasvaessa liittymien toteutuskustannukset kasvavat, joten jatkuva pilkkominen pienemmiksi moduuleiksi ei poista kaikkia kustannuksia
- Suositeltu moduulikoko noin 100-200 riviä koodia, yksittäiset operaatiot 20-30 riviä
  - kerralla hahmotettavissa

 34

Hyvä ohjelmistorakenne – kiinteys

- Kiinteys (cohesion) (toiminnallinen)
  - moduuli on kiinteä, jos sen kaikki osat palvelevat yhtä ainoaa tehtävää, ts. moduuli toteuttaa yhden selkeän toiminnallisen kokonaisuuden
  - moduuli ei ole kiinteä, jos se sisältää heikosti toisiinsa liittyviä osia
  - kiinteys parantaa moduulin ylläpidettävyyttä ja lisää mahdollisuuksia uuskäyttöön

 35

Hyvä ohjelmistorakenne – kiinteys

Kiinteyden asteet huonosta hyvään:
- moduulissa on satunnaisesti yhdistettyjä osia
- moduuliin on koottu loogisesti samantyyppiset tehtävät(esim. tulostukset)
- moduuliin on koottu samaan aikaan tarvittavat osat (esim. alustukset)
- moduuliin on koottu kontrollin etenemisjärjestyksessä peräkkäisiä osia
- moduulin on koottu yhteisiä tietorakenteita käyttäviä operaatioita (olio)
- moduulin osat muokkaavat toistensa tuloksia: toisen tuote on toisen syöte
- toimintakokonaisuuteen perustuva yhdistäminen: jokainen osa omalta osaltaan tarpeen moduulin vastuulla olevan toimintakokonaisuuden kannalta

 36

Hyvä ohjelmistorakenne – kytkentä

- **Kytkentä (coupling)**
  - kuvaa vuorovaikutuksia eri moduulien välillä
  - **löyhästi kytkettyjä** moduuleja on helpompi ylläpitää, koska muutokset eivät vaikuta muihin moduuleihin
  - kiinteästi kytketyillä on voimakkaita riippuvuuksia
  - "kytkös": yleensä moduulin jonkin operaation / metodin kutsu
    - muita tyypillisiä kytköksiä: perintäsuhde, määrittelysuhde

 37

Hyvä ohjelmistorakenne – kytkentä

Kytkentä löyhästä vahvaan:

- ei vuorovaikutusta moduulien välillä
- käsiteltävän tiedon välittäminen yksittäisinä parametreina
- käsiteltävän tiedon välittäminen rakenteisina parametreina
- ohjaustiedon välittäminen parametreina
- yhteiskäyttöiset ulkoiset laitteet
- yhteiskäyttöinen tietorakenne
- toisten moduulien sisäisten rakenteiden hyväksikäyttö

 38

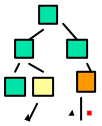
Hyvä ohjelmistorakenne – tiedon piilotus

- **Tiedon piilotus (information hiding)**
  - moduulia ajatellaan mustana laatikkona
  - vain liittymät näkyvät ulospäin :
    - syötteet
    - tulosteet
    - moduulin sisältämien toimintojen määrittely (kutsurajapinta)
  - ulos eivät näy, eivät ulkopuolelta käytettävissä:
    - moduulin sisäiset tietorakenteet
    - toimintojen toteutustapa
  - toteutus, testaus ja ylläpito paikallista
  - suojaus väärinkäyttöä vastaan tehostuu

 39

Hyvä ohjelmistorakenne

- Muita jaossa vaikuttavia asioita:
  - jakoa ei kannata jatkaa, jos liittymän koodi tulee pidemmäksi kuin toiminnan koodi **(paitsi yleiskäyttöisten osalta)**
  - yhteiskäyttöisten osien eristäminen omiksi moduuleikseen lyhentää koodia ja helpottaa ylläpitoa
  - työ ja päätöksenteko kannattaa erottaa eri moduuleihin, muutokset kohdistuvat yleensä vain jompaan kumpaan
    - tässä yhteydessä on kuitenkin pyrittävä välttämään toimivaltarikkomuksia = tilanteita , joissa päätöksen vaikutusalue menee moduulin valvonta-alueen (= alaiset kutsurakenteessa) ulkopuolelle
  - kannattaa etsiä yleiskäyttöisiä moduuleita



päätös hyödyntäminen

toimivaltarikkomus

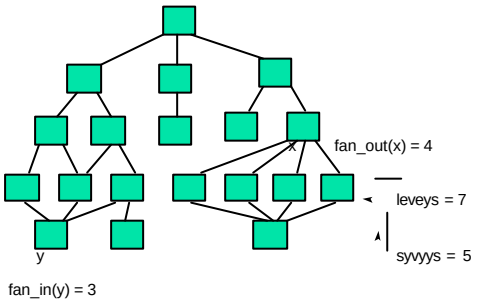
 40

Hyvä ohjelmistorakenne

- Muita jaossa vaikuttavia asioita:
  - liittymien pitäisi olla mahdollisimman yksinkertaisia, ei rakenteista tietoa
  - moduulirakenteeseen syvyyttä leveyden kustannuksella
    - kuitenkin: luokkahierarkiassa syvää perintää vältettävä, koska se haittaa ymmärrettävyyttä ja ylläpidettävyyttä
  - moduulin toiminnan tulee olla ennustettavissa – ts. sisäinen tila ei saisi vaikuttaa (black box)
  - moduuleilla pitäisi olla vain yksi sisäänmeno- ja yksi ulostulokohta (esim. määritelty kutsurajapinta)
  - mittareita: ***fan_in*** (moduulia kutsuvien moduulien määrä), ***fan_out*** (moduulin kutsumien moduulien määrä), moduulihierarkian ***maksimisyvyys*** ja ***maksimileveys***

 41

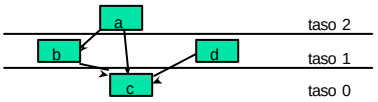
Moduulihierarkia – kutsuriippuvuus



 42

Moduulihierarkia – kutsuriippuvuus

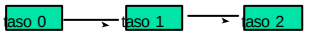
- **Moduulihierarkian suunnittelu:**
  - relaatio ***A uses B*** on voimassa, mikäli moduulille A määritelty tehtävä vaatii moduulin B käyttämistä (kutsumista)
  - relaation ***uses*** tuottama riippuvuusverkko on kehätön =>
    - verkko muodostaa moduulien (puumaisen) hierarkian
  - tavoitteena on kehätön verkko
  - esim: a ***uses*** b, b ***uses*** c, a ***uses*** c, d ***uses*** c



 43

Moduulihierarkia – kutsuriippuvuus

- nyt jokainen taso koostuu moduulien osajoukosta, joka voidaan **toteuttaa, suorittaa ja testata** erikseen **inkrementaalisesti**:



testausjärjestys integrointitestauksessa

 44