Erkki Mäkinen

Johdatus tietojenkäsittelytieteisiin

INFORMAATIOTIETEIDEN YKSIKKÖ
TAMPEREEN YLIOPISTO

INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 13/2012

TAMPERE 2012

TAMPEREEN YLIOPISTO
INFORMAATIOTIETEIDEN YKSIKKÖ
INFORMAATIOTIETEIDEN YKSIKÖN RAPORTTEJA 13/2012
ELOKUU 2012

Erkki Mäkinen

Johdatus tietojenkäsittelytieteisiin

INFORMAATIOTIETEIDEN YKSIKKÖ
33014 TAMPEREEN YLIOPISTO

ISBN 978-951-44-8871-9

ISSN-L 1799-8158
ISSN 1799-8158

Johdatus tietojenkäsittelytieteisiin

Aluksi

Tämä julkaisu on tarkoitettu käytettäväksi Tampereen yliopiston tietojenkäsittelytieteiden tutkinto-ohjelman opintojaksolla Johdatus tietojenkäsittelytieteisiin. Julkaisu perustuu Kimmo Raatikaisen kirjoittamaan Helsingin yliopiston vastaavan opintojakson julkaisuun [Raatikainen, 2007]. Tekstiä on osin karsimalla ja osin laajentamalla sovitettu Tampereen yliopiston opintojakson tarpeisiin.

Opintojakson tavoitteena on antaa opiskelijoille yleiskuva tietojenkäsittelytieteistä, niiden perusmenetelmistä ja käytännöistä sekä perehdyttää alan keskeisimpään käsitteistöön. Tietojenkäsittelytieteitä tarkastellaan oman tutkinto-ohjelman näkökulmasta.

Kiitän Aulikki Hyrskykaria, Kati Iltasta ja Timo Porasta käsikirjoituksen kommentoinnista.

Sisällysluettelo

LUKU 1: Tietojenkäsittelytieteet tieteiden joukossa

Mitä tiede on?

Ennen kuin voidaan pohtia tietojenkäsittelytieteiden asemaa suhteessa muihin tieteisiin, on syytä lyhyesti kuvailla sitä, mitä tieteellä ylipäätään tarkoitetaan ja mitkä ovat sen keskeiset tunnusmerkit. Aloittamalla jostain tieteen määritelmästä, esimerkiksi Haaparannan ja Niiniluodon [1986] määritelmästä

"Tiede on järjestelmällistä ja järkiperäistä uuden tiedon hankintaa.",

päädytään uusiin ongelmiin sen suhteen, mitä tarkoittaa "järjestelmällinen", "järkiperäinen" ja "uusi tieto". Toinen tapa yrittää selvitä tieteen määrittelemisestä on sanoa, että tiede on sitä, mitä tiedeyhteisö tekee (tai mitkä ovat sen toiminnan tulokset). Tämäkään "määritelmä" ei tietenkään ole ongelmaton.

Edellisiä parempaan tulokseen päästään, kun yritetään kuvailla tieteellistä toimintaa sitä mahdollisimman oleellisesti luonnehtivilla ominaisuuksilla. Tieteellisessä toiminnassa pyritään ainakin seuraaviin:

- objektiivisuus
- julkisuus
- kriittisyys
- itseäänkorjaavuus
- autonomisuus
- edistyvyys.

Objektiivisuus tarkoittaa sitä, että tutkimuskohteen ominaisuudet ovat riippumattomia tutkijan mielipiteistä ja tieto saavutetaan tutkimuskohteesta tehtävien havaintojen kautta. *Julkisuusvaatimuksen* mukaan tieteessä esitetyille väitteille tulee voida antaa julkinen perustelu, ja käytetyt menetelmät tulee kuvata niin tarkkaan, että toinen tutkija voisi samoin menetelmin halutessaan päätyä vastaaviin tuloksiin. Julkisuusperiaate mahdollistaa tieteellisten tulosten kritiikin. *Kritiikki* koettelee esitettyjen tulosten oikeellisuutta, ja siitä seuraa tieteen *itseäänkorjaavuus* eli mahdollisesti esitetyt virhekäsitykset korjaantuvat ajan myötä. *Autonomisuuden* mukaan niin tieteellisten ongelmien valinta kuin tieteen tulosten arviointi ovat tiedeyhteisön omia asioita, joihin tieteen ulkopuoliset ryhmät (esim. rahoittajat tai poliitikot) eivät saa vaikuttaa. Tieteen *edistyvyys* liittyy läheisesti julkisuuteen ja

kriittisyyteen: niiden avulla tieteellisen tiedon virheet ja puutteet vähenevät.

Edellä esitetyt tieteen luonnehdinnat kuuluvat tieteenfilosofian piiriin. Yksi sen keskeisistä ongelmista on ns. demarkaatio-ongelma eli rajanveto tieteen ja ei-tieteen välillä. Tässä yhteydessä ei ole syytä paneutua ongelmaan tämän enempää.

Mihin tiede pyrkii?

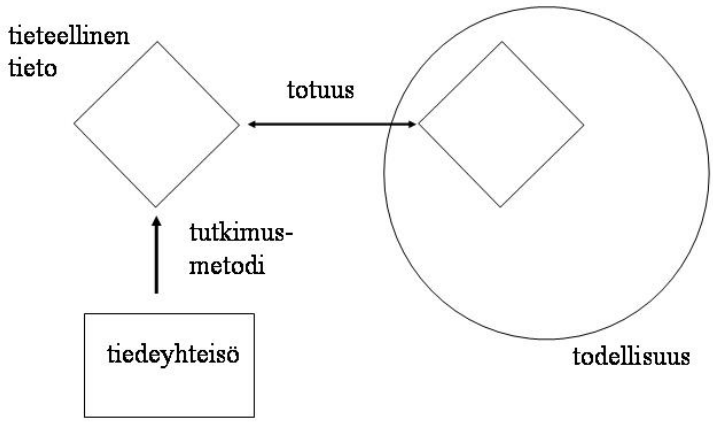
Jos pohditaan sellaisia tieteenaloja kuin kosmologia, kirkkohistoria, psykiatria, kansantaloustiede ja sähkötekniikka, niin on vaikeaa löytää niille yhteistä tavoitetta tai päämäärää. Tieteellä onkin erilaisia päämääriä, jotka ovat osittain keskenään ristiriitaisia.

Kognitivismin mukaan tiede etsii totuutta (uutta tietoa, aiemmin tuntemattomia totuuksia) ja tällä uudella tiedolla ja sen etsinnällä on arvoa sinänsä riippumatta siitä, mihin tarkoituksen saatua tietoa mahdollisesti voitaisiin käyttää. Tiedon itseisarvon korostamisesta käytetään myös nimitystä *verismi*. Vastakkaisen näkemyksen mukaan tieteen tuloksilla on arvoa vain, jos niitä voidaan pitää käytännön päätösongelmien ratkaisuiksi esitettyinä toimintasuosituksina. Tästä näkökannasta käytetään nimitystä *instrumentalismi*.

Näiden vastakkaisten näkemysten mukaan tutkijaa voidaan toisaalta pitää "totuuden etsijänä"(teoreetikkona) tai toisaalta "neuvonantajana" (soveltajana). Vastaavasti tutkimus voidaan jakaa seuraavasti:

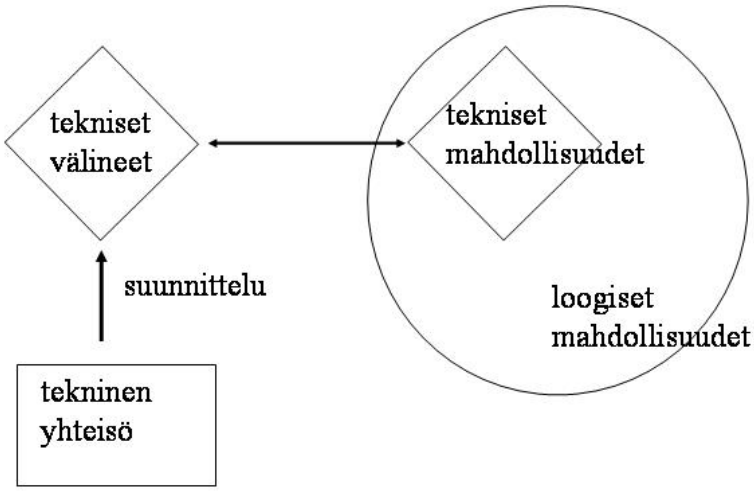
- Perustutkimus, jonka tavoitteena on uuden tiedon etsintä ilman ensisijaista pyrkimystä käytännöllisiin tavoitteisiin tai sovelluksiin eli tieto järjestettynä "teorioiksi" (todellisuutta koskevien tosiasioiden kokoelmiksi).
- Soveltava tutkimus, jonka tavoitteena perustutkimuksen tuloksiin (tieteelliseen tietoon) perustuvat sovellukset ja ongelmanratkaisut; soveltava tutkimus tuottaa muotoa "jos halutaan saavuttaa X, on tehtävä Y" olevia toimintaohjeita.
- Tuotekehitys, jonka tavoitteena on kaupallisesti (tai muulla tavoin) hyödynnettävät tuotteet ja keksinnöt.

Kuva 1 havainnollistaa, kuinka perustutkimuksen tavoitteena on muodostaa mahdollisimman totuudellinen kuva jostain todellisuuden osa-alueesta.



Kuva 1. Perustutkimuksen suhde todellisuuteen.

Tekniikan eri alueet ovat tyypillisiä esimerkkejä soveltavasta tutkimuksesta. Kuva 2 havainnollistaa, kuinka teknisten tieteiden alueella tuotetaan erilaisia teknisiä välineitä, joiden ”hyvyyttä” mitataan, totuuden tai selitysvoiman sijasta, esimerkiksi tehokkuudella, taloudellisuudella, esteettisyydellä, ekologisuudella tai ergonomisuudella.



Kuva 2. Tekniikan suhde loogisiin ja teknisiin mahdollisuuksiin.

Suuri osa tietojenkäsittelytieteitä muistuttaa teknisiä tieteitä siinä, että tietojenkäsittelytieteissäkin tulokset ovat usein uusia konstruktioita, esimerkiksi ohjelmistoja tai käyttöliittymiä.

Miten tiedettä tehdään?

Tieteellisen metodin alkuna voidaan pitää tieteellisen tiedon erilaisuuden tunnistamista suhteessa arkiajatteluun ja arkiajattelun rajoitusten havaitsemista. Taulukkoon 1 on koottu tyypillisiä arkiajattelun puutteita ja niitä menetelmiä, joilla nämä puutteet tieteellisessä ajattelussa korjataan.

Arkiajattelun puutteita	**Korjaamisen mekanismit tieteellisessä ajattelussa**
Epäluotettavat havainnot	Havaintojen systematisointi Havaintojen toistettavuus
Valikoiva havainnointi	Tutkimuksen oletuksia ja niiden vaikutuksia koskeva pohdinta
Liiallinen yleistäminen	Tilastolliset menetelmät
Puutteellinen päättely	Loogiset ja matemaattiset välineet Tilastolliset menetelmät
Lyhytjännitteisyys	Aikaisempiin tutkimuksiin perustuminen Ilmeisen näkeminen ongelmallisena
Asioiden tarkastelu irrallaan yhteyksistä ja mittakaavasta	Olennaisen ja epäolennaisen erottaminen

Taulukko 1. Arkiajattelun puutteita ja niiden korjaamisen menetelmiä [Uusitalo, 1999].

Taulukon 1 oikeanpuoleisessa sarakkeessa viitataan erilaisiin menetelmiin, joita tieteellisessä työssä käytetään apuna. Käytettävät menetelmät riippuvat tutkimuskohteesta ja siitä ”tutkimusotteesta”, jolla tutkija lähestyy kohdettaan. Tietojenkäsittelytieteille on tyypillistä erilaisten tutkimusotteiden ja -menetelmien moninaisuus. Tämä on osaltaan seurausta siitä, että tietojenkäsittelytieteet ovat ”perineet” menetelmiä niiltä tieteenaloilta, joilta tietojenkäsittelytieteiden pioneerit aikanaan siirtyivät alalle.

Järvinen ja Järvinen [2011] jakavat tutkimusotteet ylimmällä tasolla matemaattisiin ja reaalimaailmaa koskeviin tutkimusotteisiin. Monet tietojenkäsittelytieteiden osa-alueet esitetään matemaattisina teorioina, joissa matemaattisen tutkimusotteen käyttö on itsestään selvää. Tällöin tulokset muotoillaan ja todistetaan matemaattisina teoreemoina. Reaalimaailmaa koskevat tutkimusotteet Järvinen ja Järvinen [2011] puolestaan jakavat innovaatioiden hyödyllisyyttä painottaviin tutkimuksiin ja ”millainen maailma on” -tutkimuksiin. Näistä edelliset ovat erityisen tyypillisiä tietojenkäsittelytieteille, sillä tietojenkäsittelytieteissä tutkimustulokset ovat usein uusia konstruktioita. Innovaatioiden hyödyllisyyttä painottavissa tutkimuksissa voidaan erottaa innovaatioiden toteuttaminen ja niiden arviointi. ”Millainen maailma on” -tutkimuksissa

tietojenkäsittelytieteellisiä tutkimuskohteita voidaan lähestyä esimerkiksi perinteisillä käsitteellis-teoreettisilla tai empiirisillä tutkimusotteilla.

Millaisia tieteitä tietojenkäsittelytieteet ovat?

Tieteenalana tietojenkäsittely on noin 65-vuotias. Tänä aikana sen olemuksesta ja sisällöstä on esitetty monenlaisia käsityksiä. Vuosien varrella tietojenkäsittelytiedettä on pidetty muun muassa

tietokoneiden [Newell et al., 1967],
algoritmien [Knuth, 1974],
tietorakenteiden [Wegner, 1971] ja
monimutkaisuuden [Dijskstra, 1972]

tutkimisena. Useimmat muutkin 1960- ja 1970-luvuilla esitetyt määritelmät olivat lyhyitä ja selkeitä, mutta puolinaisia. Yleensä esitetty määritelmä korosti sitä tietojenkäsittelyn osa-aluetta, jota määritelmän esittäjä itse tutki.

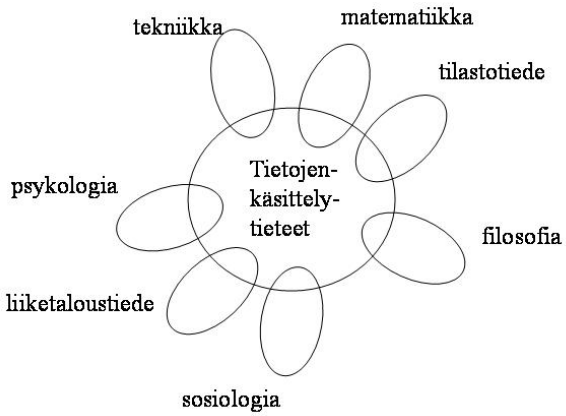
1980-luvulle tultaessa määritelmät pidentyivät ja monimutkaistuivat. ACM:n komitea [Denning et al., 1989] määritteli 1980-luvun lopussa tietojenkäsittelytieteen[1] seuraavasti:

> *Tieteenalana tietojenkäsittely tutkii systemaattisesti informaatiota kuvaavia ja muuntavia algoritmisia prosesseja; niiden teoriaa, analysointia, suunnittelua, tehokkuutta, toteuttamista ja soveltamista.*

Sopivasti tulkittuna määritelmä sisältää vieläkin kaiken oleellisen, vaikka kovin nasevana tai helppotajuisena sitä ei voi pitää.

Eräs tietojenkäsittelytieteiden määrittelyä vaikeuttava tekijä on se, että tietojenkäsittelytieteilijät käyttävät hyvin monenlaisia tutkimusmenetelmiä ja tietojenkäsittelytieteissä sovelletaan monia vanhempien naapuritieteiden ajattelutapoja. Kuvassa 3 on hahmoteltu joitakin tietojenkäsittelytieteiden naapureita.

[1] Yksiköllinen muoto ”tietojenkäsittelytiede” viittaa anglosaksisissa maissa käytettyyn termiin ”computer science”, joka on alaltaan jonkin verran suppeampi kuin tutkinto-ohjelmamme nimessä oleva ”tietojenkäsittelytieteet” tai vaikkapa saksalainen ”Informatik”.



Kuva 3. Tietojenkäsittelytieteiden naapureita.

Automatisointi on tietojenkäsittelytieteen perusta

Tietojenkäsittelytiedettä on määritelty myös kuvailemalla tietojenkäsittelytieteen osa-alueita. ACM:n komitea päätyi 1980-luvun lopulla yhdeksään osa-alueeseen: algoritmit ja tietorakenteet, ohjelmointikielet, tietokonearkkitehtuurit, numeerinen ja symbolinen laskenta, käyttöjärjestelmät, ohjelmistotekniikka, tietokannat ja tiedonhaku, tekoäly ja robotiikka sekä ihmisen ja tietokoneen vuorovaikutus. Useissa muissakin lähteissä on vastaavia, edellisestä enemmän tai vähemmän poikkeavia, luetteloita. Esimerkiksi rinnakkaisuus ja hajautus, tietoturva, luotettavuus sekä suorituskyvyn arviointi ovat esiintyneet myös tietojenkäsittelytieteen osa-alueina.

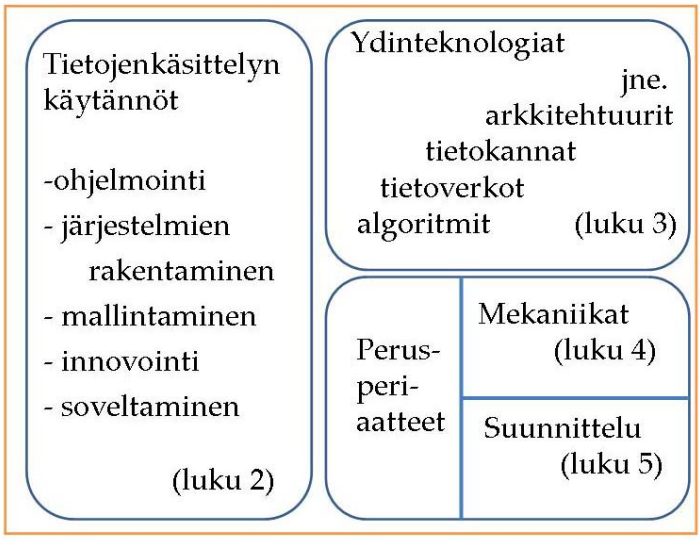
Tietojenkäsittelytieteestä saadaan muodostettua huomattavasti selkeämpi kuva, kun tarkastellaan niitä kaikkein keskeisimpiä kysymyksiä, joita tietojenkäsittelytiede yrittää ratkaista. Forsythe [1968] esitti, että tietojenkäsittelytieteen peruskysymys on:

Mitä voidaan automatisoida?

Tietojenkäsittelytieteen olemusta pohtinut ACM:n komitea [Denning et al., 1989] täydensi peruskysymyksen muotoon:

Mitä voidaan automatisoida tehokkaasti?

Denningin [2003] mukaan tietojenkäsittelyä voidaan kuvata sen perusperiaatteiden, ydinteknologioiden ja käytäntöjen kautta. Denningin rakentama tietojenkäsittelytieteen rakenne on esitetty kuvassa 4. Seuraavissa luvuissa tutustaan tarkemmin kuvan 4 osiin.



Kuva 4. Tietojenkäsittelytieteen rakenne Denningin [2003] mukaan.

Tietojenkäsittelyn käytännöt (computing practices) ovat tietojenkäsittelyjärjestelmien vakiintuneet rakentamis- ja käyttöönottotavat. Denningin systeemissä käytäntöjä ovat seuraavat (vrt. luku 2):

- ohjelmointi (programming),
- järjestelmien rakentaminen (engineering systems),
- mallintaminen (modeling),
- innovointi (innovating) ja
- soveltaminen (applying).

Vastaavasti ydinteknologiat (core technologies) ovat (vrt. luku 3)

- algoritmit (algorithms),
- tietokannat (databases),
- arkkitehtuurit (architectures),
- tietoverkot (networks),
- käyttöjärjestelmät (operating systems), jne.

Perusperiaatteet voidaan jakaa kahteen pääryhmään:

- tietojenkäsittelyn mekaniikat ja
- suunnittelun periaatteet.

Tietojenkäsittelyn mekaniikat (mechanics) tarkoittavat niitä lakeja ja yleisesti toistuvia ilmiöitä, jotka kattavat tietojenkäsittelyn toiminnot. Mekaniikan osat ovat (vrt. luku 4)

- laskenta (computation),
- kommunikointi (communication),
- koordinointi (coordination),
- automatisointi (automation) ja
- muistaminen (recollection).

Suunnittelun periaatteet tarkoittavat tietojenkäsittelyllisissä suunnittelutehtävissä va-

kiintuneita menettelytapoja. Suunnittelun tavoitteita ovat (vrt. luku 5)

- yksinkertaisuus (simplicity),
- suorituskyky (performance),
- luotettavuus (reliability),
- kehitettävyys (evolvability) ja
- tietoturva (security).

Kuvassa 4 tarkastellaan tietojenkäsittelytieteitä itsenäisinä, riippumatta sovellusalueista, joilla niitä sovelletaan. Samoin kuvasta puuttuvat sovellusalueiden, ydinteknologioiden, suunnittelun periaatteiden ja tietojenkäsittelyn mekaniikkojen vaikutukset tietojenkäsittelyn käytäntöihin. Näitä vaikutuksia voidaan luonnehtia seuraavilla peruskysymyksillä:

Sovellusalueet: Miten toimia yhdessä sovellusalueiden edustajien kanssa suunniteltaessa eri sovellusalueiden tietojenkäsittelyä?

Ydinteknologiat: Miten suunnitella menettely, joka tukee eri sovellusten yhteisiä tarpeita?

Suunnittelun periaatteet: Miten organisoida toimiva tietojenkäsittelyn toteutus?

Tietojenkäsittelyn mekaniikat: Miten tietojenkäsittely toimii?

Kuva 4 korostaa tietojenkäsittelyn toimintaan suuntautumista (action-orientation). Tietojenkäsittelyllä on lukuisia asiakkaita. Tietojenkäsittelyn käyttökohteet ovat yhtä tärkeitä kuin tietojenkäsittelyn menetelmät (tietojenkäsittelyn mekaniikat). Denningin esittämä rakenne tuo esille tietojenkäsittelyn ammattilaiselta edellytettäviä kykyjä ja taitoja, jotka koostuvat tietojenkäsittelyn mekaniikkojen, suunnittelun, tietojenkäsittelyn käytäntöjen, ydinteknologioiden ja sovellusten hallinnasta.

LUKU 2: Tietojenkäsittelyn käytännöt

Tietojenkäsittelyn käytännöt täydentävät kuvaa tietojenkäsittelytieteestä, sillä niissä ilmenevät tietojenkäsittelijöiden ammatilliset taidot. Viime kädessä tietojenkäsittelijän taidot ja osaaminen arvioidaan työn tulosten laadun perusteella.

Tietojenkäsittelyn käytännöt voidaan jakaa viiteen pääalueeseen:

Ohjelmointi: Järjestelmän käyttäjien kanssa määritellyn ohjelmiston toteuttaminen ohjelmointikieliä käyttäen. Tietojenkäsittelyn ammattilaisten on hallittava useita erilaisia ohjelmointikieliä ja osattava valita tarkoituksenmukaisin kuhunkin ongelmanratkaisutilanteeseen.

Järjestelmien rakentaminen: Tietojärjestelmien suunnitteleminen ja toteuttaminen ohjelmisto- ja laitteistokomponenteista. Tähän alueeseen kuuluvat suunnittelu, jossa keskitytään järjestelmän organisointiin siten, että järjestelmä hyödyttää käyttäjiä, toteuttaminen, jossa keskitytään järjestelmän moduuleihin, abstraktioihin, uudistuksiin (revisions), toteutustavan valintoihin ja riskeihin, sekä käyttö, jossa keskitytään konfiguraatioon, hallintaan ja ylläpitoon.

Mallintaminen ja validointi: Mallintaminen viittaa toisaalta suunnitteilla olevan tietojärjestelmän kohdealueen ja itse tietojärjestelmän kuvaamiseen niin, että järjestelmä voitaisiin rakentaa todellisessa käyttöympäristössään mahdollisimman hyvin toimivaksi, ja toisaalta valmiiden järjestelmien kuvaamista niiden käyttäytymisen ennustamiseksi erilaisissa tilanteissa ja olosuhteissa; validoinnilla tarkoitetaan kokeiden suunnittelua sen varmistamiseksi, että järjestelmät toimivat halutulla tavalla.

Innovointi: Rogers [1] määrittelee innovaation ideaksi, käytännöksi tai esineeksi, jota yksilöt pitävät uutena. Innovaation ei tarvitse olla upouusi, vaan olennaista on, että yksilö kokee sen uutena. Innovaatio voidaan määritellä myös pysyvien muutosten aikaansaamisena ryhmien ja yhteisöjen toimintatavoissa. Innovaattorit etsivät ja analysoivat mahdollisuuksia. He kuuntelevat asiakkaitaan ja muotoilevat heille hyödyllisiä ehdotuksia. Innovaattoreiden on myös huolehdittava, että luvatut tulokset saavutetaan.

Soveltaminen: Työskentely sovellusalueiden ammattilaisten kanssa näitä palvelevien tietojenkäsittelyjärjestelmien toteuttamiseksi. Työskentely muiden tietojenkäsittelyn ammattilaisten kanssa useita erilaisia sovelluksia tukevien ydinteknologioiden kehittämiseksi.

2.1 Ohjelmointi

Kautta aikojen ohjelmoijat ovat väitelleet eri ohjelmointikielten eduista. Jokaisella ohjelmoijalla on oma suosikkinsa. Hyvän ohjelmakoodin ominaisuudet ovat kuitenkin jokseenkin ohjelmointikielestä riippumattomia. Hyvin suunniteltu algoritmi voidaan toteuttaa lähes millä tahansa ohjelmointikielellä siten, että ohjelmakoodi on selkeära-

kenteinen ja helposti ymmärrettävä. Mikään käyttökelpoinen ohjelmointikieli ei pysty estämään huonon ohjelmakoodin kirjoittamista.

Ihmisille muoto ja tyyli ovat erittäin tärkeitä sisällön välittymisessä. Hyvin muotoiltu ja sujuvasti kerrottu tarina on helposti omaksuttavissa. Hyvä muoto on huomaamaton: lukija näkee sisällön eikä muotoa tai tyyliä. Hyvä muoto on myös elegantti. Se ei sotke sivua tai ajatustasi. Ohjelmakoodin lukijat arvostavat helppolukuista ohjelmakoodia, joka ei päästä ajatusta karkaamaan. Ohjelmien luettavuuteen vaikuttavia tekijöitä ovat esim. koodin kaikenpuolinen johdonmukaisuus, kommentoinnin määrä ja laatu sekä muuttujien nimeäminen.

2.2 Järjestelmien rakentaminen

Järjestelmien rakentaminen on ohjelmistotekniikan keskeisin haaste. Samantapaisia ongelmia kohdataan muillakin sovellusalueilla, kuten seuraavassa esiteltävän Vasa-laivan tarinan [Fairley and Willshire, 2003] kohdalla.

> *Elokuun kymmenes päivä 1628 Ruotsin kuninkaallisen laivaston uusin laiva, Vasa, lähti neitsytmatkalleen Tukholman satamasta. Hieman yli kilometrin purjehduksen jälkeen pieni tuulenpuuska kaatoi ja upotti laivan. Vuonna 1961 eli 333 vuotta myöhemmin alus nostettiin. Nykyisin alukseen ja sen tarinaan pääsee tutustumaan sekä Tukholman Vasamuseossa että Internetissä.*
>
> *Tammikuussa 1625 Ruotsin kuningas Kustaa II Adolf määräsi amiraali Flemingin tilaamaan Henrik ja Arend Hybertssonilta neljä sota-alusta, kaksi kölimitaltaan 108-jalkaista ja kaksi 135-jalkaista. Nämä alihankkivat aluksen varsinaisen rakentamisen laivanrakentaja Johan Isbrandssonilta. Vasa-laivan rakentaminen alkoi vuoden 1626 alussa tavanomaisena laivanrakennustehtävänä. Aluksen valmistuessa kaksi ja puoli vuotta myöhemmin se ei ollut enää tavanomainen vaan ensimmäinen lajissaan.*
>
> *Marraskuussa Kustaa II Adolf kasvatti tilattujen pienempien laivojen kölimitaksi 120 jalkaa, jotta ne pystyisivät kantamaan kolmekymmentäkaksi 24 paunan (ammuksen paino) kanuunaa. Yksi tällainen kanuuna painoi noin 1400 kiloa. Henrik Hybertssonilla oli kuitenkin käytettävissään raakapuuta vain kahteen alukseen, yhteen 111-jalkaiseen ja yhteen 135-jalkaiseen.*
>
> *Kun 111-jalkaisen aluksen kölirakenne oli tehty, Kustaa II Adolf vaati, että alus suurennetaan 135-jalkaiseksi ja että alukseen tulee kaksi kanuunakantta. Kukaan Ruotsissa ei ollut koskaan rakentanut kahden kanuunakannen alusta. Aikataulupaineista johtuen aluksen rakentajat olettivat, että 111-jalkainen alus voidaan kasvattaa 135-jalkaiseksi kölirakennetta muuttamatta.*
>
> *Vasa-laivan rakenteesta, niin 111-jalkaisesta kuin 135-jalkaisesta, ei ole mitään suunnitteludokumentaatiota, ei edes karkeita luonnoksia. Tällaiseen materiaaliin ei löydy viittauksia missään tuon ajan asiakirjoissa. Siten on erittäin todennäköistä, että missään vaiheessa Vasa-laivan rakentamisesta ei ole tehty rakentamissuunnitelmaa.*

Jälkikäteen Hybertssonien lähin avustaja, Hein Jacobsson, kertoi, että Vasa-laivan leveyttä kasvatettiin noin puolella metrillä, jotta alukseen saataisiin kaksi kanuunakantta. Koska köli oli jo rakennettu, levennys voitiin tehdä vain aluksen ylärakenteisiin. Tämä kohotti aluksen painopistettä ja lisäsi Vasa-laivan kiikkeryyttä.

Myös Vasa-laivan aseistukseen tehtiin muutoksia. Aluksen 111-jalkaiseen versioon oli alun perin ajateltu kolmekymmentäkaksi 23 paunan kanuunaa, joiden yhteispaino olisi ollut lähes 50 tonnia. Lopullisessa 135-jalkaisessa aluksessa aseistuksen määrä ja yhteispaino oli 50 %:a suurempi. Koska puolet kanuunoista oli ylemmällä kannella, niin aluksen painopiste nousi yhä ylemmäksi.

Kustaa II Adolf vaati alukseen satoja ornamentteja ja puuveistoksia. Vasa-laivan oli oltava myös ulkonäöltään vaikuttava. Raskaat tammipuiset veistokset nostivat osaltaan aluksen painopistettä ja lisäsivät kiikkeryyttä.

Henrik Hybertsson sairastui vakavasti vuonna 1626 ja kuoli seuraavana vuonna, vuotta ennen aluksen valmistumista. Sairastelunsa aikana Henrik jakoi rakentamisen johtamisen Jacobssonin ja Ibsbrandssonin kanssa. Historialliset dokumentit kuitenkin osoittavat, että rakentamisen hallinnointi oli lähes olematonta. Vastuunjako kolmen keskeisen henkilön osalta oli epäselvää, ja heidän välinen yhteydenpito lähes olematonta.

Koska aluksen rakentamisesta ei ollut minkäänlaisia suunnitelmia, Jacobssonin oli lähes mahdotonta ymmärtää ja toteuttaa Hybertssonin dokumentoimattomia suunnitelmia. Kolmen avainhenkilön välinen huono kommunikointi vain lisäsi ongelmia ja viivästytti aluksen valmistumista.

Henrik Hybertssonin kuoleman jälkeen amiraali Fleming nimitti Jacobssonin hankkeen vastuulliseksi johtajaksi. Vuodesta 1627 hankkeessa työskenteli noin 400 ihmistä viidessä eri ryhmässä. Asiakirjat eivät osoita, että ryhmät—laivan runko, veistokset, takila, aseistus, painolasti—olisivat olleet vuorovaikutuksessa toistensa kanssa.

1600-luvulla ei ollut käytettävissä menetelmiä, joilla olisi osattu laskea aluksen painopiste, kallistumisominaisuuksia tai tasapainoisuutta. Siten aluksen kapteenin oli opittava aluksen käyttäytyminen yrityksen ja erehdyksen kautta. 1960-luvulla tehtyjen laskelmien mukaan Vasa-laiva oli niin kiikkerä, että se olisi kaatunut 10 asteen kallistumasta. Viimeisempien laskelmien mukaan neljän solmun (2 m/s) tuuli olisi kaatanut aluksen.

Vasa-laivan kapteeni Hannson ja miehistön ydinjoukko tekivät kesällä 1628 aluksen vakaustestin amiraali Flemingin valvonnassa. Kolmekymmentä miestä juoksi ryhmänä aluksen laidalta toiselle. Kolmen kannenylityksen jälkeen testi oli keskeytettävä, sillä alus uhkasi kaatua. Tästä huolimatta Vasa-laiva päätettiin lähettää matkaan.

Monet Vasa-laivan ongelmat ovat vielä tänään nähtävissä useissa ohjelmistoprojekteissa. Seuraavassa on lyhyt yhteenveto ohjelmistoprojektien ongelmista ja niihin kehitetyistä ratkaisuista:

1. Kiireinen aikataulu

Monien ohjelmistoprojektien aikataulu on epärealistisen tiukka.

2. Tarpeiden muuttuminen

Tavanomaisimpia syitä ohjelmistojen vaatimusten muuttumiseen ovat ulkoiset kilpailutekijät, käyttäjien tarpeiden muuttuminen, suoritusympäristön muuttuminen ja projektin aikana tarkentunut näkemys tavoitteena olevasta järjestelmästä. Vaatimusten muuttumisen vaikutuksia voidaan hallita joko iteratiivisilla ohjelmistotuotannon menetelmillä tai perusratkaisun hallinnoinnilla (baseline management).

Iteratiivisia menetelmiä käytettäessä vaatimusten muuttumiset voidaan helpohkosti priorisoida ottaen huomioon aikataulun, resurssien ja teknologioiden asettamat rajoitukset. Perusratkaisujen hallinnassa vaatimukset ja niiden muutokset käsitellään versioiden hallinnan menetelmillä. Hyväksytyt muutokset synnyttävät vaatimusten uuden version (ja uuden perusversion) siten, että aikataulut, resurssit, teknologiat ja muut tekijät on asianmukaisesti päivitetty.

3. Teknisten määritysten puuttuminen

Ohjelmistoprojekti voi alkaa pienenä ja tavanomaisena hankkeena, jossa tekniset määritykset tuntuvat tarpeettomilta. Joskus tällainen projekti muuttuu matkan varrella laajaksi ja innovatiiviseksi. Näin kävi Vasa-laivan tapauksessa. Suullinen tiedonvaihto saattoi jossakin määrin korvata kirjallisten määritysten ja projektisuunnitelman puuttumista. Ohjelmistoprojektissa, oli se kuinka pieni tahansa, alustavien ja perusratkaisun vaatimusten kirjaaminen on paljon helpompaa ja vähemmän riskialtista kuin niiden myöhempi kaivaminen ohjelmakoodista.

4. Projektisuunnitelman puuttuminen

Projektisuunnitelman tekemiseen pätevät samat huomiot kuin teknisten määritysten tekemiseen. Pienessäkin hankkeessa kannattaa tehdä projektisuunnitelma heti alussa. Myöhemmin projektisuunnitelman konstruointi on huomattavan hankalaa.

Ohjelmistoprojektin projektisuunnitelmassa on oltava

- tehtävän työn jaottelu osatehtäviksi,
- vaatimusten sijoittaminen osatehtäviin,
- aikataulu tarkastuspisteineen ja virstanpylväineen,
- kunkin osatehtävän tuotokset määräaikoineen,
- tarvittavien ohjelmistojen hankintasuunnitelma,
- alihankinnan hallinnointisuunnitelma ja
- vastuiden selkeä kirjaaminen.

5. Yletön innovointi

Usein konstruointiprojektit epäonnistuvat, koska yritetään innovoida pitkälti yli sen

hetkisen tietotaidon. Ohjelmistoprojekteissa luodaan joko aivan uusi järjestelmä tai olemassa olevasta järjestelmästä uusi versio.

Ohjelmistoprojektien yletöntä innovointia voidaan hallita seuraavilla menetelmillä:

- työdokumenttien (vaatimukset, projektisuunnitelma, suunnitteludokumentit, testaussuunnitelmat, lähdekoodi) versioiden hallinta,
- ehdotettujen muutosten vaikutusten analysointi ja
- jatkuva riskien hallinta.

Aivan kuin Vasa-laivan tapauksessa, ohjelmistoprojekteissakin toissijaiset vaatimukset voivat muodostua dominoiviksi. Siksi ohjelmistoprojekteissa olisi oltava vastuullinen ohjelmistoarkkitehti, joka huolehtii ohjelmatuotteen eheydestä.

6. Vaatimusten luisuminen

Näyttää siltä, että Vasa-laivan rakentamisen aikana kenelläkään ei ollut kokonaiskuvaa kaikkien tehtyjen muutosten vaikutuksista. Sama tilanne pääsee helposti syntymään myös ohjelmistoprojekteissa.

Keskeiset tekniikat, joilla ohjelmistoprojektit pidetään hallinnassa, ovat alustavan dokumentaation luominen sekä vaatimusten ja suunnitelmien toistuva päivittäminen, jotta vaatimusten, aikataulun ja resurssien välillä säilyy hyväksyttävissä oleva tasapaino. Usein kuultuja tekosyitä alustavan projektidokumentaation tekemättä jättämiselle ovat riittävän tiedon puuttuminen sekä uskomus, että suunnittelu on hyödytöntä, koska kaikki kuitenkin muuttuu.

Alustavat vaatimukset ja suunnitelmat tulee tehdä, vaikka kaikkea tarvittavaa tietoa ei olisikaan käytettävissä. Alustavissa projektidokumenteissa on varauduttava muutoksiin sekä esitettävä muutosten hallintamenetelmät. Alustavien vaatimusten ja suunnitelmien toistuvan päivittämisen epäonnistuminen johtuu usein asenteesta, että päivittämiselle ei ole riittävästi aikaa. Tämä asenne puolestaan johtuu vaatimusten ja suunnitelmien jatkuvan hallinnoinnin kehittymättömistä menetelmistä sekä tähän liittyvien riskien aliarvioimisesta.

7. Tieteellisten menetelmien puuttuminen

Koska ohjelmistolla ei ole fyysisiä ominaisuuksia, niin monet perinteisen insinööritaidon menetelmät eivät sovellu ohjelmistotekniikkaan. Päinvastoin kuin Vasa-laiva ja monet muut fyysiset esineet, ohjelmisto voidaan rakentaa vaiheittain siten, että ohjelmiston ominaisuuksia, kuten muistintarvetta, suorituskykyä, turvallisuutta, tietoturvaa ja luotettavuutta, voidaan seurata ohjelmistoprojektin aikana.

8. Olennaisen unohtaminen

Vasa-laivan tapauksessa kallistustesti osoitti, että alus on vaarallisen epävakaa. Tasapainottaville lisäpainoille ei ollut tilaa, ja vaikka tilaa olisikin ollut, niin lisäpainoja ei

olisi voitu käyttää, koska alemmat ampuma-aukot olisivat jääneet vesirajan alapuolelle. Vaikka monilla ohjelmistotekniikan alueilla ei ole tieteellisiä menetelmiä, niin karkeilla laskelmilla voidaan usein tunnistaa ratkaisevia suunnitteluvirheitä ja ohjelmistojen toiminnallisia puutteita.

9. Epäeettinen käyttäytyminen

Teknologisiin innovaatioihin liittyy usein eettisiä näkökohtia. Vasa-laivan tapauksessa aivan viime hetkillä oli tullut ilmeiseksi, että alus ei ollut merikelpoinen. Kuitenkin ne, joilla olisi ollut valtuudet estää neitsytmatka, sallivat aluksen lähdön satamasta. ACM:n ja IEEE:n eettisten säännösten [Gotterbarn et al., 1999] mukaan ”Ohjelmistoammattilaisten on otettava täysi vastuu työstään ja hyväksyttävä ohjelmisto vain siinä tapauksessa, että heillä on perusteltu usko, että se on turvallinen, täyttää määritykset, läpäisee tarkoituksenmukaiset testit, ei vähennä elämän laatua tai yksityisyyttä eikä vahingoita ympäristöä.” Ongelmia ja niiden ratkaisuja on vielä koottu taulukkoon 2.

Ongelma	**Ratkaisu**
Aikataulupaine	Objektiiviset arviot, resurssien lisäämien, resurssien parantamien, vaatimusten priorisointi, vaatimusten uudelleenkohdentaminen, tuotoksen vaiheistaminen
Tavoitteiden muuttuminen	Iteratiivinen ohjelmistokehitys, perusratkaisun hallinta
Teknisten määritysten puuttuminen	Alustavien määritysten tekeminen, määritysten päivittäminen, määritysten hallinnointi
Dokumentoidun projektisuunnitelman puuttuminen	Alustavan suunnitelman tekeminen, suunnitelman toistuva päivittäminen, projektisuunnitelman hallinnointi, nimetty projektipäällikkö
Yletön ja toissijainen innovointi	Perusdokumenttien hallinnointi, vaikutusten analysointi, jatkuva riskien hallinta, nimetty ohjelmistoarkkitehti
Vaatimusten luisuminen	Alustava vaatimusten perusversio, versioiden hallinnointi, riskien hallinta, nimetty ohjelmistoarkkitehti
Tieteellisten menetelmien puuttuminen	Prototyyppien tekeminen, vaiheittainen kehittäminen, suorituskykymittaukset
Olennaisen unohtaminen	Karkeat laskelmat, opittujen opetusten sulauttaminen
Epäeettinen käyttäytyminen	Eettinen työympäristö ja työtavat, henkilökohtainen eettisten säännösten noudattaminen

Taulukko 2. Ongelmia ja niiden ratkaisumahdollisuuksia.

2.3 Mallintaminen ja validointi

Mallintaminen tarkoittaa todellisuuden osan, tietojenkäsittelytieteissä esimerkiksi ohjelmiston tai tietojärjestelmän, esittämistä abstraktilla, käsitteellisellä, graafisella tai matemaattisella mallilla. Mallintamista on esimerkiksi kartta, joka on malli todellisesta maastosta, tai pienoismalli, joka on pienennetty malli todellisesta esineestä. Esimerkki matemaattisesta mallista on painovoimamalli, joka kuvaa kahden tai useamman kappaleen välistä vuorovaikutusta. Mallinnuksen tuloksia käytetään muun muassa ilmiön simulointiin, tutkimukseen ja käyttäytymisen ennustamiseen eri tilanteissa.

Tietojenkäsittelytieteissä malli on usein erityisellä mallinnuskielellä tehty yleisen tason kuvaus tietojärjestelmän tai sen osan toiminnasta. Mallinnuskielet ovat ohjelmistotuotannon perustyökaluja, joiden avulla prosessin vaiheita voi hallita.

Esimerkiksi UML (Unified Modeling Language) auttaa visualisoimaan olio-ohjelmointiin liittyviä suunnitelmia. Se on kokoelma yksinkertaisia graafisia merkintöjä, kuten laatikoita ja nuolia, joilla on määrätty merkitys. UML-mallinnuksen tekemistä varten on erityisiä työkaluohjelmia.

Esimerkki tietojenkäsittelytieteellisestä matemaattisesta mallista on vaikkapa sellainen, joka ennustaa tietokoneviruksen leviämistä tietoverkossa. Mallissa tehdään oletuksia siitä, kuinka helposti virus voi tarttua uusiin verkon solmuihin, kuinka kauan tartunnan saanut solmu levittää virusta eteenpäin ja millä ehdoilla verkon solmu on immuuni virustartuntaa vastaan. Toimiva malli antaa ennusteen siitä, kuinka nopeasti ja laajasti tietynlainen virus voi levitä verkossa.

Validointi on prosessi, jossa tarkistetaan, että prosessin kohde (tässä tapauksessa malli) täyttää jotkin tietyt kriteerit. Tietojärjestelmän mallin validointi tehdään yleensä yhdessä tilaajan tai tulevan käyttäjän kanssa. Matemaattisen mallin validointi tapahtuu vertaamalla sen antamia tuloksia simuloinnin antamiin tai reaalimaailmassa mitattuihin arvoihin.

2.4 Innovaatiot

Muutosten aikaansaaminen käytäntöihin on paljon vaikeampaa kuin uusien teknologioiden keksiminen. Innovaatiot ovat uusia tapoja tehdä asioita. Innovaatio on yksi teknologian arvostetuimmista alueista. Elinkeinoelämän johtajat pitävät sitä yhtenä olennaisimpana osaamisalueena – ainoana tapana taata markkinajohtajuus. Monet organisaatiojohtajat puhuvat välttämättömyydestä luoda innovaatiokulttuuri, koska muuten organisaatio keskittyy vain ratkaisemaan tämän päivän kiireisiä ongelmia. Miten opitaan taitavaksi innovoijaksi? Miten innovatiivisuutta voidaan opettaa?

Innovaatio ja keksintö

Koska uusilla ajatuksilla ei ole käytännön vaikutusta ellei niitä sovelleta käytäntöön, Denning [2004] tarkoittaa innovaatiolla uuden käytännön soveltamista. Siten innovaa-

tio on sosiaalinen muutos yhteisössä.

Denning tekee selkeän eron innovaation ja keksinnön välillä. Keksintö on uuden luomista: ajatuksen, esineen, laitteen tai toimintatavan. Riippumatta siitä, miten erinomainen keksintö on, se ei välttämättä muodostu innovaatioksi. Innovaation ja keksinnön erottaminen eri käsitteiksi on olennaista, koska innovoinnin käytännöt eivät ole keksimisen käytäntöjä. Keksinnöissä voidaan keskittyä teknologioihin. Innovaatioissa on otettava huomioon sosiaalinen yhteisö, mitä muut ihmiset arvostavat ja hyväksyvät otettavaksi käyttöön. Ethernetin keksijä Bob Metcalfe kuvaa osuvasti tätä eroa. Haastattelijan toteamukseen, että Ethernetin keksiminen mahdollisti hänen ostaa asunto Bostonin rantakaistaleelta, Metcalfe vastasi, että ei keksiminen, vaan Ethernetin myyminen kymmenen vuoden ajan.

Vaikka liikemaailman innovaatiot saavat suurimman osan julkisuudesta, innovaatioita tapahtuu kaikenlaisissa organisaatioissa ja yhteisöissä. Myös innovaatioiden vaikutusten laajuudet vaihtelevat.

Innovointi on innovaattorien työtä, jolla on omat lainalaisuudet ja käytännöt. Innovointia voidaan tehdä systemaattisesti ja sen perusperiaatteita voidaan opettaa. Onnistuminen ei vaadi inspiraatiota, erityistä lahjakkuutta tai älykkyyttä. Denningin mukaan innovoinnin menestys on koulutuskysymys. Innovoinnin ydin on ajatusten myyminen tai niiden saattaminen käytännön toteutuksiksi.

Druckerin [1985] laajan analyysin perusteella innovaatioprosessin viisi kulmakiveä ovat seuraavat:

- Mahdollisuuksien etsiminen – mahdollisuuden tunnistaminen eri lähteistä.
- Analysointi – projekti- tai liiketoimintasuunnitelman laatiminen, kustannusten, resurssien ja ihmisten identifiointi.
- Kuunteleminen – yhteisöjen huolenaiheiden kuunteleminen, ymmärtäminen mihin ihmiset ovat valmiita, ehdotusten muokkaaminen yhteisöä palveleviksi.
- Keskittyminen – keskeisen idean pelkistäminen yksinkertaiseksi sanomaksi ja siinä pitäytyminen, vaikka mieli tekisi koristella tai laajentaa sitä.
- Johtajuus – kehittää teknologia lajinsa parhaaksi, saada ihmiset ja yhteisöt käyttämään sitä, muokata sille markkinarako.

Kaikkein olennaisinta on etsiä mahdollisuuksia erilaisissa tavanomaisesta poikkeavissa tilanteissa. Drucker varoittaa, että uuteen tietämykseen perustuva innovointi on erityisen haastavaa. Se on kaikkein riskialtteinta ja sillä on pisin kypsymisaika.

Tällainen innovointi vaatii tarkan ajoituksen ja etsikkoaika on yleensä lyhyt. Kaiken lisäksi tällaiset alueet ovat kaikkein kilpailluimpia, sillä uusi tietämys saa suurimman huomion.

Yltiöpäisiä riskejä ottavat yrittäjät Drucker leimaa myyteiksi. Menestyneet yrittäjät käyttävät systemaattisia toimintatapoja, jotka vähentävät riskejä. Innovaatioita mahdollistavat esimerkiksi seuraavat tekijät:

- odottamattomat tapahtumat – odottamattomat onnistumiset tai epäonnistumiset, ulkoiset tapahtumat

- epäsuhdat – todellisuuden ja yleisen uskomuksen välinen kuilu
- yhteen sopimattomat kokonaisuuden osat
- prosessin tarpeet – kriittisessä prosessissa oleva pullonkaula
- liike-elämän rakennemuutos – uudet liiketoimintamallit, jakelu kanavat, toimintatavat
- demografia – ikärakenteessa, politiikassa, uskonnossa, elintasossa, jne. tapahtuvat muutokset
- ilmapiirin tai asenteiden muuttuminen – ihmisen maailmankuvan, muodin, tapojen, jne. muuttuminen
- uusi tietämys – uuden tietämyksen hyödyntäminen, johon usein liittyy tieteen tai tekniikan läpimurto ja aiemmin erillisten toiminta-alueiden yhdentyminen
- marginaaliset käytännöt, jotka voivat ratkaista nykyisissä keskeisissä käytännöissä esiintyviä jatkuvia häiriöitä.

2.5 Soveltaminen

Denningin mukaan soveltamisessa on kyse yhteistyöstä. Työskennellään yhdessä tietojenkäsittelyn sovellusalueiden ammattilaisten kanssa näitä palvelevien tietojenkäsittelyjärjestelmien toteuttamiseksi. Toinen alue on työskentely muiden tietojenkäsittelyn ammattilaisten kanssa useita erilaisia sovelluksia tukevien ydinteknologioiden kehittämiseksi.

Ns. ihmiskeskeinen suunnittelu (Human-Centered Design) on noussut viimeisten vuosien aikana dominoivaksi teemaksi suunnittelussa. Ihmiskeskeisen suunnittelun eräs keskeinen periaate on kuunnella käyttäjiä. Tämä ei yksistään riitä. Ohjelmistoammattilaisten tulee tietää, miten käytettävyyttä (usability) mitataan ja miten havaintoaineiston perusteella tehdään päätöksiä. Seffah [2003] esittää 19 taitoa, joita tarvitaan menestyksekkäässä ihmiskeskeisessä suunnittelussa. Välttämättömiin edellytyksiin Seffah katsoo kuuluvan seuraavat kaksi taitoa[2]:

1) Ohjelmistojen kehitysmenetelmien perusperiaatteiden, perusteiden ja standardien sekä käyttöliittymien kehitystyövälineiden ja oliopohjaisten suunnittelumenetelmien perusteiden hallitseminen.
2) Tietokoneiden käyttökokemus sekä ohjelmointikokemus käyttäen nopean kehittämisen työvälineitä ja graafisten käyttöliittymien kehitysympäristöjä.

Näitä taitovaatimuksia Seffah täsmentää seuraavasti

3) Ihmisen ja koneen vuorovaikutuksen suunnittelun vaikeudet ja haasteet: vuorovaikutteisen järjestelmän rakentamisen ymmärtäminen monitieteisenä suunnittelualana, jolla on sen omat erityispiirteet ja vaikeudet.
4) Ihmiskeskeisen suunnittelun filosofia: ihmiskeskeisen suunnittelun filosofian, sen perusteiden, terminologian, faktojen, periaatteiden ja prosessien hallitseminen.

[2] Luettelon kohdat 1-19 sisältävät monia sellaisia käsitteitä, joita tässä esityksessä ei voida esitellä tarkemmin; kiinnostunut lukija löytää helposti lisätietoja Internetistä.

5) Asiantuntijavetoinen arviointi: varhaisten suunnittelukonseptien ja alhaisen totuudenmukaisuuden prototyyppien ilman käyttäjiä tapahtuva katselmointi ja arviointi.
6) Käyttäjien ominaisuudet, tehtävät ja käyttöliittymien vaatimukset: käyttäjien ominaisuuksien, heidän tehtäviensä ja työympäristön sekä käytettävyyden ja käyttöliittymän vaatimusten kokoaminen, analysointi ja määrittely.
7) Näytönsommittelu, visuaalinen ja tietosisällön suunnittelu: Näytön ulkoasun ja tietosisällön sekä visuaalisten kielikuvien, ulkoasun ja värienkäytön valinta ja suunnittelu.
8) Vuorovaikutuksen suunnittelu: ihmisen ja koneen välisen keskustelun sekä järjestelmän antaman palautteen, erityisesti virheilmoitusten ja opasteiden mallintaminen ja määrittäminen sekä tarkoituksenmukaisen vuorovaikutuksen tyylien ja laitteiden valinta.
9) Prototyyppien tekeminen: alhaisen, keskitason ja suuren totuudenmukaisuuden prototyyppien (kuten laitemallit (mockups), tarinataulut (story boards), ohjelmat, videot, paperiprototyypit) kehittäminen.
10) Käyttöön perustuva testaus: täysin toiminnallisiin järjestelmiin ja suuren totuudenmukaisuuden prototyyppeihin perustuvien käytettävyystestien suunnittelu ja toteutus kehityksen ja käyttöönoton eri vaiheissa.
11) Esikuvat ja ohjekirjat: Suunnittelumallien, ohjekirjojen ja tyylioppaiden avulla saatujen käyttäjäkokemusten ja parhaiden suunnittelukäytäntöjen kokoaminen, levittäminen ja hyväksikäyttäminen.
12) Käyttöohjeet, suoritusaikaiset opasteet ja tukijärjestelmät: käyttäjätuen toteuttaminen sisältäen käyttöohjeet, avustavat ohjelmat (*wizards*), suoritusaikaiset opasteet, käyttäjien kouluttamisen ja palautejärjestelmät.
13) Graafisen käyttöliittymän suunnittelu: ihmiskeskeisen suunnittelun menetelmien ja työvälineiden käyttäminen graafisten käyttöliittymien ja toimiston pöytäkoneille tarkoitettujen sovellusten toteuttamisessa.
14) Web-sovellusten suunnittelu: ihmiskeskeisen suunnittelun menetelmien ja työvälineiden käyttö Web-sovellusten toteuttamisessa.
15) Liikkuvien sovellusten suunnittelu: ihmiskeskeisen suunnittelun menetelmien ja työvälineiden käyttäminen liikkuvien ja langattomien sovellusten toteuttamisessa kämmenlaitteille ja matkapuhelimille.
16) Ihmiskeskeinen suunnittelu ohjelmiston elinkaaressa: käyttöliittymien suunnittelun käytettävyystekniikoiden menetelmien yhdistäminen ohjelmistokehityksen elinkaareen ja menetelmiin.

Kolmanteen ryhmään Seffah on koonnut joitakin yleistaitoja tai geneerisiä taitoja, joita tarvitaan lukuisissa ihmiskeskeisen suunnittelun tehtävissä. Toisissa yhteyksissä näitä taitoja on kutsuttu ”pehmeiksi taidoiksi” (“soft skills”). Seffahin mukaan ihmiskeskeisen suunnittelun koulutuksessa on opetettava myös näitä ”pehmeitä taitoja”. Ei riitä, että opetetaan käyttöliittymien suunnittelua. Ohjelmistoammattilaisille on myös

opetettava, miten käyttäjät ja muut asianosaiset (stageholder) otetaan mukaan ja miten heidän kanssa ollaan yhteistoiminnassa ohjelmiston koko elinkaaren aikana.

Seffahin mukaan kolme keskeisintä yleistaitoa ovat

17) Suurelle yleisölle tarkoitettujen, vaatimuksiin, suunnitteluun ja testaukseen liittyvien käyttäjäkeskeisten dokumenttien kirjoittaminen ja esittely.
18) Ohjelmistojen ja käytettävyyden työvälineiden ja menetelmien tutkiminen empiirisesti ja kustannustehokkaasti.
19) Monialaisessa ryhmässä kommunikointi ja työskentely, erityisesti asiakkaiden, käyttäjien ja käyttäytymistutkijoiden kanssa.

LUKU 3: Tietojenkäsittelyn ydinteknologiat

3.1. Nykyisiä ydinteknologioita

1950-luvulla tietojenkäsittelytieteen ydinteknologioihin laskettiin kuuluvaksi algoritmit, numeeriset menetelmät, laskennan mallit, kääntäjät, ohjelmointikielet ja logiikkapiirit. 1980-luvun loppuun mennessä joukkoon oli lisätty käyttöjärjestelmät, tiedonhaku, tietokannat, tietoverkot, tekoäly, ihmisen ja tietokoneen vuorovaikutus ja ohjelmistotekniikka.

Denningin tuorein luettelo [Denning, 2003] sisältää jo 30 ydinteknologiaa. Seuraavassa on aakkosjärjestyksessä luonnehdinnat 29 ydinteknologiasta (kaksi Denningin luettelon ydinteknologiaa on yhdistetty Raatikaisen [2007] tapaan):[3]

Algoritmit (algorithms). Termi algoritmi on johdettu 800-luvulla eläneen persialaisen matemaatikon al-Khrwarizmin nimestä. Sillä tarkoitetaan äärellistä joukkoa hyvin määriteltyjä ohjeita jonkin tehtävän suorittamiseksi. Algoritmitutkimuksessa kehitetään algoritmeja sekä analysoidaan niiden ominaisuuksia.

Hajautettu tietojenkäsittely (distributed computation). Hajautetussa tietojenkäsittelyssä tutkitaan fyysisesti eri paikoissa olevien tietokoneiden yhteistoimintaa. Tavoitteena on tarjota käyttäjille tietojenkäsittelyresursseja läpinäkyvästi, avoimesti ja laajentuvasti eli skaalautuvasti. Hajautetulla tietojenkäsittelyllä tavoitellaan parempaa saatavuutta, vikasietoisuutta ja suoritustehoa.

Ihmisen ja tietokoneen vuorovaikutus (human-computer interaction, HCI). Ihmisen tietokoneiden välinen vuorovaikutus on monitieteinen tutkimusalue. Tietojenkäsittelytieteiden osalta se keskittyy käyttöliittymään (user interface, UI), joka kattaa sekä laitteiston (tietokoneen oheislaitteet) että ohjelmiston. Teknologian perustavoitteena on tehdä tietokoneista ja tietojärjestelmistä käyttäjäystävällisiä (user-friedly) ja helppokäyttöisiä.

Johdon tietojärjestelmät (management information systems, MIS). Johdon tietojärjestelmä on (yleensä tietokonepohjainen) järjestelmä, joka kerää, muokkaa ja tallentaa tietoa sekä tarjoaa sen organisaation hallinnon käyttöön päätöksentekoa, suunnittelua, toteutusta ja seurantaa varten.

Konenäkö (vision). Konenäön tavoitteena on saada tietokone “ymmärtämään” kuvien

[3] Ydinteknologioiden suppeat kuvaukset sisältävät termejä, joita tässä esityksessä ei voida esitellä tarkemmin; kiinnostunut lukija löytää helposti lisätietoja Internetistä.

sisältöä. Tässä yhteydessä “ymmärtäminen” on hyvin rajallista. Konenäössä kuvien sisällöstä etsitään tarkkaan määriteltyä, tiettyä tarkoitusta palvelevaa informaatiota. Tämä informaatio välitetään joko ihmiselle (lääkäri saa röntgenkuvan, jossa epäilyttävät alueet on korostettu) tai jotakin prosessia ohjaavalle järjestelmälle (automaattisen varastotrukin liikkumisen ohjaus tai virheellisten tuotteiden poistaminen pakkauslinjalta). Jotkut pitävät konenäköä osana tekoälyä, jossa järjestelmän toiminnanohjaus saa syötteen kuva-aineistona ja oppii tunnistamaan haluttuja, yleensä poikkeavia tilanteita. Koska kameraa voidaan pitää valotunnistimena (light sensor), niin konenäön monet menetelmät perustuvat valon fysikaalisten ominaisuuksien ja kuvien välisiin vastaavuuksiin. Kolmas konenäköön keskeisesti vaikuttava tieteenala on biologia, erityisesti erilaisten näkemisjärjestelmien rakenne ja toiminta (silmän fysiologia). Myös signaalinkäsittely (signal processing) liittyy konenäköön. Useat yksiulotteisen signaalin käsittelyyn kehitetyt menetelmät voidaan melko suoraviivaisesti yleistää konenäössä tarvittavaan kaksi- tai moniulotteisen signaalin käsittelyyn. Toisaalta kuvainformaation erityisluonteen vuoksi konenäössä on kehitetty menetelmiä, joilla ei ole vastinetta yksiulotteisessa signaalinkäsittelyssä. Konenäön keskeisiä osa-alueita ovat mm. esineiden tunnistaminen (object recognition), kohteen seuraaminen (tracking), näkymän tulkitseminen (scene interpretation) ja itsepaikannus (ego positioning).

Käyttöjärjestelmät (operating systems). Käyttöjärjestelmä on se osa tietokonejärjestelmän ohjelmistoa, joka huolehtii laitteiston ja järjestelmän perustoiminnallisuuden valvonnasta ja hallinnasta. Käyttöjärjestelmän tarjoamien järjestelmäkutsujen (system call) avulla ohjelmat pääsevät käyttämään oheislaitteita (peripherials), tiedostoja ja keskusmuistia. Käyttöjärjestelmä huolehtii myös keskeytyksistä (interrupts), ajastimista (timers), prosesseista (processes) ja säikeistä (threads) sekä niiden vuorottamisesta (scheduling). Käyttöjärjestelmä huolehtii myös samanaikaisuuden hallinnasta (concurrency control), samanaikaisesti suorituksessa olevien ohjelmien eristämisestä ja prosessien välisestä kommunikoinnista (interprocess communication, ipc). Toisinaan käyttöjärjestelmiin lasketaan kuuluvaksi myös kirjastot (libraries), jotka helpottavat järjestelmäkutsujen käyttöä eri ohjelmointikielissä. Käyttöjärjestelmätutkimuksen keskeisiä alueita ovat mm. muistinhallinta (memory management), tiedostojärjestelmät (file systems), samanaikaisuuden hallinta, vikasietoisuus (fault-tolerance) ja virrankulutuksen hallinta (power management).

Kääntäjät (compilers). Kääntäjä on ohjelmisto, joka muuntaa lähdekieliset (source code) lauseet tuloskielisiksi (object code) lauseiksi. Tyypillisesti kääntäjän tuottama objektikoodi on konekieltä, johon on lisätty tietoa nimistä ja niiden sijainneista sekä ulkoisista funktioista. Suorituskelpoinen koodi (executable) saadaan linkittämällä yksi tai useampi objektikoodi sekä kirjastoja. Useimmat nykyiset kääntäjät on suunniteltu kaksivaiheisiksi. Ensimmäisessä vaiheessa käännetään lähdekoodi välimuotoon (intermediate representation). Tämä vaihe sisältää sanastollisen eli leksikaalisen (lexical),

muotosääntöjen eli syntaktisen (syntax) ja merkitystä koskevan eli semanttisen (semantic) analysoinnin. Toisessa vaiheessa välimuotoinen koodi muunnetaan objektikoodiksi. Tämän vaiheen asioita ovat kääntäjäanalyysi (compiler analysis), optimointi (optimization) ja koodin generointi (code generation).

Laskennallinen tiede (computational science). Laskennallinen tiede on muiden tieteenalojen tutkimusongelmia kuvaavien mallien ratkaisemista tietokoneen avulla. Tieteellisessä laskennassa tarkastellaan eri tieteenaloilla esiintyvien matemaattisten mallien numeerisia ratkaisumenetelmiä sekä niiden tietokonetoteutuksia. Historiallisesti tieteellinen laskenta jatkaa numeeristen menetelmien perinteitä tietojenkäsittelytieteessä. Laskennallisesta tieteestä on alettu puhua, kun tietokoneiden ja laskentamenetelmien käyttö on laajentunut eksakteista luonnontieteistä ja teknisistä tieteistä biotieteisiin ja lääketieteeseen.

Luonnollisen kielen käsittely (natural language processing, NLP). Luonnollisten kielten käsittely on tekoälyn ja kielitieteen (linguistics) osa-alue. Tällä tutkimusalueella tarkastellaan luonnollisen kielen käsittelyyn ja ymmärtämiseen liittyviä ongelmia. Tavoitteena on saada tietokone “ymmärtämään” ihmisten käyttämiä kieliä.

Ohjelmistotekniikka (software engineering). Ohjelmistotekniikka tarkastelee ohjelmistojen suunnitteluun, toteuttamiseen ja ylläpitoon liittyviä teknologioita ja käytäntöjä. Siinä yhdistyvät tietojenkäsittelytieteen lukuisat muut ydinteknologiat, projektinhallinta (project management), insinööritaito (engineering) ja kulloisenkin sovellusalueen tietämys. IEEE [1990] määrittelee ohjelmistotekniikan olevan systemaattisen, kurinalaisen, kvantifioitavissa olevan lähestymistavan käyttämistä ohjelmiston kehittämisessä, käytössä ja ylläpidossa sekä tällaisten lähestymistapojen tutkimista.

Ohjemointikielet (programming languages). Ohjelmointikieli on täsmällisesti määritelty tapa antaa tietokoneelle toimintaohjeet. Se koostuu syntaktisista ja semanttisista säännöistä. Ohjelmointikieli määrittelee ohjelmoijan käytettävissä olevat tietotyypit (data types), tietorakenteet (data structures) ja lauseet (statements). Ohjelmointikielten tutkimuksessa keskeisiä alueita ovat ohjelmointikielten ominaisuudet ja ohjelmointimallit.

Päätöksenteon tukijärjestelmät (decision support systems, DSS). Päätöksenteon tukijärjestelmät ovat tietokoneistettuja informaatiojärjestelmiä, jotka tukevat organisaatioiden päätöksentekoa. Käsitteenä DSS on hyvin laaja ja eri kirjoittajat antavat sille hyvinkin erilaisia määritelmiä.

Reaaliaikajärjestelmät (real-time systems). Reaaliaikaisessa tietojenkäsittelyssä tarkastellaan järjestelmiä (sekä laitteistoja että ohjelmistoja), joiden on toimittava etukäteen annettuja aikarajoitteita noudattaen. Reaaliaikaisen järjestelmän ei välttämättä tarvitse olla nopea, mutta tulokset on saatava valmiiksi ennalta määrättyyn aikarajaan

(deadline) mennessä. Nämä aikarajat eivät saa riippua järjestelmän kuormituksesta. Reaaliaikajärjestelmät luokitellaan koviin (hard) ja pehmeisiin (soft) aikarajojen ehdottomuuden perusteella. Kovan reaaliaikajärjestelmän on aina ja kaikissa mahdollisissa tilanteissa saatava jokainen tehtävä suoritettua määräaikoihin mennessä. Tällaisia järjestelmiä ovat mm. ydinvoimaloiden ohjaus- ja hallintajärjestelmät, lennonjohtojärjestelmät sekä monet erilaisia laitteita ohjaavat järjestelmät.

Rinnakkaislaskenta (parallel computation). Rinnakkaislaskennassa yksi tehtävä jaetaan osatehtäviin, joita suoritetaan samanaikaisesti usealla prosessorilla (vrt. hajautettu tietojenkäsittely). Rinnakkaislaskennan tavoitteena on nopeuttaa tehtävän suorittamista. Termillä rinnakkaisprosessori (parallel processor) tarkoitetaan tietokonetta, jossa on useita suorittimia (processor, central processor unit, CPU) yhden käyttöjärjestelmän hallinnassa. Kun järjestelmässä on tuhansia prosessoreita, niin puhutaan massiivisesta rinnakkaisuudesta (massively parallel). Moniprosessorikoneessa (multiptrocessor) on tyypillisesti muutamia suorittimia. Rinnakkaislaskennan tutkimuskohteita ovat muun muassa laitteistoarkkitehtuurit – erityisesti prosessorien välinen ja prosessorien ja muistien välinen kytkentä (interconnection) – sekä rinnakkaislaskentaan soveltuvat algoritmit ja säikeiden välinen kommunikointi.

Robotiikka (robotics). Robotiikka tutkii robottien suunnitteluun, rakentamiseen ja käyttöön liittyviä kysymyksiä. Robotiikan kehittäminen edellyttää elektroniikan, mekaniikan ja ohjelmistotekniikan hallintaa. Tyypillisesti tiettyyn tehtävään soveltuvan robotin kehittämiseen tarvitaan sopivia havaintoja tekevät tunnistimet (sensors), ohjausalgoritmi(t) ja robotin fyysistä toimintaa ohjaavat säätimet (actuators).

Supertietokoneet (supercomputers). Supertietokoneiksi kutsutaan tietokoneita, jotka aikoinaan olivat laskentateholtaan maailman parhaita. Supertietokoneet ovat perinteisesti saavuttaneet laskentatehon kulloisenkin huipun käyttämällä innovatiivisia ratkaisuja rinnakkaisuuden lisäämiseksi käskyjen käsittelyssä, muistin käytössä ja operaatioiden suorituksessa. Supertietokoneet on lähes poikkeuksetta suunniteltu tietyn tyyppiseen tietojenkäsittelyyn, useimmiten numeeriseen laskentaan eli numeronmurskaukseen. Supertietokoneiden muistihierarkia on suunniteltu huolellisesti, jotta suorittimet eivät joutuisi odottamaan käskyjen ja datan saantia muistista.

Sähköinen kaupankäynti (e-commerce). Sähköinen kaupankäynti koostuu tuotteiden tai palveluiden jakelusta, ostamisesta, myynnistä, markkinoinnista ja tarjonnasta tietoverkkojen, ennen kaikkea Internetin, välityksellä. Sähköiseen kaupankäyntiin liittyviä osatoimintoja ovat mm. sähköinen varainsiirto, tuotantoketjun hallinta (supply chain management), sähköinen markkinointi (e-marketing) ja välitön tapahtumankäsittely (online transaction processing), sähköinen tiedonvaihto (electronic data interchange, EDI), automatisoidut varastokirjanpitojärjestelmät ja automatisoidut tiedonkeruujärjestelmät. Sähköinen kaupankäyntijärjestelmä on monitieteellinen haaste.

Teknologian toimivuus on perusedellytys, mutta sähköisen kaupankäynnin menestyminen edellyttää myös sopivia liiketoimintamalleja ja riittävää, tietoturvaan ja vahvaan tunnistamiseen pohjautuvaa luottamusta. Viimekädessä osapuolten käyttökokemukset ratkaisevat sähköisten palvelujen henkiinjäämisen.

Tekoäly (artificial intelligence, AI). Tekoäly määritellään keinotekoisen luomuksen, yleensä tietokoneohjelman, osoittamaksi älykkyydeksi. Tekoälytutkimuksessa tarkastellaan järjestelmiä, jotka automatisoivat älykästä käyttäytymistä edellyttäviä tehtäviä. Tällaisia tehtäviä ovat mm. ohjaus (control), suunnittelu ja ajoitus (planning and scheduling) ja puheentunnistus (speech recognition). Tekoälyyn perustuvat järjestelmät ovat nykyisin laajalti käytössä mm. taloustieteissä ja lääketieteessä sekä erilaisissa peleissä.

Tiedonhaku (information retrieval). Tiedonhaussa keskitytään informaation etsimiseen (searching) dokumenteista, dokumenttien etsimiseen, dokumentteja kuvaavan metatiedon (metadata) etsimiseen sekä etsintään tietokannoista ja erilaisista tietoverkoista. Alun perin automatisoituja tiedonhakujärjestelmiä käytettiin hallitsemaan räjähdysmäisesti kasvavaa tieteellisten julkaisujen sisältämää informaatiota. Nykyisin painopiste on Internetin hakukoneissa.

Tiedon louhinta (data mining). Tiedon louhinnassa etsitään laajoista tietomassoista hahmoja (pattern) käyttäen esimerkiksi tilastotieteen ja hahmontunnistuksen (pattern recognition) laskennallisia menetelmiä. Teknologiasta käytetään englanninkielessä myös termiä knowledge-discovery in databases (KDD). Tavoitteena on löytää tietomassasta aiemmin tunnistamatonta ja mahdollisesti hyödyllistä informaatiota. Esimerkiksi kauppaketju voi profiloida asiakkaitaan ”plussakorttiostosten” avulla kerättyä tietomassaa analysoimalla.

Tietokannat (databases). Tietokanta on organisoitu kokoelma tietoa. Tietokannassa tieto on järjestetty tietueisiin (record), jotka koostuvat tietoalkioista (data elements). Tietokannan hallintajärjestelmäksi (database management system, DBMS) kutsutaan ohjelmaa, jonka avulla hallitaan tietokannassa olevaa tietoa ja voidaan kohdistaa kyselyjä (query) tietokannan tietosisältöön. Tietokannan käyttämä tietomalli (data model) määrää kyselykielet (query language), joiden avulla tietokantaa käytetään. Tietokannan käsittelyn nopeuttamiseksi voidaan käyttää indeksointia (indexing). Transaktioiden avulla hallitaan samanaikaisuutta (concurrency) eli tietokannan samanaikaista käyttöä. Transaktioilla voidaan taata ns. happo-ominaisuudet (ACID) eli atomisuus (atomicity), eheys (consistency), eristettävyys (isolation) ja pysyvyys (durability). Toisintamalla (replication) voidaan parantaa suorituskykyä (performance) ja saatavuutta (availability).

Tietokonearkkitehtuuri (computer architecture). Tietokonearkkitehtuuri on tietoko-

neiden rakenteen suunnittelun taustalla oleva teoria. Tähän kuuluu laitteiston suunnittelu siten, että laitteisto toimii ohjelmoijien olettamalla tavalla, ja toteutusteknologioiden, kuten puolijohteiden, käyttäminen siten, että laitteisto on paras mahdollinen. Tavallisimmin paras on kustannusten ja nopeuden välinen kompromissi. Muita keskeisiä tavoitteita voivat olla laitteen koko ja paino sekä virrankulutus.

Tietokonegrafiikka (graphics). Tietokonegrafiikka kattaa visuaalisen tietojenkäsittelyn. Siihen kuuluu sekä kuvamateriaalin synteettinen tuottaminen että todellisuudesta peräisin olevan visuaalisen informaation ja paikkatiedon (spatial information) muokkaaminen. Tietokonegrafiikassa on lukuisia osa-alueita, kuten tosiaikainen kolmiulotteinen kuvien esittäminen (3-D rendering), animointi, videosignaalin käsittely, visuaalisten tehosteiden luonti ja muokkaaminen sekä kuvan (image) muokkaaminen ja mallintaminen.

Tietorakenteet (data structures). Tietorakenne on tiedon talletustapa tietokoneessa. Tietorakenteen valinta vaikuttaa olennaisesti tiedon käsittelyn tehokkuuteen. Hyvin suunniteltu tietorakenne mahdollistaa tärkeimpien toimenpiteiden suorituksen mahdollisimman vähäisin resurssein (suoritusaika ja muistintarve). Koska tietorakenteet ovat ohjelmissa erittäin keskeisiä, useat ohjelmointikielet ja -ympäristöt tarjoavat tietorakenteiden käsittelyyn optimoituja kirjastorutiineja.

Tietoturva (data security). Tietoturva käsittelee informaatioon liittyviä luottamuksen (trust) eri aspekteja. Tietoturvaan liittyy mm. pääsynvalvonta (access control), luottamuksellisuus (confidentiality), tiedon eheys (integrity), saatavuus (availability), vastuullisuus (accountability), kiistämättömyys (non-repudiation), varmennettavuus (assurance) ja tunnistus (authentication). Tietoturvan keskeisin periaate on yksinkertainen ilmaista: *”oikea informaatio oikeille ihmisille oikeaan aikaan”*.

Tietoverkot (computer networks). Tietoverkko on erilaisilla fyysisillä tietoliikenneyhteyksillä ja tietoliikenneprotokollilla yhteen kytkettyjen tietokoneiden muodostama järjestelmä. Tietoverkot ovat nykyisin lähes kaikkien tietojärjestelmien keskeinen osa. Tietoverkkojen tutkimusalueita ovat muun muassa tietoliikennelaitteet, tiedon esitysmuodot, tietoturva, tietoliikenneprotokollat, verkonhallinta (network management), langaton tiedonsiirto (wireless communication) ja liikkuva tietojenkäsittely (mobile computing).

Työnkulku (workflow). Tietojenkäsittelyssä työnkulku liittyy organisaatioiden työtehtävien tekemisen järjestämiseen ja siihen, miten tietokonejärjestelmiä voidaan käyttää työn organisoimisen apuna. Keskeisiä kysymyksiä ovat: ”miten työtehtävät järjestetään?”, ”kuka suorittaa tietyn tehtävän?”, ”missä järjestyksessä työtehtävät on suoritettava?”, ”mitkä ovat tehtävän aloittamisen edellytykset?”, ”miten informaatiovirrat (information flows) tukevat tehtäviä?” ja ”miten tehtävien etenemistä seurataan?”.

Virtuaalitodellisuus (virtual reality, VR). Virtuaalitodellisuus on tietokoneella toteutettu simuloitu ympäristö. Useimmat virtuaaliympäristöt ovat ensisijaisesti visuaalisia kokemuksia, jotka näytetään tietokoneen näytöllä tai erityisellä stereoskooppisella näytöllä. Yhä useammin ympäristöt sisältävät myös kuvan kanssa synkronoidun äänen. Simuloitu ympäristö voi olla todellisuuden kaltainen, kuten lentäjien koulutuksessa käytettävät opetussimulaattorit. Toisaalta useiden videopelien simuloidulla ympäristöllä ei ole mitään tekemistä oikean todellisuuden kanssa. Yksityiskohtaisesti tarkan (high-fidelity) virtuaalitodellisuuden toteuttaminen on edelleen hyvin haastavaa.

Visualisointi (visualization): Visualisoinnin alueelle kuuluvat menetelmät, joilla luodaan kuvia, kaavioita tai animaatioita. Tavoitteena on parantaa tiedon välittymistä ja saada haluttu sanoma esitetyksi paremmin. Visualisointia käytetään yhä laajemmin tieteissä, tekniikassa, tuotekehityksessä ja tuotannossa, opetuksessa ja lääketieteessä. Tietokonegrafiikka on visualisoinnin tärkein apuväline. Visualisoinnin ja tietokonegrafiikan eroa voisi pelkistää siten, että visualisoinnissa keskitytään kysymykseen, mitä halutaan näyttää, tietokonegrafiikassa kysymykseen, miten haluttu visuaalinen ilme saadaan aikaan.

3.2. Tietojenkäsittelytieteet Tampereen yliopistossa

Tampereen yliopistossa tietojenkäsittelytieteitä on opetettu vuodesta 1965 lähtien. Silloin yliopistoon perustettiin Pohjoismaiden ensimmäinen tietojenkäsittelyopin professuuri. Nykyisin tietojenkäsittelytieteitä opetetaan ja tutkitaan kahdessa oppiaineessa, tietojenkäsittelyopissa ja vuorovaikutteisessa teknologiassa. Yhdessä yksikössä ei voida kattaa kaikkia tässä luvussa esitettyjä ydinteknologioita. Seuraavassa luetellaan niitä ydinteknologioita, jotka ovat edustettuina Tampereen yliopistossa. Ydinteknologioita kannattaa tarkastella maisteriopintojen opintosuuntien kautta. Tampereen yliopiston tietojenkäsittelytieteiden opintosuunnat ovat

- Laskentamenetelmien ja ohjelmoinnin opintosuunta (tko)
- Organisaatioiden tietojärjestelmien opintosuunta (tko)
- Tietokantojen ja tiedonhaun opintosuunta (tko)
- Ohjelmistokehityksen maisteriopinnot (tko)
- Ihmisen ja teknologian vuorovaikutuksen maisteriopinnot (vt).

Viimeksi mainitussa on vielä kolme suuntautumisvaihtoehtoa: Interaction Design and Research, Development of Interactive Software ja User Experience Design and Evaluation.

Laskentamenetelmien ja ohjelmoinnin opintosuunnan opetus keskittyy ydinteknologioista seuraaviin: algoritmit, tiedon louhinta, tietorakenteet ja tietokonegrafiikka. Lisäksi tutkimusta on tehty virtuaalitodellisuuden ja robotiikan alueilla. Organisaatioiden tietojärjestelmien opintosuunta antaa opetusta johdon tietojärjestelmien, sähköisen kaupankäynnin ja tietoturvan alueilla. Näiden alojen lisäksi tutkimusta on tehty mm. innovaatioiden ja tutkimusmenetelmien alueilla. Tietokanto-

jen ja tiedonhaun opintosuunnan ydinteknologioita ovat tietokannat ja tiedonhaku ja ohjelmistokehityksen ydinteknologioita hajautettu tietojenkäsittely, ohjelmistotekniikka, reaaliaikajärjestelmät ja tietoturva.

Ihmisen ja teknologian vuorovaikutuksen alueen keskeinen ydinteknologia on luonnollisesti ihmisen ja teknologian vuorovaikutus (!). Esimerkiksi Development of Interactive Software -suuntaan liittyy kuitenkin monia muita ydinteknologioita, kuten esimerkiksi ohjelmistotekniikka, konenäkö ja virtuaalitodellisuus. Edustettuna on myös visualisoinnin ydinteknologia.

Edellä olevasta luettelusta selviää, että eri opintosuunnilla ja maisterikoulutuksilla on yhteisiä ydinteknologioita, mikä on tietenkin luonnollista, kun ne kaikki kuuluvat tietojenkäsittelytieteisiin.

Informaatiotieteiden yksikön muissakin tutkinto-ohjelmissa on edustettuina Denningin ydinteknologioiden luettelossa olevia tutkimusalueita, selvin esimerkki on tiedonhaku, jota opetetaan ja tutkitaan myös informaatiotutkimuksen ja interaktiivisen median tutkinto-ohjelmassa.

LUKU 4: Tietojenkäsittelyn mekaniikat

Tietojenkäsittelyn ydinteknologioiden perusperiaatteiden analysoinnin perusteella Denning nimesi viisi perustoiminnallisuutta eli mekaniikkaa (ks. kuva 5):

- **laskenta,** jonka peruskysymykset liittyvät laskennan rajoihin: mitä voidaan laskea tai päätellä algoritmisin menetelmin;
- **kommunikointi,** jonka peruskysymykset liittyvät sanoman välittämiseen paikasta toiseen;
- **koordinointi,** jonka peruskysymykset liittyvät yhteistyöhön: miten vähintään kaksi toimijaa työskentelee yhteisen päämäärän hyväksi;
- **automatisointi,** jonka peruskysymykset liittyvät tietokoneella suoritettaviin kognitiivisiin (ihmiselle tyypillisiin tietojenkäsittelyn) tehtäviin;
- **muistaminen,** jonka peruskysymykset liittyvät informaation tallentamiseen ja hakemiseen.

Kuva 5. Tietojenkäsittelyn mekaniikkojen viisi näkymää [Denning, 2003]

Mekaniikat eivät ole toisiaan poissulkevia. Tässä luvussa kuvataan kustakin perustoiminnallisuudesta yksi esimerkki:

- **laskenta:** Turingin kone,
- **kommunikointi:** OSI-malli,
- **koordinointi:** synkronointi,
- **automatisointi:** Turingin testi,
- **muistaminen:** välimuistit.

4.1 Laskenta

Laskennan (*computing*) peruskysymykset liittyvät laskennan rajoihin: Mitä voidaan laskea tai päätellä algoritmisilla menetelmillä? Laskentaan liittyviä keskeisiä käsitteitä ovat esimerkiksi algoritmit (algorithms), ohjausrakenteet (control structures), tietorakenteet (data structures), automaatit (automata), formaalit kielet (formal languages), Turingin koneet (Turing machines), kompleksisuus (complexity), predikaattilogiikka (predicate logic), likimääräismenetelmät (approximation algorithms), heuristiset algoritmit (heuristics), satunnaisalgoritmit (probabilistic algorithms) ja ratkeavuus (solvability).

Turingin kone

Turingin koneet ovat yksinkertaisia abstrakteja laskentalaitteita. Ne ovat peräisin ajalta ennen varsinaisia tietokoneita, englantilaisen matemaatikon Alan Turingin julkaisusta vuodelta 1936.

Turing pyrki selvittämään sitä, mitä algoritmisilla menetelmillä voidaan laskea tai päätellä. Historiallisesti tämä liittyi tilanteeseen, jossa Kurt Gödel oli todistanut ns. epätäydellisyyslauseensa, jonka mukaan melko yksinkertaisissa loogisissa järjestelmissä voidaan muotoilla lauseita, joiden oikeellisuutta ei voida todistaa kyseisessä järjestelmässä. Turing määritteli abstraktin laskulaitteen, joka antaa matemaattisesti tarkan vastineen algoritmille.

Turingin kone on automaatti, jonka kontrolli jokaisella hetkellä on jossakin tilasysteemin tilassa. Automaatilla on toiseen suuntaan ääretön nauhamuisti, jonka jokaiseen muistipaikkaan voidaan tallentaa yksi merkki. Muistiin liittyy luku-kirjoitus -pää, jota voidaan siirtää edelliseen tai seuraavaan muistipaikkaan ja jonka avulla voidaan lukea nykyisen muistipaikan merkki ja kirjoittaa sen tilalle uusi merkki. Ohjelman suorituksen aikana tilasiirtymät määräytyvät nykyisen tilan ja muistipaikasta luetun merkin perusteella.

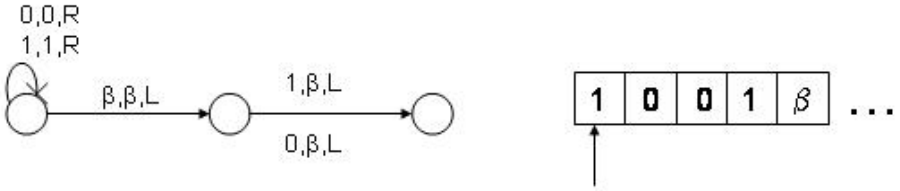
Turingin koneen toiminnan täydelliseen määrittämiseen tarvitaan koneen nykyinen tila, nykyisen muistipaikan arvo ja tilasiirtymät eli koneen ”ohjelma”. Jokainen tilasiirtymä on muotoa

< nykyinen tila, merkin lukeminen, uusi tila, merkin kirjoittaminen, pään siirtäminen >.

Nykyisen tilan ja pään kohdalla olevan merkin perusteella koneen tila muuttuu, luetun merkin kohdalle kirjoitetaan uusi merkki ja luku-kirjoitus -päätä siirretään nauhalla yksi askel vasemmalle tai oikealle. Jos kone päätyy tilanteeseen, jossa toimintaa ei ole määritelty, niin kone pysähtyy ja laskenta tulkitaan epäonnistuneeksi.

Alkutilanteessa Turingin kone on erityisessä alkutilassa, syöte on kirjoitettuna

nauhamuistille ja luku-kirjoitus -pää on syötteen ensimmäisen merkin kohdalla. Kaikissa niissä nauhan muistipaikoissa, joihin ei ole kirjoitettu merkkiä, oletetaan olevan ns. tyhjä merkki β. Kuvassa 6 on esimerkki yksinkertaisesta Turingin koneesta, joka saa syötteekseen binääriluvun ja poistaa siitä viimeisen bitin (korvaa sen tyhjä merkillä β).



Kuva 6. Esimerkki Turingin koneesta ja sen nauhan alkutilanteesta syötteellä 1001.

Turingin koneet näyttävät hyvin yksinkertaisilta. Kuitenkin niiden analysointi on osoittanut, että ne ovat laskennallisesti yhtä voimakkaita kuin nykyiset tietokoneet. Churchin-Turingin teesi olettaa, että Turingin koneilla voidaan toteuttaa kaikki mahdolliset algoritmit. Tässä yhteydessä on syytä korostaa, että laskennallinen voimakkuus tai laskentavoima tarkoittaa periaatteellista kykyä ratkaista jokin tehtävä, eikä se liity mitenkään tehtävän ratkaisemiseen kuluvaan aikaan eli laskentatehoon.

Turingin koneen yksinkertaisuus on tehnyt siitä tehokkaan työvälineen laskettavuuden teoriassa. Koneellaan Turing pystyi osoittamaan, että tietyt ongelmat ovat ratkeamattomia. Tällainen on esimerkiksi pysähtymisongelma (halting problem): ”Pysähtyykö Turingin kone annetulla syötteellä?”. Turingin todistus perustui vastaesimerkkiin. Olettamalla, että pysähtymisongelman ratkaiseva Turingin kone olisi olemassa, päädytään loogisesti mahdottomaan tilanteeseen.

Yksi Turingin kone laskee vain yhden laskettavissa olevan funktion. Tässä mielessä se on kuin tietokone, johon on asennettu yksi kiinteä ja muuttumaton ohjelma. Toisin sanoen jokainen ongelma tarvitsee oman Turingin koneensa. Turing osoitti, että on olemassa universaali kone, joka pystyy simuloimaan minkä tahansa Turingin koneen toimintaa sen millä tahansa syötteellä.

Laskennallisen vaativuuden merkityksestä

Laskennan teoriassa (theory of computation) käytetään Turingin koneen lisäksi muitakin laskennan malleja, kuten esimerkiksi rekursiivisia funktioita ja λ-kalkyyliä. Laskentamallien avulla tutkitaan laskettavuuden rajoja eli sitä, mitkä ongelmat ovat todistettavasti algoritmisesti ratkeamattomia, mitkä taas ratkeavia.

On olemassa ongelmia, jotka voidaan kyllä ratkaista tietokoneella, mutta ratkaisun tuottaminen kestää niin kauan, että ratkaisu on valmistumishetkellään käytännössä käyttökelvoton. Tällaisia hankalia eli NP-kovia (NP-hard, intractable) ongelmia on runsaasti, erityisesti tilanteissa, joissa halutaan löytää paras mahdollinen vaihtoehto

suuresta mahdollisten ratkaisujen avaruudesta. Tyypillisiä esiintymisalueita ovat mm. erilaiset graafiteorian ongelmat, skedulointi, pakkausongelmat ja pelit. Lisäksi laskennan vaativuusteoriassa analysoidaan eri algoritmien suorittamiseen tarvittavien operaatioiden lukumäärää ja tilan tarvetta.

Algoritmin aikavaativuus eli operaatioiden määrä voi olla esimerkiksi O(nlog n). Tulos luetaan muodossa: Jos algoritmille annetaan syötteenä n alkiota, niin ratkaisuun tarvittavien operaatioiden määrä on kertaluokkaa nlog n. Jos algoritmin vaativuus on $O(2^n)$, niin vaativuus kasvaa eksponentiaalisesti. Tällainen algoritmi on käyttökelpoinen vain hyvin pienillä syötteillä.

4.2 Kommunikointi

Kommunikaation (*communication*) peruskysymykset liittyvät sanoman välittämiseen paikasta toiseen. Kommunikaation keskeisiä käsitteitä ovat tiedonsiirto (data transmission), tiedon fyysinen esittäminen (encoding to medium), kanavan kapasiteetti (channel capacity), kohinanpoisto (noise suppression), tiedon tiivistäminen (data compression), salakirjoitus (cryptography), pakettiverkko (reconfigurable packet networks) sekä virheiden havaitseminen ja korjaaminen (error detection and correction).

OSI-malli

Tiedonsiirron käsitteellisenä kehikkona toimii useimmiten ISO:n (International Standardization Organization) OSI (Open System Interconnect) malli. OSI-mallissa (kuva 7) on seitsemän kerrosta:

Sovelluskerroksella keskitytään kunkin sovelluksen vuoropuheluun eli dialogin määrittelyyn, määritellään viestit, niiden rakenne ja merkitys. Yleisimpiä sovellustason protokollia ovat sähköposti, uutisryhmät ja Web.

Esitystapakerroksella keskitytään sanoman sisällön esitystapaan. Perinteisessä Internet-maailmassa tämä kerros on ollut lähes olematon. Se on jätetty sovelluksen sisäiseksi asiaksi tai sitten on käytetty yksinkertaista TLV (type-length value) -esitysmuotoa. Web-konsortion eli W3C:n (World Wide Web Consortium) määrittelemä XML eli eXtensible Markup Language on nykyisin yleinen sanoman sisällön esitystapa.

Kuva 7. ISO:n OSI-malli

Istuntokerros mahdollistaa istunnon (session) muodostamisen.

Kuljetuskerros keskittyy sanoman siirtämiseen päätepisteiden välillä. Internet-maailman keskeiset kuljetusprotokollat ovat TCP ja UDP. TCP on luotettavuuden takaava tietovuo (data stream). UDP on epäluotettava tietosähke (datagram).

Verkkokerroksella keskitytään pääasiassa sanomien reititykseen – mitä reittiä pitkin sanoma kulkee lähettäjältä vastaanottajalle. Verkkokerroksella huolehditaan myös ruuhkanhallinnasta. Internet-maailmassa verkkokerroksen keskeisin protokolla on IP eli Internet Protocol. IP:n ruuhkanhallinta on suoraviivaista: jos sanomia on liikaa, jotkut sanomat yksinkertaisesti tuhotaan.

Linkkikerroksen tärkein tehtävä on muuntaa fyysinen tiedonsiirtokanava tiedonsiirtolinjaksi siten, että fyysisessä tiedon siirrossa tapahtuvat virheet eivät näy verkkokerrokselle. Tavanomaisin tapa on ryhmitellä bitit muutaman sadan tai tuhannen tavun paketeiksi. Pakettiin liitetään joitakin ylimääräisiä bittejä, joiden avulla vastaanottaja pystyy päättelemään, onko kehyksen sisältö muuttunut matkan varrella. Muuttumattomat paketit kuitataan vastaanotetuiksi, rikkoutuneet pyydetään lähettämään uudelleen.

Fyysisellä kerroksella keskitytään bittien lähettämiseen tiedonsiirtokanavaa pitkin. Tyypillisesti määritellään, kuinka monta volttia käytetään esittämään arvoa 1 ja kuinka monta arvoa 0. Lisäksi määritellään bitin kesto. Muita määriteltäviä asioita ovat mm. fyysisen tiedonsiirtoyhteyden muodostaminen ja lopettaminen.

3.3 Koordinointi

Koordinoinnin (*coordination*) peruskysymykset liittyvät yhteistyöhön. Miten vähintään kaksi toimijaa työskentelee yhteisen päämäärän hyväksi? Denning jakaa nämä kysymykset kolmeen ryhmään: ihmisten välisen, ihmisen ja tietokoneen välisen, sekä tietokoneiden välisen yhteistyön kysymyksiin. Ihmisten välisen yhteistyön keskeisiä käsitteitä ovat toimintaketjut (action loops) ja tietokoneiden tukema työnkulku. Ihmisen ja tietokoneen välisen yhteistyön keskeisiä käsitteitä ovat rajapinta (interface), syöte (input), tuloste (output) ja vasteaika (response time). Tietokoneiden välisen yhteistyön keskeisiä käsitteitä ovat puolestaan synkronointi (synchronization), kilpatilanteet (race conditions), lukkiutuminen (deadlock), sarjallistuvuus (serializability) ja atomiset toimenpiteet (atomic actions).

Synkronointi

Synkronointi eli tahdistaminen on aikaan liittyvää koordinointia. Esimerkiksi monikanavaviestinnässä eri kanavien tietovirtojen tulee edetä samaa tahtia. Toisissa tilanteissa synkronoinnilla varmistetaan, että itsenäisten toimijoiden aikaansaannokset valmistuvat ”oikeassa” järjestyksessä.

Hajautetuissa järjestelmissä sanomiin liitetään usein lähettäjän aikaleima ja on tärkeää, että kellot on synkronoitu. NTP on Internetissä käytössä oleva protokolla, jonka avulla eri koneiden kellot saadaan synkronoitua siirtoviiveistä riippuen muutaman kymmenen millisekunnin (avoin Internet) tai muutaman sadan mikrosekunnin (nopea lähiverkko) tarkkuudella.

Erilaiset synkronointiongelmat liittyvät usein yhteiskäyttöisten resurssien hallintaan. Ratkaisujen on estettävä sekä nälkiintyminen (joku joutuu odottamaan vuoroaan ikuisesti) että lukkiutuminen (kaikki odottavat, että joku toinen tekisi jotain).

Synkronointiongelmista tunnetuimpia ovat seuraavat:

Tuottaja-kuluttaja -ongelma. Kuluttaja ei voi kuluttaa ennen kuin tuottaja on tuottanut. Toisaalta välivaraston täyttymisen jälkeen tuottajan on odotettava, että kuluttaja ehtii kuluttamaan. Kun tuottajia ja kuluttajia on useita, niin on myös huolehdittava poissulkemisesta. Tuottajien tuotokset on saatava välivarastossa eri paikkohin. Kaksi kuluttajaa ei saa saada samaa tuotosta. Lisäksi samanaikaiset lisäykset ja poistot eivät saa sotkea varastokirjanpitoa.

Lukija-kirjoittaja -ongelma. Kaikki voivat lukea samanaikaisesti, mutta vain yksi kerrallaan voi kirjoittaa. Kirjoittajan nälkiintymisen estäminen on erityisen tärkeää. Jos kirjoittaja ei saa kirjoitusvuoroa, niin kaikki lukijat lukevat vanhentunutta tietoa.

Kilpatilanne. Käsite kilpatilanne (race hazard) tarkoittaa järjestelmässä olevaa suunnitteluvirhettä, jonka seurauksena järjestelmän toiminta riippuu ennalta arvaamattomalla tavalla tapahtumien käsittelyjärjestyksestä. Käsite juontaa juurensa tilanteesta, jossa kaksi signaalia kilpailee siitä, kumpi pääsee ensin vaikuttamaan tulokseen. Kil-

patilanne syntyy, jos kaksi tai useampi ohjelma pääsee kontrolloimattomasti käsiksi yhteiskäyttöiseen resurssiin samanaikaisesti.

Aterioivien filosofien ongelma

Dijkstra [1972] käytti ryhmää aterioivia filosofeja esimerkkinä synkronointiongelman ratkaisemisessa: Viisi filosofia istuu pyöreän pöydän ympärillä. Jokaisen filosofin edessä on spagettilautanen. Jokaisen lautasen välissä on yksi haarukka. Siis viisi filosofia, viisi lautasta ja viisi haarukkaa.

Filosofien elämässä on kaksi vuorottelevaa tilaa: syöminen ja ajattelu. Jokainen filosofi tarvitsee kaksi haarukkaa syödäkseen, mutta haarukat otetaan yksitellen. Saatuaan kaksi haarukkaa filosofi syö jonkin aikaa, minkä jälkeen hän vapauttaa molemmat haarukat ja jatkaa ajatteluaan.

Jos jokainen filosofi saisi samanaikaisesti päähänsä tarttua oikeanpuoleiseen haarukkaan, niin syntyisi lukkiutuminen. Kullakin filosofilla olisi yksi haarukka ja jokainen odottaisi ikuisesti toisen haarukan vapautumista.

Vaihtoehtoisesti filosofit voisivat olla kohteliaita: Jos toinen haarukka ei ole vapaa, niin filosofi vapauttaa varaamansa haarukan, odottaa hetken ja yrittää varata haarukoita uudelleen. Nyt järjestelmä ei lukkiudu, mutta filosofit eivät silti pääse silti syömään ellei odotusaika ole satunnainen. Vaikka odotusaika olisikin satunnainen, niin ei ole mitään takeita, että jokainen filosofi pääsee joskus syömään.

Aterioivien filosofien ongelman ratkaisuun tarvitaan algoritmi, joka estää sekä edellä kuvatun lukkiutumisen että nälkiintymisen. Tällaisia ratkaisuja on useita.

4.4 Automatisointi

Automatisoinnin (automation) peruskysymykset liittyvät tietokoneella suoritettaviin kognitiivisiin tehtäviin. Automatisoinnin keskeisiä käsitteitä ovat kognitiivisten tehtävien simulointi (simulation of cognitive tasks), asiantuntemus ja asiantuntijajärjestelmät (expertise and expert systems), älykkyyden lisäminen (enhancement of intelligence), Turingin testit (Turing tests) sekä koneoppiminen ja tunnistaminen (machine learning and recognition).

Turingin testi

Turingin testin taustalla on englantilainen seuraleikki Imitation Game. Alan Turing tarjosi siihen perustuvaa testiään älykkyyden määritelmäksi. Vuonna 1950, jolloin Turing esitteli testinsä, keskustelu tietokoneen kyvyistä oli sangen värikästä. Tuohon aikaan tietokoneista, joita maailmassa oli muutama, puhuttiin lehdistössä myös ’ajattelevina koneina’.

Turingin testissä ihminen kirjoittaa kahdelle testattavalle kysymyksiä. Saamiensa kirjallisten vastausten perusteella hän yrittää erottaa, kumpi testattavista on tietokone.

Tekoälyn maailmassa Turing testi edustaa tekoälyn ’toimii kuin ihminen’ -määri-

telmää. Muita tekoälyn määritelmiä ovat 'ajattelee kuin ihminen', 'ajattelee rationaalisesti' ja 'toimii rationaalisesti'.

Turingin testin käyttökelpoisuutta koneälyn määritelmänä on arvosteltu voimakkaasti. Kone, joka läpäisisi Turingin testin, kyllä jäljittelisi ihmisen käyttäytymistä keskustelussa, mutta tällainen jäljittely on vielä kaukana todellisesta älykkyydestä. Kone vain noudattaisi jotain nokkelasti laadittua säännöstöä. Toisaalta voidaan myös kysyä, mistä tiedämme, etteivät ihmisetkin vain noudata joitain nokkelasti laadittua säännöstöä. Tällaiseen argumentointiin perustuva tunnettu vastaesimerkki on John Searlen kiinalainen huone.

Lisäksi on huomattava, että Turingin testi ei edellytä tietoisuutta (consciousness) tai tavoitteellisuutta (intentionality). Esimerkiksi tieteen popularisoija Larry Gonick ei pidä Turingin testiä älykkyyden määritelmänä, koska hänestä jäljittelyllä ei ole mitään tekemistä todellisen ajattelun kanssa.

Turingin testin kritiikin kärki on kohdistunut testin tavoiteasetteluun: järjestelmän tulisi käyttäytyä kuin ihminen. Useimpiin tietojenkäsittelyn sovelluksiin tavoite on virheellinen. Tunnetusti ihminen tekee virheitä, joita muut ihmiset eivät välttämättä huomaa. Siten Turingin testin läpäisevä palkanlaskujärjestelmä voisi tehdä virheitä.

Näin Turingin testin läpäisevä järjestelmä olisi "oikea vastaus väärään kysymykseen". Useasti tällainen vastus on hyödytön. Toisaalta Turingin testin läpäisevän järjestelmän olisi ratkaistava ainakin neljä tietojenkäsittelyn keskeistä ongelmaa:

Luonnollisen kielen käsittely: Järjestelmän on ymmärrettävä sille esitetyt kysymykset ja muotoiltava vastaukset. Luonnollisen kielen käsittelyssä erityisenä ongelmana on, että usealla sanalla on useita asiayhteydestä riippuvia merkityksiä.

Tietämyksen esittämisen: Järjestelmän on tallennettava tietämänsä ja kuulemansa. Tietämyksen laajentuessa tallennuksen ja etsinnän tehokkuus nousevat keskeiseen asemaan.

Automaattinen päättely: Järjestelmän on pystyttävä tallennetun informaation perusteella tekemään päätelmiä, joiden perusteella vastaus voidaan muotoilla. Koska loogisen päättelyn jotkut ongelmat ovat todistetusti ratkeamattomia, niin järjestelmän on myös "tiedettävä", mitä se ei pysty päättelemään.

Koneoppiminen: Järjestelmän on sopeuduttava uusin tilanteisiin. Koneoppimisessa järjestelmän maailmankuva (eli malli järjestelmää kiinnostavista asioista) muuttuu järjestelmän saaman datan perusteella

4.5 Muistaminen

Muistamisen (*recollection*) peruskysymykset liittyvät informaation tallentamiseen ja hakemiseen. Muistamisen keskeisiä käsitteitä ovat tallennusvälineiden hierarkiat (hierarchies of storage), viittausten paikallisuus (locality of reference), välimuistit (caching), osoiteavaruudet ja niiden väliset kuvaukset (address space and mappings), nimentä (naming), yhteiskäyttö (sharing), etsintä (searching), haku nimen perusteella (retrieval by name) ja haku sisällön perusteella (retrieval by content).

Välimuisti

Eräs nykyisten tietojenkäsittelylaitteiden suurimmista suorituskyvyn haasteista on saada haluttu data riittävän nopeasti käsiteltäväksi. Prosessorit ovat huomattavasti nopeampia kuin keskusmuistit, keskusmuistit huomattavasti nopeampia kuin levymuistit. Web-sivujen nouto kaukaiselta palvelimelta vie aikaa. Erilaisia välimuisteja (*cache*) käytetään nopeuttamaan tulosten valmistumista. Tieto tallennetaan tilapäisesti välimuistiin lähemmäksi käsittelypaikkaa.

Välimuistit ovat osoittautuneet erittäin tehokkaiksi useilla tietojenkäsittelyn alueilla, koska datan käyttö on tyypillisesti paikallista. Paikallisuus (*locality*) esiintyy tietojenkäsittelyssä useissa eri muodoissa. Tavanomaisimmin käsite tarkoittaa, että samoja tietoalkioita käsitellään useita kertoja ajallisesti lähekkäin tai että lähellä toisiaan sijaitsevia tietoalkioita käsitellään ajallisesti lähekkäin.

Kun välimuistin käyttäjä (prosessori, Web-selain, käyttöjärjestelmä) haluaa käyttää varsinaisessa muistissa olevaa tietoa, niin se ensimmäiseksi tarkistaa, löytyykö kyseinen tieto välimuistista. Jos tarvittavan tiedon kopio on välimuistissa, niin käytetään kopiota. Jos välimuistissa ei ole kopiota, niin tietoalkio noudetaan varsinaisesta muistista ja useimmiten tallennetaan myös välimuistiin.

Välimuisti on useimmiten huomattavasti pienempi kuin varsinainen muisti. Tällöin välimuistista joudutaan poistamaan alkioita uusien tietoalkioiden tieltä. Heuristiikkaa, jolla poistettava alkio valitaan, kutsutaan poistopolitiikaksi (*replacement policy*). Poistopolitiikka perustuu aikaisemmin mainittuun paikallisuuteen. Ohjelmallisesti toteutetuissa välimuisteissa yleisin poistopolitiikka on LRU (*least recently used*) eli poistetaan alkio, joka on ollut pisimpään käyttämättä.

Kun välimuistissa olevaa tietoa muutetaan, niin jossakin vaiheessa muuttunut tieto on kirjoitettava myös varsinaiseen muistiin. Tämän kirjoituksen ajoitusta ohjaa kirjoituspolitiikka (*write policy*). Läpikirjoittavassa (*write-through*) välimuistissa jokainen välimuistiin kirjoitus aiheuttaa välittömän kirjoituksen varsinaiseen muistiin. Vaihtoehtoisesti kirjoitusta voidaan viivästää (*write-back cache*). Tällöin välimuisti pitää kirjaa niistä tietoalkioista, joita on muutettu. Nämä kirjoitetaan varsinaiseen muistiin, kun alkio poistetaan välimuistista. Läpikirjoittaminen on tavallista välimuistin laitteistototeutuksessa (prosessorien välimuistit), viivästetty kirjoittaminen ohjelmallisesti toteutetuissa välimuisteissa (tiedostovälimuistit). Läpikirjoittavaa välimuistia käytetään myös tilanteissa, joissa välimuistin ja varsinaisen muistin välinen tietoliikenneyhteys on epäluotettava.

Välimuistissa oleva tietoalkio voi vanhentua (*stale*) myös siten, että varsinaisen muistin sisältö muuttuu. Tällöin välimuistin toteutuksessa tarvitaan eheysprotokolla (*coherency protocol*), jonka avulla välimuistien sisällöt pidetään yhtenäisinä. Välimuistin ajantasaisuus on Web-välimuistien ja hajautettujen tiedostovälimuistien erityinen haaste.

LUKU 5: Suunnittelu

Tietojenkäsittelyn ammattilaiset tukeutuvat suunnittelussa periaatteisiin, jotka soveltavat tietojenkäsittelyn mekaniikkaa. Vuosien saatossa hyväksi havaittuja suunnittelun käytäntöjä ovat mm.:

- abstrahointi (abstraction) – epäolennaisten yksityiskohtien häivyttäminen,
- informaation piilottaminen (information hiding) – rakenteen osan (moduulin) sisäisten tietojen ja tietorakenteiden piilottaminen muilta osilta,
- moduulit (modules) – kokonaisuuden jakaminen osiin siten, että osien väliset vuorovaikutukset ja rajapinnat ovat hyvin määriteltyjä,
- erikseen kääntäminen (separate compilation) – ohjelman osien kääntäminen erikseen ja linkittäminen myöhemmin suorituskelpoiseksi kokonaisuudeksi,
- pakkaus (package) – jakelu- ja asennusyksikkö, johon on koottu ohjelmisto(je)n osat ja dokumentaatio(t),
- versionhallinta (version control) – menetelmät, joilla hallitaan kehitystyötä ja sen aikana syntyviä versioita,
- hajota ja hallitse (divide-and-conquer) – periaate, jonka mukaisesti suuret kokonaisuudet jaetaan paremmin ja helpommin hallittaviksi osakokonaisuuksiksi,
- toimintatasot (functional levels) – toiminnallisuuden ryhmitteleminen eritasoisiksi toimenpiteiksi,
- kerrosajattelu (layering) – toimintakokonaisuuden jakaminen kerroksiin siten, että kukin kerros tarjoaa joitakin (muutamia) palveluja ylemmille kerroksille käyttäen hyväksi alempien kerrosten tarjoamia palveluja,
- ongelmien eriyttäminen (separation of concerns) – periaate, jonka mukaan ratkaisussa keskitytään yhteen, mahdollisimman täsmällisesti määriteltyyn tehtävään,
- uudelleenkäyttö (reuse) – olemassa olevien moduulien, määritysten ja suunnitelmien käyttäminen sen sijaan, että tehtäisiin uusia,
- kapselointi (encapsulation) – elementtien sisällyttäminen laajempaan ja abstraktimpaan kokonaisuuteen; käsite on hyvin läheistä sukua informaation piilottamiselle ja ongelmien eriyttämiselle; kapseloinnilla yleensä piilotetaan toteutuksen sisäiset yksityiskohdat ja tarjotaan rajapinta kapselin sisältämien tietojen manipulointiin,
- rajapinta (interface) – rajapinnan välityksellä ohjelmisto- tai laitteistokomponentti tarjoaa toiminnallisuuksiaan muiden käyttöön,
- virtuaalikone (virtual machine) – ohjelmisto, joka toteuttaa määritellyn abstraktin koneen toiminnallisuuden.

Edellä luetelluilla suunnittelussa vakiintuneilla toimintatavoilla pyritään saavuttamaan:

yksinkertaisuus (simplicity): erilaiset abstraktiot ja rakenteet, joilla pyritään hallitse-

maan sovellusten luontaista monimutkaisuutta,

suorituskyky (performance): suoritustehon ja vasteajan ennustaminen, pullonkaulojen paikallistaminen, kapasiteetin suunnittelu,

luotettavuus (reliability): toisteisuus, toipuminen, varmistaminen, eheys, luottamus,

kehitettävyys (evolvability): varautuminen toiminnallisuuden ja käytön laajuuden muutoksiin, ja

tietoturva (security): pääsynvalvonta, salassapito, yksityisyys, tunnistus, eheys ja turvallisuus.

Näiden tavoitteiden lisäksi suunnitteluun vaikuttavat useat rajoitteet, kuten kustannukset, aikataulut, yhteensopivuus (compatibility) ja käytettävyys (usability).

5.1 Yksinkertaisuus

Yksinkertaisuus on ollut kautta aikojen keskeinen tavoite tietojenkäsittelyjärjestelmien suunnittelussa. IBM:n ohjelmistosuunnittelijoiden huoneentauluna kerrotaan olleen ”KISS!”, tarkoittaen joko

Keep It Simple and Small!

tai

Keep It Simple, Stupid!

Kumpikin korostaa sitä tosiasiaa, että järjestelmien monimutkaistuminen tuo mukanaan ongelmia. Samaa asennetta heijastaa myös Dijkstran [1972] tietojenkäsittelytieteen määritelmä: Tietojenkäsittelytiede on monimutkaisuuden hallitsemisen tutkimista.

Aikoinaan tätä Dijkstran tietojenkäsittelytieteen määritelmää vastaan hyökättiin voimakkaasti väittämällä, että useat muutkin tieteenalat yrittävät hallita monimutkaisuutta. Tämän määritelmän sanoma on vuosikymmenten mittaan osoittautunut kestäväksi, sillä nykyisin tietojenkäsittelyjärjestelmät ovat monimutkaisimpien ihmisten tekemiä konstruktioiden joukossa.

Yksinkertaisuus ei synny itsestään. Se vaatii runsaasti ajattelua ja suunnittelua. Hyviä ohjelmia ja muita tuotteita ei vain kirjoiteta, vaan ne on suunniteltu huolellisesti. Parnas [1996] antaa seuraavat perussäännöt, joilla voi pyrkiä kohti yksinkertaisia ratkaisuja:

- Suunnittele ennen toteutusta.
- Dokumentoi suunnittelu.
- Katselmoi ja analysoi dokumentoitu suunnittelu.
- Katselmoi, että toteutus vastaa suunnittelua.

5.2 Suorituskyky

Suorituskykyä (*performance*) kuvaavia keskeisiä suureita ovat

suoritusteho, kuinka paljon hyödyllistä työtä tehdään aikayksikössä;

vasteaika, kauanko tietyn tehtävän suorittamiseen kuluu;
käyttöaste, miten suuren osan ajasta laite on aktiivisena.

Suorituskykyä tutkitaan mittaamalla järjestelmän toimintaa simuloimalla järjestelmän keskeisten osien toimintaa sopivalla tarkkuudella sekä laatimalla järjestelmästä matemaattisia malleja, joista keskeiset suorituskykysuureet lasketaan.

Suorityskykyanalyysin yksi osa-alue on ohjelmistojen suorituskyvyn suunnittelu, joka on myös osa ohjelmistotekniikkaa. Tavoitteena on jo ohjelmiston suunnitteluvaiheessa arvioida tulevan järjestelmän tehokkuutta ja tarvitsemia laitteistoresursseja.

Useimmat kiinnostavat ohjelmistojärjestelmät ovat monimutkaisia suunnitella ja toteuttaa. Ohjelmistojen vaatimukset (*requirements*) jaetaan toiminnallisiin ja ei-toiminnallisiin vaatimuksiin. Niin kutsuttuihin ei-toiminnallisiin vaatimuksiin kuuluvat sellaiset ominaisuudet kuin tietoturva (*security*), saatavuus (*availability*), luotettavuus (*reliability*) ja suorituskyky. Vasteajan ja suoritustehon lisäksi suoritettujen (tai suorittamatta jääneiden) pyyntöjen osuus on keskeinen suorituskykysuure.

Ohjelmistotekniikan useimmat menestystarinat ovat liittyneet menetelmiin ja työkaluihin, joilla hallitaan ohjelmistojen laatimisen monimutkaisuutta. Nämä kattavat suunnittelun, vaatimusten, testitapausten, konfiguraatioiden, versioiden ja kehittämisen hallinnoinnin.

5.3 Luotettavuus

Luotettavuuden (reliability) osa-alueita ovat toisteisuus, toipuminen, varmistaminen, eheys ja luottamus.

Tietojenkäsittelytieteilijät ovat yrittäneet käyttää erilaisia formaaleja menetelmiä, joiden avulla voitaisiin päätellä ohjelmistojen käyttäytymistä. Nämä menetelmät perustuvat kuitenkin usein ohjelmiston abstraktiin malliin, joka ei täydellisesti kuvaa ajoaikaisen järjestelmän toimintaa. Siksi formaaleilla menetelmillä ei juuri koskaan pystytä kuvaamaan toteutuksen todellista käyttäytymistä riittävän tarkasti, koska koko järjestelmän kaikilla fyysisesti mahdollisilla tiloilla ei ole vastinetta abstraktissa mallissa.

Järjestelmän toipumisesta huolehtiva ohjelmakoodi käsittelee poikkeustilanteita ja siksi sen olisi toimittava virheettömästi. Valitettavasti poikkeukselliset tilanteet ovat vaikeasti käsiteltäviä. Ne esiintyvät harvoin, eikä niitä ole helppo saada aikaan kehitysvaiheessa, jotta toipumista päästäisiin testaamaan, joten toipumisesta huolehtiva ohjelmakoodi on usein epäluotettavaa.

5.4 Kehitettävyys

Ohjelmistotekniikan kriisistä on puhuttu 1960-luvun lopulta lähtien. Parnas [1994] pitää perusongelmana ohjelmistojen ikääntymistä. Kun organisaatio lisää tietotekniikan hyväksikäyttöään, eri tietojenkäsittelyjärjestelmien integrointi nousee keskeiseksi vaatimukseksi. Alun perin irralliset järjestelmät on saatava keskustelemaan keskenään. Siitä huolimatta niiden on edelleen pystyttävä toimimaan organisaation aiempi-

en prosessien ja käytäntöjen kanssa. Alkuaan hyvinkin löyhä kytkös muuttuu ajan myötä yhä kiinteämmäksi. Määriteltäessä uusia sovelluksia, etsittäessä yhdistämisiä ja integroimalla jo toiminnassa olevia järjestelmiä, kehittäjien on rajattava sovellusten käyttöaluetta, laajuutta ja yksityiskohtia. Lopputuloksena on rajoituksia, jotka saattavat muodostua pullonkauloiksi ja ärtymyksen aiheuttajiksi. Käyttäjien turhautuminen, kasvava vaatimustaso ja toimintaympäristöjen lisääntyvä riippuvuus toisistaan synnyttää jatkuvia muutospaineita ja loputtomia yhtenäistämisvaatimuksia.

Järjestelmiin kohdistuu myös ulkoisia käyttöalueista johtuvia paineita. Esimerkiksi vuosituhannen vaihtumisen (Y2K-ongelma) aiheuttamat ongelmat olisi ollut vältettävissä, mikäli 1900-luvulla tehdyt järjestelmät olisi alun perin suunniteltu toimiviksi myös 2000-luvulla. Jotkut ulkoiset paineet johtuvat poliittisista päätöksistä, jotka eivät ota huomioon niiden vaikutuksia tietojenkäsittelyyn. Tällaisia ovat olleet esimerkiksi euroon siirtyminen ja työeläkejärjestelmän maksuperusteiden muutokset. Myös teknologiset muutokset ovat vaatineet ohjelmistojen muuttamista. Esimerkiksi puhelinnumerot ovat useaan kertaan muuttuneet laajentuneen puhelinkannan vuoksi.

Riippumatta siitä, mikä on ollut ongelman alkuperä, niin tulos on ollut sama: turhautuneisuutta, kustannuksia, pienempiä tai suurempia onnettomuuksia, jotka ovat vaatineet välittömiä muutoksia ohjelmistoihin. Lisäksi käyttäjäkunnan vaatimustaso kasvaa jatkuvasti. Ohjelmistojen käyttöympäristöt, teknologia, lainsäädäntö ja talous muuttuvat ja toisaalta suunnittelun ja toteutuksen tarjoamat mahdollisuudet ovat rajattomat. Nämä tekijät yhdessä johtavat loputtomaan ohjelmistojen muutoksiin, parannuksiin ja kehitykseen sekä jatkuvaan uusien ohjelmistoversioiden tarpeeseen. Rakenteeltaan, sisällöltään ja toiminnallisuudeltaan ohjelmistot ovat monimutkaisimpia ihmisen luomia järjestelmiä. Ohjelmisto itsessään on malli sovelluksesta, osallistujista (ihmiset, organisaatiot, laitteet, koneet), käyttöalueesta ja kyseisen alueen toiminnoista.

Lehman [1998] luettelee toimintatapoja, joilla ohjelmistojen kehitettävyyttä voidaan parantaa (alkuperäinen lista on pidempi):

- Kun tietokonejärjestelmä otetaan käyttöön tai sen käyttöä laajennetaan, vaikutuksia on tarkasteltava organisaation koko toiminnan kannalta. Ei riitä, että tarkastellaan vain sovellutuksen tehokkuutta tai taloudellisia hyötyjä.
- Sovelluksen ja käyttöalueen rajat on tunnistettava heti alussa. Nämä päätökset on kirjattava siten, että myös riippuvuudet ja keskinäiset suhteet tulevat selvästi esille.
- Rajojen määrityksiä on katselmoitava säännöllisesti sekä ohjelmistokehityksen aikana että sen jälkeen, jotta niiden paikkansapitävyys voidaan varmistaa olosuhteiden muuttuessa.
- Määrittelyn, suunnittelun ja kehitystyön kaikissa vaiheissa on pyrittävä tunnistamaan ja kirjaamaan sekä eksplisiittiset että implisiittiset oletukset, jotka tehdään suunnittelun ja toteutuksen valinnoissa. Nämä oletukset eivät liity vain teknisiin ja hallinnollisiin seikkoihin, vaan myös käyttäjien reaktioihin, ohjelmien vaikutuksiin

käyttöalueella sekä taloudellisiin ja sosiaalisiin tekijöihin.
- Ohjelmistoon tehtävien muutosten vaikutukset rajoihin ja oletuksiin on käytävä läpi ennen muutosten tekemistä, jotta yhteensopimattomuudet tai muut ei-toivotut sivuvaikutukset voidaan välttää.
- Esitettyjen muutosten kaikki vaikutukset, niin globaalit kuin paikalliset, on analysoitava. Tämä edellyttää muun muassa sitä, että kaikki muutoksiin liittyvät takaisinkytkentäketjut tunnetaan ja ymmärretään.
- Ohjelmistoarkkitehtuurien on minimoitava osien (moduulit, komponentit, alijärjestelmät) väliset riippuvuudet.
- Aina kun mahdollista, järjestelmän jokaisen osan tulisi palvella suoraan sen käyttäjää eikä tuottaa tietoa muiden osien käsiteltäväksi. Tavoitteena tulisi olla ohjelmiston rakentaminen osista, jotka ovat toiminnallisesti riippumattomia toisistaan.

5.5 Tietoturva

Tietoturva-käsitteen alle Denning on ryhmitellyt pääsynvalvonnan, salassapidon, yksityisyyden, tunnistamisen, eheyden ja turvallisuuden.

Onnettomuudet ovat harvoin yksinkertaisia. Useimmiten onnettomuuden takana on monimutkainen vyyhti toisiinsa liittyviä tapahtumia, jotka johtuvat teknisistä, inhimillisistä ja työyhteisöön liittyvistä tekijöistä. Tyypillinen vakavien onnettomuuksien aiheuttaja on uskomus, että asiat saadaan kuntoon yhden löydetyn virheen korjaamisella. Monimutkaisissa järjestelmissä on lähes aina useita ohjelmistovirheitä.

Onnettomuuksien katsotaan usein joutuvan yhdestä syystä, esimerkiksi inhimillisestä erehdyksestä. Toisaalta lähes kaikkien syiden voidaan sanoa olevan inhimillisiä erehdyksiä. Jopa laitteiston kulumisesta johtuvien vikojen voidaan väittää olevan inhimillisiä erehdyksiä, koska laitteistoon ei oltu suunniteltu riittävää redundanssia tai koska järjestelmän ylläpitäjät eivät olleet kunnollisesti huoltaneet laitteistoa tai vaihtaneet kuluneita osia ajoissa. Siksi onnettomuuden syynä inhimillinen erehdys ei ole kovin hyödyllinen ellei sitä riittävästi tarkenneta.

Hyvin yleinen virhe on luottaa liikaa ohjelmistoon, sillä täydellisiä ohjelmistoja ei voi toteuttaa. Vaikka ohjelmistossa ei esiinny laitteistojen tavoin satunnaisia kulumisvirheitä, ohjelmiston suunnitteluvirheiden löytäminen ja estäminen on huomattavasti hankalampaa kuin kulumisvirheiden. Lisäksi laitteiston virhekäyttäytymisen muotoja on yleensä vain muutama. Siksi niitä vastaan suojautuminen on useimmiten olennaisesti helpompaa kuin ohjelmistovirheitä vastaan suojautuminen.

Ehdotonta turvallisuutta vaativissa ohjelmistoissa turvallisuus on rakennettava ohjelmistoon. Lisäksi tällaisten ohjelmistojen turvallisuus on analysoitava. Turvallisuus on pystyttävä takaamaan järjestelmätasolla mahdollisista ohjelmistovirheistä huolimatta. Ohjelmistovirheitä vastaan voidaan suojautua myös itse ohjelmistossa.

Usein naivisti oletetaan, että ohjelman uudelleen käyttäminen tai kaupallisen val-

misohjelman käyttäminen lisää turvallisuutta, koska ohjelma on jo pitkään ollut käytössä. Ohjelmamoduulin uudelleen käyttäminen ei takaa uuden järjestelmän turvallisuutta. Uudelleen käyttäminen saattaa jopa kasvattaa suunnitelman monimutkaisuutta, mistä usein seuraa turvallisuusriskien lisääntyminen. Viime kädessä turvallisuus on järjestelmän, jossa ohjelmistoa käytetään, ominaisuus, ei varsinaisesti ohjelmiston ominaisuus.

LUKU 6: Tietojenkäsittelijän ammattietiikka

Ammattietiikasta puhutaan yleensä lääkärien tai tuomareiden kohdalla. Samantapaisia vaatimuksia voidaan kuitenkin asettaa myös monissa muissa ammateissa toimiville. Tietojenkäsittelijän ammattietiikka tarkoittaa eettisiä periaatteita, sääntöjä, normeja, arvoja ja hyveitä, joita tietojenkäsittelijän tulisi noudattaa harjoittaessaan omaa ammattiaan. Tietojenkäsittelijän toiminta muodostuu tehtävistä, jotka tekevät siitä erityisen, muista ammateista erottuvan ammatin, ja tietojenkäsittelijän ammattietiikka koskee näitä erityisiä tehtäviä. Kysymys on siitä, millaisia eettisiä vaatimuksia tietojenkäsittelijälle tulee asettaa, kun hän suorittaa ammattinsa edellyttämiä erityistehtäviä.

Tietojenkäsittelijän ammattietiikkaa pohdittaessa kannattaa aluksi jakaa tehtävä kahteen osaan: tietojenkäsittelytieteen alalla toimivan tutkijan ammattietiikkaan ja toisaalta käytännön tehtävissä toimivan it-ammattilaisen, esimerkiksi ohjelmistosuunnittelijan tai projektipäällikön, ammattietiikkaan.

Tutkijan ammattieettiset normit esitetään usein niin yleisellä tasolla, että normit voidaan asettaa yhteisesti kaikkien alojen tutkijoille, vaikka todellisuudessa eri aloilla toimivien tutkijoiden tilanne voi poiketa huomattavasti toisistaan.

Tutkijan ammattietiikan tarkastelussa tulisi selvittää yleensä ainakin viidenlaisia kysymyksiä: mitä eettisiä vaatimuksia tulee asettaa

- ammattitaidon hankkimiselle,
- informaation tuottamiselle,
- informaation välittämiselle,
- informaation käyttämiselle sekä
- kollegiaaliselle toiminnalle ja yleisemmin lojaalisuudelle [Pietarinen, 1999].

Tutkijan ammattitaidon vähimmäisvaatimuksena voidaan pitää sitä, että hän pystyy seuraamaan oman alansa kehitystä ja välittämään sen tuloksia muille. Tutkijalta odotetaan myös uuden informaation tuottamista. Se ei ole mahdollista, ellei hän tunne aikaisempaa tutkimusta eikä tutkimusmenetelmiä. Ammattitaidon hankkimisessa ja ylläpitämisessä tutkija joutuu suorittamaan valintoja, joihin liittyy eettisiä kysymyksiä, sillä jo oikeasta ajankäytöstä päättäminen on ongelmallista. Samoin tutkimuskohteiden valinta ja tulosten hyödyntäminen sisältävät lukemattomia eettisiä ongelmia. Pietarinen [1999] on listannut seuraavat tutkijan ammattietiikkaan liittyvät vaatimukset:

1. Älyllisen kiinnostuksen vaatimus: tutkijan on oltava aidosti kiinnostunut uuden informaation hankkimisesta.

2. Tunnollisuuden vaatimus: tutkijan on paneuduttava tunnollisesti alaansa, jotta hänen hankkimansa ja välittämänsä informaatio olisi niin luotettavaa kuin mahdollista.
3. Rehellisyyden vaatimus: tutkija ei saa syyllistyä vilpin harjoittamiseen.
4. Vaaran eliminoiminen: sellaisesta tutkimuksesta tulee pidättäytyä, joka voi tuottaa kohtuutonta vahinkoa.
5. Ihmisarvon kunnioittaminen: tutkimuksen tekeminen ei saa loukata ihmisarvoa yleisesti eikä kenenkään ihmisen tai ihmisryhmän moraalista arvoa.
6. Sosiaalisen vastuun vaatimus: tutkijan tulee osaltaan vaikuttaa siihen, että tieteellistä informaatiota käytetään eettisten vaatimusten mukaisesti.
7. Ammatinharjoituksen edistäminen: tutkijan tulee toimia tavalla, joka edistää tutkimuksen tekemisen mahdollisuuksia.
8. Kollegiaalinen arvostus: tutkijoiden tulee suhtautua toisiinsa arvostavasti, ei vähättelevästi.

Opiskelijan ”ammatin” voi monelta osin rinnastaa tutkijaan, ja opiskelijan ”ammattieettiset” vaatimukset ovatkin samantapaiset kuin edellä kuvatut tutkijaan kohdistuvat vaatimukset. Opiskelijoille on lisäksi laadittu omia ohjeita, kuten esimerkiksi ”Hyvät opiskelukäytännöt Tampereen yliopistossa”.

ACM:n ja IEEE:n ohjelmistotekniikan eettiset ohjeet

ACM ja IEEE:n Computer Society ovat jo pitkään toimineet ohjelmistotekniikan ammattimaisuuden edistämiseksi. Näiden järjestöjen ohjelmistotekniikan eettiset ohjeet ja ammatilliset käytännöt (*Software Engineering Code of Ethics and Professional Practise*) [Gotterbarn et al., 1999] on yksi ammattimaisuuden kulmakivi. Eettisissä ohjeissa kuvataan ohjelmistotekniikan opetuksen ja harjoittamisen eettiset ja ammatilliset velvoitteet. Ohjeet heijastavat niitä odotuksia, jotka yhteiskunta, kollegat ja suuri yleisö odottavat ohjelmistoammattilaisten täyttävän.

Ohjelmistojärjestelmien kehittäjinä ohjelmistoammattilaisilla on mahdollisuudet aikaansaada hyvää tai aiheuttaa harmia, edistää muiden mahdollisuuksia aikaansaada hyvää tai aiheuttaa harmia, tai vaikuttaa muiden mahdollisuuksiin aikaansaada hyvää tai aiheuttaa harmia. Jotta ohjelmistoammattilaiset pystyisivät mahdollisimman hyvin vakuuttumaan, että heidän työpanoksensa käytetään hyvän aikaansaamiseksi, niin heidän on sitouduttava edistämään ohjelmistojen laatimisen muodostumista hyödylliseksi ja arvostetuksi ammatiksi.

ACM:n ja IEEE:n eettiset ohjeet sisältävät kahdeksan periaatetta, jotka liittyvät ohjelmistoammattilaisen käyttäytymiseen ja päätöksentekoon ammatissaan. Eettiset ohjeet on luettava kokonaisuutena, eivätkä ne kata kaikkia mahdollisia tilanteita, joissa ohjelmistoammattilainen joutuu arvioimaan oman toimintansa oikeutusta. Siksi eettiset ohjeet eivät ole algoritmi, joka generoisi eettisesti hyväksyttävät päätökset ja valin-

nat.

Eettisiä jännitteitä on parempi tarkastella kokonaisvaltaisesti perusperiaatteiden kautta kuin luottaa yksityiskohtaisiin säädöksiin. ACM:n ja IEEE:n eettisten ohjeiden kahdeksan periaatteen tarkoituksena on saada ohjelmistoammattilaiset

- ajattelemaan oman työnsä laaja-alaisia vaikutuksia,
- tarkastelemaan, kohtelevatko he ja heidän kolleegansa muita ihmisiä riittävällä kunnioituksella,
- arvioimaan, miten suuri yleisö, jos se olisi tarpeeksi hyvin informoitu, suhtautuisi heidän päätöksiinsä,
- analysoimaan, miten heidän päätöksensä vaikuttavat vähempiosaisiin ja
- arvioimaan omien toimenpiteidensä hyväksyttävyyttä.

Kaikissa näissä tilanteissa yleinen terveys, turvallisuus ja hyvinvointi ovat ensisijaisia.

Alla on periaatteiden lyhyet kuvaukset. Kaikkien ohjelmistoammattilaisten on syytä tutustua myös periaatteiden yksityiskohtaisiin kuvauksiin, jotka erittelevät kutakin periaatetta monipuolisesti. Alla käydään läpi vain yleisen edun periaate.

Yleinen etu. Ohjelmistoammattilaiset toimivat aina yleisen edun mukaisesti.
Asiakas ja työnantaja. Ohjelmistoammattilaiset toimivat tavalla, joka vastaa asiakkaan ja työnantajan etuja, mutta on sopusoinnussa yleisen edun kanssa.
Tuote. Ohjelmistoammattilainen varmistaa, että hänen tuotteensa ja niihin liittyvät muutokset täyttävät parhaalla mahdollisella tavalla ammattimaisen toiminnan vaatimukset.
Harkinta. Ohjelmistoammattilaiset ovat johdonmukaisia ja säilyttävät riippumattomuutensa ammatillisissa mielipiteissään.
Johto. Ohjelmistotekniikan johtajat ja esimiehet sitoutuvat ohjelmistojen kehityksen ja ylläpidon eettisesti kestävään johtamiseen ja edistävät sitä.
Ammatti. Ohjelmistoammattilaiset edistävät ammattinsa rehellisyyttä ja mainetta yleisen edun mukaisesti.
Työtoverit. Ohjelmistoammattilaiset ovat työtovereilleen rehtejä ja kannustavia.
Oma toiminta. Ohjelmistoammattilaiset ovat sitoutuneet oman ammattitaitonsa elinikäiseen kehittämiseen ja edistävät eettisesti kestäviä ammatillisia käytäntöjä.

Yleisen edun periaate

Yleisen edun periaatetta ACM:n ja IEEE:n eettiset ohjeet tarkentavat seuraavilla kahdeksalla kohdalla:

1. Ohjelmistoammattilaiset ottavat täyden vastuun omasta työstään.
2. Ohjelmistoammattilaiset sovittavat yhteen oman, työnantajan, asiakkaan ja käyttäjien edut yleisen edun kanssa.
3. Ohjelmistoammattilaiset hyväksyvät ohjelmiston luovutuksen vain, jos heillä on hyvin perusteltu uskomus, että ohjelmisto on turvallinen, täyttää määritykset ja

läpäisee asianmukaiset testit, ei vähennä elämän laatua tai yksityisyyttä eikä vahingoita ympäristöä. Työn perimmäisten vaikutusten on oltava yleisen edun mukaisia.
4. Ohjelmistoammattilaiset paljastavat asianmukaisille henkilöille tai viranomaisille minkä tahansa todellisen tai mahdollisen käyttäjiin, suureen yleisöön tai ympäristöön kohdistuvan uhan, jonka he kohtuudella uskovat liittyvän ohjelmistoon tai sen dokumentaatioon.
5. Ohjelmistoammattilaiset osallistuvat aktiivisesti ohjelmistojen, niiden installoinnin, ylläpidon tai dokumentaation aiheuttamien vakavien julkisten huolenaiheiden selvittelyyn.
6. Ohjelmistoammattilaiset ovat rehtejä ja välttävät harhakuvia kaikissa, erityisesti julkisissa, lausunnoissaan, jotka koskevat ohjelmistoja sekä niihin liittyviä dokumentteja, menetelmiä tai työvälineitä.
7. Ohjelmistoammattilaiset ottavat huomioon fyysisestä rajoittuneisuudesta, käytettävistä voimavaroista, taloudellisesta eriarvoisuudesta ja muista syistä johtuvat tekijät, jotka voivat vähentää ohjelmistojen saatavuuden hyötyjä.
8. Ohjelmistoammattilaiset käyttävät ammatillisia taitojaan vapaaehtoistyössä ja osallistuvat alan julkiseen kouluttamiseen.

Tietotekniikan liiton tietotekniikan etiikan ohjeisto

Tietotekniikan liiton etiikan työryhmä julkisti 2002 tietotekniikan ammattilaisen eettiset ohjeet [Tietotekniikan liitto, 2002] tukemaan tietotekniikan ammattilaisia heidän työssään kohtaamiensa eettisten ongelmien ratkaisemisessa. Tavoitteena on tuoda esille eettisiä toimintatapoja ja auttaa käsittelemään moraalisia ongelmia.

Ohjeiston tarkoituksena on syventää tietotekniikan ammattilaisuuden eettistä ulottuvuutta. Aivan samoin kuin ACM:n ja IEEE:n eettiset ohjeet TTL:n ohjeisto ei yritä olla täydellinen. Koska eettisiin ongelmiin ei voida ennalta antaa täydellisiä ohjeita, TTL:n ohjeiston kohtia ei tule lukea ehdottomina totuuksina vaan suunnannäyttäjinä. Vastuu päätöksistä on jokaisella itsellään.

Alla oleva teksti on suora kopio TTL:n ohjeistosta.

Valta ja vastuu. Tietotekniikan ammattilainen ei saa käyttää asemaansa väärin. Hänen on kannettava vastuunsa, joka näkyy tekoina ja toimina. Tieto on valtaa ja tiedon käyttäminen vaatii viisautta kuten muukin vallankäyttö.

Tieto ja kokemus. Ammattilaisen on tunnettava rajansa: tiedettävä mitä osaa ja myös mitä ei osaa. Kehittyvällä alalla ammattilaisen on ylläpidettävä osaamistaan. Hänen on tunnettava työtään koskeva, esimerkiksi tietosuojaan liittyvä, lainsäädäntö. Ammattilainen ei panttaa tietoa, vaan pyrkii lisäämään omaa ja muiden osaamista ja jakaa omat kokemuksensa muulle yhteisölle. Ammattilainen suojaa kuitenkin asiakkaan omat asiat ja muut suojaamista vaativat tiedot. Saadessaan kritiikkiä työstään, aiheellisesti tai aiheetta, ammattilainen osaa ottaa sen vastaan ja ottaa tapahtuneesta

oppia.

Asenne. Ammattilainen ei toimi vain itseään vaan myös muita varten. Hän ottaa huomioon toimintansa kohteiden näkökannan. Hän ei anna valtaa ahneudelle ja piittaamattomuudelle. Hän ymmärtää myös, että hänen työllään on merkitystä vain muiden ihmisten kautta.

Viestintä. Ammattilainen ymmärtää viestinnän merkityksen. Hän kommunikoi asiakkaan kanssa, dokumentoi tekemisensä ja tiedottaa toimistaan kaikille asianomaisille. Ammattilaisen on pyrittävä viestimään selväkielisesti ja määrittelemään tarvittaessa käyttämänsä käsitteet. Viestinnän tavoitteena on yhteisen näkemyksen ja ymmärryksen luominen toiminnan pohjaksi. Asioidessaan asiakkaan kanssa ammattilaisen on kerrottava myös niistä seikoista, joita asiakas ei osaa itse kysyä. Ammattilaisen on kerrottava myös huonot uutiset.

Työn vaikutukset. Tietoteknisen työn tulokset saavat usein arvonsa vasta kun niitä hyödynnetään. Tietotekniikan ammattilaisen on pyrittävä ymmärtämään oman työnsä vaikutus usein pitkävaiheiselle ketjulle, jonka päässä on lopullinen hyödyntäjä. Ammattilaisen on myös otettava huomioon kuluttajan, laskun maksajan ja työnantajan vaatimukset. Toimiessaan ammattilainen pyrkii katsomaan työnsä laajempaa merkitystä koko sille yhteisölle, jolle työ tehdään, eikä rajoitu vain hänen kanssaan asioivien edustajien näkemyksiin.

Muut ihmiset. Tietotekniikan ammattilainen kunnioittaa toisten työtä ja ottaa huomioon muiden ihmisten oikeuden luomaansa ja tekemäänsä. Tietotekniikan ammattilaisen työ koskee sidosryhmien kautta yhteiskuntaa laajemmin. Ammattilaisen on käsitettävä työnsä seuraukset ja otettava huomioon esimerkiksi ihmisoikeudet, ympäristön suojelu, lainsäädäntö ja tekijänoikeudet.

Eettisyyden kasvu. Tietotekniikan ammattilaisen tulee edistää eettisesti kestävien toimintatapojen yleistymistä tietotekniikka-alalla. Toimiminen eettisesti on valinta, jonka jokainen yksilö voi tehdä tai olla tekemättä. Eettisyys ei ole mustavalkoinen asia, vaan ihminen voi kehittyä koko ajan ottamalla ympäristöään enemmän huomioon. Nämä ohjeet pyrkivät esittämään tietotekniikan ammattilaiselle eettisen toimintamallin, joka tukee sekä hänen itsensä että ympäristönsä eettistä kasvua.

Lähdeluettelo

[Denning, 2003] Denning, P. J., Great principles of computing. *Communications of the ACM* **46**, 11 (2003), 15–20.

[Denning, 2004] Denning, P. J., The social life of innovation. *Communications of the ACM* **47**, 4 (2004), 15–19.

[Denning et al., 1989] Denning, P. J. et al., Computing as a discipline. *Communications of the ACM* **32**, 1 (1989), 9–23.

[Dijkstra, 1972] Dijkstra, E. W., Notes on structured programming. In: O.-J. Dahl, E. W. Dijkstra and C. A. R. Hoare (eds.) *Structured Programming*. New York: Academic Press, 1972.

[Drucker, 1985] Drucker, P., *Innovation and Entrepreneurship*. Harper, 1985.

[Fairley and Willshire, 2003] Fairley, R. E. and Willshire, M. J. , Why the Vasa sank: 10 problems and some antidotes for software projects. *IEEE Software* **20**, 2 (2003), 18–25.

[Forsythe, 1968] Forsythe, G. E., Computer science and education. In: *Proceedings of IFIP Congress* 68, vol. 2, 1025–1039. North-Holland, 1968.

[Gotterbarn et al., 1999] Gotterbarn, D., Miller, K. and Rogerson, S., Computer Society and ACM approve Software Engineering Code of Ethics. *IEEE Computer* **32**, 10 (1999), 84–88.

[Haaparanta ja Niiniluoto, 1991] Haaparanta, L. ja Niiniluoto, I., Johdatus tieteelliseen ajatteluun. Helsingin yliopiston filosofian laitoksen julkaisuja 3/1986. Helsingin yliopisto, 1986 (6., korjattu painos 1991).

[IEEE, 1990] IEEE standard glossary of software engineering terminology. IEEE Standard 610.12-1990.

[Järvinen ja Järvinen, 2011] Järvinen, P., ja Järvinen, A., *Tutkimustyön metodeista*. Tampere: Opinpaja Oy, 2011.

[Knuth, 1974] Knuth, D. E. Computer science and its relation to mathematics. *American Mathematical Monthly* **81**, 4, (1974), 323–343

[Lehman, 1998] Lehman, M. M., Software's future: managing evolution. I*EEE Software* **15**, 1 (1998), 40-44.

[Newell et al., 1967] Newell, A., Perlis, A. J. and Simon, H., What is computer science. *Science* **157**, 3711 (1967), 1373–1374.

[Parnas, 1994] Parnas, D. L., Software aging. In: *Proceedings of the 16th International Conference on Software Engineering*, 279-287, 1994.

[Parnas, 1996] Parnas, D. L., Why software jewels are rare. *IEEE Computer* **29**, 2 (1996), 57–60.

[Pietarinen, 1999] Pietarinen, J., Tutkijan ammattietiikan perusta. Teoksessa Lötjönen, S. (toim.), Tutkijan ammattietiikka. Helsinki: Tieteellinen neuvottelukunta, 1999.

[Raatikainen, 2007] Raatikainen, K., Johdatus tietojenkäsittelytieteeseen – Tarinoita tietojenkäsittelytieteen osa-alueilta. Helsingin yliopiston tietojenkäsittelytieteiden lai-

tos, Raportti D-2007-1.
[Rogers, 2003] Rogers, Everett M., *Diffusion of Innovations*. 5. edition. New York: Free Press,
[Seffah, 2003] Seffah, A., Learning the ropes: human-centered design skills and patterns for software engineers's education. *interactions* **10**, 5 (2003), 36–45.
[Tietotekniikan liitto, 2002] Tietotekniikan liitto, Tietotekniikan ammattilaisen eettinen ohjeisto, 2002. http://www.tt-tori.fi/pls/ttl/docs/F148570701/Eettisetohjeet3.htm
[Uusitalo, 1999] Uusitalo, H., *Tiede, tutkimus ja tutkielma. Johdatus tutkielman maailmaan.* WSOY, 1999.
[Wegner, 1971] Wegner, P., A view of computer science education. In: *Proceedings of IFIP Congress 71*, vol. 2, 1515–1522. North-Holland, 1971.