HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
5. Tietojenkäsittelyn mekaniikat

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Kurssin sisältö

Lähde: Peter J. Denning: Great Principles of Computing (Communications of the ACM, 46, 11, marraskuu 2003, sivut 15-20).

Luku 1: Historiaa
Luku 2: Kokonaiskuva
Luku 3: Eettiset perusteet
Luku 7: COMPUTING PRACTICES

programming
engineering systems
modeling
innovating
applying

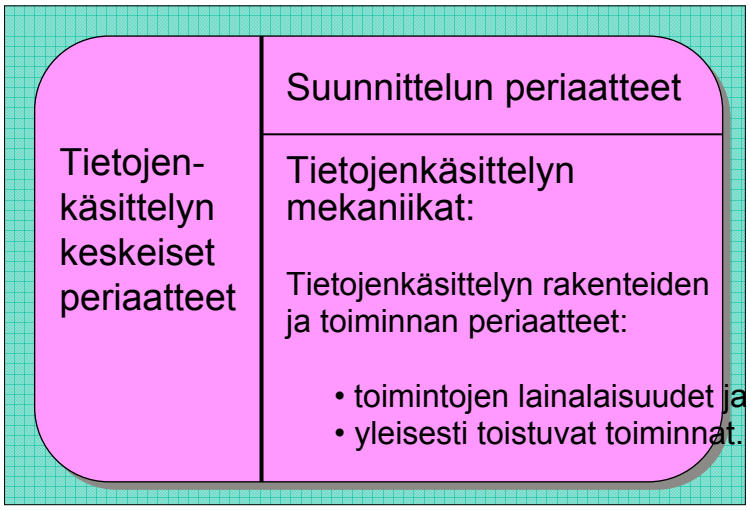
Luku 4: CORE TECHNOLOGIES

architectures
artificial intelligence
databases
operating systems
algorithms
networks
ETC....

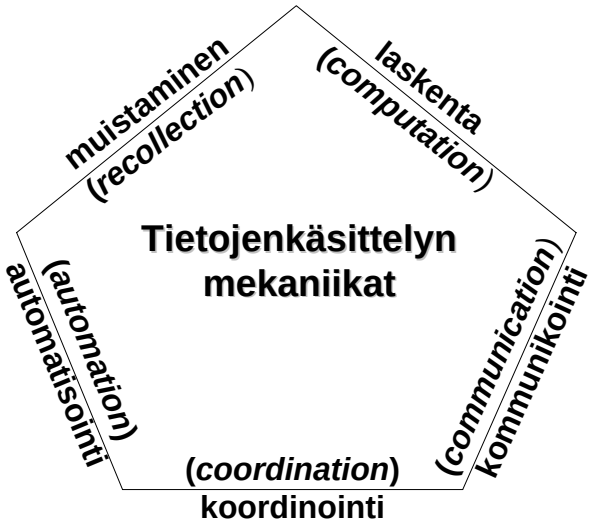
Luku 6: DESIGN
simplicity, performance, reliability,

GREAT PRINCIPLES OF COMP

Tietojenkäsittelyn mekaniikat (*mechanics*)



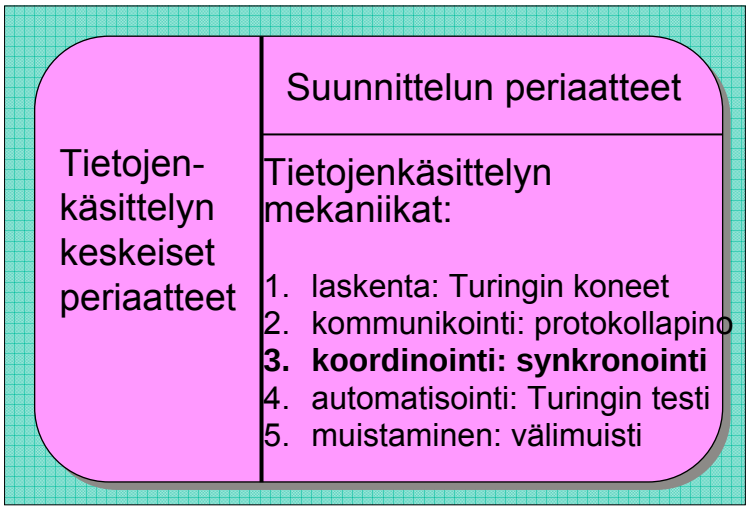
Viisi näkymää tietojenkäsittelyn mekaniikkoihin



Näkymät lyhyesti

1. Laskenta.
   - Mitä voidaan laskea – laskennan rajat.
2. Kommunikointi.
   - Sanoman tai viestin lähettäminen paikasta toiseen.
3. Koordinointi.
   - Vähintään kaksi toimijaa ja yhteinen tavoite.
4. Automatisointi.
   - Tietokoneella suoritettavat kognitiiviset tehtävät.
5. Muistaminen.
   - Tiedon tallettaminen ja hakeminen.

Viisi tarinaa tietojenkäsittelyn mekaniikoista



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
5. Tietojenkäsittelyn mekaniikat
5.3 Koordinointi: synkronointi

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Koordinoinnin tarinoita

- Ihmisten välinen (*human-to-human*)
- Ihmisen ja tietokoneen välinen (*human-computer*)
- Tietokoneiden välinen (*computer-computer*)
  - Synkronointi (*synchronization*)
  - Kilpatilanteet (*race*)
  - Lukkiutuminen (*deadlock*)
  - Sarjallistuvuus (*serializability*)
  - Atomiset toimenpiteet (*atomic actions*)

Synkronointi

- Synkronointi eli tahdistus on aikaan liittyvää koordinointia.
- Esimerkiksi monikanavaviestinnässä eri kanavien tietovirtojen tulee edetä samaa tahtia.
- Toisissa tilanteissa synkronoinilla varmistetaan, että itsenäisten toimijoiden (väli)tulokset valmistuvat oikeassa järjestyksessä.

Kellojen synkronointi

- Eräs hajautetun tietojenkäsittelyn perusongelma.
- GPS – *Global Positioning System*.
- NTP – *Network Time Protocol*.

Synkronoinnin keskeiset ongelmat

- Liittyvät usein yhteiskäyttöisten resurssien hallintaan.
- Ratkaisuiden on estettävä sekä nälkiintyminen että lukkiutuminen.
  - Nälkiintyminen (*starvation*): joku joutuu odottamaan vuoroaan ikuisesti.
  - Lukkiutuminen (*deadlock*): kaikki odottavat, että joku toinen tekisi jotain.

Tuottaja – kuluttaja ongelma

- Tuottajan tuotettava ennen kuin kuluttaja voi kuluttaa.
- Välivaraston (puskurin) täyttyessä on tuottajan odotettava, että kuluttaja kuluttaa.
- Kun tuottajia ja kuluttajia on useita, niin on myös huolehdittava poissulkemisesta.
  - Tuottajien tuotokset on saatava välivarastossa eri paikkoihin.
  - Kaksi kuluttajaa ei saa saada samaa tuotosta.
  - Samanaikaiset lisäykset ja poistot eivät saa sotkea varastokirjanpitoa.

Lukija – kirjoittaja ongelma

- Kaikki voivat lukea samanaikaisesti, mutta vain yksi kerrallaan voi kirjoittaa.
- Kirjoittajan nälkiintyminen on estettävä.
  - Jos kirjoittaja ei saa kirjoitusvuoroa, niin kaikki lukijat lukevat vanhentunutta tietoa.

Kilpatilanne (*race condition*)

- Virhe järjestelmän suunnittelussa.
- Kaksi signaalia kilpailee siitä, kumpi pääsee ensin vaikuttamaan tulokseen.
  - Kaksi tai useampi ohjelma pääsee synkronoimattomasti käsiksi yhteiskäyttöiseen resurssiin samanaikaisesti.

```
global integer A = 0;
task Received()
    { A = A + 1; }
task Timeout()
// Print only the even numbers
    { if (A is divisible by 2)
          { print A; } }
```

- Received aktivoituu aina, kun sarjaportti aiheuttaa keskeytyksen eli vastaanottaa dataa.
- Timeout aktivoituu kerran sekunnissa (kellolaitekeskeytys).

Mahdollinen suoritusjärjestys

```
        global integer A = 0;
3.      task Received()
4.              { A = A + 1; }
1.      task Timeout()
        // Print only the even numbers
2.              { if (A is divisible by 2)
5.                      { print A; } }
```

- Timeout tulostaakin parittoman A:n arvon, jos askeleessa 2 oli parillinen A.

Aterioivat filosofit

http://www.doc.ic.ac.uk/~jnm/concurrency/classes/Diners/Diners.html

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
5. Tietojenkäsittelyn mekaniikat
5.4 Automatisointi: Turingin testi

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Automatisoinnin tarinoita

- Kognitiivisten tehtävien simulointi (*simulation of cognitive tasks*)
- Automatisoinnin filosofia (*philosophical distinctions about automation*)
- Asiantuntemus ja asiantuntijajärjestelmät (*expertise and expert systems*)
- Älykkyyden lisääminen (*enhancement of intelligence*)
- Turingin testit (*Turing tests*)
- Koneoppiminen ja tunnistaminen (*machine learning and recognition*)
- Bioniikka (*bionics*)

Turingin testi

- Ihminen kirjoittaa kahdelle testattavalle (ihminen ja tietokone) kysymyksiä.
- Saamiensa kirjallisten vastausten perusteella hän yrittää päätellä, kumpi vastaajista on tietokone.
- Perustuu englantilaiseen seurapiirileikkiin *Imitation game*.
- Turing ehdotti testiään älykkyyden määritelmäksi 1950.
  - Keskustelua käytiin värikkäästi tietokoneen kyvyistä.
    - Ajatteleva kone (*thinking machine*).
    - Sähköaivot.
- Nykyisin Turingin testiä pidetään marginaali-ilmiönä tekoäly-yhteisössä.

Turingin testin arvostelua

- Toimii kuin ihminen ei ole järkevä tavoite.
  - Ihminen tekee virheitä.
- Lentäminen perustuu aerodynamiikkaan.
  - Linnun hyväkään matkiminen ei riitä.
- John Searlen kiinalainen huone.
- Älykkyys ei ole matkimista.
  - Tietoisuus.
  - Tavoitteellisuus.

Turingin testin läpäisemisen edellytyksiä

1. Luonnollisen kielen käsittely.
2. Tietämyksen esittäminen.
3. Automaattinen päättely.
4. Koneoppiminen.

- Kaikki tietojenkäsittelyn, erityisesti tekoälyn, keskeisiä ongelmia.

Osaongelmien tunnuspiirteitä

1. Luonnollisen kielen käsittely.
   - Järjestelmän on ymmärrettävä sille esitetyt kysymykset ja muotoiltava vastaukset.
   - Eräs ongelma: yksi sana – monta eri asiayhteydestä riippuvaa merkitystä.
2. Tietämyksen esittäminen.
   - Järjestelmän on talletettava tietämänsä ja kuulemansa.
   - Tietämyksen laajetessa talletuksen ja etsinnän tehokkuudet voivat olla ongelmallisia.

Osaongelmien tunnuspiirteitä

3. Automaattinen päättely.
   - Järjestelmän on talletetun informaation perusteella osattava tehdä päätelmiä, joiden perusteella vastaus voidaan muotoilla.
   - Loogisen päättelyn jotkut ongelmat ovat todistetusti ratkeamattomia, joten järjestelmän on myös osattava päätellä, mitä se ei pysty päättelemään.

4. Koneoppiminen.
   - Järjestelmän on sopeuduttava uusiin tilanteisiin.
   - Järjestelmän ”maailmankuva” muuttuu uuden datan perusteella.
     - Maailmankuvalla tarkoitetaan järjestelmän käytössä olevaa mallia kiinnostavista asioista.

Turingin testin läpäisemisestä

- Yleisesti pidetään mahdottomana.

- ”Oikea vastaus väärään kysymykseen”.
  - Yleensä hyödytön.
  - Turingin testin tapauksessa hyödyllinen: osaongelmien ratkaisut olisivat hyödyllisiä.
    - Luonnollisen kielen automaattinen käsittely (ihmisen ja koneen vuorovaikutus).
    - Tietämyksen automaattinen esittäminen ja käyttäminen.
    - Automaattinen päättely (mitä käyttäjä seuraavaksi haluaa).
    - Koneoppiminen.

- Järjestelmän tulee toimia siten kuin ihminen HALUAA.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
5. Tietojenkäsittelyn mekaniikat
5.5 Muistaminen: välimuisti

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Muistamisen tarinoita

- Muistihierarkiat (*hierarchies of storage*)
- Viittausten paikallisuus (*locality of reference*)
- Välimuistit (*caching*)
- Osoiteavaruudet ja niiden kuvaukset (*address space and mapping*)
- Nimeäminen (*naming*)
- Yhteiskäyttö (*sharing*)
- Haku nimen perusteella (*retrieval by name*)
- Haku sisällön perusteella (*retrieval by content*)

Välimuistit

- Miten saadaan dataa nopeasti käsiteltäväksi.
  - Prosessori ↔ keskusmuisti.
  - Keskusmuisti ↔ massamuisti.
  - Internetin siirtoviiveet.
- Data talletetaan väliaikaisesti käsittelypaikan lähelle.
  - Prosessorin välimuisti.
  - Tiedostovälimuisti.
  - Webin välimuisti.

Välimuistit

- Välimuistit ovat osoittautuneet keskeisiksi tietokoneiden suoritusnopeuden kasvattamisessa.
- Datan käsittely on usein paikallista (*locality of reference*).
  - Tietoalkiota käsitellään useita kertoja ajallisesti lähekkäin.
  - Lähekkäin olevia tietoalkioita käsitellään usein ajallisesti lähekkäin.
    - Tietorakenteen alkiot.
    - Tiedoston läpikäynti.

Välimuistin hallinta

- Välimuisti on pieni verrattuna varsinaiseen muistiin.
- Välimuistista poisto (*replacement policy*).
  - LRU: *least recently used.*
- Kirjoitus varsinaiseen muistiin, kun välimuistiin on kirjoitettu (*write policy*).
  - Läpikirjoitus (*write-through cache*).
  - Viivästetty kirjoitus (*write-back cache*).
- Eheysprotokolla (*coherency protocol*).
  - Sama tieto välimuistissa ja varsinaisessa muistissa.
  - Vanhentuminen (*stale*).

Prosessorin välimuisti

- Prosessorit nopeutuneet enemmän kuin muistipiirit ja väylät.
- Prosessorilla pieni ja nopea välimuisti.
  - Laitteistototeutus.
- Nykyisin useita erikoistuneita välimuisteja.
- Tavoite: saantiviiveen (*access latency*) lyhentäminen.

Muistien nopeusongelmia

- 1970-luku: supertietokoneiden ongelma.
- 1980-luku: graafisten työasemien ongelma.
- 1990-luku: pöytäkoneiden ongelma.
- 2000-luku: prosessori suorittaa satoja käskyjä yhden keskusmuistista noudon aikana.

Virtuaalimuisti

- Jokaisella suoritettavalla ohjelmalla on illuusio, että se on yksin tietokoneessa ja voi käyttää omaa osoiteavaruutta (*address space*).
- Prosessori muuttaa virtuaaliosoitteet todellisiksi muistiosoitteiksi.
- Prosessoreissa on yleensä osoitteenmuunnokset tekevä laitteisto.
  - Muistinhallintayksikkö (*memory management unit, MMU*).
  - MMU:lla on yleensä oma välimuisti.
    - Osoitteenmuunnospuskuri (*translation lookaside buffer, TLB*).

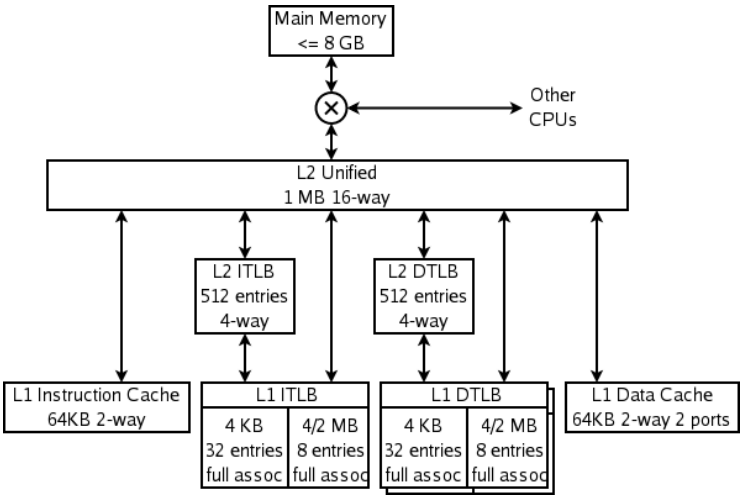
Prosessorin käskynkäsittely

- Liukuhihnoitus (*pipelining*) nopeuttaa prosessoreja.
- Idea: käskyn suoritus voidaan jakaa vaiheisiin:
  - Käskyn nouto (*instruction fetch*).
  - Osoitteenmuunnos (*address translation*).
  - Tietoalkion nouto (*data fetch*).
- Muistiin on päästävä käsiksi jokaisessa vaiheessa.
  - Jokaisella vaiheella oma välimuisti.
  - Laitteistotasolla ei tarvitse varautua kilpatilanteeseen.

Välimuistien hierarkia

- Nykyisissä prosessoreissa on yleensä hierarkinen välimuistijärjestelmä.
- Suunnittelussa joudutaan tekemään kompromisseja (*tradeoff*).
  - Mitä laajempi välimuisti sitä hitaampi, mutta osumistodennäköisyys (*hit ratio*) suurempi eli noudettava tieto on välimuistissa useammin.
  - Nykyisissä prosessoreissa on yleensä ainakin kaksi välimuistitasoa.

AMD Athlon 64 prosessorin välimuistit



HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
6. Suunnittelu

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Kurssin sisältö

Lähde: Peter J. Denning: Great Principles of Computing (Communications of the ACM, 46, 11, marraskuu 2003, sivut 15-20).

Luku 1: Historiaa
Luku 2: Kokonaiskuva
Luku 3: Eettiset perusteet
Luku 7: COMPUTING PRACTICES

programming
engineering systems
modeling
innovating
applying

Luku 4:
CORE TECHNOLOGIES

architectures
artificial intelligence
databases
operating systems
algorithms
networks
ETC. ...

GREAT PRINCIPLES OF COMPUTING

Luku 5: MECHANICS
computation, communication, coordination, automation, recollection

Suunnittelu

Tietojen-käsittelyn keskeiset periaatteet

Suunnittelun periaatteet:

- yksinkertaisuus
- suorituskyky
- luotettavuus
- kehitettävyys
- tietoturva

Tietojenkäsittelyn mekaniikat

Suunnittelun vakiintuneita käytäntöjä

- Abstrahointi (*abstraction*).
  - Epäolennaisten yksityiskohtien häivyttäminen.
- Informaation piilottaminen (*information hiding*).
  - Ohjelmiston osan (moduulin) sisäisten tietojen ja tietorakenteiden piilottaminen muilta ohjelmiston moduuleilta.
- Moduulit (*modules*).
  - Ohjelmiston jakaminen osiin siten, että osien väliset vuorovaikutukset ja rajapinnat ovat hyvin määriteltyjä.
- Erikseen kääntäminen (*separate compilation*).
  - Moduulien kääntäminen erikseen ja linkittäminen myöhemmin suorituskelpoiseksi kokonaisuudeksi.

Suunnittelun vakiintuneita käytäntöjä

- Pakkaukset (*packages*).
  - Jakelu- ja asennusyksikkö, johon on koottu ohjelmisto(je)n osat ja dokumentaatiot.
- Versionhallinta (*version control*).
  - Menetelmät, joilla hallitaan ohjelmiston kehitystyötä ja sen aikana syntyviä ohjelmaversioita.
- Hajota ja hallitse (*divide-and-conquer*).
  - Suuret kokonaisuudet jaetaan pienempiin ja helpommin hallittaviin osakokonaisuuksiin.
- Toimintatasot (*functional levels*).
  - Toiminnallisuuden ryhmittely eri tasoisiksi toimenpiteiksi.

Suunnittelun vakiintuneita käytäntöjä

- Kerrosajattelu (*layering*).
  - Toimintakokonaisuuden jakaminen kerroksiin siten, että kukin kerros tarjoaa joitakin (muutamia) palveluja ylemmälle kerrokselle ja käyttää alemman kerroksen palveluja.
- Hierarkiat (*hierarcy*).
  - Suunnittelussa hierarkiat liittyvät yleensä olio-ohjelmoinnin luokkarakenteeseen.
- Ongelmien eriyttäminen (*separation of concerns*).
  - Periaate, jonka mukaan keskitytään kerrallaan yhteen, hyvin määriteltyyn (osa)tehtävään.
- Uudelleenkäyttö (*reuse*).
  - Olemassa olevien moduulien, määritysten, suunnitelmien jne käyttäminen sen sijaan, että tehtäisiin uudestaan.

Suunnittelun vakiintuneita käytäntöjä

- Rajapinta (*interface*).
  - Rajapinnan välityksellä ohjelmisto- tai laitteistokomponentti tarjoaa toiminnallisuutensa muiden komponenttien käyttöön.
- Virtuaalikone (*virtual machine*).
  - Ohjelmisto, joka toteuttaa määritellyn abstraktin koneen toiminnallisuuden.

Suunnittelun viisi tavoitetta

1. Yksinkertaisuus (*simplicity*).
   - Abstraktiot ja rakenteet, joilla hallitaan sovelluksen luontaista monimutkaisuutta.
2. Suorituskyky (*performance*).
   - Suoritustehon (*throughput*) ja vasteajan (*response time*) ennustaminen, pullonkaulojen (*bottlenecks*) paikallistaminen sekä kapasiteetin suunnittelu (*capacity planning*).
3. Luotettavuus (*reliability*).
   - Päällekkäisyys (*redundancy*), toipuminen (*recovery*), varmistaminen (*checkpointing*), eheys (*integrity*) ja luottamus (*trust*).

Suunnittelun viisi tavoitetta

4. Kehitettävyys (*evolvability*).
   - Varautuminen toiminnallisuuden ja käytön laajuuden muutoksiin.
5. Tietoturva (*security*).
   - Pääsynvalvonta (*access control*), salassapito (*secrecy*), yksityisyys (*privacy*) autentikointi (*autenthication*) ja turvallisuus (*safety*).

Suunnittelun rajoitteet

- Kustannukset.
- Aikataulut.
- Yhteensopivuus (*compatibility*).
- Käytettävyys (*usability*).

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
6. Suunnittelu
6.1 Yksinkertaisuus

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Tutkimusmatkailija Parnas ohjelmistojen ihmeellisessä maailmassa

- Parnas löytää silloin tällöin ohjelmistohelmiä:
  - Rakenne on erinomainen.
  - Ohjelmointityyli on johdonmukainen.
  - Ei käytetty typeriä ohjelmointikikkoja.
  - Jokainen komponentti on
    - yksinkertainen,
    - järjestelmällinen ja
    - helppo muuttaa.
- Parnas kummastelee miksi ohjelmistohelmet ovat harvinaisia, vaikka ohjelmistojen tekemistä on tutkittu yli 30 vuotta.

Havaintoja tutkimusmatkan varrelta

- Tietoa on: hyviä oppikirjoja, erinomaisia artikkeleita jne
- Ohjelmistohelmet on yleensä tehty olosuhteissa, joita ei esiinny kaupallisen menestyksen paineen alla ohjelmistoteollisuudessa.
- Useimmat ohjelmistot ovat
  - rumia,
  - epäluotettavia ja
  - vaikeita muuttaa.

”Fat software”

- Kaupalliset paineet aiheuttavat piirteiden lisäämistä ohjelmistoon.
  - Laajentuminen epäolennaisuuksiin saattaa hämärtää rakennetta.
- Pitäisi keskittyä olennaiseen – ei turhia kilkuttimia.
- Hyvä periaate, mutta olennaisen ja turhan välinen rajanveto on hankalaa.
- Joidenkin ominaisuuksien pudottaminen pois ohjelmistosta on kuin jalan amputointi laihdutuskeinona.

Ohjelmistojen paisumisen syitä

- Huono suunnittelu.
- Yhteensopivuus aikaisempien ohjelmistojen kanssa.
- Suorituskykyvaatimukset ja laitteistorajoitukset.
- Ohjelmiston elinkaaren aikana menneisyyden painolasti kasvaa.
  - Uudelleen tehtynä puhtaalta pöydältä osattaisiin paremmin.

Luovuus ja omaperäisyys

- Omaperäisyys on yliarvostettua:
  - Pitää oppia aikaisemmista
    - virheistä ja
    - onnistumisista.
- Luovuutta ja omaperäisyyttä tarvitaan, kun ongelmaan ei ole hyvää ratkaisua vielä olemassa.
- Pyörää ei kannata keksiä uudestaan!?
  - Vai onko niin, että
    - pyörä on keksitty uudestaan ja uudestaan, koska se on erinomainen ratkaisu?
    - ratkaisu, joka on keksitty vain kerran, on epäilyttävä?

Suunnittelu vai ohjelmointikieli

- Väite: Uusi ohjelmointikieli oli ratkaiseva ohjelmistohelmen synnyssä.
  - Parnas ei usko kielen valinnan merkitykseen.
    - Monta kieltä on kehitetty ratkaisemaan ohjelmistotuotannon ongelmat.
- Suunnittelu on ratkaiseva – ei toteutuksen kieli.
- Ohjelmointikieli, joka estää virheiden tekemisen on kuin veitsi, joka leikkaa pihviä muttei sormea.
- Hyvää jälkeä syntyy vain terävillä työkaluilla.

Onnistumisen perusedellytyksiä

- Etupainoinen työskentely – eli suunnittelu.
  - Aikaa suunnitteluun.
  - Ohjelmien kirjoittaminen ensin pseudokoodilla ennen lopullisella ohjelmointikielellä kirjoittamista.

1. Suunnittele ennen toteutusta.
2. Dokumentoi suunnittelu.
3. Katselmoi ja analysoi dokumentoitu suunnittelu.
4. Katselmoi toteutuksen ja suunnitelmien vastaavuus.

Sanottua

- Tietojenkäsittelyn määritelmä 1972 (Dijkstra).
  - Monimutkaisuuden hallitsemista.
- Seymor Cray:
  - Monimutkainen on hidasta.
  - Kauas on pitkä matka.
- KISS! (IBM:n vanha huoneentaulu.)
  - Keep It Simple and Stupid!
  - Keep It Simple and Small!

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
6. Suunnittelu
6.2 Suorituskyky

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Suorituskyvyn mitattavia suureita

- Suoritusteho (*throughput*).
  - Kuinka paljon hyödyllistä työtä tehdään aikayksikössä.
- Vasteaika (*response time*).
  - Kuinka kauan tehtävän suorittaminen kestää.
- Käyttöaste (*utilization*).
  - Osuus ajasta, jonka systeemi on aktiivisena.

Suorituskyvyn suureiden mittaustapoja

- Mitataan valmiin järjestelmän toimintaa.
- Simuloidaan (jäljitellään ohjelmalla) järjestelmän toimintaa likimääräisesti.
- Mallinnetaan järjestelmää matemaattisin menetelmin ja lasketaan mallin avulla keskeiset suorituskykysuureet.
  - Jonomallit.
  - Jonoverkkomallit.
  - jne

Suorituskyvyn suunnittelu

- Tavoite: ohjelmiston suunnitteluvaiheessa arvioidaan tulevan järjestelmän tehokkuutta ja tarvittavia laitteistoresursseja.
- Suorituskyky tietyn työkuorman vallitessa:
  - Käyttäjä kokee.
  - Järjestelmä tarjoaa.

Ohjelmiston vaatimukset

- Toiminnalliset (*functional*).
- Ei–toiminnalliset (*non-functional*).
  - Tietoturva (*security*).
  - Saatavuus (*availability*).
  - Luotettavuus (*reliability*).
  - Suorituskyky (*performance*).
- Asennevamma ohjelmistotuotannossa: jako toiminnallisiin ja ei-toiminnallisiin vaatimuksiin.
  - Korjaa-myöhemmin (*fix-it-later*) –asenne suorituskyvyn suhteen osoitti, että se ei kuulunut ohjelmiston suunnitteluvaiheeseen.

Miksi suorituskyvyn suunnittelu ei ole osa ohjelmiston suunnitteluvaihetta

- Menascén havainnot:
  1. Tieteellisten mallien ja periaatteiden puute.
  2. Koulutus.
  3. Tietojenkäsittelyn työvoima.
  4. Yhden käyttäjän ajattelu.
  5. Pienen tietokannan ajattelu.

1. Tieteellisten mallien ja periaatteiden puute.

- Ohjelmistotekniikassa ei ole yleisesti käytetä malleja suorituskyvyn suunnittelussa ohjelmiston elinkaaren aikana.
  - Malleja ohjelmistotuotannon hallitsemiseen muuten on.
- Perinteisissä insinööritaidoissa (esim. rakentamisen eri muodot) käytetään matematiikkaan, fysiikkaan ja laskennallisiin tieteisiin perustuvia tieteellisiä periaatteita ja malleja.

2. Koulutus.

- ACM:n/IEEE:n tkt:n opetussuosituksissa ei ole tietokonejärjestelmien suorituskyvyn analysoinnin pakollista kurssia.
  - Suorituskyvystä vain hajatunteja käyttöjärjestelmä- ja tietoliikennekursseilla.
  - Ohjelmistotekniikka ei sisällä mainintaa suorituskyvystä.
- Miksi?
  - Opettajat eivät osaa.
  - Tieteellisten periaatteiden ja mallien puute.
  - Muutosvastarinta.
  - Kaikki hyödyllinen ei mahdu tutkintoon.

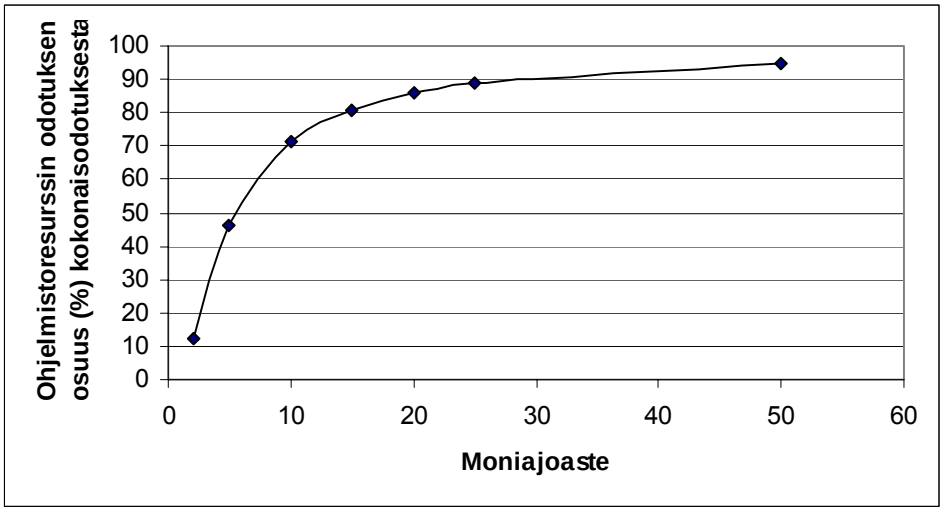
3. Tietojenkäsittelyn työvoima.

- Suunnittelu- ja ohjelmointiväessä paljon henkilöitä vailla tietojenkäsittelyn koulutusta.

4. Yhden käyttäjän ajattelu.

- Suunnittelijat ja ohjelmoijat luulevat, että järjestelmää käyttää vain yksi käyttäjä kerrallaan.
  - Samanaikaisia käyttäjiä on kuitenkin yleensä useita.
  - Samanaikaisuus aiheuttaa kilpailua (ja odotusta) sekä laitteistoresursseista
    - prosessorit, muistit, talletusvälineet, tietoliikenneyhteydet,…
  - että ohjelmistoresursseista
    - tietokantalukot, kriittiset alueet, ohjelmistosäikeet, …

Esimerkki ohjelmistoresurssikilpailun vaikutuksesta – 33% käsittelystä kriittisellä alueella



5. Pienen tietokannan ajattelu.

- Tietokannan käytön ohjelmoinnissa ei yleensä oteta tietokannan kokoa huomioon.
  - Kysely 1000 rivin tietokantaan voidaan yleensä tehdä eri tavalla kuin 1 000 000 rivin tietokantaan.

Menascén johtopäätökset.

- Ohjelmiston monimutkaisuus aiheuttaa usein tehottomuutta.
- Monimutkaisuuden hallitsemiseksi kannattaa parantaa ohjelmoijien ammattitaitoa eikä kehittää heidän käyttämiään työkaluja.
- Tehokkain menettely hyvän suorituskyvyn ohjelmistojen tuottamiselle on suunnittelijoiden ja ohjelmoijien koulutus suorituskykyyn liittyvissä asioissa.
- Ohjelmiston hyvä suorituskyky riippuu enemmän hyvästä suunnittelusta kuin hyvästä ohjelmoinnista.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
6. Suunnittelu
6.3 Luotettavuus

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Luotettavuus

- Päällekkäisyys tai toisteisuus (*redundancy*).
- Toipuminen (*recovery*).
- Tarkistuspisteiden käyttö eli varmistaminen (*checkpointing*).
- Eheys (*integrity*).
- Luottamus (*trust*).

Luotettavuuden tarina – toipuminen

- Candea ja Fox: Chrash-only software.
  - Erilainen näkökulma ohjelmiston toipumiseen kaatumisesta – ei ole tietojenkäsittelyn vakiintunutta käytäntöä.
- Tietojenkäsittelyjärjestelmä, joka ei koskaan kaadu, ei ole realistinen tavoite.
- Järjestelmän suunnittelussa on siis varauduttava kaatumiseen ja toipumiseen.
- Miksi suunnitella sekä hallittu alasajo ja siitä toipuminen että kaatuminen ja siitä toipuminen?

Crash-only ohjelmistojen perusajatus

- Kaatuminen (*crash*) on tehtävä turvalliseksi.
- Toipuminen (*recovery*) on tehtävä nopeaksi.
- Ainoa tapa lopettaa ohjelman suoritus on kaataa se.
- Ohjelma käynnistetään aina toipumisen kautta.

Toipumisesta

- Järjestelmän toipumisesta huolehtiva ohjelman osa käsittelee poikkeustilanteita. Sen on oltava virheetön.
- Poikkeukselliset tilanteet ovat hankalia käsitellä.
  - Esiintyvät harvoin, eikä niitä ole helppo tuottaa ohjelmiston kehitysvaiheessa, jolloin toipumista olisi testattava.
  - Toipumisen suorittava ohjelman osa on vaikea saada virheettömäksi.

Crash-only ohjelmistojen ominaisuuksia

- Kaikki ei-tilapäinen tilatieto on talletettava erityiseen tilatietomuistiin (*state store*).
- Ohjelmiston osien on varauduttava muiden osien kaatumiseen ja niiden palveluiden tilapäiseen puuttumiseen (*unavailability*).
- Keskeisiä ominaisuuksia:
  - Modulaarisuus.
  - Vahvat rajapinnat, joissa häiriöt hallitaan ilman vaikutuksia toisaalla.
  - Ajastimiin perustuva kommunikointi.
  - Laina-aikaan (*lease*) perustuva resurssien varaus.
  - Täydellisesti itsensä kuvaavat palvelupyynnöt.

Johdatus tietojenkäsittelytieteeseen
6. Suunnittelu
6.4 Kehitettävyys

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Ohjelmistotekniikka on kriisissä

- Ollut jo 1960-luvun lopulta alkaen.
- Kriisi on äkillinen ja lyhytaikainen vakava hätätila.
- Ilmiö on vakava.
- Kyseessä ei ole kriisi vaan pitkäaikaishoitoa vaativa krooninen tauti.

Tilannekatsaus (Lehman 1998)

- Yhteiskunnan tietokone(ohjelmisto)riippuvuus kasvaa.
- Tietoteknologian käytön lisääntyminen lisää järjestelmien integroinnin tarvetta.
- Ympäröivä maailma muuttuu.
  - Y2K – vuosituhannen vaihtumista ei ohjelmistosuunnitelmissa.
  - Euro.
  - Puhelinnumeroiden piteneminen.
  - jne

Huomioita ohjelmistoista

- Ohjelmistot ovat monimutkaisimpia ihmisen aikaansaannoksia.
- Ohjelmisto itsessään on malli sovelluksesta, osallistujista (ihmiset, organisaatiot, laitteet,…) käyttöalueesta ja kyseisen alueen toiminnoista.
- Käyttöalue on moniulotteinen – käytännössä rajattoman laaja.
  - Ohjelmisto on rajallinen.
- Ohjelmisto on äärellinen ja epätäydellinen malli rajattoman käyttöalueen rajattomasta sovelluksesta.

Ohjelmiston ja todellisen maailman välillä on kuilu

- Kuilua paikataan oletuksilla.
  - Algoritmien valinta.
  - Järjestelmän valinta, määrittely, suunnittelu ja toteutus sisältävät paljon oletuksia.
  - Osa oletuksista on selkeitä (*explicit*), kuten määrittelyssä tehdyt valinnat.
  - Osa oletuksista on epäsuoria (*implicit*), kuten valitun teorian mukana tulevat, algoritmin suunnittelun tuottamat, rajapinnan määrittelystä johtuvat jne.
- Lehman arvelee, että jokaista 10 ohjelmariviä kohti on olemassa yksi oletus. (Miljoona riviä ??? oletusta!)

FEAST (*feedback, evolution and software technology*)

- Ohjelmistoprosessi muodostaa
  - monitasoisen (*multilevel*) ja
  - monisilmukkaisen (*multiloop*) takaisinkytkentäjärjestelmän (*feedback system*)
- ja sitä on käsiteltävä sellaisena, jos haluamme saada huomattavaa parannusta sen suunnitteluun, ohjaukseen ja kehittämiseen.

Lehmanin suosituksia (lyhyt lista … 18 kohtaa)

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
6. Suunnittelu
6.5 Tietoturva

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos

Tietoturva (*security*)

- Pääsynvalvonta (*access control*).
- Salassapito (*secrecy*).
- Yksityisyys (*privacy*).
- Todennus (*authentication*).
- Eheys (*integrity*).
- Turvallisuus (*safety*).

Tietoturvan tarina - turvallisuus

- Sädehoidon ohjausohjelmisto – Therac-25 laitteistossa.
- Vuosina 1985 – 1987 on kuusi tunnettua tapausta potilaan ylisäteilytyksestä.
- Pahimmillaan säteilyannos oli 10 000-kertainen tarkoitettuun annokseen verrattuna.
- Ainakin viiden potilaan kuolema osoitettiin johtuvaksi sädehoitolaitteen suunnittelu- ja ohjelmointivirheistä johtuvaksi.

Onnettomuuksien syistä

- Useimmiten onnettomuudet johtuvat monimutkaisista toisiinsa kietoutuvista tapahtumista, jotka johtuvat teknisistä, inhimillisistä ja työyhteisöön liittyvistä tekijöistä.
- Therac-25:n tapausten kaksi vakavaa virhettä:
  - Uskottiin, että onnettomuuden syyt oli poistettu ensimmäisen tapauksen jälkeen.
    - Johtopäätökselle ei ollut kestäviä perusteita.
    - Vaihtoehtoisia syitä ei selvitetty kuin ylimalkaisesti.
  - Oletettiin, että yhden ohjelmistovirheen korjaaminen estäisi uudet onnettomuudet.
    - Monimutkaisissa järjestelmissä löytyy lähes aina seuraava virhe.

Onnettomuuksien selityksiä

- Usein selitetään onnettomuuksien johtuneen yhdestä syystä – esim. inhimillisestä erehdyksestä.
  - Lähes kaikkien syiden voidaan katsoa olevan inhimillisiä erehdyksiä.
  - Jos onnettomuuden syy on laitteiston kuluminen, niin miksei kulunutta osaa vaihdettu ajoissa?
- Onnettomuuden selityksenä inhimillinen erehdys ei ole kovin hyödyllinen, ellei sitä tarkenneta riittävästi.
- Yhtä hyödytön selitys on laitteistovika tai ohjelmistovirhe.

Therac-25:n onnettomuuksien syistä

- Valmistajan hallinnoinnissa olleet epätarkoituksen-mukaisuudet ja raportoitujen onnettomuuksien käsittelyn puutteellisuudet.
- Liiallinen luottamus ohjelmistoon ja laitteistovarmistuksista luopuminen, minkä seurauksena ohjelmistosta tuli varmistamaton virhelähde.
- Ohjelmistotekniikan käytäntöjen ilmeiset puutteet.
- Epärealistinen riskien arviointi ja ylenpalttinen luottamus arvioinnin antamiin tuloksiin.

Järjestelmän rakentaminen

- Yleinen virhe on luottaa liikaa ohjelmistoihin.
- Ohjelmiston suunnitteluvirheiden löytäminen ja estäminen on huomattavasti hankalampaa kuin laitteiston kulumisesta johtuvien virheiden.
- Laitteiston virhekäyttäytymisen muotoja on vain muutama.
  - Niitä vastaan suojautuminen on yleensä olennaisesti helpompaa kuin ohjelmistovirheistä vastaan suojautuminen.

Therac-25:n opetuksia

- Ehkä tärkein: laitteistovarmistuksista ei pidä luopua, kun järjestelmän ohjaamiseen aletaan käyttää ohjelmistoa.
- Trendi on ollut vähentää laitteistovarmistuksia.
  - Niissäkin tapauksissa, missä laitteistovarmistuksia käytetään, niitä yhä useammin ohjataan ohjelmistolla.
  - Ehdotonta turvallisuutta vaativissa järjestelmissä ei saa olla varmistamattomia virhelähteitä.
- Järjestelmää ei saa suunnitella siten, että yksittäinen ohjelmistovirhe voi aiheuttaa katastrofin.

Ohjausjärjestelmien ohjelmistovirheistä

- Onko kyseessä tilapäinen laitteistovirhe?
- Ohjausohjelmisto lukee arvoja tunnistimista (*sensors*) ja lähettää komentoja säätimille (*actautors*).
- Hyvin vaikea (ellei mahdotonta) päätellä
  - antoiko tunnistin väärää tietoa,
  - lähettikö ohjelmisto väärän komennon vai
  - toimiko säädin tilapäisen laitevirheen vuoksi väärin.

Therac-25:n virhediagnostiikka

- Potilaiden oireet olivat ainoat todelliset indikaattorit järjestelmän virheistä.
- Järjestelmässä ei ollut riippumattomia toiminnan oikeellisuuden tarkistuksia.
  - Therac-25 ei voinut havaita antamaansa säteilyn määrää.
- Keskeinen opetus: ohjausjärjestelmät on suunniteltava pahimman toiminnan varalle.
  - Ehdotonta turvallisuutta vaativiin järjestelmiin on rakennettava jäljitysmekanismit (*audit trails*) sekä poikkeavien tilanteiden analysointimekanismit.

Loppuhuomioita

- Turvallisuus on pystyttävä takaamaan järjestelmätasolla (laitteistovarmistukset) mahdollisista ohjelmistovirheistä riippumatta.
- Therac-25:n edeltäjässä oli sama ohjelmistovirhe, mutta laitteistovarmistus esti onnettomuudet.
  - Usein naivisti oletetaan, että ohjelman uudelleen käyttö lisää turvallisuutta, koska ohjelma on ollut jo pitkään käytössä.
- Turvallisuus on koko järjestelmän ominaisuus – ei pelkästään ohjelmiston ominaisuus.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

Johdatus tietojenkäsittelytieteeseen
7. Tietojenkäsittelyn käytännöt

Matemaattis-luonnontieteellinen tiedekunta

Tietojenkäsittelytieteen laitos