Käytettävyyden suunnittelu ja rakentaminen ohjelmistotuotantoprosessissa

Mirja Immonen

Pro gradu –tutkielma

Kuopion yliopisto

Tietojenkäsittelytieteen laitos

Huhtikuu 2003

TIIVISTELMÄ

KUOPION YLIOPISTO, Informaatioteknologian ja kauppatieteiden tiedekunta
Tietojenkäsittelytieteen koulutusohjelma
Ohjelmistotekniikan linja

IMMONEN MIRJA: Käytettävyyden suunnittelu ja rakentaminen ohjelmistotuotantoprosessissa
Pro gradu -tutkielma, 88 sivua, 2 liitettä (5 s.)
Pro gradu -tutkielman ohjaajat:
FT, professori (mvs) Anne Eerola
KTT, yliassistentti Anja Mursu

Huhtikuu 2003

Avainsanat: käytettävyys, ihminen käyttäjänä, käytettävyyden hyödyt, käytettävyysskenaariot, käytettävyyden suunnittelu, käytettävyyden rakentaminen

Tietokoneet ja tietojärjestelmät ovat yhä useammin tärkeässä asemassa ihmisten jokapäiväisessä elämässä. Aivan liian usein ohjelmistojen käytettävyys on kuitenkin huono ja työntekijöiden aikaa ja energiaa kuluu itse työnteon sijasta ohjelmistojen sisältämien ongelmien kanssa kamppailemiseen.

Tutkielmassa tarkastellaan ihmiseen ja ihmisen toimintaan liittyviä asioita, jotka osaltaan täytyy ottaa huomioon käytettävää ohjelmistoa suunniteltaessa ja rakennettaessa. Lisäksi paneudutaan lyhyesti käytettävyystutkimuksen taustalla oleviin teoreettisiin lähestymistapoihin, kuten toiminnan teoriaan ja tapahtumapohjaiseen lähestymistapaan. Tutkielmassa pyritään myös kannustamaan käytettävyyden huomioimiseen läpi koko ohjelmistotuotantoprosessin esittelemällä käytettävyyden avulla saavutettavia hyötyjä ja toisaalta antamalla esimerkkejä huonosta käytettävyydestä aiheutuvista konkreettisista ongelmista.

Tutkielman pääpaino on ohjelmistotuotantoprosessissa ja siinä, kuinka käytettävyys voidaan ottaa prosessin eri vaiheissa huomioon. Tutkielmassa pohditaan, kuinka käytettävyys voidaan rakentaa järjestelmään ohjelmistotuotantoprosessin aikana, lähtien liikkeelle vaatimusmäärittelystä ja jatkuen aina järjestelmän elinkaaren loppuun saakka. Käytettävyysnäkökulmat tuodaan esille prosessiriippumattomasti siten, että esiteltäviä menetelmiä ja näkökulmia voidaan hyödyntää ja soveltaa erityyppisissä ohjelmistotuotantoprosesseissa.

SISÄLLYS

1 JOHDANTO

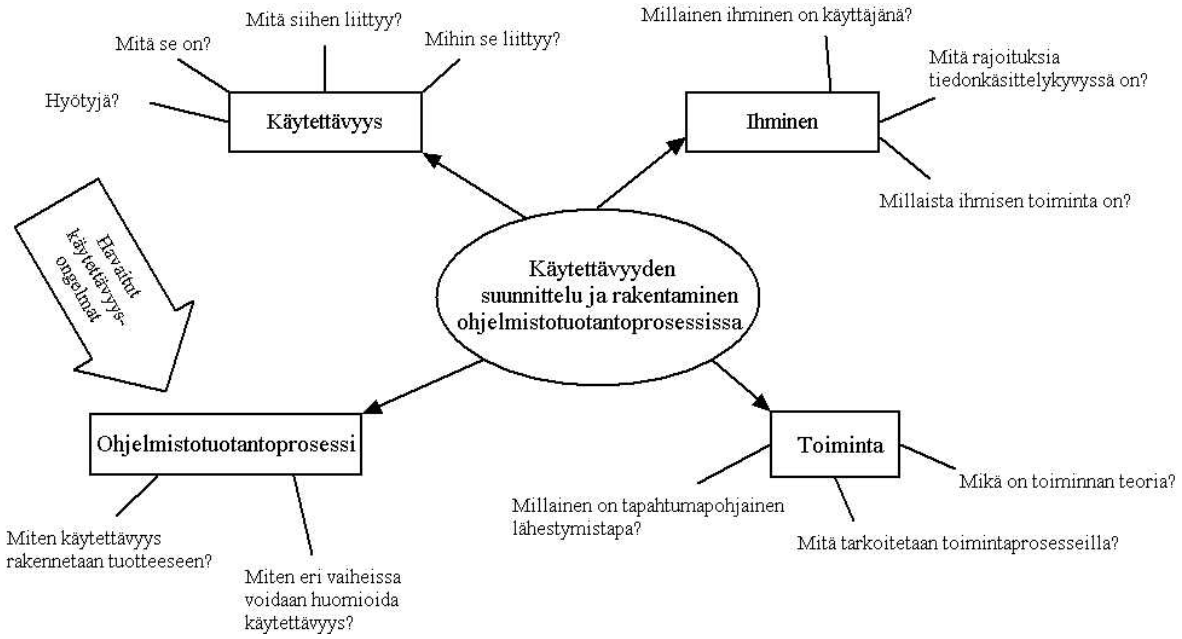
Aikuisen ihmisen ympärillä on arvioitu olevan noin 30 000 erilaista esinettä, joita on opittava käyttämään ja sen jälkeen myös osattava käyttää [Sne03]. Lähes missä tahansa ammatissa tai muussa elämäntilanteessa törmää vääjäämättä myös tietokoneisiin ja erilaisiin tietokoneistettuihin systeemeihin. Aivan liian usein näissä ohjelmistoissa ja laitteissa on jonkintasoisia käytettävyysongelmia. Useimmiten käytettävyysongelmia olisi voitu estää tai ainakin merkittävästi vähentää ottamalla käytettävyys huomioon tarpeeksi aikaisessa vaiheessa ohjelmistotuotantoprosessia ja pitämällä se mielessä koko prosessin ajan. Tosiasia on, että käytettävyys otetaan usein huomioon vasta sitten, kun ongelmia ilmenee. Siinä vaiheessa käytettävyysongelmien korjaaminen on yleensä erittäin hankalaa ja paljon kalliimpaa, kuin jos käytettävyys olisi otettu huomioon jo ohjelmistotuotantoprosessin alkuvaiheessa.

Suurin osa tällä hetkellä tehtävästä tutkimuksesta käytettävyyden piirissä keskittyy käyttöliittymiin. On myös tutkittu sitä, kuinka käytettävyys otetaan vaatimusmäärittelyvaiheessa huomioon, tai miten käytettävyyttä testataan, mutta näiden vaiheiden välillä käytettävyys näyttää useimmiten unohtuvan. Missä vaiheessa käytettävyys rakennetaan ohjelmistoon, jos se unohdetaan vaatimusmäärittelyvaiheen jälkeen ja kaivetaan esille vasta testausvaiheessa? Tämän tutkimuksen näkökulmana on koko ohjelmistotuotantoprosessi, ja kuinka sen aikana voidaan rakentaa lopputuotteeseen käytettävyyttä. Tarkoituksena on tuoda käytettävyysnäkökulmat esille prosessiriippumattomasti siten, että näkökulmat olisivat hyödynnettävissä mahdollisimman monenlaisissa ohjelmistotuotantoprosesseissa.

Ajatuksena on, että ohjelmiston käytettävyys ei ole pelkästään käyttöliittymän käytettävyyttä, vaan ohjelmiston on kokonaisuudessaan oltava toimiva ja täytettävä ne vaatimukset, jotka sille on asetettu. Erinomainenkaan käyttöliittymä ei tee ohjelmistosta käytettävää, jos esimerkiksi tietokantayhteydet eivät toimi, tai jos ohjelmistolla ei voi tehdä sitä, mihin se on tarkoitettu. Tavoitteena on kiinnittää huomiota myös todellisten loppukäyttäjien tehtäväkokonaisuuksiin ja siihen, kuinka ne vaikuttavat osaltaan tietojärjestelmän käytettävyyteen.

Kuvassa 1 on esitetty tutkielman karkea sisältö. Tutkielman rakenne koostuu siten, että aluksi perehdytään käytettävyyteen liittyviin tekijöihin ja tutustutaan yleisiin ihmiseen ja ihmisen toimintaan liittyviin tekijöihin. Sen jälkeen perehdytään yleisellä tasolla, pintaa raapaisten,

ihmisen toiminnan teoreettiseen pohjaan käymällä läpi toimintopohjaiseen lähestymistapaan, kuten toiminnan teoriaan ja toimintaprosesseihin sekä tapahtumapohjaiseen lähestymistapaan liittyviä näkökulmia. Luvussa 7 päästään lopulta ohjelmistotuotantoprosessiin ja sen eri vaiheisiin. Jokaisen vaiheen kohdalla mietitään, miten käytettävyys voitaisiin sen aikana ottaa huomioon. Tutkielman pääpaino on siten ohjelmistotuotantoprosessissa ja sen eri vaiheissa. Kahdeksannessa luvussa puolestaan pohditaan, miten käytettävyyden hyödyt voidaan saavuttaa ohjelmistotuotantoprosessin eri menetelmiä hyödyntämällä. Luku 9 liittyy tutkielmaan kuuluvaan kokeelliseen osioon, jossa kyselyn tuloksena kootut käytettävyysongelmat on ryhmitelty eri kategorioihin. Lisäksi on pohdittu, mistä ongelmat mahdollisesti johtuvat.



Kuva 1: Tutkielman rakenne miellekarttana

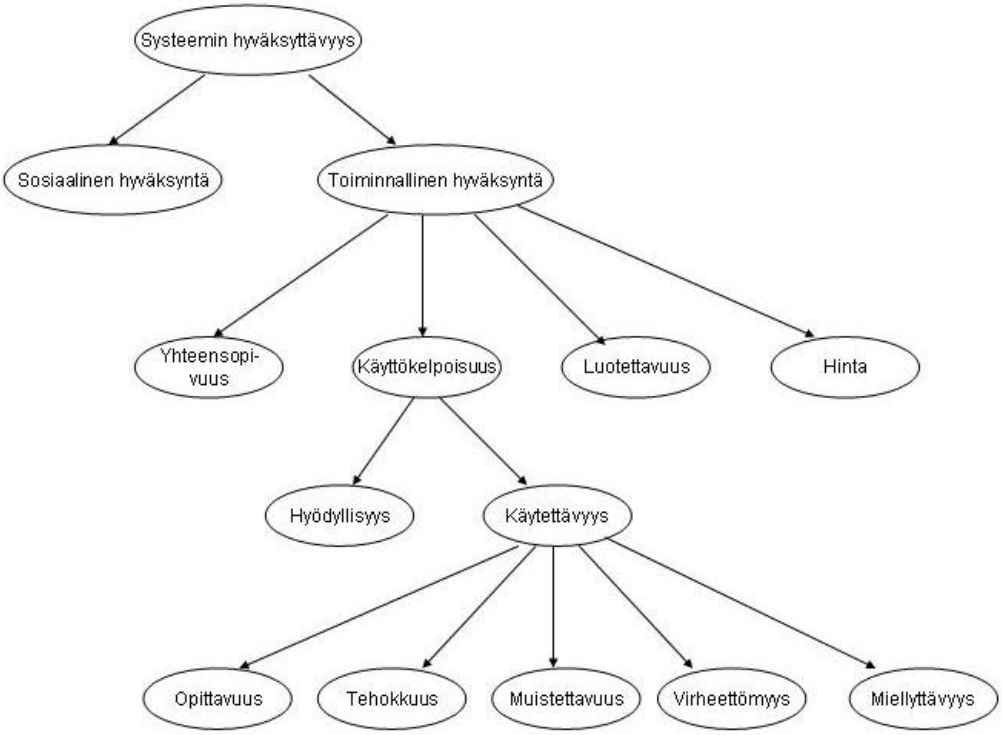
2 MITÄ ON KÄYTETTÄVYYS?

Käytettävyys (usability) on tuotteen ei-toiminnallinen ominaisuus, ja sille löytyy lähes yhtä monta määritelmää kuin on määrittelijöitäkin. Toisille käytettävyys on tuotteen helppokäyttöisyyttä tai sitä, kuinka hyvin tuote vastaa sitä käyttötarkoitusta, johon se on suunniteltu. Toisten mielestä käytettävyydellä taas tarkoitetaan sitä, että tuote on käytettävissä 24 tuntia vuorokaudessa (useability). Tässä työssä keskitytään käytettävyyteen helppokäyttöisyyden ja käyttötarkoitukseen soveltumisen näkökulmasta ja tuotteen käytettävissä (useability) oleminen rajataan tutkimuksen ulkopuolelle.

ISO 9241-11 :n ("Standardi näyttöpäätetyön ergonomiasta–ohjeita käytettävyydestä") mukaan tuotteen käytettävyys kertoo kuinka hyvin käyttäjät pystyvät tietyssä käyttöympäristössä käyttämään tuotetta tehokkaasti, tuottavasti ja miellyttävästi määriteltyjen tavoitteiden saavuttamiseksi [Usa02]. ISO-standardi huomioi siis käyttötilanteen merkityksellisyyden käytettävyyden määritelmässä, mikä on erittäin tärkeää, sillä käyttöympäristöt voivat vaihdella suurestikin. Esimerkiksi kiire, stressi, melu, lika ja työskentely-ympäristön lämpötila vaikuttavat tuotteen käytettävyyteen.

Käytettävyyden voidaan ajatella myös liittyvän yhtenä osana systeemin hyväksyttävyyteen (ks. Kuva 2) [Nie93]. Tällä tarkoitetaan sitä, kuinka hyvin systeemi täyttää käyttäjien ja muiden potentiaalisten *asianosaisten* (stakeholders), kuten asiakkaiden ja esimiesten, tarpeet ja vaatimukset. Nielsenin mukaan *hyödyllisyys* (utility) on tuotteen kyky toimia tietyssä tehtävässä ja *käytettävyys* (usability) puolestaan osoittaa, miten käyttäjä voi toteuttaa tuotteen toimintakykyisyyttä. Hyödyllisyys ja käytettävyys muodostavat yhdessä tuotteen *käyttökelpoisuuden*, joka puolestaan yhdessä muiden tuotteen havaittavien ominaisuuksien kanssa luo tuotteen toiminnallisen hyväksyttävyyden (practical acceptability).

Yleisesti ottaen käytettävyyteen kuuluvat kaikki ne systeemin piirteet, joiden kanssa ihminen voi olla vuorovaikutuksessa suoraan tai välillisesti. Tällaisia asioita ovat myös esimerkiksi systeemin asennus- ja ylläpitotoimenpiteet [Nie93].



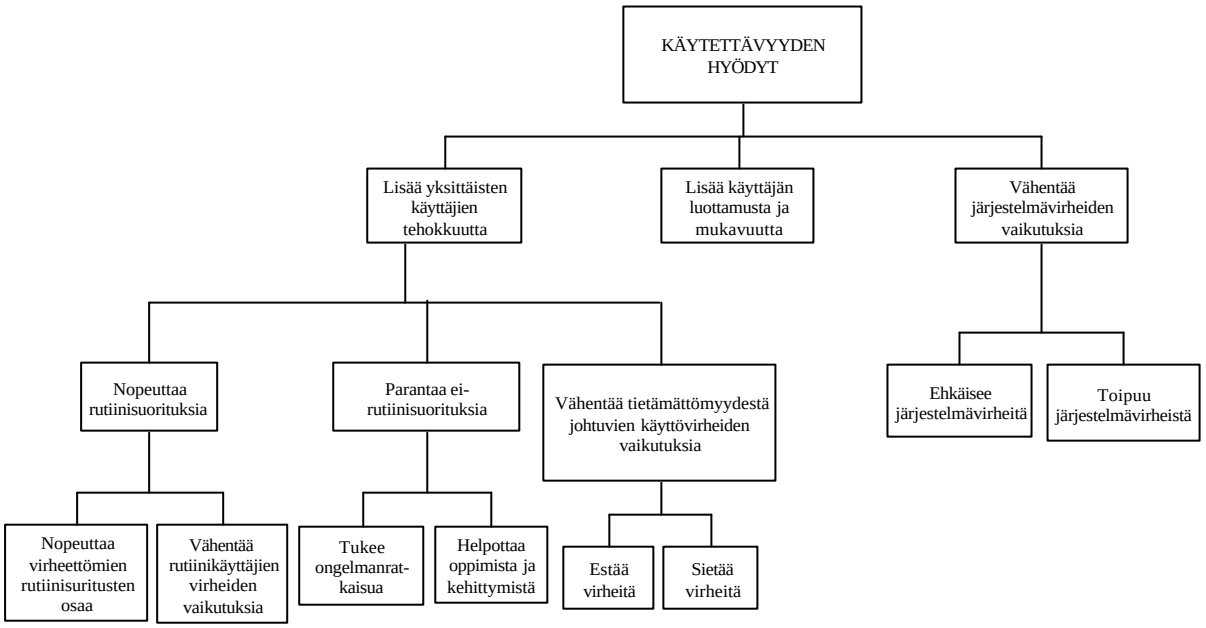
Kuva 2: Mitä on käytettävyys? (Soveltaen [Nie93] s.25)

Nielsenin mukaan [Nie93] on tärkeää muistaa, että käytettävyyteen kuuluu monia erilaisia osa-alueita. Nielsen liittää käytettävyyteen viisi ominaisuutta, joiden perusteella tuotteen käytettävyyttä voidaan arvioida: opittavuus, tehokkuus, muistettavuus, virheettömyys ja miellyttävyys.

1) *Opittavuudella* (learnability) tarkoitetaan sitä, että systeemin tulisi olla mahdollisimman nopeasti uudenkin käyttäjän omaksuttavissa. Opittavuutta siis mittaa aloittelijan käyttämä aika tuotteen kohtalaisen käyttötaidon omaksumiseen. 2) *Tehokkudella* (efficiency) tarkoitetaan sitä, että tuotteen käytön tulisi olla tehokasta, kun sen käyttö on ensin opittu. 3) *Muistettavuudella* (memorability) tarkoitetaan puolestaan sitä, että satunnaisenkin käyttäjän tulisi pystyä muistamaan tuotteen käyttötapa käyttötaukojen jälkeen opettelematta kaikkea alusta saakka uudelleen. 4) *Virheettömyydellä* (low rate of errors) pyritään siihen, että ensinnäkin systeemi olisi mahdollisimman virheetön ja toisaalta, jos käyttäjä tekee virheitä, systeemin tulisi toipua niistä helposti. 5) *Miellyttävyydellä* (satisfaction) tarkoitetaan käyttäjän subjektiivista mielty-mystä käyttää tuotetta. Toisaalta miellyttävyyteen liittyy myös *lähestyttävyyden näkökulma*, eli se, miten käytettävältä ja miellyttävältä tuote näyttää ennen sen käyttämistä.

3 KÄYTETTÄVYYDEN HYÖDYT

Aivan liian usein luullaan, että käytettävyyteen panostaminen vie paljon resursseja, mutta käytettävyydestä saavutettavat hyödyt eivät ole yhtä suuret. Käytettävyyteen panostamalla saadaan kuitenkin aikaan uskomattoman paljon hyötyjä. Näitä käytettävyyden avulla saavutettavia hyötyjä on esitetty kuvassa 3 Bass et.al [BaB01] ajatuksia soveltaen.



Kuva 3: Käytettävyyden hyödyt

Käytettävyyden avulla voidaan lisätä yksittäisen käyttäjän tehokkuutta. Tuottavuuden kasvu voi näyttää mitättömältä, jos työntekijöitä tarkastellaan yksittäin, mutta koko organisaation tai työyhteisön tuottavuus voi parantua merkittävästikin. Tehokkuuden kasvuun vaikuttavat rutiinisuoritusten nopeutuminen, ei-rutiinisuoritusten parantuminen ja tietämättömyydestä johtuvien käyttövirheiden vaikutusten väheneminen [BaB01].

Rutiinisuorituksessa käyttäjä tunnistaa tilanteen, tietää, mikä seuraava päämäärä tulee olemaan ja tietää, mitä hänen täytyy tehdä tähän päämäärään päästäkseen. Rutiinisuorituksessa ei siis tarvita ongelmanratkaisua. Vaikka käyttäjä tietää, miten hän saavuttaa päämääränsä, hän saattaa silti vahingossa tehdä virheitä suorituksessaan. Näitä *lipsahduksia* (slips) tapahtuu

enemmän kiireessä ja paineen alla. Suunnittelijan on lähes mahdotonta ennustaa lipsahdukset täydellisesti etukäteen, mutta toiset suunnittelijat kykenevät ehkäisemään niitä muita paremmin. Tyypillisiä lipsahduksia ovat esimerkiksi sellaiset kirjoitusvirheet, joissa kirjaimet vaihtavat paikkaa (ognelma – ongelma). Suunnittelijoiden täytyy pyrkiä vähentämään lipsahdusten mahdollisuuksia ja tyypillisimmät lipsahdukset tulee korjata automaattisesti. Toisaalta tapahtuneista lipsahduksista on myös toivuttava.

Ei-rutiinisuorituksessa käyttäjä ei tarkalleen tiedä, mitä hänen pitäisi tehdä. Hän saattaa esimerkiksi kokeilla ohjelman eri painikkeita, tai ottaa käyttöohjeen avukseen päämäärään päästäkseen. Käyttäjät siis käyttävät erilaisia ongelmanratkaisutapoja, kun he eivät tarkalleen tiedä, mitä heidän seuraavaksi tulisi tehdä. Käytettävä järjestelmä tukee käyttäjän ongelmanratkaisua. Järjestelmän tulisi myös tukea oppimista, ettei käyttäjän tarvitse tehdä samoja virheitä joka kerta järjestelmää käyttäessään.

Sen lisäksi, että käyttäjät tekevät virheitä (lipsahduksia) silloin, kun he tietävät päämääränsä ja kuinka siihen päästään, he tekevät virheitä myös silloin, kun he eivät tiedä, miten jostakin tilanteesta tulisi jatkaa päämäärään päästäkseen. Tietämättömyydestä johtuvia käyttövirheitä kutsutaan *erehdyksiksi* (mistakes). Tällaisissa tilanteissa käyttäjät luulevat tietävänsä, mitä haluavat, mutta he ovat erehtyneet oletuksessaan. Suunnittelun avulla ei voida estää kaikkia erehdyksiä, mutta käyttäjän tekemien valintojen tekemistä voidaan helpottaa. Käyttöliittymässä voidaan esimerkiksi himmentää sopimattomat vaihtoehdot ja tällä tavalla ohjata käyttäjän valintoja. Jos vääriä valintoja on kaikesta huolimatta mahdollista tehdä, järjestelmän tulisi mukautua niihin, mahdollistamalla esimerkiksi käyttäjän palata helposti erehdystä edeltäneeseen tilaan, kun hän huomaa tehneensä väärän valinnan. Järjestelmän tulisi myös mahdollisuuksien mukaan korjata tyypillisimmät käyttäjän tekemät virheet.

Käyttäjän kokema luottamus ja miellyttävyys järjestelmää kohtaan on myös käytettävyyden avulla saatuja hyötyjä. Järjestelmää kohtaan koettua luottamusta ja miellyttävyyttä voidaan parantaa esimerkiksi haastattelemalla tulevia käyttäjiä jo suunnitteluvaiheessa, jotta tiedetään mitä he pitävät luotettavana ja miellyttävänä [Kuj02].

Järjestelmät kohtaavat aina jonkin verran virheitä, tai erikoisia toimintatilanteita, jotka eivät johdu käyttäjästä. Verkko voi kaatua tai sähköt voivat yllättäen katketa. Suunnittelulla ei voida estää kaikkia virheitä, mutta järjestelmän tulisi toipua myös järjestelmävirheistä [BaB01].

Yleisesti ottaen käytettävyyden avulla saavutettavat hyödyt ja säästöt syntyvät monien eri asioiden summasta. Ensinnäkin hyvä käytettävyys vähentää tehtävien suorittamiseen kuluvaa aikaa ja virheiden määrää, työskentelyn keskeytymiset vähenevät, tukihenkilöstön työmäärä ja henkilöstön koulutusmäärä vähenee ja lisäksi ohjelman julkaisemisen jälkeen tehtävien muutosten määrä vähenee [KoN94].

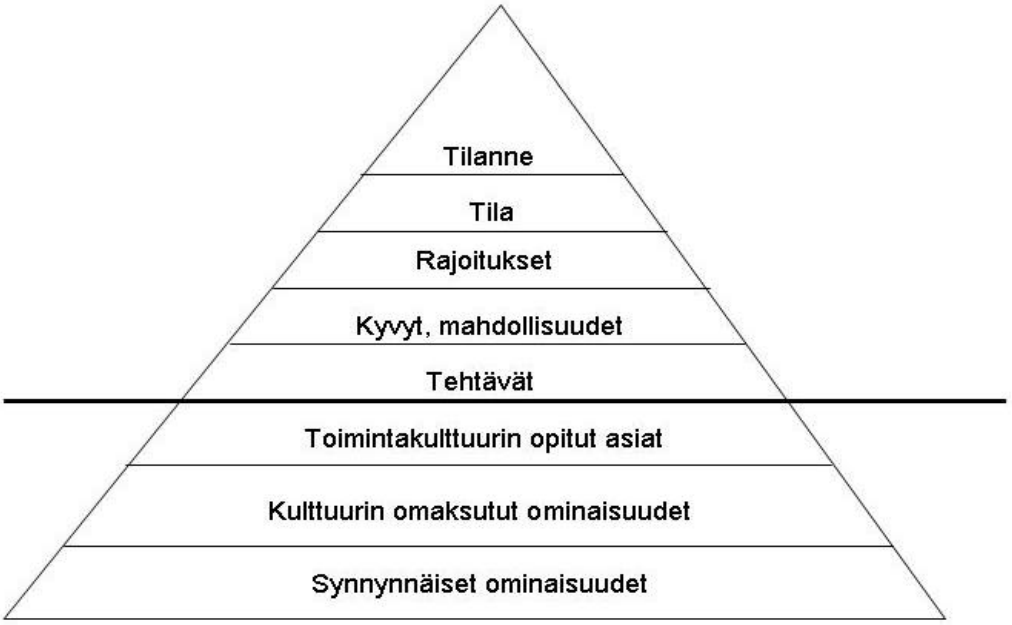
4 IHMINEN KÄYTTÄJÄNÄ

Psykologiassa muistutetaan usein ihmisen olevan psyko-fyysis-sosiaalinen kokonaisuus. Tällä tarkoitetaan sitä, että ihmisen kaikkeen toimintaan vaikuttavat hänen psyykkiset, fyysiset ja sosiaaliset toimintonsa. Psyykkiseen puoleen kuuluvat ihmisen sisällä tapahtuvat konkreettiset toiminnot, kuten ajattelu ja muisti. Fyysiseen puoleen kuuluvat elintoiminnot ja ulospäin näkyvät liikkumiseen liittyvät toiminnot. Sosiaaliseen puoleen kuuluvat sitä vastoin ihmisen toisten ihmisten ja ympäristön kanssa muodostamat suhteet, jotka perustuvat yhteistoimintaan ja kommunikaatioon. Koska tietojärjestelmien käyttäjät ovat ihmisiä, nämä seikat on tärkeää ottaa huomioon myös ohjelmistotuotantoprosessissa.

4.1 Ihmisen toimintaan vaikuttavia asioita

Ihmisen toiminnassa on paljon sellaisia asioita, jotka pätevät enemmän tai vähemmän kaikkiin ihmisiin. Toiminnalla, samoin kuin symboleilla ja väreillä, voi olla myös kulttuurisidonnaisia piirteitä. Esimerkiksi punaisella ja vihreällä sekä mustalla ja valkoisella on länsimaissa hyvin pitkälle sovitut symboliset arvot, mutta itämaissa nämä merkitykset ovat lähes päinvastaiset [SiK02].

Kehittyessään jollakin toiminnan alueella pitkälle, ihminen saattaa saavuttaa niin sanotun eksperttitason, jonka saavuttamista voidaan kuvata esimerkiksi tehtävän suorittamisen vaatimalla työmäärällä tai muuttuvilla tiedonkäsittelyllisillä ominaisuuksilla. Usein nämä määreet ovat hyvin vaikeasti todistettavia ja kvantifioitavia. Vaikka ihminen kehittyykin jollakin tietyllä osa-alueella, pysytään kuitenkin monilla muilla osa-alueilla entisellä osaamistasolla. Esimerkiksi tietokoneiden huippuohjelmoijista ei tule eksperttejä siinä työssä, jota heidän ohjelmoimansa sovelluksen käyttäjät tekevät [SiK02].



Kuva 4: Ihmisen toimintaan vaikuttavat asiat [SiK02, s. 27]

Kuvassa 4 on esitetty ihmisen toimintaan vaikuttavia asioita. Paksun viivan alapuolella olevat asiat ovat ihmiseen ja tuotteeseen liittyvää yleistietoa, joita ei tarvitse tutkia jokaisessa projektissa erikseen. Viivan yläpuolella olevia asioita ei sitä vastoin voi päätellä ilman kunnollisia projektikohtaisia tutkimuksia [SiK02].

Ihmisen toiminnan perustana ovat synnynnäiset ominaisuudet, jotka peritään geenien kautta. Kulttuuriin kuuluvat suhteellisen pysyvät asiat, kuten kieli tai osa normeista ja tavoista. Kulttuurin sisällä on myös joukko alakulttuureja, joiksi voidaan luokitella esimerkiksi erilaiset toimintakulttuurit [SiK02].

Viivan yläpuolelle jäävät ominaisuudet riippuvat tuotteen käyttötilanteesta, ihmisistä, jotka tuotetta käyttävät ja tehtävistä, joihin se on tarkoitettu. Tuotteen tulisi tukea mahdollisimman hyvin tehtäviä, joiden tueksi se on tarkoitettu. Kyvyt ja mahdollisuudet ovat niitä opittuja taitoja, joita esimerkiksi työntekijällä on. Lisäksi organisaation toimintatavat asettavat omia toimintavaltuuksia tai – mahdollisuuksia [SiK02].

Rajoitukset ovat valtuuksien tai rajoitusten puutteita. Rajoitus voi olla myös jokin tarve, jota ei ole toteutettu tuotteessa, vaan joka on toteutettava jollakin toisella tuotteella. Lisäksi siihen liittyvät sellaiset asiat kuin käyttäjän rajoitteet, kuten esimerkiksi värisokeus, tai se, että tuotteen on oltava mukana kuljetettava tai taskuun mahtuva. Lisäksi toimintaan vaikuttaa myös tila, jossa toimitaan, ja sen olosuhteet sekä itse käyttötilanne [SiK02].

On myös syytä muistaa, että käyttäjät eivät pysy samana, vaan systeemin käyttäminen muovaa käyttäjiä ja kun he muuttuvat, he käyttävät systeemiä uudella tavalla. Käyttäjien muuttumista on mahdotonta aavistaa etukäteen. Joustava suunnittelu mahdollistaa toimintatapamuutokset paremmin ja tukee näin käyttäjien kehittymistä [SiK02].

4.2 Ihmisen toiminta

Ihmisellä on useita erilaisia toimintatiloja, joita voidaan kuvata esimerkiksi käsitteillä tietoinen, tiedostamaton, kontrolloitu, automaattinen, pohtiva ja kokemuksellinen toiminta [SiK02]. Toiminta on *tietoista* silloin, kun ihminen pystyy käsittelemään mielensä sisältöjä, ajatuksia ja tunteita. Ihminen tietää tällöin itse käsittelevänsä tai prosessoivansa jotakin asiaa. *Tiedostamaton* on tila tai mielen taso, jonne talletettuja asioita ihminen ei pysty suoraan käsittelemään, eikä ole tietoinen näistä prosesseista niiden suoritushetkellä. Tiedostamattoman prosessin tulos voi kuitenkin tulla myös tietoiseksi. Tietoisten ja tiedostamattomien prosessien väliin sijoittuvat *esitietoiset prosessit*, joita ihminen ei tiedosta tapahtumahetkellä, mutta joiden lopputulos on tietoinen ja vie ihmisen huomion.

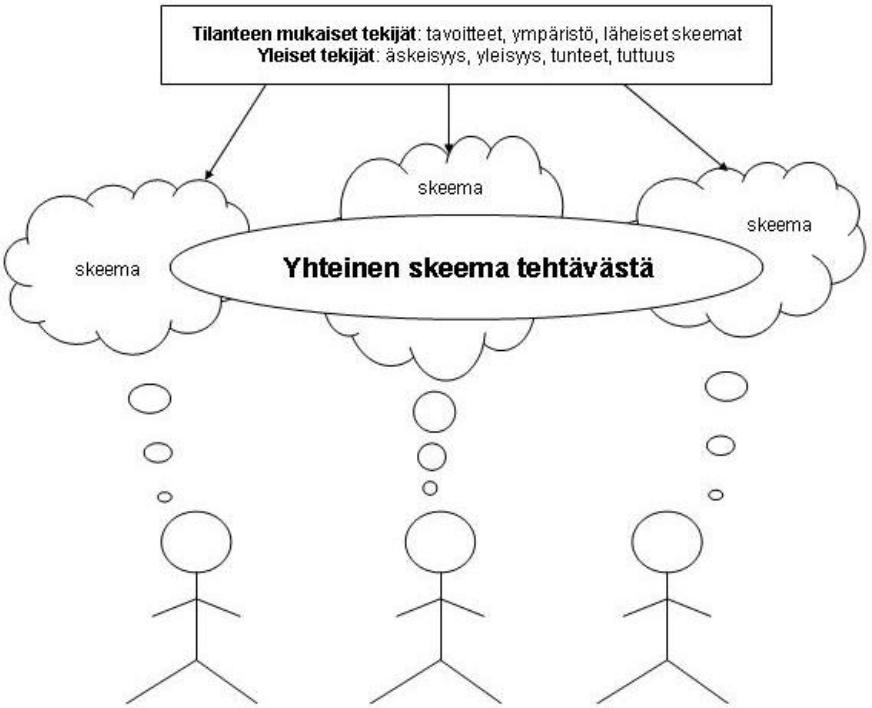
Automaattiseksi toiminnaksi kutsutaan rutiininomaisesti, ajattelematta sujuvaa toimintaa. Erityisesti motoriset toiminnot automatisoituvat harjoittelun myötä. Automatisoituneita prosesseja voidaan suorittaa yhtä aikaa ja niitä voidaan yhdistää tietoisiin toimintoihin, sillä automatisoituneet toiminnot vaativat vain vähän tarkkaavaisuutta. Automatisoituneita prosesseja on hankala muuttaa, sillä totutuista prosesseista täytyy ensin opetella pois. Automatisoituminen on tietojärjestelmän operatiivisen käytön perusedellytys, sillä tietoinen ajattelu täytyy varata järjestelmän tukemalle varsinaiselle työlle. *Kontrolloitu* toiminta on hitaampaa ja vaatii huomioita, mutta sitä on joustavampi muuttaa kuin automatisoitunutta toimintaa. Uusien asioiden opiskelussa tarvitaan kontrolloitua tietoista toimintaa. Esimerkiksi ensimmäisiä kertoja tietokonetta käyttävän ihmisen on keskityttävä erikseen hiiren liikuttamiseen ja huomio voikin kiinnittyä ohjelman eri toiminnallisuuksien opettelun sijasta siihen, kuinka kuvakkeen tai pai-

nikkeen valinta hiirellä onnistuisi sujuvasti. Kokeneelle tietokoneenkäyttäjälle hiiren liikuttaminen on automatisoitunutta toimintaa, johon ei tarvitse erikseen enää kiinnittää huomiota.

Pohtiva toiminta on suunniteltua ja harkittua. Ihminen käyttää tätä toimintatapaa esimerkiksi ratkaistessaan jotakin ongelmaa tai miettiessään asioita. Tietoa saatetaan pohtia pitkiäkin aikoja. *Kokemuksellinen* toiminta on sitä vastoin spontaania, ulkoa ohjautuvaa, tilanteesta riippuvaista ja lähtee liikkeelle havainnosta ja siihen reagoimisesta. Tällaisissa tilanteissa käytetään usein avuksi useita aisteja ja vaikka tietoa otetaan paljon vastaan, sitä käsitellään vain vähän. Ideointi on esimerkki kokemuksellisesta toiminnasta.

4.3 Ihmisen skeemat ja yhteistoiminta

Ihmisen tietoisuus koostuu käsitteiden sekä asiakokonaisuuksien ja niiden välisten yhteyksien muodostamasta skeemaverkostosta. *Skeema* on jäsentynyt ja jäsennelty informaatiokokonaisuus [SiK02]. Skeeman voidaan ajatella olevan eräänlainen kartta, jonka mukaan ihminen toimii. Nykypäivän työyhteisössä monia työtehtäviä suoritetaan useamman työntekijän ryhmissä. Samaa tehtävää suorittavien tehtäväskeemoilla on paljon yhteisiä piirteitä, mutta jokaisen omassa skeemassa on myös ainutlaatuiset piirteensä (Kuva 5). Skeeman valintaan vaikuttavat tilanteesta riippuvat tekijät kuten kulloisetkin tavoitteet, ympäristö ja kyseistä skeemaa lähellä olevat muut skeemat. Lisäksi kuhunkin tilanteeseen parhaiten sopivan skeeman valintaan vaikuttavat myös yleiset tekijät, kuten tunteet, tehtävän tuttuus, vireystaso ja se, onko kyseistä skeemaa tarvittu lähiaikoina.



Kuva 5: Ihmisten skeemat ja osittain yhteinen tehtäväkohtainen skeema

Ihmisten skeemat siis vaihtelevat samankin tehtävän kohdalla toisistaan. Skeemaerot ovat vielä suuremmat, jos kysymyksessä on ohjelmiston kohdalla esimerkiksi noviisikäyttäjä ja ekspertti tai ohjelmistosuunnittelija ja ohjelmiston loppukäyttäjä. Rakennettavan tuotteen, esimerkiksi ohjelmiston, käsitteellisen rakenteen tulisi vastata käyttäjän käsitemaailmaa suhteineen, hierarkioineen ja käsitteineen [SiK02]. Käytettävät käsitteet ja niiden suhteet voidaan määritellä analyysivaiheessa käsiteanalyysillä (ks. kappale 7.2.1). Myös loogisen rakenteen tulisi tukea käyttäjän ajatusmallia, jos ohjelmaan halutaan rakentaa käytettävyyttä.

4.4 Ihmisen tiedonkäsittelykyvyn rajoituksia

Ihmisen toimintaympäristössä on aina tarjolla runsaasti informaatiota ja toisaalta ihmisen muistiin voi myös taltioitua lähes rajattomasti tietoa, jonka pohjalta hän voi tulkita uutta informaatiota. Ihminen voi kuitenkin kerralla mieltää ja työstää vain rajoitetun määrän informaatiota. Tätä rajoitusta voidaan kutsua *tiedonkäsittelyn pullonkaulaksi* [RaW97]. Pullonkaulaa voidaan tarkastella kahdesta eri näkökulmasta. Ensinnäkin voidaan kiinnittää huomiota

valikoivaa tarkkaavaisuutta sääteleviin tekijöihin tai toisaalta voidaan keskittyä tarkastelemaan pullonkaulassa tapahtuvia prosesseja eli työmuistia [RaW97].

4.4.1 Valikoiva tarkkaavaisuus

Ihmisen tietojenkäsittelykapasiteetin rajallisuus pakottaa jatkuvaan aisti-informaation valikointiin. Prosessoitava tieto valikoituu tarkkaavaisuuden avulla. *Valikoivalla tarkkaavaisuudella* [RaW97] tarkoitetaan sitä, että ihminen suuntaa huomionsa tietoisesti johonkin kohteeseen. Mitä selvemmin jokin viesti erottuu fyysisiltä piirteiltään muusta ärsykevirrasta, sitä helpommin siihen kiinnitetään valikoivasti huomiota. Valikointia säätelevät sekä ulkoiset että sisäiset tekijät. Ulkoisiin tekijöihin kuuluvat esimerkiksi suuret, oudot tai yhtäkkiset ärsykkeet jotka laukaisevat ns. *orientoitumisreaktion*. Biologiset tarpeet ja esimerkiksi tunteisiin vaikuttavat ärsykkeet ovat sisäisiä tekijöitä, jotka suuntaavat tarkkaavaisuutta. Valikoivan tarkkaavaisuuden avulla ihminen säätelee tietoisuutensa kulloisiakin sisältöjä ja valikoi, mitä informaatiota säilötään muistiin. Tämän vuoksi valikoiva tarkkaavaisuus on edellytyksenä esimerkiksi oppimiselle ja ongelmanratkaisulle [SiK02].

Tietojärjestelmän suunnittelija voi ohjata käyttäjän huomion kiinnittymistä tärkeisiin asioihin näytöllä visuaalisin keinoin. Esimerkiksi tyhjä tila tärkeän asian ympärillä auttaa tiedon huomaamisessa [SiK02].

4.4.2 Työmuisti

Työmuisti on muistin osa, jossa tietoa säilytetään lyhyen aikaa ja tehdään esimerkiksi laskutoimituksia [SiK02]. Laajojen tutkimusten pohjalta on todettu, että työmuistin kapasiteetti on 5 ± 2 mieltämisyksikköä, eli kokonaisuutena hahmotettavaa asiaa [RaW97]. Mieltämisyksikkö voi siis olla esimerkiksi yksittäinen numero, lause tai jokin suurempi asiakokonaisuus, joka hahmotetaan yhtenä kokonaisuutena. Ihmisen muistikapasiteettia pystytään hyödyntämään tukemalla mieltämisyksikköjen muodostumista. Esimerkiksi käyttöliittymän visuaalinen ryhmittely ja hahmolait helpottavat muistamista [SiK02]. Ihmisen toimintaa voidaan helpottaa myös vähentämällä työmuistin kuormitusta asettelemalla käyttöliittymään kaikki tarvittava tieto kerralla näkyville. Kaikkiin tilanteisiin tällainenkaan menettelytapa ei kuitenkaan sovi, sillä jossain tapauksissa käyttöliittymästä voi tulla liian sekava ja sen vuoksi huonommin käytettävä.

5 TOIMINTAPOHJAINEN LÄHESTYMISTAPA

Toimintapohjaisen lähestymistavan teoreettisena perustana on toiminnan teoria (ks. kappale 5.1), jonka tavoitteena alun perin oli pyrkiä ymmärtämään ihmisten toiminnallisuutta. Ihmisten toiminta esimerkiksi työpaikalla ei ole toisista ihmisistä ja heidän toiminnastaan riippumatonta, vaan työntekijöiden toiminta muodostaa erilaisia yrityksen toimintaprosesseja (ks. kappale 5.2). Toimintaprosessien tarkoituksena on usein asiakkaan tarpeiden tyydyttäminen parhaalla mahdollisella tavalla. Aina tuotteen huono käytettävyys ei johdu pelkästään tuotteesta, vaan syy voi olla myös tehottomissa toimintaprosesseissa. Toimintaprosessia voidaan pyrkiä tehostamaan ja kehittämään toiminnankehittämishankkeen avulla (ks. kappale 5.4).

5.1 Toiminnan teoria

Toiminnan teoria (activity theory) on kehittyvä teoreettinen viitekehys [Pre94]. Toiminnan teoria käsittää kognitiivisen-, kehitys- ja kulttuuripsykologian [BaG95]. Kognitiivisen- ja kulttuurisen lähestymistavan yhdistäminen sovittaa yhteen pyrkimyksen suunnitella yksilöiden toimintaa työyhteisössä [BaG95]. Toiminnan teorian lähtökohtana on, että ihminen ei koskaan reagoi suoraan ympäristöön, vaan ihmisen ja ympäristön objektien väliseen suhteeseen vaikuttavat esimerkiksi kulttuuri ja työvälineet [Toi98].

Toiminnan teorian ydin on erilaisten tarpeiden aikaansaama ihmisen toiminnallisuus. Ihmisen toiminnallisuus on monimutkaisen evoluutiokehityksen ansiota [Toi98]. Toimintaan vaikuttaa usein yksi tai useampia välineitä tai artefakteja, kuten esimerkiksi vasara, kynä, tietojärjestelmä tai puhelin. *Työtoiminta* (work activities), kuten esimerkiksi pankkitoiminta tai ohjelmistosuunnittelu, tapahtuu käytännössä usein ihmisten muodostamassa yhteisössä. Artefaktien suunnittelu ja hyödyllisyys lähtee tavasta, jolla niitä hyödynnetään käytännössä. Paitsi että artefakteja käytetään työtoiminnassa, ne ovat myös toiminnan tuotteita. Tämän takia niiden nähdään jatkuvasti muuttuvan. Artefakteista on tärkeää opiskella niiden vaikutus ryhmätoimintaan, eikä vain keskittyä yksittäisiin käyttäjiin [Pre94].

Toiminnan (activities), toimenpiteen (action) ja operaation (operation) välillä on havaittavissa selvät käsitteelliset erot. *Toiminta* viittaa erilaisten ryhmien yhteiseen ponnistukseen tavoitteeseen pääsemiseksi. Vaikka toiminnalla on yhteinen päämäärä, toimita ei välttämättä tapahdu samaan aikaan samassa paikassa, vaan se voi olla myös hajautettua. *Toimenpide* viittaa

tietoiseen henkilökohtaiseen toimintaan johonkin pyrkimykseen pääsemiseksi. Toimenpiteet ovat suhteellisen lyhytkestoisia ja niillä on selvä alku ja loppu. *Operaatiot* ovat puolestaan henkilön keinoja ymmärtää toimenpide, joka on toteutettu suhteellisen tiedostamattomasti [Pre94].

Artefaktit, mukaan lukien tietokonejärjestelmät, tulisi suunnitella siten, että käyttäjät voivat suunnata operaationsa artefaktien sijasta toiminnan kohteeseen, jotta työskentely olisi saumatonta. Toisin sanoen artefaktien tulisi olla niin helppokäyttöisiä, että niiden käyttämiseen ei tarvitse kiinnittää erikseen huomiota. Monissa tapauksissa idea toiminnan teorian analyysille tulee yrityksestä tai organisaatiosta, joka on huomannut, että heidän toimintatavoissaan on muutoksen tarvetta. Tämä tarve voi olla esimerkiksi jonkin prosessin automatisoiminen tai halu parantaa tuottavuutta ottamalla käyttöön uusi tietokonesovellus. Toiminnan uudistamista ja kehittämistä tavoitteleva toiminnan teorian analysoija pyrkii hankkimaan tietoa ja tunnistamaan toimintahäiriöitä (breakdowns) ja ristiriitaisuuksia (contradictions) tämänhetkisessä työtoiminnassa, mikä on estänyt organisaatiota toimimasta tehokkaasti olemassa olevalla teknologialla [Pre94].

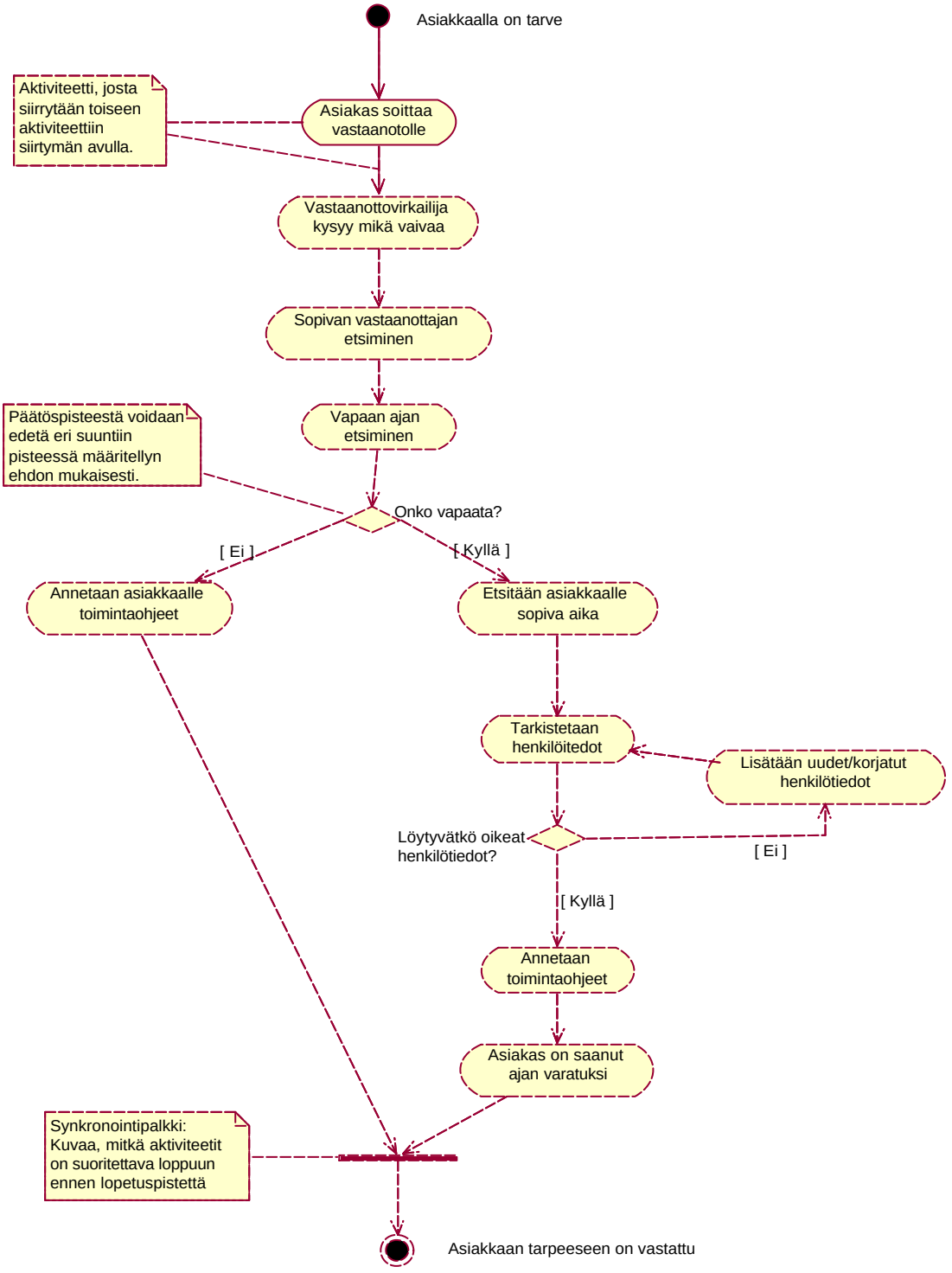
Toimintahäiriö on ristiriita sen välillä mitä oletettiin tapahtuvan ja mitä todella tapahtuu. *Ristiriita* ilmenee, kun ympäröivä kehitys estää työntekijöitä irrottautumasta tehottomista ja epämieluisista tilanteista, jotka ovat kehittyneet yhteisön käytänteiksi [Pre94].

5.2 Toimintaprosessit

Toimintaprosessit muodostuvat kaikista niistä tehtävistä, joita yrityksessä tarvitaan asiakkaan saaman palvelun tuottamiseen. Sairaalamaailmassa toimintaprosesseja ovat esimerkiksi ajanvaraus, vastaanotto, osastohoito tai sairaskertomuksen tietojen hallinta. *Liiketoimintaprosessi* on toisiinsa liittyvien toimintojen ja tehtävien muodostama kokonaisuus, joka alkaa sisäisen tai ulkoisen asiakkaan tarpeesta ja päättyy asiakkaan tarpeen tyydyttämiseen [Tie01].

Yrityksen kannalta keskeiset toimintoketjut ovat *avainprosesseja.* Avainprosessit tuottavat lisäarvoa asiakkaalle, ja ne ovat liiketoiminnan kannalta tärkeitä prosesseja. *Tukiprosessi* puolestaan mahdollistaa avainprosessin toiminnan [Ber96]. Jotta asiakas saisi mahdollisimman hyvää palvelua, eri toimintaprosessien täytyy toimia ja liittyä toisiinsa saumattomasti. Esimerkiksi sairaalan ajanvaraus-toimintaprosessin (Kuva 6) tulisi itsessään toimia mahdolli-

simman tehokkaasti ja toisaalta sen tulisi toimia tehokkaasti myös osana suurempaa toiminta-prosessiketjua, sillä se liittyy lähes aina keskeisesti johonkin toiseen sairaalan toimintapro-sessiin, esimerkiksi vastaanottoon tai laboratoriotoimintaan.



Kuva 6: Ajanvaraus-toimintaprosessin kuvaus aktiviteettikaaviona

5.3 Tietojärjestelmät ja toimintaprosessit

Toimintaprosesseissa on nykyään yhä enemmän mukana myös tietojärjestelmiä. Näin ollen tietojärjestelmien toimivuus ja käytettävyys vaikuttavat olennaisesti toimintaprosessien sujuvuuteen ja saumattomaan yhteistyöhön. Voidaan myös todeta, että liiketoimintaprosessit ja tietotekniikka ovat jatkuvassa vuorovaikutuksessa keskenään [Käh00]. Muutokset prosessissa heijastuvat teknologiaratkaisuihin ja teknologian mahdollisuudet puolestaan muuttavat prosessia.

On lisäksi syytä huomata, että ohjelmistoyrityksillä on myös omat prosessinsa, kun ne rakentavat asiakkaille ohjelmistoja. Ohjelmistoyrityksen prosessit riippuvat yrityksen toimintatavasta. Asiakaskohtaisia ohjelmistoja myyvän yrityksen avainprosesseja ovat asiakasprosessi ja ylläpitoprosessi, kun taas pakettituotteita tekevän yrityksen asiakasprosessi on yksinkertaisempi ja tuotekehitys on oma prosessinsa [HaM01]. Tietojärjestelmän käytettävyyden parantamiseksi on tarkasteltava sekä ohjelmiston tehneen yrityksen toimintaprosesseja että ohjelmistoa käyttävän tahon toimintaprosesseja. Lisäksi näiden toimintaprosessien välille on rakennettava jonkinlainen silta. Jos ohjelmistossa esiintyy käytettävyysongelmia, syitä voi löytyä sekä ohjelmistoa käyttävien organisaatioiden ja ihmisten toimintaprosesseista että ohjelmiston rakentaneen yrityksen prosesseista. Koska ohjelmiston käytettävyys rakennetaan ohjelmistoyrityksessä, sen on otettava rakennusvaiheessa asiakasyrityksen toimintaprosessit huomioon. Näin ohjelmisto on käytettävä siinä toimintaympäristössä, jossa sitä tullaan käyttämään.

5.4 Toiminnan kehittäminen

Tietojärjestelmän tai jonkin muun tuotteen huono käytettävyys ei aina johdu pelkästään siitä, että tuote olisi huono tai huonosti käytettävä. Syynä käytettävyysongelmiin voivat myös osaltaan olla käyttäjien epäsopivat toimintatavat tuotteeseen nähden tai tehottomat toimintaprosessit. Toimintatapoja ei kuitenkaan kannata lähteä väkisin muuttamaan siinä vaiheessa, kun tuote on jo otettu käyttöön. Muuten tuote voi herättää vastenmielisyyttä käyttäjissä ja muutosvastarintaa uuteen toimintatapaan siirtymistä kohtaan. Toiminnan kehittämisen tulisi tapahtua joko ennen tai viimeistään yhtä aikaa tuotteen kehittämisen kanssa ja yhteistyössä kaikkien toimintaan vaikuttavien tahojen kanssa.

Toiminnankehittämisen tulisi ensisijaisesti keskittyä yrityksen avainprosesseihin siten, että muutokset tuovat lisäarvoa asiakkaalle [HaM01]. Esimerkiksi sairaalan toimintaprosessista tulisi pyrkiä karsimaan sellaiset työt, jotka eivät tuota lisäarvoa potilaalle ja vapauttaa tämä aika potilastyöhön. Tällaisia karsittavia asioita voisivat olla esimerkiksi moniin eri tietojärjestelmiin kirjautuminen tai samojen potilastietojen tallentaminen useisiin ohjelmiin. Apua toiminnan kehittämiseen voidaan saada toiminnankehittämishankkeen avulla.

Toiminnankehittämishankkeella voi olla monenlaisia tavoitteita. Yleensä tärkeimmäksi koetaan nykyisen toimintatavan muuttaminen siten, että toimintaa haittaava epäkohta saadaan poistumaan. Joissakin tapauksissa tavoitteena voi myös olla tuotekehityksen aikaansaaminen. Tällöin toiminnan muuttamisen ohella yritetään löytää uusia fyysisiä välineitä toiminnan tehostamiseksi. Toiminnankehittämishankeen tavoitteena voi myös olla yleispätevän tiedon kerääminen toiminnan alueelta. Karttuvan tiedon avulla voidaan myöhemmin auttaa saman ongelman kohdanneita muita ihmisryhmiä [Rou02].

Kehittämishanke voi olla itseohjaava tai ulkoa ohjattu hanke. Kehittämiseen on usein tarpeetonta hakea ulkopuolelta hyväksymistä tai apua, jos toiminnan kehittämisellä ei ole sivuvaikutuksia, eli se vaikuttaa vain niihin ihmisiin, jotka toimintaa suorittavat. Tällaista toiminnankehittämistä voidaan kutsua *itseohjaavaksi*. Monissa tapauksissa toiminta ei kuitenkaan ole muista ihmisistä tai ihmisryhmistä riippumatonta. Jotta tällaisen kehittämishankkeen tulokset jäisivät pysyviksi, on tärkeää saada kaikki osapuolet kehittämiseen mukaan. Tällaista kehittämishanketta kutsutaan *ulkoa ohjatuksi* [Rou02].

Toiminnankehittämishankkeessa on tärkeää keskittyä käyttäjien nykyisiin käytänteisiin ja prosesseihin, vaikka se voikin tuntua kummalliselta, jos on esimerkiksi päätetty kehittää uusi tuote, jonka on tarkoitus tarjota aivan uusi tapa toteuttaa olemassa olevia tehtäviä. Nykyisestä toimintatavasta on tärkeää tunnistaa hyvät ja huonot puolet. Nykyisten prosessien hyvät puolet tulisi säilyttää ja viat korjata. Lisäksi käyttäjien tulisi pystyä hyväksikäyttämään omia taitojaan myös uudessa toimintatavassa [Kuj02].

Nykyiseen toimintatapaan ja toimintaprosesseihin tutustuttaessa on tärkeää kiinnittää huomiota niihin kohtiin, joissa toimitaan manuaalisesti, eli työntekijä esimerkiksi vie jonkin paperin toiselle työntekijälle jatkokäsittelyyn. Tällaisen tilanteen tullessa esille olisi syytä miettiä, miksi näin tehdään ja voisiko tämä paperin välitys tapahtua esimerkiksi sähköpostia tai muuta

viestinvälitystä apuna käyttäen. On mahdollista, että tälle manuaaliselle toimintatavalle on jokin järkevä peruste, joten toimintatapaa ei välttämättä voida muuttaa, mutta tällaiset tilanteet on syytä kartoittaa toiminnankehittämishankkeessa.

6 TAPAHTUMAPOHJAINEN LÄHESTYMISTAPA

Tapahtumapohjaisessa lähestymistavassa [Rob99] puhutaan toimintaprosessien sijasta työstä. *Työllä* (work) ei tarkoiteta pelkkää ohjelmistoa, vaan se on systeemi, jonka avulla tehdään liiketoimintaa. Työ sisältää ihmisten tekemiä tehtäviä, tietokoneita ja muita erilaisia laitteita, kuten puhelimia ja kopiokoneita. Työhön kuuluu kaikki, mitä tarvitaan tuottamaan asiakkaille tarvikkeita tai palveluja.

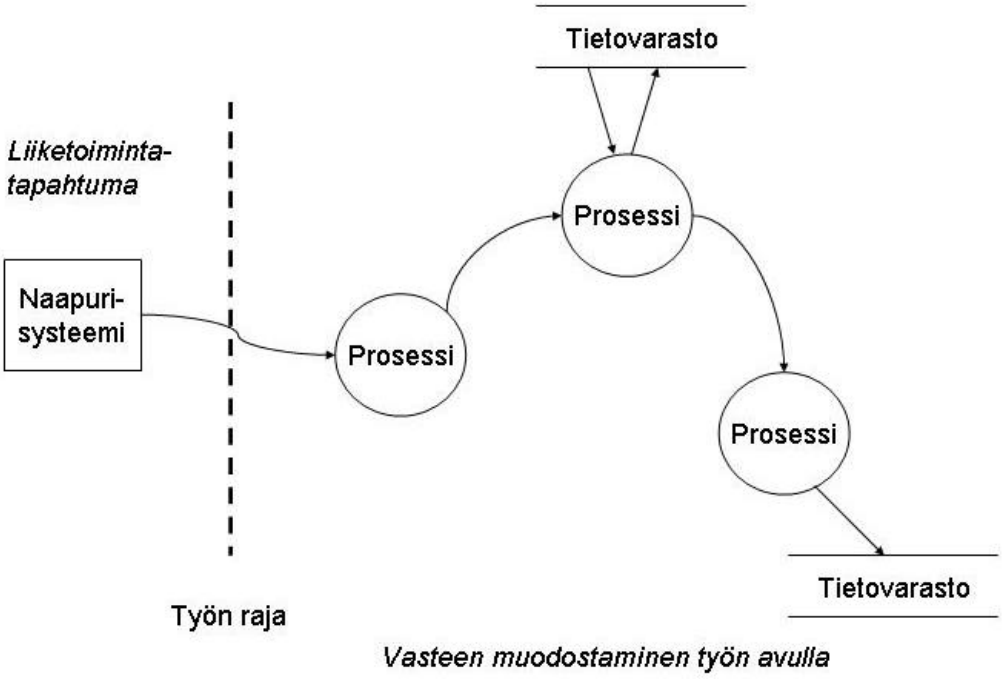
Rakennettavasta tuotteesta tulee osa asiakkaan työtä, ja tämän vuoksi asiakkaan toimintatapojen selvittäminen on tärkeää ohjelmistotuotantoprosessin alkuvaiheessa. On väärin aloittaa suunnittelu tuotteesta ja sovittaa valmis tuote työhön. On tärkeää aloittaa suunnittelu työn ymmärtämisestä ja pyrkiä rakentamaan tuote, joka tukee tätä työtä. Jotta työtä pystyisi ymmärtämään, on tiedettävä, miten se liittyy ulkopuoliseen maailmaan. Yleensä työ saa jotain ulkopuoliselta maailmalta ja lähettää sinne jotakin. Nämä yhteydet ulkomaailmaan voidaan kuvata *kontekstikaaviolla* (work context). On erittäin tärkeää ottaa alkuvaiheessa huomioon kaikki, jotka tuotteeseen vaikuttavat. Kun asiakkaan tekemästä työstä on saatu syvällisempi ymmärrys, on päätettävä, kuinka suuren osan rakennettava tuote tästä työstä tekee.

6.1 Liiketoimintatapahtumat ja naapurisysteemit

Liiketoimintatapahtumat (Business events) ovat tapahtumia, jotka tapahtuvat työn ulkopuolella ja toimivat herätteinä, joihin systeemin on vastattava parhaalla mahdollisella tavalla. Liiketoimintatapahtuma siis käynnistää systeemissä prosessin, joka muodostaa vasteen tapahtumalle. *Vaste* (response) on systeemin suorittama yhtenäinen ketju prosesseja, joita suoritetaan, kunnes tarvittavat tiedot on esimerkiksi luettu ja talletettu tai on muodostettu muu sopiva tuotos (output) (Kuva 7). Vaste ei välttämättä ole aina konkreettinen tuloste, vaan se voi olla myös muunlainen vastaus herätteeseen. Vasteen käsite vastaa toimintaprosessien liiketoimintaprosessi-käsitettä (ks. kappale 5.2). Jotta vaste olisi paras mahdollinen, ketjun prosessien tulee muodostaa paras mahdollinen kokonaisuus.

Naapurisysteemit muodostavat sen osan maailmaa, johon työ on yhteydessä. Naapurisysteemi voi olla ihminen, organisaatio, teknologiayksikkö tai edellä mainittujen yhdistelmä. Toimintatavaltaan naapurisysteemi voi olla aktiivinen, autonominen tai yhteistyötä tekevä. *Aktiivinen*

naapurisysteemi on yleensä ihminen ja käyttäytyy dynaamisesti. Tällainen naapurisysteemi voi vaihtaa systeemin kanssa esimerkiksi tietoa, kunnes sen tavoite on täytetty. Esimerkiksi pankkiautomaattia käyttävä asiakas on pankkijärjestelmän aktiivinen naapurisysteemi. *Autonominen naapurisysteemi* toimii itsenäisesti tarkasteltavaan työhön nähden, mutta sillä on työhön kuitenkin jonkinlainen liittymä. Kommunikointi tapahtuu yksisuuntaisen yhteyden avulla. Autonominen naapurisysteemi on usein jokin toinen yritys. Luottokunta on esimerkki autonomisesta naapurisysteemistä, kun se lähettää kuukausittain tiliotteen asiakkaalleen asiakkaan sitä erikseen pyytämättä. *Yhteistyötä tekevät naapurisysteemit* toimivat ennustettavasti ja luotettavasti ja antavat välittömästi vasteen systeemille. Yhteistyötä tekevät naapurisysteemit ovat usein automatisoituja ja noudattavat yksinkertaista pyyntö-vastaus-rajapintaa. Tietokanta on yhteisyötä tekevä naapurisysteemi antaessaan ennustettavissa olevan, dokumentoidun, välittömän vasteen.



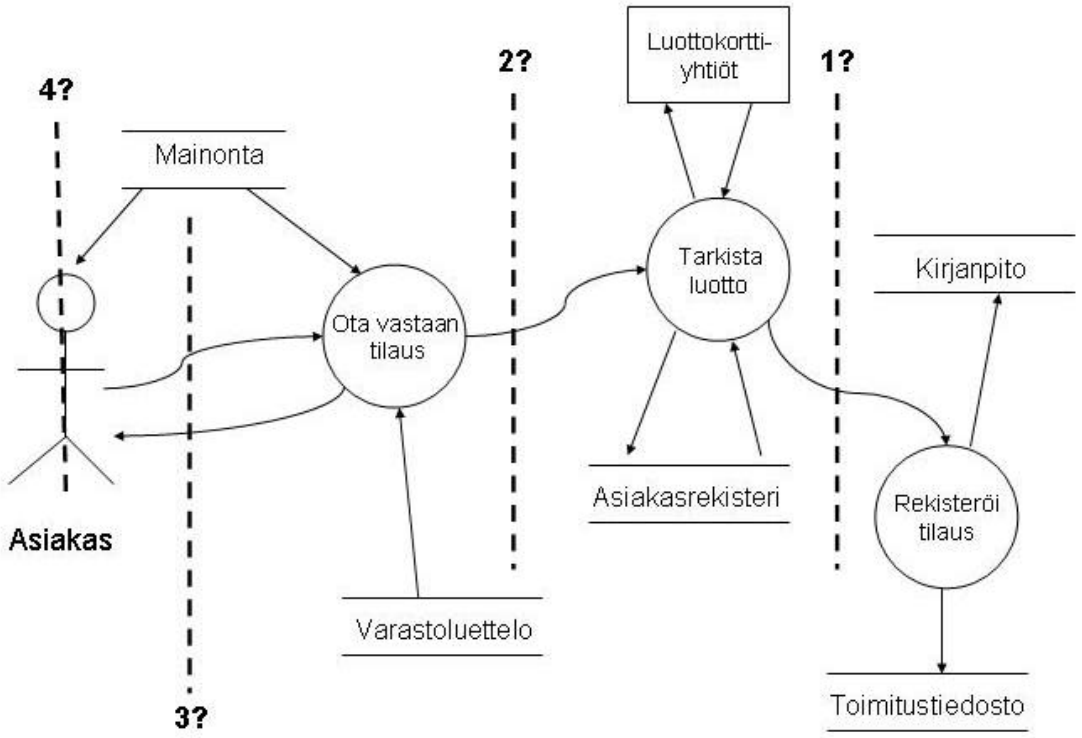
Kuva 7: Liiketoimintatapahtuman aikaansaama prosessiketju (soveltaen [Rob99, s. 61])

6.2 Tuotteen rajaaminen

Tuotteen oikeanlainen ja käyttöyhteyteensä sopiva rajaaminen eli se, mistä tuote alkaa ja mihin se päättyy, on tärkeä tekijä tuotteen käytettävyydessä. Oikeanlainen rajaus osaltaan myös rationalisoi toimintaprosesseja ja helpottaa työntekijöiden jokapäiväistä työskentelyä. On tärkeää muistaa, että tuotteen oikea raja riippuu siitä, mitä käyttäjä haluaa ja tarvitsee.

Kuvassa 8 on tilanne, jossa asiakas tilaa puhelimella tuotteita jostakin postimyyntifirmasta suoraan kotiinsa. Jotta tilausjärjestelmä olisi mahdollisimman hyvin käyttöyhteyteensä sopiva ja käytettävä, järjestelmän tekijöiden on mietittävä tarkkaan, mistä kohtaa systeemi alkaa. Jos tuotteen rajana on kohta 1, se tarkoittaa sitä, että asiakaspalveluhenkilö tms. ottaa vastaan tilauksen, tarkistaa tai lisää asiakkaan tiedot asiakasrekisteriin, tarkistaa luottotiedot ja lopulta tallentaa tilaustiedot järjestelmään. Järjestelmän käyttäjä tekee siis tällaisessa tapauksessa valtaosan työstä manuaalisesti. Jos tuotteen rajana on kohta 2, asiakaspalveluhenkilö ottaa vastaan tilauksen, mutta luottotietojen tarkistaminen ja tilauksen rekisteröiminen tapahtuvat automaattisesti. Monissa postimyyntifirmoissa on käytössä tällainen menettelytapa, eli asiakas voi soittaa tilauksensa puhelimitse ja samalla asiakaspalveluhenkilö tekee tilauksen järjestelmään ja pystyy suoraan sanomaan, onko kyseisiä tuotteita varastossa. Jos raja on kohdassa 3, myös tilauksen vastaanottaminen on automaattista. Tällainen on mahdollista esimerkiksi siten, että asiakas täyttää tilauslomakkeen Internetissä postimyyntifirman kotisivuilla.

Järjestelmän rajan ollessa kohdassa 4, järjestelmä huomaa itse, milloin asiakas tarvitsee jotakin ja ottaa yhteyttä asiakkaaseen sopiakseen esimerkiksi tuotteen toimitusajankohdasta. Tällöin asiakkaan ei itse tarvitse huolehtia tilauksen tekemisestä. Tällainen menettelytapa on käytössä monilla lehtiä myyvillä yrityksillä. Jos asiakkaalla on määräaikaistilaus, ei hänen tarvitse itse huolehtia tilauksen uusimisesta, vaan yleensä lehtimyyjä soittaa jo hyvissä ajoin ennen tilausjakson päättymistä kysyäkseen, haluaako asiakas uusia tilauksensa. Tulevaisuudessa tullaan varmaan hyödyntämään yhä enemmän teknologian suomia mahdollisuuksia ja varmaan on vain ajan kysymys, milloin tieto esimerkiksi tilausjakson päättymisestä ja mahdollisesta uudesta tilauksesta hoidetaan sähköpostilla tai tekstiviesteillä.



Kuva 8: Tuotteen rajaaminen (soveltaen [Rob99] s. 71)

7 OHJELMISTOTUOTANTOPROSESSI

Käyttäjät ja käytettävyys tulisi ottaa huomioon koko ohjelmistotuotantoprosessin ajan, mikäli halutaan rakentaa lopputuote, joka soveltuu hyvin siihen käyttötarkoitukseen ja siihen käyttöympäristöön, johon se on suunniteltu. Mitä myöhemmässä vaiheessa käytettävyysongelmat tulevat esille, sitä kalliimpaa ja työläämpää virheiden korjaaminen on [Chr00]. Joskus virheiden korjaaminen voi olla myös mahdotonta, jos niitä ei ole huomattu tarpeeksi ajoissa. Useat tutkimukset osoittavat, että käytettävyysasioiden huomioiminen tarpeeksi aikaisessa vaiheessa vähentää ohjelmistotuotantoprosessin kokonaiskustannuksia. On todettu, että on jopa 200 kertaa kalliimpaa korjata käytettävyysongelma ylläpitovaiheessa, kuin vaatimusmäärittelyvaiheessa [Kuj02]. Toisaalta tutkimukset osoittavat myös, että 80 % ylläpitokustannuksista aiheutuu ennalta arvaamattomista käyttäjien vaatimuksista ja vain 20 % varsinaisista virheistä [Chr00]. Tämän vuoksi ohjelmien muunneltavuuteen pitäisi panostaa entistä enemmän. Se rahamäärä, joka kulutetaan käyttäjien tehtävien omaksumiseen suunnitteluvaiheessa, säästetään kymmenkertaisesti ongelmien korjaamisessa tuotteen kehitysvaiheessa ja jopa satakertaisesti ylläpitotyössä tuotteen julkaisemisen jälkeen [Chr00]. Käyttäjien huomioiminen ei siten ole turhaa ajan ja rahan tuhlausta ohjelmistotuotantoprosessin alkuvaiheessa.

On syytä muistaa, että seuraavissa kappaleissa esitetyn ohjelmistotuotantoprosessin vaiheet eivät seuraa toisiaan välttämättä lineaarisesti, vaan usein edellinen vaihe on vielä kesken, kun siirrytään jo seuraavaan vaiheeseen. Myös projektin luonteesta riippuu, käydäänkö kaikkia vaiheita läpi lainkaan. Projekti, jossa tehdään uutta versiota jo markkinoilla olevaan ohjelmistoon, poikkeaa varmasti uuden tuotteen tuotekehitysprojektista.

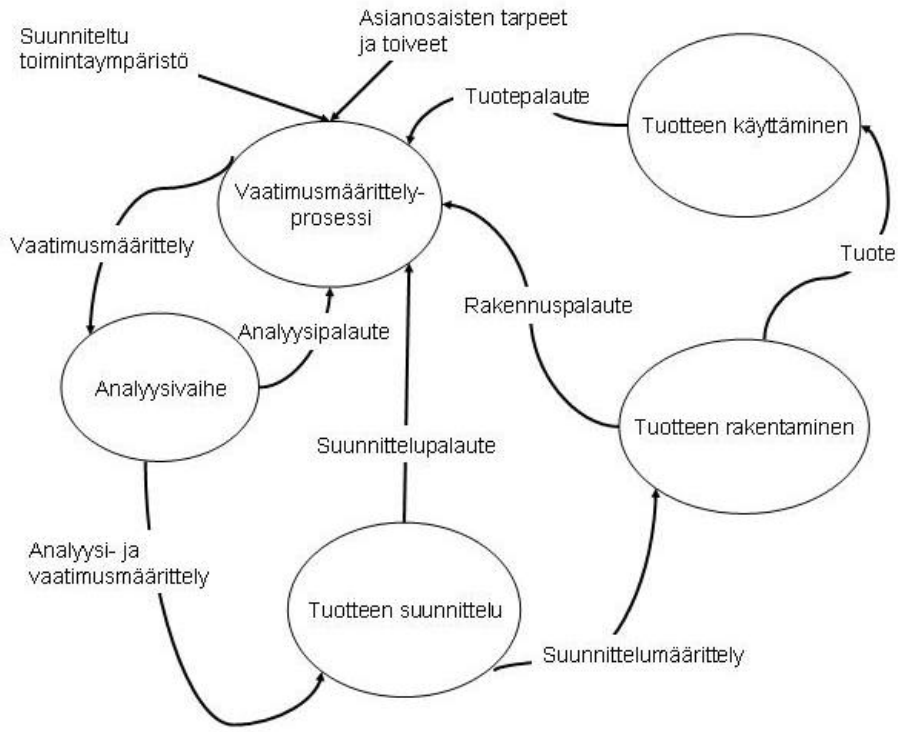
7.1 Vaatimusmäärittely

Vaatimukset ovat asioita, jotka tulisi löytää ennen tuotteen rakentamisen aloittamista. *Toiminnalliset vaatimukset* ovat asioita, jotka tuotteen täytyy tehdä ja tietoja, jotka täytyy tallentaa tai hoitaa. *Ei-toiminnalliset vaatimukset* ovat ominaisuuksia tai piirteitä, jotka tuotteella täytyy olla [Rob99]. Käytettävyys kuuluu tuotteen ominaisuutena ei-toiminnallisiin vaatimuksiin, mutta toisaalta käytettävä tuote täyttää toiminnalliset vaatimukset. Jos oikeita vaatimuksia ei ole määritelty tai niitä ei ole löydetty, ei voida rakentaa oikeaa tuotetta. Käyttäjätarpeiden ymmärtäminen nähdään usein tuotekehityksen menestystekijänä [Kuj02]. Vaatimusmääritte-

lyn tekeminen voi olla vaikeaa, sillä ei ole mitenkään epätavallista, että käyttäjät eivät tiedä mitä he haluavat, tai he eivät osaa ilmaista tuotetta koskevia tarpeitaan. Monissa käyttäjille erittäin tutuissa töissä osa tärkeästä työhön liittyvästä tietämyksestä on heille niin itsestään selvää, että he eivät edes ymmärrä mainita niistä erikseen. Tämän vuoksi käyttäjien tarkkailu heidän tehdessään omaa työtään on tehokas keino saada selville tarpeita ja vaatimuksia. Myös työn ja työskentelyn sosiaalinen luonne tulee ottaa vaatimusmäärittelyvaiheessa huomioon.

Suunnittelijoiden harmiksi vaatimukset eivät ole pysyviä, vaan ne muuttuvat sekä ohjelmistotuotantoprosessin että tuotteen käytön aikana (Kuva 9). Vaatimusten muuttumista suunnittelijoiden on vaikea ennustaa, mutta silti se on hyväksyttävä [Rob99]. Vaatimusmäärittelyprosessin (Requirements Prosess) aikana määritellään tuotteen toiminnalliset- ja ei-toiminnalliset vaatimukset sekä mahdolliset rajoitukset. Rajoituksia voivat liittyä projektiin, kuten raha ja henkilöresurssit tai vanhaan käytössä olevaan tietojärjestelmään. Tällaiset rajoitukset voivat hankaloittaa tai jopa estää käytettävyyden rakentamista tuotteeseen. Vaatimusmäärittelyprosessin tuloksena syntyy vaatimusmäärittelydokumentti, jossa kuvataan tuotteen toiminnallisuus ja käyttäytyminen.

Analyysivaiheessa muodostetaan malleja toiminnoista ja tiedoista, joita tuotteessa tarvitaan. Tuotteen suunnitteluvaiheessa tämä abstrakti tietämys muutetaan fyysiseksi ja teknisemmäksi, eli päätetään esimerkiksi mitä apuvälineitä käytetään, mitä komponentteja tarvitaan ja miten ne muodostetaan. Kun tuote on rakennettu, sitä käytetään, ja väistämättä silloin syntyy uusia vaatimuksia. Lisäksi jokaisessa edellä lyhyesti kuvatussa vaiheessa syntyy palautetta, jonka avulla muutetaan ja täydennetään vaatimuksia [Rob99].



Kuva 9: Vaatimusten muuttuminen ja tarkentuminen ([Rob99], s. 3)

Seuraavissa kappaleissa kuvataan käytettävyysprosessin vaiheita, jotka kuuluvat vaatimusmäärittelyvaiheeseen. Vaatimusmäärittelyvaiheen aikana on tärkeää oppia tuntemaan käyttäjät, käyttäjien toimintaympäristö ja toimintatavat sekä se, mitä tuotteen odotetaan tekevän tässä kontekstissa. Tuotteen laatua ja käytettävyyttä ei voida erottaa käyttäjien maailmasta. Laatu ja tuotteen käytettävyys riippuvat siitä, kuka tuotetta käyttää ja millaisessa ympäristössä. Käyttäjävaatimukset kuvaavat, kuinka tulevaisuuden tuote auttaa käyttäjiä saavuttamaan päämääränsä tehokkaasti siten, että siihen ollaan tyytyväisiä tuotteen käyttöyhteydessä [Kuj02]. Käyttäjävaatimuksista johdetaan tuotteen vaatimukset.

7.1.1 Käyttäjien tunnistaminen ja ryhmittely

Ensimmäinen vaihe käytettävyysprosessissa on oppia tuntemaan tulevat käyttäjät ja se, kuinka he käyttävät tuotetta. Erityisesti tulisi ottaa huomioon yksittäisten käyttäjien ominaisuudet ja tehtävien vaihtelevuus, mitkä ovat kaksi käytettävyyteen eniten vaikuttavaa muuttujaa. On myös otettava huomioon, että käyttäjiä ovat myös esimerkiksi ohjelman asentajat ja ylläpitäjät [Nie93].

Tuotteen käytettävyyden kannalta on tärkeää luokitella tulevat käyttäjät. Tietämällä, millaisia tulevat käyttäjät ovat, voidaan ennakoida, mitkä asiat mahdollisesti hankaloittavat tuotteen käytön oppimista tai minkä tasoinen käyttöliittymä olisi sopiva. Esimerkiksi lukutaidottomille lapsille suunnatussa ohjelmassa täytyy olla sellainen käyttöliittymä, jonka käyttämiseen ei tarvitse osata lukea. Myös käyttäjien työympäristö ja sosiaalinen asema täytyy olla tiedossa [Nie93].

Tulevista käyttäjistä saatavilla oleva tiedon määrä vaihtelee tapauskohtaisesti. Joissain tapauksissa valmistettavaa ohjelmaa tullaan käyttämään vain tietyn yrityksen tietyllä osastolla, jolloin tulevat käyttäjät ovat jo ennakolta tiedossa [Nie93]. Toisaalta tuote voi olla suunnattu suuremmalle joukolle ihmisiä, eikä tarkkaa käyttäjäryhmää voida rajata. Tällaisissa tapauksissa on tärkeää tunnistaa erilaiset käyttäjäryhmät ja oppia tuntemaan kunkin tärkeänä pidetyn ryhmän edustajan toimintatavat [Kuj02].

Suurin osa siitä tiedosta, jota tarvitaan luonnehtimaan tulevia käyttäjiä, saadaan markkina-analyysien avulla tai havainnoimalla käyttäjiä. Apuna voidaan käyttää myös kyselylomakkeita tai haastatteluja. Ei pitäisi kuitenkaan luottaa pelkästään kirjoitettuun tietoon, sillä useimmiten uusia oivalluksia saadaan kartoitettua parhaiten havainnoimalla ja juttelemalla todellisten käyttäjien kanssa heidän omassa työskentely-ympäristössään [Nie93].

7.1.2 Tehtäväanalyysi

Käyttäjien työn ja tehtävien kautta oppimat asiat ja toimintatavat poikkeavat suuresti suunnittelijoiden oppimasta [SiK02]. Suunnittelijasta tulee usein työnsä myötä sen laitteen tai ohjelmiston asiantuntija, jota hän suunnittelee. Käyttäjä on puolestaan asiantuntija siinä tehtävässä, jota hän yrittää laitteen tai ohjelmiston avulla suorittaa [Nor91]. Lisäksi rakennettavasta tuotteesta tulee osa asiakkaan työtä [Rob99]. Tämän vuoksi jokaiseen tuotekehitysprojektiin tulisi kuulua käyttäjien toiminnan seuraaminen ja ymmärtäminen, jotta tiedetään mitä käyttäjät tuotteella oikeasti tekevät, ja mitkä ovat potentiaalisia virhetilanteita.

Pelkkä käyttäjien havainnointi laboratorioympäristössä ei ole riittävää, sillä laboratorioympäristö on aivan erilainen kuin käyttäjien todellinen toimintaympäristö. Käyttäjien normaali toimintaympäristö voi auttaa käyttäjiä muistamaan käyttäytymisensä yksityiskohtia, ja toisaalta ne kohdat, missä teknologiaa hyödynnetään työssä saattavat jäädä huomaamatta haastatte-

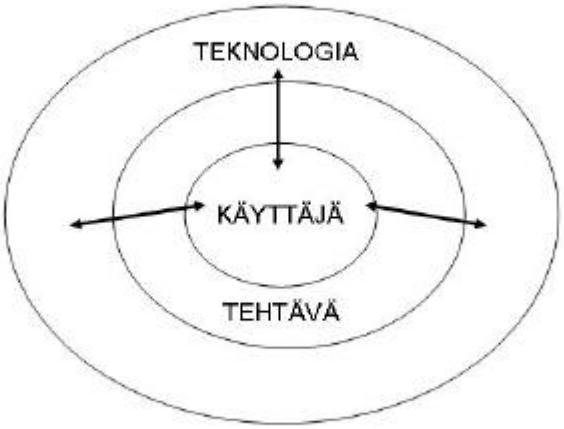
luissa [Kuj02]. On kuitenkin syytä muistaa, että havaintojen tekeminen luonnollisessa ympäristössä ei välttämättä onnistu aivan kokemattomalta havainnoijalta, vaan tarkoituksenmukainen havainnointi vaatii kokemusta. Havainnointitilanteen vaarana on, että havainnoija kiinnittää huomiota epäoleellisiin asioihin ja merkitykselliset asiat jäävät huomaamatta. Tämän vuoksi esimerkiksi videointi on hyvä apuväline havaintoja tehtäessä. Kaikista hedelmällisintä olisi, jos voidaan sekä havainnoida että haastatella loppukäyttäjiä. Esimerkiksi ennen havainnointia tehtävä haastattelu tutustuttaa havainnoijan ja käyttäjän toisiinsa, eikä havainnointitilanne vaikuta välttämättä sen jälkeen hämmentävältä tilanteelta. Toisaalta ennen havainnointia tehtävässä haastattelussa voidaan kysellä taustatietoja työskentelystä ja tehtävistä. Myös havainnoinnin jälkeen on syytä vielä haastatella loppukäyttäjiä, sillä havainnoija ei välttämättä ymmärrä kaikkia toimintaprosessin osia ja havainnointitilanne voi sen vuoksi herättää lisää kysymyksiä ja tarkennettavia kohtia. Myös käyttäjä voi muistaa havainnointitilanteen jälkeen lisää yksityiskohtia työstään.

Menetelmät, joiden avulla käyttäjien toimintaa pyritään ymmärtämään, voidaan jakaa kahteen ryhmään: testaaminen ja käyttäjien toiminnan hahmottaminen [SiK02]. *Testaamiseen* liittyvät käytettävyystestit eri muodoissaan (ks. kappale 7.9). Testaaminen edellyttää, että käytössä on tuote tai tuotteen prototyyppi. *Hahmottaminen* on käyttäjien toiminnan seuraamista ja mallintamista niin, että tuote voidaan suunnitella käyttäjien toimintaa tukevaksi. Hahmottamiseen on käytettävissä kaksi menetelmää toimintatarinat ja käyttötarinat (ks. kappaleet 7.1.3 ja 7.1.4), jotka voidaan rakentaa tehtäväanalyysistä saatujen tietojen perusteella.

Tehtäväanalyysi on tärkeä osa käytettävyyden suunnittelua jo projektin alkuvaiheessa, sillä käyttäjien tavoitteet, nykyiset toimintatavat, tiedon tarpeet ja toimintatavat erikoistilanteissa tulisi selvittää huolellisesti [Nie93]. Tehtäväanalyysi auttaa sekä oikean toiminnallisuuden että riittävän ymmärrettävyyden suunnittelussa.

Tehtäväanalyysi sisältää joukon tekniikoita, joiden avulla voidaan paremmin saada selvyys siitä, miten käyttäjät käyttävät systeemiä. *Päämäärää* (goal) kutsutaan myös *ulkoiseksi tehtäväksi* (external task), ja se on systeemin tila tai tulos, jonka käyttäjä haluaa saavuttaa. Päämäärä saavutetaan käyttämällä jotakin apuvälinettä, metodia, edustajaa, työkalua, tekniikkaa tai taitoa. Yleisesti ottaen päämäärä saavutetaan jokin sellaisen keinon avulla, joka mahdollistaa systeemin tilan muuttua toivotuksi [Pre94]. Tuotteen käyttäminen sinänsä on harvoin kenenkään päämääränä, vaan tuote on vain apuväline, jonka avulla päämäärään toivotaan päästävän

[SiK02]. Kun käyttäjä on muotoillut oman tavoitteensa, hän valitsee keinon, jonka avulla hän saavuttaa oman päämääränsä. Yhä useammin tähän päämäärään päästään teknologian avulla. Voidaan ajatella, että ihminen on suorassa vuorovaikutuksessa pikemminkin teknologian kuin itse tehtävän kanssa (Kuva 10). Esimerkiksi ohjelmistosuunnittelija piirtää useimmiten UML-kaavioita jonkin sovelluksen, kuten Rational Rosen avulla.



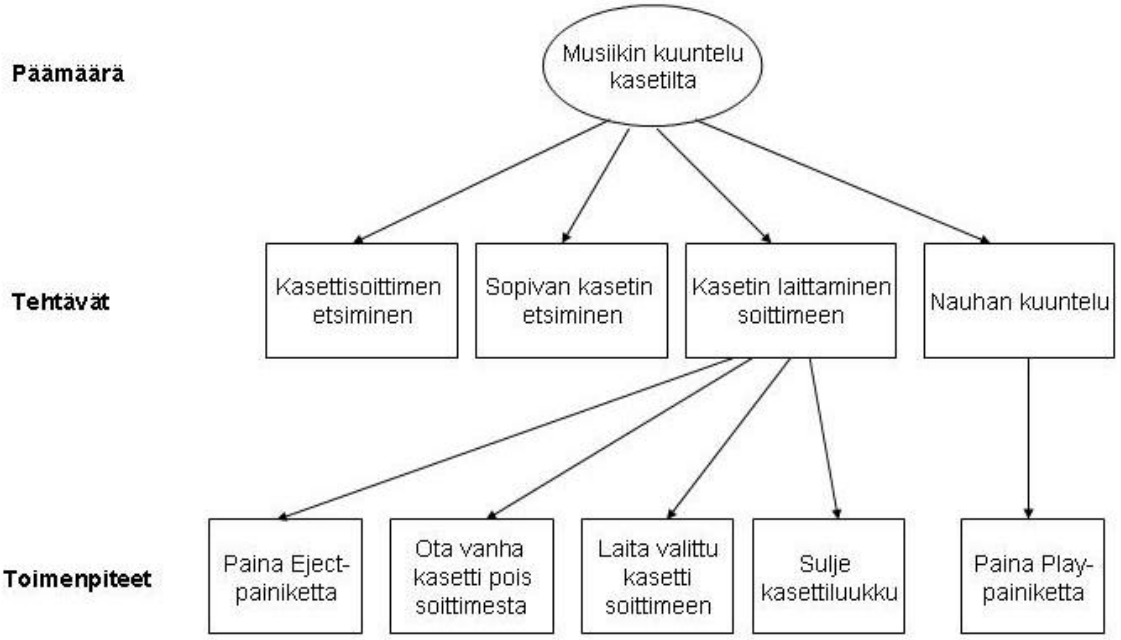
Kuva 10: Ihmisen vuorovaikutus teknologian kanssa [Iso02]

Tehtävä, jota kutsutaan myös sisäiseksi tehtäväksi (internal task), on jäsentynyt joukko aktiviteetteja, joissa toimet suoritetaan tietyssä järjestyksessä päämäärän saavuttamiseksi. *Toimenpide* (action) on puolestaan tehtävä, joka ei sisällä ongelmanratkaisua tai kontrollirakennetta [Pre94]. Toiminnan teorian käsitteillä (ks. kappale 5.1) on vastineensa tehtäväanalyysissä. Toiminnan teorian mukaan toiminnan käynnistää jokin tarve, tehtäväanalyysin puolella käytetään päämäärän käsitettä. Toiminnan teorian toimenpide-käsite viittaa tehtäväanalyysin tehtävä-käsitteeseen ja toiminnan teorian operaatio-käsite viittaa puolestaan tehtäväanalyysin toimenpide käsitteeseen. Toiminnan teorian toiminta-käsitteelle ei ole vastinetta tehtäväanalyysissä (Taulukko 1).

Toiminnan teorian käsite	Tehtäväanalyysin käsite
Tarve (need)	Päämäärä (goal)
Toiminta (activities)	
Toimenpide (action)	Tehtävä (task)
Operaatio (operation)	Toimenpide (action)

Taulukko 1: Toiminnan teorian ja tehtäväanalyysin käsitevastaavuudet

Esimerkissä (Kuva 11) päämääränä on kuunnella musiikkia. Tähän päämäärään pääsemiseksi on suoritettava lukuisia tehtäviä. On etsittävä kasettisoitin ja sopiva kasetti, laitettava kasetti soittimeen ja kuunneltava nauhaa. Lisäksi esimerkiksi tehtävä "kasetin laittaminen soittimeen" sisältää toimenpiteitä, kuten Eject-painikkeen painaminen.



Kuva 11: Päämäärä ja siihen liittyviä tehtävi ä ja toimenpiteitä (Soveltaen [Pre94])

Tehtäväanalyysin suorittamiseen on olemassa useita menetelmiä. Haastattelut ja systemaattinen havainnointi ovat esimerkkejä hyvistä analysointitavoista. Haastateltaessa käyttäjiä, on

tärkeää pyytää esimerkkejä heidän työskentelytavoistaan ja välineistään, eikä tyytyä vain keskustelemaan abstraktilla tasolla [Nie93]. Myös roolipelit tai käyttäjän ääneen ajattelu voivat olla tehokkaita tapoja saada tietoja käyttäjien todellisista toimintatavoista [Kuj02]. Toisaalta käyttäjän tehtävämallin yksilöiminen voi toimia esimerkiksi lähteenä käyttöliittymän kielikuviin. Käyttäjien strategiat ja pätevät käyttäjät voivat antaa vihjeitä, millaisia toimintoja uuden systeemin tulisi tukea. Lisäksi nykyisen tilanteen huonojen puolien kartoittamisen avulla voidaan saada aikaan parannuksia uuteen systeemiin.

Tyypillisesti tehtäväanalyysistä saadaan tulokseksi:

- lista asioista, joita käyttäjät tahtovat systeemin avulla saavuttaa,
- kaikki se tieto, jota tarvitaan saavuttamaan käyttäjän päämäärät (esiehdot),
- ne tehtävät ja toimenpiteet, jotka on suoritettava,
- riippuvuudet näiden tehtävien ja toimenpiteiden välillä,
- kaikki tuotettavat tulosteet ja raportit,
- tulosten laatu- ja hyväksyttävyyskriteerit, sekä
- se tieto, jota vaihdetaan toisten kanssa samalla kun tehtävää suoritetaan.

7.1.3 Toimintatarinat

Ihmiset toimivat tavallisesti suunnitelmiensa tai tilanteen mukaan. Tilanne kuitenkin vaikuttaa myös suunnitelmiin. Tilannetta kutsutaan *kontekstiksi* ja se määrää ihmisen toimintaa, tavoitteita sekä havaintoja ja niistä tehtyjä tulkintoja. Käyttötilanteita kerätään, mallinnetaan ja tarkastetaan toimintatarinoiden avulla [SiK02]. *Toimintatarina* (scenario) on siis kuvaus käyttäjän vuorovaikutuksesta systeemin kanssa [Inf02]. Tarinoista saadaan esille toiminnot, joita tuotteen avulla täytyy saada aikaiseksi. Toimintatarinoita käytetään suunnittelun aikana varmistamaan, että kaikki osallistujat ymmärtävät ja hyväksyvät suunnitteluparametrit, sekä määrittelemään tarkasti, millaisia vuorovaikutustapoja systeemin täytyy tukea [Inf02].

Ennen toimintatarinoiden kirjoittamista on kerättävä tietoa niistä tehtävistä, joita tuotteella tullaan tekemään. Tehtäväanalyysin menetelmillä saadut tiedot ovat sirpaleisia, eivätkä säily pitkään muistissa, ellei tietoa muuteta helpommin jaettavaan ja säilytettävään muotoon. Kertomukset ovat ihmiselle luonnollinen tapa muistaa asioita, siksi toiminta- ja käyttötarinat toimivat hyvin [SiK02]. Sen jälkeen kun kerätystä materiaalista on löydetty mielenkiintoiset tapahtumat, niistä muodostetaan lyhyitä kertomuksia, toimintatarinoita. Toimintatarinat voi-

daan yhdistää myös käyttötapauksiin, jotka kuvaavat käyttäjän vuorovaikutusta systeemin kanssa teknisellä tasolla [Inf02], käyttötapauskaaviot auttavat erityisesti ohjelmistosuunnittelijoita käytettävän tuotteen rakentamisessa. Toimintatarinoiden etuna käyttötapauksiin verrattuna on se, että toimintatarinat ovat ymmärrettäviä myös sellaisille ihmisille, joilla ei ole teknistä tietämystä.

7.1.4 Käyttötarinat

Käyttötarinat (product scenario) ovat kertomuksia, jotka esittelevät käyttäjän toimintatavan uudella tuotteella [Sin02]. Käyttötarinat muunnetaan myöhemmin malleiksi, kuvauksiksi, protoiksi ja lopulta tuotteiksi. Kun käyttäjällä on tavoitteita, toiminta jakaantuu alatavoitteisiin. Käyttäjän tavoitteiden hahmottamisen avulla löydetään käyttöliittymätasolle helpommin oikeat termit ja toiminnot sekä pystytään valitsemaan, mitkä toiminnot tulee olla helposti saavutettavissa missäkin tuotteen tilassa. Käyttötarinoihin kerätään alatavoitteet, tavoitteen muodostumisen aikaansaavat tekijät sekä tarvittava palaute, josta tiedetään siirtyä seuraavaan alatavoitteeseen [SiK02]. Käyttötarinat ovat hyvä pohja myös testaukselle, sillä niiden pohjalta voidaan muodostaa testitapauksia.

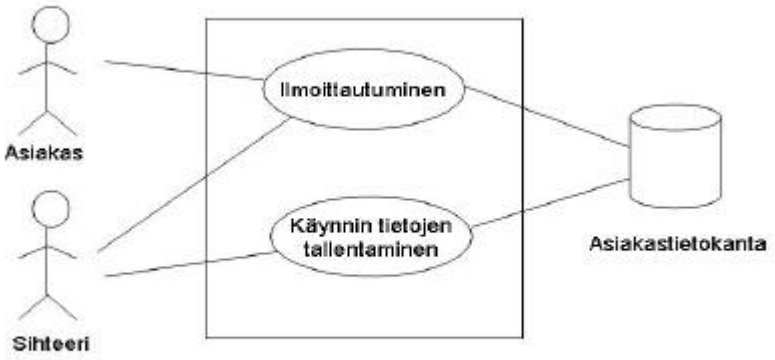
Toiminta- ja käyttötarinoita tehtäessä on syytä kiinnittää huomiota sekä käytön kokonais- että osatavoitteiden muodostumiseen sekä riittävään palautteeseen. Palautteen avulla käyttäjä tietää, että hänen kannaltaan oleellinen tavoite tai järjestelmän rakenteen osatavoite on saavutettu [SiK02].

7.1.5 Käyttäjien välinen vuorovaikutus

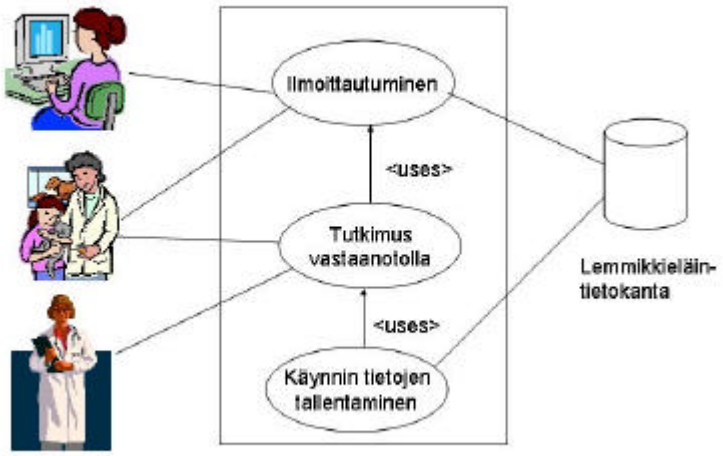
Vaatimusmäärittelyvaiheessa kuvataan usein käyttötapauskaaviolla tai toiminta- ja käyttötarinoilla tietokoneen ja käyttäjän välistä vuorovaikutusta. Myöskään ihmisten välistä vuorovaikutusta ei tulisi unohtaa, sillä työelämässä yhä suurempi osa työstä tehdään jonkinlaisessa vuorovaikutuksessa toisten ihmisten kanssa, ja mitä erilaisempien töiden yhteydessä puhutaan tiimeistä ja ryhmätyöstä. Suunniteltavan ohjelmiston on useimmiten tarkoitus tukea nykyisiä työskentelytapoja tai esimerkiksi mahdollistaa uusien vuorovaikutuksellisempien työskentelytapojen käyttäminen, joten ihmisten välinen vuorovaikutus on otettava jo vaatimusmäärittelyvaiheessa huomioon. Jos käyttötapauksissa kuvataan vain tietokoneen ja ihmisen välinen vuorovaikutus, käyttötapaukset voivat olla hyvinkin vajavaisia ja sen vuoksi niitä voi olla vaikea

tarkastaa ja ymmärtää. Tämän vuoksi valmistuva tuote ei välttämättä tue yrityksen työskentelytapoja, eikä sitä sen vuoksi pidetä käytettävänä käyttöympäristössään.

Kuvassa 12 esitetään käyttötapauskaavio, johon on tyypilliseen tapaan kuvattu vain se, kuka tekee ja mitä järjestelmän avulla. Käyttötapauskaavio on hyvin yksinkertainen ja vajavainen, mitä ilmoittautumisen ja käynnin tietojen tallentamisen välillä tapahtuu? Kuvassa 13 puolestaan kuvataan sama käyttötapauskaavio niin, että siihen on lisätty myös *toimintatapaukset*, eli ihmisten välinen vuorovaikutus. Kaaviosta tulee heti ymmärrettävämpi. Lisäksi kaavion kuvaamisessa on käytetty kuvia ja kuvaavampia termejä ymmärtämisen helpottamiseksi [Eer03].



Kuva 12: Käyttötapauskaavio ilman ihmisten välisen vuorovaikutuksen kuvaamista



Kuva 13: Ihmisten välisen vuorovaikutuksen kuvaava käyttötapauskaavio

7.1.6 Käyttöympäristön kuvaus

Ohjelmiston käytettävyyden kannalta ei ole merkityksetöntä, millaisessa kontekstissa ohjelmistoa käytetään (Kuva 14). *Käyttöympäristö* (context of use) on kuvaus niistä todellisista olosuhteista, joissa tuotetta käytetään. Käyttöympäristön osia ovat mm. fyysinen, sosiaalinen ja organisaatioympäristö [Sin02]. Ohjelmistolle asetetaan aivan erilaisia vaatimuksia, jos sitä käytetään lämpimässä konttorissa hyvässä valaistuksessa, tai jos sitä käytetään kovassa pakkasessa epätasaisessa maastossa liikkuvassa metsäkoneessa paksu hansikas kädessä. On olemassa myös ohjelmia, joita käytetään vain hätätilanteissa, jolloin ihmisten mahdollinen ahdistuneisuus ja jännittynyt mielentila on otettava huomioon jo ohjelmistoa suunniteltaessa. Nykyaikana yhä useammin samaa ohjelmistoa tai tuotetta käytetään useammissa käyttöympäristöissä. Kannettavat tietokoneet, kämmenmikrot jne. ovat mahdollistaneet etätyöskentelyn kotona tai työskentelemisen esimerkiksi työmatkoilla lentokoneessa tai junassa. Myös muuttuvat käyttöympäristöt tulisi ottaa huomioon käytettävyyttä suunniteltaessa.

Kuva 14: Konteksti vaikuttaa tuotteen käytettävyyteen

Jokaisella maalla ja kansakunnalla on oma kulttuurinsa. Myös työpaikoilla on omat kulttuurinsa, joiden mukaan työntekijät tietoisesti ja tiedostamattaankin toimivat. Yrityksellä voi olla esimerkiksi tietyt pukeutumissäännöt tai kirjoittamattomat säännöt, miten kahvitauolla toimi-

taan. Yrityskulttuuri on osa työntekijöiden toimintaympäristöä, ja siten sillä on oma vaikutuksensa myös toimintaprosesseihin ja ohjelmistojen käytettävyyteen tässä ympäristössä.

Työpaikan kulttuuria ilmentävät fyysiset ja materiaaliset artefaktit, kielelliset symbolit ja työskentelykäytänteet [GrK91]. Yrityksen ydinarvot havainnollistuvat fyysisten ja materiaalisten artefaktien, kuten työskentely-ympäristön ulkoisten piirteiden, varustelun, työskentelyvälineiden ja erilaisten pukeutumissääntöjen välityksellä. Nämä ydinarvot ilmaistaan kielellisten symbolien, esimerkiksi sanontojen tai ammattislangin ja kielikuvien avulla. Yrityksen työskentelykäytänteet, eli erilaiset rutiinit, yhteistyömallit, eleet ja rituaalit puolestaan dramatisoivat ydinarvot. Jotta valmistettavan tuotteen tai ohjelmiston tuleva todellinen käyttöympäristö ei jäisi liian vähälle huomiolle, se on syytä kuvata riittävän tarkasti vaatimusmäärittelyvaiheessa, sillä toimintaympäristön huomioiminen helpottaa käyttöyhteyteensä sopivan tuotteen rakentamisessa. Työympäristön kuvaus voidaan liittää osaksi toimintatarinoita, mutta siihen on kiinnitettävä tiedonkeruuvaiheessa erikseen huomiota.

Yrityksen kulttuurin selvittäminen on usein hankalaa, sillä kulttuuri näyttäytyy ulkopuoliselle erilaisena kuin kulttuuriin kuuluvalle. Jotta ulkopuolinen saisi työpaikan kulttuurista todellisen kuvan, siihen olisi päästävä perehtymään kunnolla. Kun ulkopuolinen on päässyt kulttuuriin syvemmälle ja saanut siitä tietoa, hän on samalla jo keskellä tähän kulttuuriin sosialisoitumisprosessia. Tämän prosessin seurauksena hän on omaksunut kulttuurille tyypillisiä normeja ja toimintatapoja, eikä enää osaa itsekään kiinnittää niihin huomiota, vaan niistä on tullut tiedostamattomia. Vaarana on, että kulttuuria tutkimaan tullut ei enää itsekään erota metsää puilta [GrK91].

7.1.7 Käytettävyystavoitteiden luominen

Käytettävyyssuunnittelun näkökulmasta vaatimusmäärittelyvaihe päättyy käytettävyystavoitteiden luomiseen [Vuo96]. Käytettävyydelle on asetettava tavoitearvot samalla tavalla kuin kehitettävän tuotteen teknisemmillekin piirteille asetetaan, jotta käytettävyys ei jäisi vain puheiden tasolle [Nie98]. Tehtävää työtä voidaan ohjelmistotuotantoprosessin eri vaiheissa peilata vaatimusmäärittelyvaiheessa asetettuihin käytettävyysvaatimuksiin. Testausvaiheessa viimeistään testataan, toteutuvatko käytettävyysvaatimukset, jotka liittyvät yleensä tiettyyn käyttötapaan tai käyttäjäryhmään [Ova02].

Tavoitteiden asettelussa voidaan käyttää apuna erilaisia käytettävyysmalleja, jotka määrittelevät keskeisiä käytettävyyden piirteitä (ks. kappale 2), esimerkiksi opittavuus, muistettavuus, miellyttävyys tai käytön virheettömyys [Nie98]. Jotta käytettävyystavoitteista olisi hyötyä, niiden täytyy olla mitattavia tai ainakin selkeästi havaittavia. Vaatimuksena voi esimerkiksi olla, että ohjelman pääpiirteiden opetteluun ei kulu tuntia enempää aikaa. Käytettävyystavoitteisiin tulisi liittää tiedot mittayksiköstä ja mittaustavasta, joiden avulla tulokset voidaan todistaa. Lisäksi tulisi mainita mittarin minimiarvo, jota ei saa alittaa, tavoitearvo, johon pyritään, maksimiarvo, johon voidaan päästä ja tieto nykytilanteen tasosta. Käytettävyystavoitteisiin olisi hyvä liittää myös perustelut sille, miksi tietyt mittarit ja arvot on valittu. Periaatteena tulisi olla, että ohjelmiston kehittämistä ei lopeteta, ennen kuin asetetut vaatimukset on täytetty. Käytettävyystavoitteiden luominen ja tavoitteiden asettaminen on usein vaikeaa ensimmäisellä kerralla, sillä käytettävyystavoitteiden asettaminen vaatii käyttäjä- ja tehtäväkuvausten suhteellisen tarkkaa analyysiä [Vuo96].

7.2 Analyysi

Analyysivaiheessa mallinnetaan toteutettavan järjestelmän ongelma-alue vaatimusmäärittelyn pohjalta, sekä mietitään haasteet ja ratkaisumahdollisuudet. Analyysivaiheen alussa on tärkeää rajata ongelma-alue selvittämällä, mitkä asiat kuuluvat systeemiin ja mitkä asiat rajataan systeemin ulkopuolelle (ks. kappale 6.2). Lisäksi tarkastellaan, millaisten muiden systeemien, tahojen tai ihmisten kanssa systeemi on vuorovaikutuksessa, ja mitä tietoa näiltä naapurisysteemeiltä saadaan ja mitä tietoa systeemi niille tuottaa. Analyysivaiheessa pyritään vastaamaan kysymykseen "mitä olemme lähdössä tekemään".

7.2.1 Käsiteanalyysi

Ohjelmistotuotantoprosessin alkuvaiheessa on tärkeää tehdä käsiteanalyysiä, sillä usein ohjelmistosuunnittelijat eivät ole sen kohdealueen asiantuntijoita, johon järjestelmää suunnitellaan. *Käsiteanalyysin* aikana määritetään järjestelmän kannalta keskeisiä käsitteitä ja selvitetään niiden väliset suhteet. Analyysiprosessin aikana myös varmistetaan, että kaikki osapuolet puhuvat samoista asioista samoilla nimillä. Käytettävistä käsitteistä voidaan piirtää esimerkiksi tietokantasuunnittelun puolelta tuttuja ER- (Entity Relationship) eli käsitemalleja tai *käsitteellisen tason luokkakaavioita* [FoS97]. Kaavioiden ideana on yhdistää yhteen kuuluvat käsitteet. Järjestelmän kannalta keskeisistä käsitteistä kannattaa koota sanasto, jossa luetellaan ja

määritellään keskeiset käsitteet. Käsiteanalyysi on oivallinen keino minkä tahansa asian jäsentämisessä, sillä se selkiyttää ja jäsentää käytettävää sanastoa [Tik91].

Käsiteanalyysin avulla saatuja tietoja voidaan käyttää hyväksi esimerkiksi käyttöliittymäsuunnittelun aikana, jotta käyttöliittymään osataan valita käyttäjille mahdollisimman selkeät käsitteet. Käsiteanalyysiä kannattaa tehdä suunnittelijoiden ja käyttäjien välisenä ryhmätyönä. Käyttäjät tuntevat parhaiten kuvattavan systeemin ja käytettävän termistön, suunnittelijat tuntevat puolestaan analyysin työmenetelmät [Tik91].

7.2.2 Käytettävyysskenaariot

Eri *asianosaisilla* (stakeholder), eli esimerkiksi loppukäyttäjillä, kehittäjillä ja ylläpitäjillä on erilaisia järjestelmän käytettävyyteen liittyviä toiveita ja käytettävyyteen liittyviä skenaarioita. Käytettävyysskenaariot (ks. Liite 1) eivät ole sama asia kuin käytettävyystavoitteet. Käytettävyystavoite (ks. kappale 7.1.7) on vaatimusmäärittelyvaiheen lopuksi asetettava mitattava ja todistettavissa oleva käytettävyyteen liittyvä tavoite. *Käytettävyysskenaarioiden* (usability scenarios) voidaan puolestaan ajatella olevan käytettävyyteen liittyviä näkökulmia, joiden avulla kyseisen järjestelmän kannalta keskeisimmät käytettävyystavoitteet voidaan täyttää [BaB01]. Tämän vuoksi vaatimusmäärittelyvaiheessa saatujen tietojen pohjalta olisi analyysivaiheessa syytä pohtia, mitkä ovat suunniteltavan ohjelman kannalta keskeisimmät ja oleellisimmat käytettävyysskenaariot ja kuinka ne tulisi huomioida ja toteuttaa ohjelmistotuotantoprosessin myöhemmissä vaiheissa. Eri skenaarioiden huomioiminen voi tapahtua hieman eri vaiheissa ohjelmistotuotantoprosessia. Skenaarioiden huomioimiseksi on olemassa menetelmiä sekä perinteisen ohjelmistotuotantoprosessin menetelmien joukossa että arkkitehtuurimallien puolella.

7.3 Arkkitehtuurisuunnittelu

Ohjelmistoarkkitehtuuri on kuvaus järjestelmän alisysteemeistä ja komponenteista, sekä niiden välisistä suhteista [BuM01]. Arkkitehtuuri on samalla tavalla suunniteltavan systeemin eräs näkökulma kuin näyttöjen tiedot tai järjestelmän toiminnallisuuskin [BaB01]. *Arkkitehtuurisuunnittelu* on pääasiassa järjestelmän eri osien välisen työnjaon ja rajapintojen suunnittelua [HaM01]. Suunnittelun avulla pyritään siihen, että ohjelmiston alisysteemien ja komponenttien välillä olisi mahdollisimman vähän riippuvuuksia, jotta järjestelmän ylläpito ja kehit-

täminen osasia vaihtamalla tai muuttamalla olisi helpompaa. Suunnittelun tavoitteita ovat selkeys, ymmärrettävyys, tehokkuus, luotettavuus, ylläpidettävyys ja siirrettävyys. Kirjallisuudessa tästä listasta jätetään usein käytettävyys pois, mutta miksei arkkitehtuurisuunnittelun tavoitteena voisi olla myös ohjelmiston parempi käytettävyys?

Monet arkkitehtuurisuunnittelun aikana tehdyt päätökset vaikuttavat olennaisesti myös järjestelmän käytettävyyteen. Käytettävyysongelmia voidaan ennaltaehkäistä käyttämällä hyväksi havaittuja arkkitehtonisia ratkaisutapoja. Vaikka järjestelmän käyttöliittymä ja toiminnallisuus olisi suunniteltu erittäin hyvin, järjestelmän käytettävyys voi silti olla vaarassa, jos taustalla oleva arkkitehtuuri ei tue esimerkiksi muunneltavuutta [BaB01].

Arkkitehtuurin avulla ohjelmistosuunnittelijat voivat analysoida, miten hyvin suunniteltu ohjelmisto täyttää sille asetetut vaatimukset, erityisesti ei-toiminnalliset vaatimukset, ja tavoitteet [Bos00]. Lisäksi suunnittelijat pystyvät miettimään rakenteellisia ratkaisuja niin aikaisessa vaiheessa, että muutosten tekeminen on useimmiten vielä helppoa. Arkkitehtuurisuunnittelun avulla voidaan myös vähentää ohjelmistokehityksen riskejä, ja arkkitehtuurikuvaus mahdollistaa kaikkien eri osapuolten välisen keskustelun rakennettavasta järjestelmästä. Arkkitehtuuri ei yleensä ole kertakäyttöinen, vaan samaa arkkitehtuuria voidaan käyttää useissa samankaltaisissa järjestelmissä [Jär02].

Yleisellä tasolla arkkitehtuurisuunnittelun avulla pyritään ihmiselle hankalan monimutkaisuuden hallintaan. Ohjelmistolle asetetut vaatimukset heijastuvat voimakkaasti arkkitehtuuriin. Määriteltyjen vaatimusten siirtäminen arkkitehtuuriin on sovelluskehityksen vaikein toiminto, sillä se vaikuttaa olennaisesti järjestelmän toimivuuteen [Bos00].

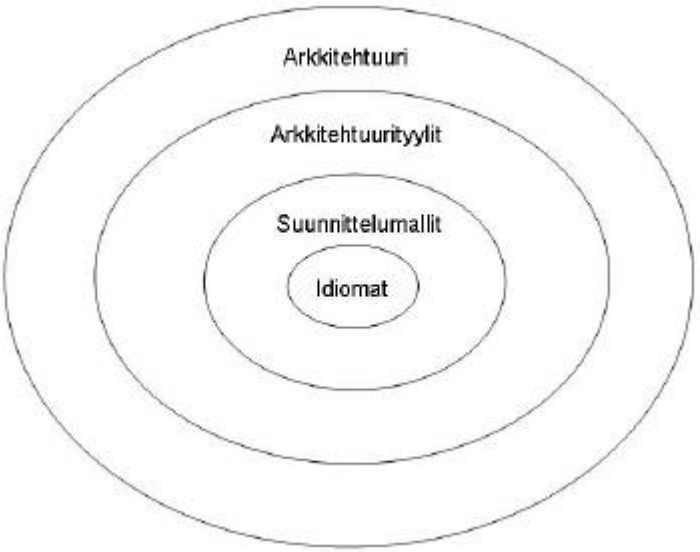
7.3.1 Arkkitehtuurin rakentuminen

Arkkitehtuuri rakentuu kolmeen eri kategoriaan kuuluvista arkkitehtuurimalleista (Kuva 15): arkkitehtuurityyleistä, suunnittelumalleista ja idiomista [BuM01]. *Arkkitehtuurityyli* (architectural style) ilmaisee järjestelmän perustavanlaatuisen rakenteellisen mallin. Se määrittelee arkkitehtuuriperheen, tarjoaa kokoelman ennaltamääriteltyjä komponentteja, jotka toteuttavat järjestelmän toiminnallisuuden ja kuvaa joukon *liittimiä* (connectors), joiden avulla komponentit kommunikoivat, ovat vuorovaikutuksessa ja toimivat yhteistyössä. Lisäksi arkkitehtuurityyli kuvaa joukon *rajoitteita*, jotka määrittelevät, kuinka järjestelmän rakennuselement-

tejä voidaan käyttää. Arkkitehtuurityyli kuvaa myös semanttisia malleja, jotka auttavat suunnittelijaa ymmärtämään järjestelmän ominaisuuksia osatekijöiden tunnettuja ominaisuuksia analysoimalla.

Erilaisten tyylien tuntemus yksinkertaistaa arkkitehtuurisuunnittelua, mutta arkkitehtuurityylin valitseminen on myös tärkeä suunnittelupäätös järjestelmää kehitettäessä. Kuhunkin järjestelmään sopivimman arkkitehtuurityylin valintaan vaikuttavat järjestelmälle asetetut laatuvaatimukset. Rakenteellisesti saman ongelman voi ratkaista useampikin arkkitehtuurityyli, mutta valintaan vaikuttaa se, mitkä laatuominaisuudet katsotaan järjestelmän kannalta kriittisiksi.

Suunnittelumalli (design pattern) kuvaa yleisen, ja käytössä hyväksi todetun ratkaisun johonkin tietyssä kontekstissa esiintyvään ongelmaan (ks. kappale 7.8.1). Suunnittelumallit tarkentavat arkkitehtuurityyliä, mutta niidenkin tulisi olla ohjelmointikieliriippumattomia. *Idiomat* (idioms) ovat matalimman tason malleja, jotka ovat ohjelmointikieliriippuvaisia. Idioma kuvaa, kuinka tietty komponenttien piirre tai niiden välinen suhde toteutetaan tietyn ohjelmointikielen jotakin ominaisuutta hyödyntäen.



Kuva 15: Arkkitehtuurin rakentuminen

Eri arkkitehtuurimallit ovat kiinteässä yhteydessä toisiinsa ja niiden voidaan ajatella olevan osittain myös sisäkkäisiä rakenteita (Kuva 15). Arkkitehtuuri kuuluu johonkin arkkitehtuuri-

perheeseen, eli edustaa jotakin arkkitehtuurityyliä. Arkkitehtuurityylejä ovat esimerkiksi kerrosarkkitehtuuri (layers pattern), putket ja suodattimet (pipes and filterns pattern) sekä liitutauluarkkitehtuuri (blackboard pattern) [BuM01]. Toisaalta jonkin järjestelmän kokonaisarkkitehtuuri voi koostua useista eri arkkitehtuurityyleistä. Jokin järjestelmän osa voi olla toteutettu kerrosarkkitehtuurilla ja toinen putket ja suodattimet -arkkitehtuuria hyödyntämällä. Tietyssä arkkitehtuurityylin toteutuksessa on voitu käyttää hyväksi erilaisia suunnittelumalleja ja suunnittelumallit on puolestaan toteutettu jollakin ohjelmointikielellä idiomia käyttäen.

7.3.2 Arkkitehtuurin kuvaaminen

Käytettävyyden kannalta tilanteeseen sopivimman arkkitehtuurityylin valinta on kriittinen tekijä. Kuhunkin tilanteeseen sopivin arkkitehtuurityyli riippuu tavoitteista sekä järjestelmän rakenteesta ja ominaisuuksista. Toisissa järjestelmissä voidaan esimerkiksi pitää tarvittavien resurssien vähäistä määrää tärkeämpänä kuin suoritusnopeutta tai yksittäisellä näytöllä navigoitavuutta tai päinvastoin. Valittu arkkitehtuuri tulisi kuvata riittävällä tarkkuudella, jotta siitä olisi hyötyä seuraavissa ohjelmistotuotantoprosessin vaiheissa. Arkkitehtuurikuvauksesta voidaan käyttää myös nimitystä *järjestelmämalli*. Siinä kuvataan komponenttien lisäksi myös niiden väliset yhteydet ja mahdolliset reunaehdot. Järjestelmämallin avulla voidaan ratkaista esimerkiksi sovelluksen samanaikaiseen käyttöön, laiteriippumattomuuteen tai järjestelmän tilan tarkkailuun liittyviä ongelmia.

7.4 Komponenttien suunnittelu

Komponentti on ohjelmiston muista komponenteista riippumaton uudelleenkäytettävä ohjelmistoelementti, jolla on hyvin määritelty, toteutuksesta erillään oleva rajapinta, jonka kautta sitä käytetään. Komponenttien mustalaatikkoajattelun tavoitteena on piilottaa komponentin sisäinen rakenne komponentin käyttäjiltä ja tarjota ainoastaan joukko korkean tason rajapintoja komponenttien palveluiden kuvaamiseen [Käh00]. Sitä rajapintaa, jonka operaatiot komponentti toteuttaa, kutsutaan *tarjotuksi rajapinnaksi* (provided interface). *Kutsuttu rajapinta* (required interface) on puolestaan rajapinta, jonka operaatioita komponentti kutsuu [HeC01].

Komponenttien avulla on mahdollista päästä tehokkaampaan sovelluskehitykseen ja jo tehdyn kehityksen uudelleenkäyttöön. Tehokkaampi sovelluskehitys perustuu kolmeen seikkaan [Käh00]. Ensinnäkin itse tehtyjä komponentteja voidaan käyttää aiempaa paremmin uudel-

leen. Toisaalta komponentteja voidaan hankkia myös valmiina muualta, eikä kaikkea tarvitse toteuttaa itse. Kolmanneksi komponentti-infrastruktuuri tarjoaa palveluja, jotka nopeuttavat sovellustyötä merkittävästi. Komponentit määritellään useimmiten oliotekniikalla, mutta niiden toteutus ei välttämättä aina ole oliopohjainen.

Liiketoimintaprosessit (ks. kappale 5.2) muodostavat hyvän lähtökohdan komponenttirajojen löytymiselle. Jos useampi liiketoimintaprosessi tarvitsee jotakin tiettyä toiminnallisuutta, kannattaa tällainen toiminnallisuus useimmiten määritellä palveluksi, jonka toteuttaa erillinen komponentti. Tällaisessa tilanteessa komponentin rajapinta täytyy määritellä kaikkien sitä tarvitsevien prosessien näkökulmasta [Käh00]. Komponenttien oikeanlainen rajaaminen ja hyvä toteutus ovat käytettävyyden kannalta tärkeässä asemassa. Hyvä käytettävyys ja ohjelmiston ihmisten työn tukeminen parantavat liiketoimintaprosessia ja asiakkaan saamaa vastetta. Järjestelmän komponenttipohjainen toteutus helpottaa myös virheiden paikannettavuutta. Yhden komponentin sisälle tehdyt muutokset eivät säteile ympäri ohjelmistoa, jos rajapinta pysyy muuttumattomana. Komponenttipohjaisuus mahdollistaa myös osien vaihdon ja järjestelmän laajennettavuuden. Komponenttipohjaisuuden avulla saavutettava skaalautuvuus vaikuttaa myös osaltaan järjestelmän käytettävyyteen.

7.4.1 Erittely

Komponenttiteknologia käyttää hyväkseen erittely-menetelmää, jonka avulla voidaan käsitellä monimutkaisia ongelmia. *Erittelyssä* (separation) hankala ongelma pilkotaan pienempiin osiin. Sen jälkeen jokainen palanen ratkaistaan erillään ja lopuksi ratkaisu yhdistetään [BaB01]. Menetelmän onnistuminen riippuu siitä, kuinka helppoa yhdistäminen on. Yhdistäminen on helppoa, jos jokainen hajotetun ongelman palanen on suhteellisen itsenäinen. Jos taas tarvitaan suuri määrä viestinvälitystä erillisten osien välillä ongelman ratkaisemiseksi, erittelytekniikka on luultavasti ollut tilanteeseen sopimaton tai se käyttäminen on epäonnistunut.

Termiä *kytkentä* (coupling) käytetään kuvaamaan tarvittua osasten välisen yhteistyön ja viestinvälityksen määrää. Vähäinen kytkentä osoittaa, että erittelyn käyttäminen on onnistunut ja todellisen ongelman ratkaiseminen on helpottunut. Toimintojen kapselointi, tietojen erottaminen toiminnoista, tietojen erottaminen tietonäkymästä ja toimintojen valtuuttamisen erottaminen toteutuksesta ovat esimerkkejä eriasteisista erittelytekniikoista [BaB01].

Toimintojen kapselointi (encapsulation) on erittelytekniikoista alkeellisin [BaB01]. Menetelmässä toiminnallisuus kapseloidaan moduulin sisään ja ulospäin paljastetaan vain se, mikä on välttämätöntä toiminnallisuuden ja tulosten aikaansaamiseksi. Kapseloitu toiminnallisuus siis eristetään muusta toiminnallisuudesta.

Kapseloinnin avulla voidaan:

- muuttaa yksittäistä algoritmia muita algoritmeja muuttamatta,
- helpottaa sovellusten samanaikaisen käytön mahdollistamista,
- helpottaa aikaisempien tietojen uudelleenkäytön mahdollistamista,
- mahdollistaa käyttöliittymien muuntelu,
- helpottaa monien samanaikaisten toimintojen ja kokonaisvaltaisen etsimisen tukemista, sekä resurssien tarkistamista ja
- mahdollistaa unohdettujen salasanojen takaisinsaaminen.

Tietojen erottaminen toiminnosta (separating data from function) on menetelmä, joka sallii monien erillisten komentojen suorittamisen tietojoukossa (on set on data), tai yksittäisen komennon suorittamisen monissa erillisissä tietojoukoissa [BaB01]. Kun tätä menetelmää käytetään, tieto tai tietojoukko kapseloidaan komennosta tai komennoista. Tämä menetelmä soveltuu käytettäväksi tilanteissa, joissa joko komento- tai tietosarjat vaihtelevat. Tietojen erottaminen toiminnoista helpottaa tiedon koostamista, yhdenmukaisen toiminnan rakentamista näytöstä toiseen, näyttöjen tekemistä käyttäjälle helppopääsyisiksi sekä käyttäjän mielikuvien tukemista. Lisäksi menetelmä voi helpottaa virheettömyyden tarkistamista ja järjestelmän tilan tarkkailua.

Tiedon erottaminen tietonäkymästä (separating data from the view of that data) mahdollistaa erillisten näkökulmien laatimisen samalle tietojoukolle. Tietoa siitä, kuinka käyttäjät mahdollisesti haluavat tiedot nähdä, ylläpidetään erillisessä kokoelmassa. Menetelmän avulla voidaan esimerkiksi rajata joidenkin tietojen näkyvyyttä vain tietyille käyttäjille ja käyttäjäryhmille tai mahdollistaa käyttäjien nähdä tieto eri tavalla riippuen käytettävästä alustasta. Tämän menetelmän avulla voidaan siten tukea sovellusten kansainvälistä käyttöä rakentamalla esimerkiksi erikielisille omat käyttöliittymänsä. Toisaalta voidaan tukea myös käyttäjän aikaisempien tietojen hyväksikäyttöä, sillä paljon vastaavia sovelluksia käyttäneelle voidaan tehdä hieman monimutkaisempi käyttöliittymä tai tiedot voidaan muuten esittää näytöllä sillä tavalla, kun

käyttäjä on ennen tottunut näkemään. Tällä tavalla helpotetaan myös epätavallisessa käyttöympäristössä tapahtuvaa työskentelyä. Lisäksi menetelmä auttaa rakentamaan useiden näyttöjen välille yhdenmukaisen toiminnan ja tekemään näytöt helppopääsyisemmiksi käyttäjille [BaB01].

Toiminnan valtuuttamisen erottaminen suorituksesta (separating authoring from execution) on sovelluskehityksen peruselementti. Käyttäjillä voi esimerkiksi olla mahdollisuus määritellä järjestelmän toiminnallisuus järjestelmän sisällä. Toiminnallisuuden määritteleminen voidaan mahdollistaa käyttäjälle helpoiten valikkojen avulla, tai huomattavasti vaikeammin niin, että käyttäjä voi määritellä haluamansa asetukset erilaisten skripiten avulla. Käyttäjän tekemät määrittelyt voivat olla pysyviä tai olla voimassa vain senhetkisessä suorituksessa. Valtuutuksen erottaminen suorituksesta mahdollistaa siis tietojen ja komentojen koostamisen tilanteeseen tai käyttötarkoitukseen soveltuvaksi [BaB01].

7.4.2 Välillistäminen

Välillistäminen (indirection) on tapa vähentää kytkentää (coupling) erillisten elementtien välillä (ks. myös kappale 7.4.1). Välillistämistä voidaan hyödyntää sekä tietojen että funktioiden kohdalla [BaB01]. *Tietojen välillistämistä* (data indirection) tarvitaan, kun suora yhteys tiedon tuottajan ja tiedon tarvitsijan välillä aiheuttaa tiukan kytkennän näiden elementtien välille. Tällöin muutokset tiedon tarvitsijassa vaikuttavat tiedon tuottajaan ja päinvastoin. Tietojen välillistäminen voidaan toteuttaa esimerkiksi ilmoittautumismenetelmää hyödyntämällä. Ilmoittautumismenetelmässä erillinen komponentti jakaa tiedon sitä tarvitseville. Asiakas voi ilmoittaa välittäjälle, että on kiinnostunut tietynlaisesta tiedosta ja vastaavasti palveluntarjoaja ilmoittaa *välittäjälle* (broker), millaista tietoa se tarjoaa. Ilmoittautumisprosessi voidaan tehdä joko määrittelyvaiheessa tai ajoaikana. Sekä asiakkaalla että palveluntarjoajalla on suora yhteys välittäjään, mutta ei toisiinsa. Näin ollen uusia tiedon tarvitsijoita voi ilmoittautua välittäjälle tiedon tuottajan toiminnan häiriintymättä. Tietojen välillistäminen helpottaa järjestelmän arvioimista, sillä menetelmän avulla on helpompi esimerkiksi toteuttaa testipisteitä, joiden avulla järjestelmän arvioimista voidaan suorittaa. Lisäksi menetelmä helpottaa tietojen uudelleenkäyttöä ja käyttöliittymien muuntelua.

Funktioiden välillistäminen (function indirection) tarkoittaa sitä, että kahden erilaisen vaihtoehtoisesti tietyn palvelun toteuttavan metodin välille asetetaan välittäjäfunktio. Esimerkiksi

Abstract Factory (abstrakti tehdas) –suunnittelumalli käyttää tätä menetelmää. Sidonta palvelun pyytäjän ja palvelun tarjoajan välille voidaan tehdä joko ennen ajoa tai ajoaikana. Palvelun pyytäjä käyttää yksittäistä rajapintaa vuorovaikutukseen välittäjäfunktion kanssa ja välittäjäfunktio puolestaan kääntää tiedon sille vastaanottajalle, joka vaihtoehdoista valittiin. Myös funktioiden välillistämisen avulla voidaan parantaa järjestelmien käytettävyyttä, sillä menetelmä helpottaa sovellusten laiteriippumattomuuden ylläpitämistä, käyttöliittymien muuntelua kuhunkin tilanteeseen sopivaksi ja tukee monipuolista ja kokonaisvaltaista tiedonhakua tietovarastaosta.

7.5 Toimintosuunnittelu

Toimintosuunnitteluvaiheessa mietitään, miten vaadittu toiminnallisuus rakennetaan järjestelmään, eli kuvataan jokaisen moduulin looginen rakenne. Toimintosuunnitteluvaihe on kiinteässä yhteydessä käyttöliittymän suunnitteluun (ks. kappale 7.6). Ennen kuin moduulien rakennetta päästään suunnittelemaan, on tiedettävä, mitkä ovat ne toiminnot, jotka tuotteessa täytyy olla. Kappaleessa 7.5.2 kuvataan lyhyesti käyttäjäkeskeisen toimintosuunnittelun menetelmiä.

Voisi kuvitella, että toimintosuunnitteluvaihe on helppo toteuttaa, kun vaatimusmäärittelyvaihe on tehty kunnolla, mutta tähänkin vaiheeseen liittyy käytettävyyden näkökulmasta monia ongelmia, joita kuvataan tarkemmin kappaleessa 7.5.1. Lopuksi kuvataan arkkitehtuuriratkaisu, keskeytyksen sallivan ajoituksen –menetelmä (ks. kappale 7.5.3), jonka avulla voidaan ratkaista toimintojen vaatimien resurssien jakamiseen liittyviä ongelmia.

7.5.1 Toimintosuunnittelun ongelmia

Tuotteen suunnitteluprosessissa on monia ongelmia. Käytettävyyden kannalta ihanteellisinta olisi tuotteen luonnollinen kehittyminen [Nor91]. Luonnollinen kehitysprosessi on tyypillistä käsitöille: mallia testataan, ongelma-alueet havaitaan, ja sen jälkeen tuotetta kokeillaan ja muokataan niin kauan kuin se on tarpeellista, tai kunnes aika ja voimavarat loppuvat.

Ohjelmistosuunnittelussa luonnollista suunnitteluprosessia ei useinkaan ole mahdollista hyödyntää, sillä esimerkiksi laitteistot muuttuvat ja kehittyvät erittäin nopeasti. Eräs keino luonnollisen kehitysprosessin tukemiselle nykyteknologian kehityksen pyörteissä olisi edellisistä

versioista tai malleista saatujen kokemusten hyödyntäminen. Usein on kuitenkin niin, että uutta versioita tai mallia ryhdytään suunnittelemaan jo ennen kuin edellinen versio on ehtinyt kuluttajien käyttöön. Myöskään palautetta ei yleensä kerätä järjestelmällisesti, ja näin sen hyödyntäminen on hankalaa. Lisäksi markkinoilla menestyäkseen on erotuttava joukosta. Jokaisen mallin on oltava erilainen ja jokaisesta versiosta on löydyttävä uusia ominaisuuksia, vaikka jokainen uusi ominaisuus tai toiminto lisää tuotteen monimutkaisuutta ja heikentää samalla käytettävyyttä. Tuotteissa on myös oltava yksilöllisyyden leima, ja jos useat yritykset valmistavat samaa tuotetta, jokaisen on tehtävä se omalla tavallaan erottuakseen muista. Tämä tuottaa luonnollisesti hankaluuksia käyttäjälle, kun sama toiminto löytyy jokaisesta tuotteesta eri paikasta.

Uusien ominaisuuksien ja toimintojen lisääminen ja kehittäminen ei ole pelkästään suunnittelijoiden syytä, sillä ohjelmistojen ja tuotteiden ostajat haluavat paljon toiminnallisuutta tuotteisiinsa, vaikka he eivät usein hyödynnäkään suurinta osaa tuotteen tarjoamista mahdollisuuksista. Toimintojen määrästä on tullut yritysten kilpailukeino. Tuotteen toimittajalta vaatii siten suunnatonta rohkeutta kehittää turhia toimintoja sisältämätön tuote. Avain tällaisen tuotteen markkinamenestykseen löytyisi varmasti osaltaan helppokäyttöisyydestä. Pääsyy tuotteiden käytettävyysongelmiin on kuitenkin useimmiten siinä, että tuotteen suunnitellaan niin, että painopiste on tekn ologiassa eikä tuotteen käyttäjissä [HaR98].

Hyvän ja perusteellisen toimintojen suunnittelun avulla voidaan välttää monenlaisia käytettävyysongelmia. Käytettävyys ei liity pelkkään käyttöliittymään, vaan hyvin ja loogisesti toteutettu ohjelman toiminnallisuus on käytettävyyden kannalta vähintäänkin yhtä tärkeää kuin hyvä käyttöliittymä.

7.5.2 Käyttäjäkeskeinen toimintosuunnittelu

Toimintosuunnitteluvaiheessa on erittäin tärkeää pitää mielessä, ketä tuotteen todelliset käyttäjät ovat, ja millaisia heidän tuotteen avulla suorittamansa toimintokokonaisuudet ja tehtävät ovat. Toimintosuunnitteluvaiheessa voidaan kuvata käyttäjien toimintaa ja tehtäviä uuden tuotteen näkökulmasta esimerkiksi käyttösekvenssien ja käyttöhierarkioiden avulla. Molemmat menetelmät perustuvat vaatimusmäärittelyvaiheessa kirjoitettaviin käyttötarinoihin (ks. kappale 7.1.4).

Käyttösekvenssi (use sequnce) kuvaa käyttäjän uuden tuotteen avulla tekemän tehtäväketjun jonkin toiminnon aikaansaamiseksi [Kuj99]. Käyttösekvenssi sisältää käyttäjän tekemät toimenpiteet, päätöksentekotilanteet ja tuotteen toiminnan. Käyttösekvenssien avulla voidaan kuvata tuotteen toiminta yksityiskohtaisesti, tehtävien liittyminen toisiinsa ja näiden tehtävien yhteensopivuus. Käyttösekvenssejä voidaan hyödyntää myös käyttäjän tehtäväkokonaisuudet huomioivia käyttöohjeita kirjoitettaessa (ks. kappale 7.8.4).

Käyttöhierarkia (use hierarchy) puolestaan kuvaa käyttösekvenssin hierarkkisessa muodossa, osoittaen eri toimenpiteiden väliset suhteet [Kuj99]. Käyttöhierarkia auttaa suunnittelijaa ymmärtämään käyttäjän ja erilaisten toimenpiteiden tavoitteet käyttäjän kannalta. Käyttöhierarkia voi sisältää myös valikkorakenteiden ja ikonien suunnittelua. Tehtäväanalyysin avulla määriteltävät käyttäjän päämäärät, tehtävät ja toimenpiteet muistuttavat paljon käyttöhierarkiaa, jos tehtäväanalyysin tuloksista piirretään kuva (ks. kappale 7.1.2). Tehtäväanalyysin ja käyttöhierarkian ero on kuitenkin siinä, että tehtäväanalyysissä analysoidaan käyttäjien nykyisiä toimintatapoja, kun taas käyttöhierarkia kuvaa käyttäjän toimintaa ja suoritettavia toimenpiteitä rakennettavan uuden tuotteen avulla.

7.5.3 Keskeytyksen salliva ajoitus

Keskeytyksen salliva ajoitus (preemptive scheduling) on menetelmä, jonka avulla voidaan parantaa monien käytettävyysskenaarioiden toteutumista (Liite 1). Menetelmässä resurssit osoitetaan järjestelmän toiminnoille. Resurssit voivat olla fyysisiä, kuten muisti, tai loogisia, kuten jonot tai muut kokonaisuudet (entity). Ajoitus perustuu keskeytyksiin (preemptive) tai keskeytymättömyyteen (non-preemptive). *Keskeytyksen sallivassa* menettelytavassa resurssi voidaan ottaa joltakin toiminnolta pois ja *keskeytymättömässä* menettelytavassa toiminto voi pitää resurssin niin kauan kuin haluaa. Se, kumpi menettelytapa on soveltuvampi, riippuu resurssin tyypistä, tehtävien prioriteettijärjestyksestä, resurssin maksimoinnista ja resurssin odotusajan minimoinnista. Keskeytyksen sallivan ajoituksen -menetelmää voidaan arvioida esimerkiksi resurssin hyväksikäytön, pahimman tapauksen odotusajan tai keskimääräisen odotusajan perusteella.

Keskeytyksen salliva ajoitus mahdollistaa järjestelmän useat samanaikaiset toiminnot. Todellisuudessa toiminnot eivät ole samanaikaisia, vaan ne vain vaikuttavat samanaikaisilta *säikeiden* (thread) ansiosta. Useat samanaikaiset toiminnot toteutetaan useiden säikeiden avulla. Tietyl-

lä hetkellä säie voi olla joko aktiivisena tai odottamassa syötettä resurssilta. Tämä monien säikeiden käyttö on usein toteutettu keskeytyksen sallivan ajoituksen menetelmää käyttämällä.

Keskeytyksen sallivan ajoituksen avulla voidaan myös helpottaa komentojen perumisen ja virheettömyyden tarkistamisen toteuttamista järjestelmässä. Lisäksi menetelmän avulla voidaan helpommin mukauttaa järjestelmän toiminta käyttäjän toimintanopeuteen, ennustaa tehtävien kestoa, tarkkailla järjestelmän tilaa ja tarjota hyvä apu (ks. Liite 1) käyttäjälle.

7.6 Käyttöliittymän suunnittelu ja arviointi

Käyttöliittymä (user interface) on kaikki, mitä käyttäjä näkee tuotteesta tai järjestelmästä [Sin02]. Ihminen on vuorovaikutuksessa järjestelmän kanssa juuri käyttöliittymän kautta, ja sen vuoksi käyttöliittymä muovaa erityisesti ihmisen ensivaikutelmaa järjestelmästä. Pelkkä ohjelman toiminnallisuus ei riitä tekemään ohjelmistosta laadukasta, vaan myös käyttöliittymän on oltava kunnollinen ja käytettävä. Sama pätee myös toisinpäin, sillä hyvälläkään käyttöliittymällä ei voi korvata huonoa toiminnallisuutta.

Käyttöliittymän suunnittelu (user interface design) on prosessi, jossa suunnitellaan, miten ihminen ja tuote ovat vuorovaikutuksessa keskenään [Sin02]. Käyttöliittymiin voidaan suunnittelun aikana sisällyttää konkreettisten, havaittavien ominaisuuksien lisäksi myös sellaisia ominaisuuksia, joita ei välttämättä voi silmillä havaita, mutta joiden avulla käyttöliittymän laatua voidaan olennaisesti parantaa [SiK02]. Käyttöliittymän konkreettisia ominaisuuksia ovat esimerkiksi värit, pikakuvakkeet ja kirjasinten käyttö eli typografia. Abstrakteja, ei-havaittavia ominaisuuksia ovat mm. muunneltavuus, asioiden ryhmittely ja jäsentely sekä metaforien eli kielikuvien käyttö.

Käyttöliittymää suunniteltaessa ei saa myöskään unohtaa ohjelmistotuotantoprosessin aikaisemmissa vaiheissa selvitettyjä asioita, kuten tehtäväanalyysin avulla saatuja tietoja käyttäjistä, käyttäjien suorittamista tehtävistä ja käyttöympäristöstä tai käsiteanalyysissä määriteltyjä käsitteitä. Käyttöliittymän suunnittelu ei siten ole mikään oma erillinen projektinsa, vaan se nivoutuu kiinteästi ohjelmistotuotantoprosessin muihin vaiheisiin.

7.6.1 Tyylioppaan laatiminen

Tyyliopas, eli käyttöliittymästandardi on hyvä apuväline käyttöliittymän suunnittelijoille. *Tyylioppaalla* (user interface guidlines) tarkoitetaan sääntöjoukkoa, jota yrityksen tuotesuunnittelussa on käytettävä. Tyyliopas kuvaa yleensä eri käyttöliittymäelementtien käyttötavan ja ottaa usein kantaa myös ulkoasuun [Sin02].

Tyylioppaan laatiminen on kohtuullisen suuritöinen hanke, mutta useimmiten sen tekeminen kannattaa. Sen laatii yleensä tuotteen kehittäjäorganisaatio, mutta sen voi tehdä myös tuotteita ostava organisaatio liittääkseen sen vaatimusmäärittelyyn. Tällä tavalla useilla toimittajilla tietojärjestelmiään teettävä yritys voi varmistaa tuotteidensa yhdenmukaisuuden. Mikäli laadittua tyyliopasta noudatetaan, kaikista järjestelmistä saadaan rakennettua yhdenmukaisesti ja johdonmukaisesti toimivia, ja näin eri järjestelmien opetteluun ei kulu niin paljon aikaa. Tyylioppaan avulla voidaan myös poistaa suurin osa tavallisimmista käytettävyysongelmista. Lisäksi tyyliopas helpottaa ja nopeuttaa järjestelmien kehittämistä, sillä suunnittelijoiden ei tarvitse miettiä kerran keksittyjä asioita enää uudelleen, vaan he voivat keskittyä sovelluksen toiminnallisuuteen [Sin98].

Tyyliopas tulisi rakentaa yrityksessä käytössä olevan käyttöjärjestelmän hengessä. Opasta tehtäessä tulisi ottaa huomioon myös viralliset standardit. Oppaaseen kannattaa laatia mahdollisimman paljon esimerkkejä. On myös syytä ottaa huomioon, että hyvin toimiva tyyliopas on tehtävä huolellisesti ja ammattitaitoisesti, sillä huolimattomasti tehty tyyliopas voi olla jopa käytettävyydelle haitaksi. Toisaalta myös turhia säädöksiä tulee välttää ja tyytyä standardoimaan vain ne asiat, jotka on pakko standardoida [Sin98].

7.6.2 Asiantuntijamenetelmät

Usein käytettävyyden arviointi aloitetaan vasta siinä vaiheessa, kun järjestelmä on lähes valmis. Käytettävyyttä voidaan luotettavimmin arvioida todellisten käyttäjien kanssa, mutta käytettävyystestien järjestäminen on kallista, vie paljon aikaa ja vaatii todellisten käyttäjien osallistumista. *Asiantuntija-arvioinnit* ovat menetelmiä, joiden avulla käyttöliittymien suunnittelijat voivat itse arvioida ideoidensa käytettävyyttä ilman käyttäjiä tai muutaman käyttäjän kanssa [Rii02a]. Asiantuntijamenetelmät eivät korvaa varsinaisia käytettävyystestejä, mutta asiantuntijamenetelmillä arviointi voidaan aloittaa jo tuoteideoiden syntyessä ja huomata näin osa käytettävyysongelmista jo suunnittelun alkuvaiheessa. Asiantuntijamenetelmistä tunnetuimpia

ovat heuristinen arviointi ja kognitiivinen läpikäynti, joista jälkimmäinen on varsin vähän käytetty menetelmä. Muita asiantuntijamenetelmiä ovat mm. käyttötapaussimulointi ja käyttöliittymän läpikäynti.

7.6.2.1 Heuristinen arviointi

Heuristisessa arvioinnissa (heuristic evaluation) järjestelmän käyttöliittymän osat tarkastetaan erilaisten käytettävyysperiaatteiden eli heuristiikkojen avulla [Nie93]. *Heuristiset säännöt* ovat siis käytettävyystestien ja käytännön kokemuksen kautta saatuja periaatteita, jotka kuvailevat käyttöliittymän hyviä ja ei-toivottavia ominaisuuksia (ks. Liite 2). Heuristisen arvioinnin tavoitteena on löytää käyttöliittymästä ongelmakohtia. Arviointivaiheessa tutkittavan järjestelmän ei tarvitse olla valmis, vaan arviointiin riittävät järjestelmän ja käyttöliittymän suunnitelmat tai prototyypit. Tämän vuoksi arvioinnin tuloksia voidaan hyödyntää jo tuotteen suunnittelu- ja toteutusvaiheissa, jolloin muutosten tekeminen on vielä melko helppoa ja yleensä myös edullista [Rii02a].

Heuristinen arviointi on alun perin kehitetty käyttöliittymäsuunnittelijoiden omaan käyttöön, mutta tutkimukset osoittavat, että parhaimmat tulokset saavuttavat arvioijat, jotka hallitsevat sekä käytettävyysasiat että sovellusalueen. Toiseksi parhaisiin tuloksiin pääsevät pelkän käytettävyyden hallitsevat asiantuntijat. Pelkän sovellusalueen asiantuntijat menestyivät näistä kolmesta ryhmästä huonoiten. Yleensä eri ihmiset kiinnittävät huomiota hieman eri asioihin, joten kaikkein hedelmällisintä olisi käyttää useita arvioijia [Nie93].

Heuristinen arviointi voidaan jakaa neljään vaiheeseen. Ensimmäisessä vaiheessa kukin arvioija käy käyttöliittymän läpi yksinään pari kertaa, ensin yleisluontoisesti, sitten yksityiskohtaisemmin. Läpikäynnin avulla saadaan koottua lista havaituista ongelmista ja perustelut sille, miksi nämä ovat ongelmia. Toisessa vaiheessa erillisissä läpikäynneissä havaituista ongelmista kootaan yksi yhteinen lista, ja sen jälkeen ongelmat asetetaan tärkeysjärjestykseen vakavuuden perusteella. Vakavuuteen vaikuttavia tekijöitä ovat esimerkiksi se kuinka usein ongelma esiintyy, kuinka vaikea siitä on selvitä ja kuinka helposti ongelma voidaan välttää. Ongelmalistan kokoamisen jälkeen ryhmän olisi syytä vielä keskustella käyttöliittymän hyvistä puolista, koska muuten hyvät puolet jäävät helposti vaille minkäänlaista huomiota [Rii02a].

7.6.2.2 Kognitiivinen läpikäynti

Kognitiivisen läpikäynnin (cognitive walkthrough) avulla tutkitaan järjestelmän opittavuutta suorittamalla järjestelmään liittyviä tyypillisiä tehtäviä esimerkiksi prototyypin tai käyttöliittymän järjestelmäkuvauksen avulla. Järjestelmän ei siis tarvitse olla vielä valmis menetelmää hyödynnettäessä ja tämän vuoksi sitä voidaan soveltaa jo tuotekehityksen alkuvaiheissa. Kognitiivinen läpikäynti keskittyy opittavuuden tarkasteluun, koska monet käyttäjät opettelevat käyttämään järjestelmiä ilman käyttöohjetta [Rii02a]. Menetelmä opastaakin analysoijia keskittymään käyttöliittymän yksityiskohtien ja ominaispiirteiden sijasta käyttäjän mentaalisiin prosesseihin [Rii00]. Menetelmää ei ole tarkoitettu käytettäväksi ainoana arviointikeinona, vaan muita menetelmiä täydentävänä arviointikeinona.

Kognitiivinen läpikäynti jakaantuu viiteen vaiheeseen: esiselvitys, ryhmän kokoaminen, kognitiivinen läpikäynti istunnossa, havaintojen tallentaminen sekä havaittujen virheiden ja puutteiden korjaaminen [Rii02a]. Esiselvitysvaiheessa selvitetään, ketkä ovat järjestelmän tyypillisimpiä käyttäjiä, millainen kokemus heillä on vastaavien järjestelmien käytöstä ja millainen tekninen osaaminen heillä on. Mikäli vaatimusmäärittelyvaiheessa on tehty tehtäväanalyysi, nämä tiedot ovat jo valmiina, eikä niitä tarvitse enää uudelleen tutkia.

Läpikäynnissä tulisi käydä läpi järjestelmän perustehtävät ja lisäksi jokin hieman monimutkaisempi tehtävä, joka vaatii perustehtävien yhdistelyä. Tehtävien ratkaisu täytyy miettiä vaiheittain käyttäjien osaamistason vaatimalla tarkkuudella. Läpikäynnin arviointi voidaan tehdä yksin tai ryhmässä, mutta parhaisiin tuloksiin päästään ryhmätyön avulla. Antoisimmassa ryhmässä on henkilöitä eri osaamisalueilta, kuten markkinoinnista, suunnittelusta, toteutuksesta, dokumentoinnista, koulutuksesta ja käytettävyydestä. Läpikäynnin aikana tutkitaan valitut tehtävät vaihe vaiheelta ja tehtävien suoristusta tarkkaillaan sen mukaan, miten järjestelmän suunnittelija on ajatellut tehtävän suoritettavaksi. Havaitut ongelmat kirjataan mustiin, mutta tehtävää jatketaan ongelmista huolimatta. Jälkitarkastelua varten läpikäynti-istunto olisi hyvä myös videoida. Viimeisessä vaiheessa ongelmat pyritään korjaamaan esimerkiksi poistamalla käyttäjän näkökulmasta ylimääräiset vaiheet tai antamalla vaiheisiin paremmat ohjeet.

7.6.2.3 Muita asiantuntijamenetelmiä

Käyttöliittymän arvioinnissa käytetään heuristisen arvioinnin ja kognitiivisen läpikäynnin lisäksi myös muita menetelmiä. *Käyttötapausten simuloinnissa* (usage simulation) asiantuntija

valitsee yhden käyttötapauksen kerrallaan, simuloi todellisen loppukäyttäjän toimintaa ja pyrkii ennustamaan mahdollisia ongelmakohtia. Asiantuntijan ei pidä pyrkiä toimimaan varsinaisena testikäyttäjänä eikä raportoida omia ongelmiaan, vaan hänen täytyy pyrkiä perehtymään järjestelmän toimintaan siten, että hän tietää, miten käyttötapaukset suoritetaan suunnitellulla tavalla. Asiantuntijan skeemat järjestelmän toiminnasta ovat todellisen loppukäyttäjän skeemoja täydellisemmät ja täsmällisemmät. Tämän vuoksi asiantuntija pystyy vertaamaan simuloidun käyttäjän oletettua tietämystä omaan kattavampaan tietämykseensä. Asiantuntija ei simuloi pelkästään käyttäjän mekaanisia toimintavaiheita (painikkeiden valinnat jne.), vaan myös hänen ajatustyötään [Laa02].

Käyttöliittymän läpikäynnissä (pluralistic usability walkthrough) on käytettävyystestin ja käytettävyysarvioinnin elementtejä [Rii00]. Läpikäyntipalaverissa on paikalla samanaikaisesti useampia käyttäjiä. Palaveria ohjaava käytettävyysasiantuntija esittää yhden käyttötapauksen kerrallaan ja jokainen käyttäjä merkkaa paperisiin näyttökuviinsa kohdan, josta seuraavaksi lähtisi etenemään. Merkkaamisen jälkeen valinnat ja valintojen syyt käydään läpi ja niistä keskustellen. Tämän jälkeen palaverin ohjaaja valitsee yhden käyttäjien esittämistä vaihtoehdoista ja kaikki lähtevät seuraamaan tätä yhteiseksi valittua polkua [Laa02]. Käyttöliittymän läpikäynti voidaan myös suorittaa erittäin aikaisessa vaiheessa suunnitteluprosessia, sillä pelkät kuvan järjestelmän näytöistä riittävät läpikäynnin toteuttamiseksi. Läpikäynti paljastaa käytettävyystestiä paremmin epävarmat valinnat, sillä tavallisessa käytettävyystestissä osa valinnoista voi osua oikeaan onnekkaalla arvauksella, eikä epävarmuutta siten huomata.

7.7 Tulosteiden käytettävyyden huomioiminen osaksi käytettävyyttä

Varsinaisen ohjelman käyttöliittymän käytettävyyden lisäksi nykyistä enemmän huomiota tulisi kiinnittää myös erilaisiin ohjelmistojen ja järjestelmien avulla tuotettaviin dokumentteihin ja muihin tulosteisiin. Esimerkiksi sähkö- tai puhelinlaskun tulisi olla sellainen, että alaa tarkemmin tuntematon kuluttaja voi tarkistaa laskunsa oikeellisuuden ja ymmärtää, mistä kaikesta hän joutuu maksamaan. Myös erilaiset lomakkeet ovat harmittavan usein todella vaikeaselkoisia ja hankalia käyttää ja täyttää. Samalla tavalla kuin käyttöliittymän arvioinnissa käytetään esimerkiksi heuristisen arvioinnin menetelmiä, myös tulosteiden käytettävyyttä tulisi kehittää, arvioida ja testata. Erittäin usein varsinaisen kehitettävän tuotteen asiakkaiden asiakkaat törmäävät tuotteen käytettävyyteen juuri tulosteiden välityksellä. Myös www-sivujen

käytettävyyteen tulee kiinnittää entistä enemmän huomiota, sillä Internetin yleistyttyä sitä käyttävät hyvin eritasoiset käyttäjät.

7.8 Ohjelmiston toteutus

Ohjelmiston toteutusvaiheesta käytetään myös nimitystä *ohjelmointivaihe*, jolla tarkoitetaan ohjelman kirjoitusvaihetta ensimmäiseen virheettömään käännökseen asti [HaM01]. Projektin edettyä toteutusvaiheeseen, on hyvin tehdyt suunnitelmat toteutettava. Toteutusvaiheessa rakennetaan komponentit ja metodit sekä kirjoitetaan järjestelmälle kunnolliset käyttöohjeet. Toteutustason ongelmia voidaan pyrkiä ratkaisemaan hyväksi todettujen menetelmien, eli suunnittelumallien, avulla. Tässä vaiheessa olisi syytä tehdä myös pienimuotoisia käytettävyystestejä. Havaittujen ongelmien korjaaminen on yleensä tässä vaiheessa vielä suhteellisen helppoa, kun kaikkea ei ole ehditty tehdä valmiiksi. Toteutusvaiheen päätteeksi peilataan tuotteen toteutettuja ominaisuuksia ja käytettävyyttä vaatimusmäärittelyvaiheessa tehtyihin käytettävyystavoitteisiin ja arvioidaan, missä kohdin tavoitteisiin päästiin ja mitkä tavoitteet eivät toteutuneet. Toteutumattomien tavoitteiden kohdalla on syytä miettiä, miksi tavoitteeseen ei päästy ja onko saavutettu tilanne kyllin hyvä, vai onko tavoitteeseen pääsemiseksi tehtävä vielä lisää työtä.

7.8.1 Suunnittelumallit

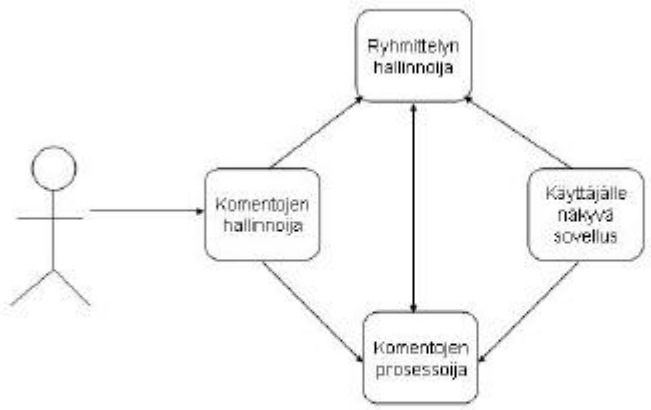
Suunnittelumalli-käsite on lainattu rakennusarkkitehtuurin puolelta. *Suunnittelumalli* (design pattern) [GaH02] nimeää, tarjoaa taustatietoa ja selittää käytännössä hyväksi todetun tavan ratkaista jokin järjestelmässä usein esiintyvä arkkitehtuuri- tai suunnitteluongelma [Imm02]. Ratkaisu on yleinen kokoelma luokkia ja olioita, joita sovelletaan ja muovataan ratkaisemaan ongelma erilaisissa käytännön tilanteissa. Suunnittelumallien avulla voidaan parantaa myös ohjelmistojen käytettävyyttä. Kun myös käytettävyyden kannalta hyvät ratkaisumallit dokumentoidaan huolellisesti, muutkin voivat hyödyntää sitä, eikä kaikkien tarvitse toistaa samoja virheitä. Kaikkien liitteessä 1 kuvattujen käytettävyysskenaarioiden ratkaisemiseksi on olemassa suunnittelumallit. Seuraavissa kappaleissa kuvataan esimerkkeinä *tiedonkoostamismalli* ja *komentojen perumismalli* [BaB01].

7.8.1.1 Tiedonkoostamismalli

Tietojen koostamisella (aggragating data) tarkoitetaan sitä, että käyttäjän tulisi voida valita ja toimia mielivaltaisilla tietoyhdistelmillä. Käyttäjän haluamia tiedon tai toimenpiteiden koosteita ei voida aina ennustaa etukäteen, sillä ne riippuvat jokaisen tehtävän vaatimuksista. Tämän vuoksi järjestelmien tulisi sallia käyttäjien valita ja toimia mielivaltaisilla tietoyhdistelmillä.

7.8.1.1.1 Rakenne

Tiedonkoostamismalli koostuu neljänlaisista komponenteista: komentojen hallinnoijasta ryhmittelyn hallinnoijasta, käyttäjälle näkyvän sovelluksen tiedoista ja komentojen prosessoijasta (Kuva 16). Näillä kaikilla komponenteilla on mallissa omat vastuunsa ja velvollisuutensa.



Kuva 16: Tiedonkoostamismallin rakenne

Komentojen hallinnoija (command manager) -komponentti hallinnoi käyttäjän tuottamat komennot. Komento liittyy aina johonkin toimenpiteeseen (action) ja sillä on yksi tai useampia subjekteja, jotka joko tarjoavat syötteen tai hyväksyvät komennon tuotoksen (output). Komentojen hallinnoija tuottaa palautteen komennosta ja siirtää sen näytölle.

Komentoja on olemassa kolmenlaisia. Jotkut komennoista ovat riippumattomia ryhmästä ja ryhmittelyn hallinnoijasta. Nämä komennot siirtyvät suoraan komentojen prosessoijalle suo-

rittaviksi. Toisen tyyppiset komennot puolestaan liittyvät ryhmien hallinnointiin, tiedon liittämiseen ryhmiin, tiedon poistamiseen ryhmästä tai ryhmän poistamiseen. Nämä komennot siirtyvät ryhmittelyn hallinnoijalle. Kolmannen komentotyypin komennot ovat toimenpiteitä, joissa yksi tai useampi subjekteista on ryhmä. Tämän tyyppisiä komentoja ei siirretä ryhmittelyn hoitajalle eikä komentojen prosessoijalle.

Ryhmittelyn hallinnoija (grouping manager) -komponentti hallinnoi ryhmien määrittelyjä, sekä tietojen lisäämistä ja poistamista ryhmästä. Ryhmittelyn hallinnoijan on vähintäänkin tuettava komentoja, jotka luovat tai poistavat ryhmiä ja liittävät tai poistavat tietoja näistä ryhmistä. Myös tiedon ominaisuuksia (data points) ja muita ryhmiä tulisi pystyä liittämään tai poistamaan ryhmästä. Ryhmittelyn hallinnoija kontrolloi komentojen toistoa (iteration) komentojen prosessoijan välityksellä. Se hakee käyttäjälle näkyvän tiedon ja tämänhetkiset ryhmät tukeakseen ryhmien muokkauskomentoja.

Käyttäjälle näkyvän sovelluksen tiedot (user-visible data) -komponentti tarjoaa pääsyn siihen sovelluksen tietoon, joka näkyy käyttäjälle. Tieto voi kuulua yhteen tietolähteeseen tai se voi olla hajautettu useamman komponentin kesken. Joka tapauksessa sovelluksen käyttäjälle näkyvä tieto on sekä näyttöä kontrolloivan komponentin että tietoa muuttavan komponentin käytettävissä.

Komentojen prosessoija (command prosessor) suorittaa komennot.

Komennon kuulumiselle johonkin ryhmään on olemassa kaksi erilaista vaihtoehtoa: *toisto* (iteration) ja *sulautettu ryhmittely* (embedded grouping). Toisto kuvaa tilanteen, jossa tietty komento käyttää yksittäistä muuttujaa. Tällaisessa tilanteessa ryhmittelyn hallinnoija hoitaa sovelluksen komennon ryhmään. Ryhmittelyn hallinnoija herättää toistuvasti jokaiselle ryhmän yksilölle oikean komentojen prosessoijan. Tämä vaihtoehto olettaa, että jonkin komennon kuuluminen ryhmään merkitsee komennon kuulumista jokaiseen ryhmän elementtiin.

Toinen vaihtoehto on sulautettu ryhmittely. Tällaisissa tilanteissa komentojen prosessoija ymmärtää ryhmät ja voi suoraan toimia ryhmällä. Komento ja sen muuttujat lähetetään suoraan komennon prosessoijalle. Ryhmittelyn hallinnoijan tehtävänä on mahdollistaa ryhmän elementtien päästä käsiksi komentojen prosessoijaan.

Jotta sekä ryhmittelyn hallinnoija että komennon prosessoija voivat muutella samaa tietoa, tämä tieto on pidettävä erillään molemmista komponenteista. Tällainen erittely mahdollistaa komentojen lukea samaa tietoa, jota komponentit muuntelevat. Käyttäjät voivat luoda komentokokonaisuuksia antamalla kokonaisuudet komentojen hallinnoijalle. Ryhmittelyn hallinnoija kontrolloi toistoa luomalla uusia komentojoukkoja ja antamalla ne järjestyksessä komentojen prosessoijalle. Tämän vuoksi myös komentojen luominen on pidettävä erillään komentojen suorittamisesta.

7.8.1.2 Komentojen perumismalli

Usein tapahtuu tilanteita, joissa käyttäjä aloittaa toiminnon, mutta ei haluakaan suorittaa sitä loppuun asti. Käyttäjä saattaa siis suorituksen päättymisen odottamisen sijasta haluta keskeyttää toiminnon kokonaan. Sillä ei ole merkitystä, miksi käyttäjä tahtoo näin tehdä. Syynä saattaa olla vahinko, käyttäjä saattoi vahingossa valita väärän toiminnon jne. Näistä ja monista muista syistä johtuen järjestelmän olisi mahdollistettava käyttäjän perua aloittamansa suoritus.

7.8.1.2.1 Rakenne

Komentojen perumismalli koostuu toimintokomponenteista, peruutusten kuuntelijasta, peruutusten kontrolloijasta ja avustajista (Kuva 17). Seuraavassa kuvataan lyhyesti kunkin komponentin tehtävät mallissa.

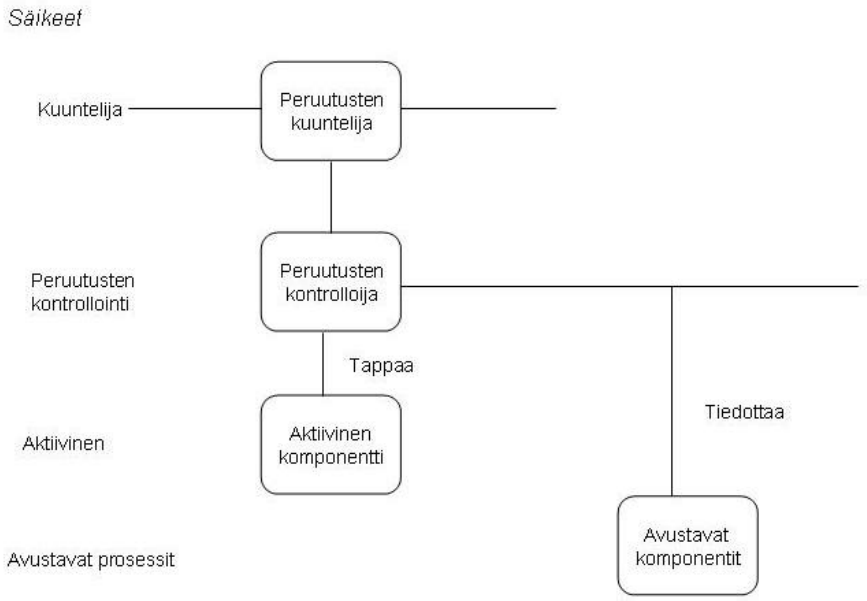
Toimintakomponentit (activity component) suorittavat peruttavat toiminnat. Niiden täytyy toimia yhteistyössä hallinnoijan kanssa ja antaa tietoa käyttämistään resursseista ja yhteistoiminnastaan. Niillä täytyy myös olla mekanismi, jonka avulla säilytetään merkityksellinen tieto järjestelmän tilasta, jotta se voidaan palauttaa minä hetkenä hyvänsä.

Peruutusten kuuntelija (cancellation listener) -komponentti kuuntelee käyttäjiä pyytääkseen peruutusta toimintakomponentilta. Kuuntelijan täytyy ilmoittaa käyttäjälle, että se on saanut peruutuspyynnön ja tiedottaa myös peruutusten kontrolloijalle peruutuksesta. Jos on tarvetta, kuuntelija voi synnyttää uuden säikeen kontrolloimaan peruutuskomponentin operaatioita.

Peruutusten kontrolloija (cancellation controller) -komponentti keskeyttää aktiivisen säikeen, palauttaa keskeytymättömät resurssit edeltäneeseen tilaan herättämällä aktiiviset komponentit,

vapauttaa resurssit, antaa palautteen käyttäjälle edistymisestä ja peruutuksen tuloksesta, sekä tiedottaa yhteistyössä oleville komponenteille aktiivisen säikeen keskeytymisestä.

Avustaja- (collaborators) komponentit ovat vastuussa aktiivisten komponenttien keskeytystietojen vastaanottamisesta.



Kuva 17: Komentojen perumismallin rakenne

Mallissa (Kuva 17) käyttäjä lähettää peruutusherätteen peruutusten kontrolloijalle käynnistämällä kuuntelija-säikeen. Sen jälkeen tämä säie antaa käyttäjälle palautteen, että peruutuspyyntö on saapunut ja tiedottaa peruutusten kontrolloijalle siitä, että sen tulisi huolehtia peruutustoiminnasta. Peruutusten kontrolloija on siis säie, joka toteuttaa peruutuksen. Aktiivisen säikeen ei mallissa odoteta kuuntelevan peruutuspyyntöjä, koska se voi olla jostakin syystä tukossa tai ikuisessa silmukassa.

Peruutusten kontrolloija keskeyttää aktiivisen säikeen ja ilmoittaa käyttäjälle keskeytyksestä. Kontrolloija myös palauttaa resurssit keskeytystä edeltäneeseen tilaan, joten sen on tiedettävä palautusmekanismit ja resurssien edeltävät tilat. Lisäksi kontrolloijan on tiedettävä aktiivisten komponenttien vaatimat resurssit ja vapautettava ne. Peruutusten kontrolloija tiedottaa myös

avustaville komponenteille aktiivisen komponentin toiminnan peruutuksesta, joten sen on tiedettävä, mitkä komponentit toimivat avustajina kullekin aktiivisena olevalle komponentille.

7.8.2 Tallentaminen

Useiden käytettävyysskenaarioiden (liite 1) huomioimiseksi kannattaa järjestelmän tila tallentaa jaksoittain. Tallentamismenetelmää käyttämällä voidaan helpottaa komentojen koostamista ja perumista, toimintojen perumista, virheistä toipumista ja järjestelmän arviointia. Muuttujia, jotka ovat riippuvaisia menetelmän soveltamisesta, ovat esimerkiksi se, kuinka usein tila tallennetaan, tallennetun tiedon pysyvyys ja tallennettujen tietojen yhtenäisyys. Joissakin tapauksissa voi esimerkiksi olla tärkeää tallentaa tieto pysyvään varastoon ja joskus tieto voidaan tallentaa myös tilapäisempään varastoon.

7.8.3 Sijoittelu ja paketointi

Paketoinnilla ryhmitellään luokkia yhteen suuremmiksi kokonaisuuksiksi, eli paketeiksi (package). Muodostettujen pakettien väliset riippuvuudet voidaan kuvata pakettikaavion avulla. Kahdella paketilla on riippuvuus, mikäli vähintään yhden kummankin paketin luokan välillä on riippuvuus. Riippuvuudesta johtuen toisen paketin jonkin elementin muuttaminen voi aiheuttaa muutoksia myös toisen paketin elementteihin. Suunnittelussa pyritään luonnollisesti minimoimaan riippuvuudet (coupling), koska tällöin systeemin ylläpito helpottuu ja tulee taloudellisemmaksi, eivätkä tehtävät muutokset heijastele laajalle alueelle. Pakettien välisiä riippuvuuksia voidaan komponenttien välisten riippuvuuksien vähentämisen tavoin vähentää erittely- ja välillistämismenetelmiä hyödyntämällä (ks. kappaleet 7.4.1 ja 7.4.2). Paketointiideaa voidaan soveltaa myös muiden kuin luokkien ryhmittelyssä. Paketti sijoitetaan yleensä kokonaisuudessaan samalle laitteistolle.

Pakettien koostamisessa ja sijoittelussa voidaan käyttää hyväksi toistamismenetelmää. *Toistaminen* (replication) on menetelmä, jossa toistetaan kopioita tai muunnelmia järjestelmän jostakin kokonaisuudesta (entity). Tämä kokonaisuus voi olla tietoa, funktio tai kokonainen komponentti. Toistamisen avulla voidaan lisätä suorituskykyä, luotettavuutta, vikasietoisuutta tai tarjota vaihtoehtoisia reittejä tietyn tuloksen aikaansaamiseksi.

Tiedon toistaminen (data replication) tarkoittaa saman tiedon säilyttämistä useissa eri paikoissa järjestelmässä [BaB01]. Tiedon toistamisen avulla voidaan parantaa suorituskykyä tai joissain tapauksissa parantaa tiedon saatavuutta. Esimerkki tämän menetelmän käytöstä on tiedon tallentaminen välimuistiin. Tieto tallennetaan samassa muodossa useissa erilaisissa paikoissa. Esimerkiksi Internet-sivut ladataan koneen omaan välimuistiin, jotta tiedonhakunopeus paranee Internetissä. Tiedon toistamisen avulla voidaan myös helpottaa, tehostaa ja nopeuttaa käyttäjän navigointia yksittäisellä näytöllä.

Komentoja (commands) toistetaan, jotta sama toiminta voidaan saavuttaa useista eri käyttöliittymistä [BaB01]. Eri komennoilla voidaan siis saavuttaa sama päämäärä. Käytettävyysskenaarioista komentojen toistaminen helpottaa osaltaan komentojen koostamista ja epätavallisessa käyttöympäristössä työskentelyä mahdollistaen tuttujen komentosarjojen käyttämisen uudessakin ympäristössä.

Kokonaisia komponentteja toistamalla voidaan myös tehostaa ja nopeuttaa toimintojen suorittamista. Sijoitteluvaiheen yksi tärkeä päätös onkin se, kuinka monta samanlaista komponenttia jotakin tiettyä toimintoa varten tarvitaan järjestelmään. Erityisesti suurissa hajautetuissa järjestelmissä, joita käytetään eri paikoissa yhtä aikaa, komponentteja voidaan kopioida useitakin suorittamaan jotakin tiettyä tehtävää, jotta odotus- ja vasteajat eivät nousisi liian korkeiksi. Hajautettujen järjestelmien kohtaamien ongelmien ratkaisemiseksi on myös kehitetty omat arkkitehtuurimallinsa, joita ovat esimerkiksi Half Component - Protocol, Shared Object ja Object Replication [Gra02].

7.8.4 Käyttöohjeet

Harvemmin tulee ehkä ajatelleeksi, että myös toimivat käyttöohjeet ovat osa tuotteen käytettävyyttä. Käyttöohjeiden laatiminen tulisi aloittaa mahdollisimman aikaisessa vaiheessa, sillä ohjeistusta voidaan laajentaa iteratiivisesti projektin edetessä. Lienee sanomattakin selvää, että käyttöohjeet täytyy kirjoittaa selvästi, tarpeeksi kattavasti ja vaihteleva lukijakunta huomioiden. Lisäksi on syytä pitää mielessä, että käyttöohjeet otetaan useimmiten esille vasta viimeisessä hädässä, kun ohjelmaa ei muuten osata käyttää tai eteen tulee jokin hälyttävältä vaikuttava tilanne. Myös käyttöohjeet on hyvä arvioida, tarkistaa ja antaa ulkopuolisen henkilön testattavaksi. Luonnollisesti käyttöohjeen testaajan olisi hyvä olla osaamis- ja tietotasoltaan todellisia loppukäyttäjiä vastaava.

Käyttöohjeita laadittaessa on edelleen pidettävä mielessä, että ohjelman päämääränä on tukea ihmisten tekemää työtä. Käyttäjät siis käyttävät laitteita tai ohjelmistoja jonkin päämäärän saavuttamiseksi. Näin ollen pääasiallinen käyttäjien tarvitsema tuki on tietoa ja opastusta niiden tehtävien suorittamiseen, joita tarvitaan tämän päämärän saavuttamiseksi. Useimmiten, kun käyttöohje otetaan esille, etsitään tietoa, joka auttaa käyttäjää joko selvittämään, mitä tehtäviä hänen on suoritettava, tai kuinka nämä tehtävät suoritetaan [HaR98].

Suunnittelijat näyttävät aivan liian usein olettavan, että käyttäjät lukevat käyttöohjeet kannesta kanteen, mutta tosiasia on, että käyttäjät lukevat käyttöohjetta vain juuri sen verran, että he pystyvät taas jatkamaan työskentelyään. He silmäilevät, hyppivät kohdasta toiseen ja lukevat tarkemmin vain sen kohdan, josta vastaus kohdattuun ongelmaan löytyy. Tämän vuoksi käyttöohjeiden navigoitavuuteen tulisi kiinnittää huomiota. Erityisesti otsikoiden ja kuvaotsikoiden tms. tulisi olla kuvaavia ja niissä tulisi käyttää käyttäjien termistöä ja kieltä. Lisäksi käyttöohjeet kirjoitetaan yleensä toimintonäkökulmasta, eli kuvataan lähinnä sitä, mitä ohjelman avulla voidaan tehdä. Käytettävyydeltään hyvän käyttöohjeen näkökulma on kuitenkin käyttäjässä ja siinä työssä, jota käyttäjät yrittävät ohjelman avulla tehdä [HaR98].

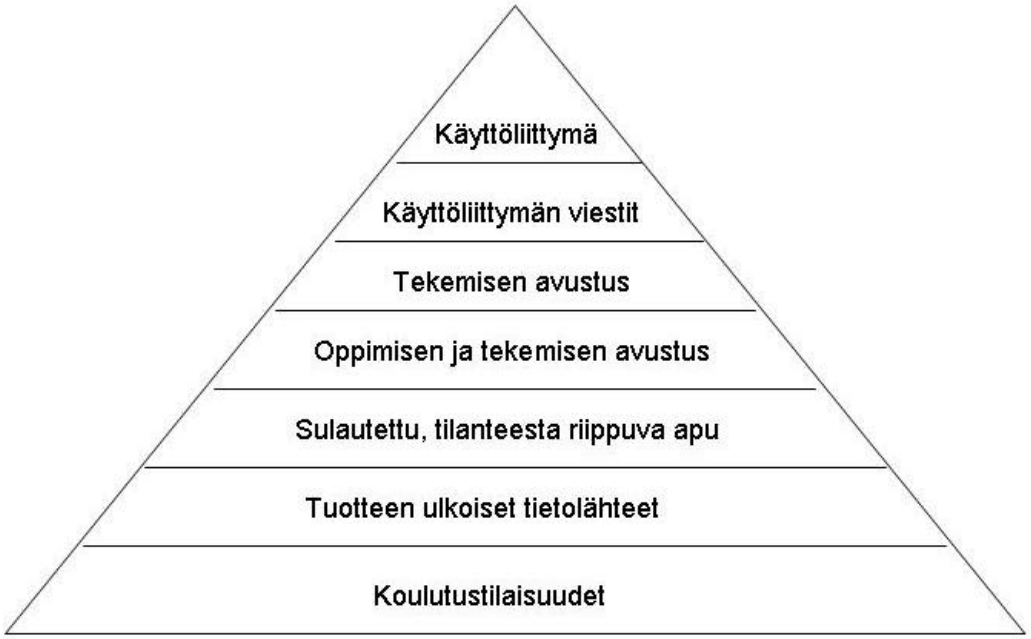
7.8.4.1 Minimalistinen käyttöohje

Monilla on sellainen harhaluulo, että minimalistisella dokumentoinnilla tarkoitetaan samaa kuin lyhyellä dokumentilla. *Minimalismi* (minimalism) on kuitenkin dokumentoinnin painopisteen siirtämistä käyttäjien tekemiin tehtäviin ja työhön [HaR98]. Lisäksi se on käyttäjien ymmärtämistä ja tietämystä siitä, mitä käyttäjät jo entuudestaan tietävät. On myös tärkeää tietää, mitkä ovat tyypillisimpiä käyttäjien tekemiä virheitä, jotta niitä voitaisiin ehkäistä pienillä käyttäjille annettavilla vihjeillä. Minimalismi on myös käyttäjien rohkaisemista yrittämään erilaisten tehtävien suorittamista.

Minimalistinen käyttöohje on hyvä ja tehokas vaihtoehto tilanteissa, joissa tunnetaan tyypilliset käyttäjät ja heidän tyypilliset tehtävänsä sekä tyypillisimmät virhetilanteet. Käyttäjiä ja heidän suorittamiaan tehtäviä voidaan parhaiten oppia tuntemaan ja ymmärtämään vaatimusmäärittelyvaiheessa tehtävien kattavien käyttäjä- ja tehtäväanalyysien avulla (ks. kappale 7.1).

7.8.4.2 Eritasoiset käyttöohjeet

Ohjelmistot pitävät sisällään monentasoista tietoa ja toimintaohjeita (Kuva 18) [HaR98]. Näiden ohjeiden tarkoituksena on auttaa käyttäjää suorittamaan niitä tehtäviä, joita hän sovelluksen avustuksella haluaa suorittaa. Ohjeistuksen laatiminen on pikemminkin olennainen osa koko suunnittelu- ja toteutusprosessia eikä vain välttämättömyys, joka otetaan huomioon vasta jälkikäteen. Osa suunnitteluprosessia on päättää, mitkä näistä käyttöohjehierarkian asioista otetaan mukaan ja mitkä voidaan jättää toteuttamatta. Lisäksi on luonnollisesti päätettävä, miten valitut osa-alueet toteutetaan.



Kuva 18: Eritasoiset käyttäjän opastusmahdollisuudet [HaR98]

Ylimmällä tasolla käyttöohjehierarkiassa on *käyttöliittymä* (interface), joka on itsessään yksi informaatiomuoto. Se sisältää informaatiota sisältäviä sanoja, kuvia, värejä, kuvakkeita, valikoita jne. Seuraavalla tasolla oleva ohjeistus liittyy myös kiinteästi käyttöliittymään, mutta käyttäjä ei näe niitä suoraan avatessaan sovelluksen. *Käyttöliittymän viestit* (messages that appear in interface) auttavat käyttäjää nopeasti etenemään tehtävänsä suorituksessa. Tällaisia viestejä ovat esimerkiksi virheilmoitukset, Ohje-painikkeet ja sopivan työkalun valintaan liittyvät vihjeet (tool tips).

Kolmannen tason käyttöohjeet auttavat käyttäjää suorittamaan tehtävät välittömästi. Joissakin tapauksissa käyttäjä voi itse päättää, haluaako hän nähdä tämän tyyppisiä ohjeita, tai suunnittelijat ovat voineet laittaa ohjeistusta niihin paikkoihin, joissa he olettavat suurimman osan käyttäjistä tarvitsevan ohjeistusta. *Tekemisen avustamis-menetelmiä* (aids to doing) ovat esimerkiksi erilaiset wizardit ja agentit, jotka ovat osa käyttöliittymää ja auttavat käyttäjää suorittamaan tehtävänsä loppuun asti. *Oppimisen ja tekemisen avustamisella* (aids to learning and doing) tarkoitetaan sitä, että käyttäjää pyritään opettamaan tekemällä. Esimerkki tällaisesta menetelmästä on erilaisten vihjeiden käyttäminen.

Sulautetulla tilanneriippuvaisella avustuksella (embedded context-sensitive help) tarkoitetaan esimerkiksi erilaisia tutoriaaleja, jotka on rakennettu sovellukseen, ja joihin on helppo päästä käsiksi suoraan suorituksen aikana. Olennaista tämän tyyppiselle avustukselle on tilanneriippuvaisuus, eli sovellus ymmärtää, mitä käyttäjä on juuri tekemässä. Tyypillistä on myös hypertekstuaalisuus, eli käyttäjä voi linkkien kautta hakea myös lisätietoa tarvitsemastaan asiasta tai toiminnosta. *Tuotteen ulkoisilla tietolähteillä* (information external to rest of product) tarkoitetaan perinteisiä käyttöohjeita, manuaaleja ja opastusmateriaalia, jotka ovat usein paperimuodossa. Alimmalla tasolla hierarkiassa ovat erilaiset *koulutustilaisuudet* (classroom training), joita voidaan järjestää sovelluksen käytön opettamiseksi.

7.9 Käytettävyystestaus

Järjestelmän käytettävyyden testaamiseen on olemassa lukuisia erilaisia menetelmiä ja käytettävyystestausta on myös tutkittu paljon. *Käytettävyystesti* (usability test) on apukeino, jonka tarkoituksena on tuoda esiin tuotteen käyttöliittymässä olevat ongelmat, jotta ne voidaan korjata tai arvioida, onko tuote ongelmistaan huolimatta tarpeeksi hyvä käyttöön [Sin02]. Käytettävyystestit voidaan jakaa tarkoituksensa mukaan kahteen tyyppiin: kehitystesteihin ja hyväksymistesteihin [SiK02]. *Kehitystestejä* tehdään käyttöliittymän suunnittelu- ja toteutusvaiheessa, ja niiden tarkoituksena on löytää mahdollisimman käytettävä ratkaisu. Ilman käyttäjiä tehtävät asiantuntija-arviot kuuluvat kehitystesteihin (ks. kappale 7.6.2), myös seuraavissa kappaleissa kuvattavia menetelmiä voidaan käyttää myös tuotteen kehitysvaiheessa prototyyppien avulla. *Hyväksymistesteissä* testataan, miten hyvin käyttöliittymä täyttää sille asetetut käytettävyysvaatimukset.

Käytettävyystestissä tuotteen käytettävyyttä mitataan joko oikeilla käyttäjillä tai oletettuun käyttäjäryhmään kuuluvan testikäyttäjän avustuksella. Testin aikana käyttäjät tekevät oikeita työtehtäviä testattavalla järjestelmällä oikeassa työskentely-ympäristössä tai laboratorioympäristössä. Testiä tekee tavallisesti vain yksi käyttäjä kerrallaan, mutta yhteensä testikäyttäjiä on 3-8 tuotteen käyttäjäryhmän hajanaisuudesta riippuen [Rii02b]. Testin ohjaaja antaa käyttäjälle yhden tehtävän kerrallaan ja pyytää käyttäjää ajattelemaan ääneen tehtävää suorittaessaan. Ääneenajattelu auttaa esimerkiksi selvittämään, miksi käyttäjä tekee tiettyjä valintoja, tai mitkä kohdat käyttöliittymässä ovat epäselviä. Testin kulku tallennetaan tavalla tai toisella ja saadut tiedot analysoidaan. Käytettävyystesti on ainoa objektiivinen mittaustapa, jolla tuotteen käytettävyys voidaan todeta. Käytettävyystestiin sijoitetut rahat tulevat aina takaisin, jos testit tehdään asianmukaisesti ja havaitut ongelmakohdat korjataan [SiK02].

Käytettävyystestin välitön hyöty on siinä, että tuotteiden käyttölaatu paranee, järjestelmän käyttökustannukset ja käyttöön liittyvien virheiden määrä pienenevät selvästi ja käytön tehokkuus sekä järjestelmän eri ominaisuuksien käyttöaste paranee. Lisäksi koulutukseen ja käytön tukeen kuluu vähemmän aikaa ja rahaa. Välillisesti käytettävyyden arviointi parantaa suunnittelijoiden valmiutta ottaa todelliset loppukäyttäjät huomioon. Suunnittelijat saavat käytettävyystesteistä paljon arvokasta palautetta ja näin heidän ammattitaitonsa paranee. Toisaalta testaus vaikuttaa myös suunnittelijoiden asenteisiin käyttäjiä ja käytettävyyden huomioimista kohtaan [SiK02].

Perinteiselle käytettävyystestille on olemassa useita erilaisia muunnelmia, joista paritestaus, ryhmäläpikäynti, vapaa läpikäynti ja tilannesidonnainen läpikäynti kuvataan seuraavissa luvuissa lyhyesti [Rii02b].

7.9.1 Paritestaus

Paritestauksessa (co-discovery method) kaksi käyttäjää suorittaa annettuja tehtäviä yhdessä. Paritestauksen ääneenajattelu on luonnollisempaa ja tehokkaampaa kuin se, että yksinään testiä tekevää käyttäjää pyydetään ajattelemaan ääneen. Paritestauksessa ääneenajattelusta tulee enemmänkin omien näkemyksien opettamista toiselle kuin pelkkien omien ajatusten ääneen esittämistä. Testien analysointi on myös helpompaa paritestin kuin perinteisen käytettävyystestin jälkeen, koska ääneenajattelu on jäsennellympää ja luonnollisempaa kuin yksin tehtävässä testissä. Testiparin on oltava tietämykseltään ja kokemuksiltaan tasavertaisia, jotta

kumpikaan ei dominoisi testiä. Käyttäjät voivat myös olla toisilleen entuudesta tuttuja ja tottuneita työskentelemään yhdessä.

7.9.2 Ryhmäläpikäynti

Ryhmäläpikäynti (pluralistic walkthrough) on käytettävyystestin muoto, joka yhdistää asiantuntija-arvioiden ja käytettävyystestien ominaisuuksia. Ryhmäläpikäynnissä on mukana sekä käyttäjiä että suunnittelijoita samassa istunnossa. Jokainen ottaa istunnossa käyttäjän roolin ja pyrkii ratkomaan tehtäviä hänen näkökulmastaan. Ensin tehtävät ratkotaan yksin ja sen jälkeen niistä keskustellaan ryhmässä. Keskustelun päätteeksi suunnittelija tai testin ohjaaja kertoo, mikä olisi ollut suunniteltu ratkaisu. Ryhmäläpikäynti soveltuu myös suunnitteluvaiheen ideoiden arviointiin, sillä läpikäynnissä voidaan käyttää varsinaisen järjestelmän sijasta myös käyttöliittymän paperiversioita [Rii02b].

Ryhmäläpikäynnin etuna on suunnittelijoiden ja käyttäjien kontakti sekä suora keskustelu, josta saadaan välitöntä palautetta. Varsinkin, jos ryhmäläpikäynti tehdään suunnittelun alkuvaiheessa, suunnittelijat saavat hyviä vinkkejä käyttöohjeiden suunnitteluun, kun he näkevät, mitkä kohdat käyttöliittymässä ovat hankalia ja vaativat opastusta.

7.9.3 Vapaa läpikäynti

Vapaa läpikäynti (exploratory testing) on käytettävyystestausmenetelmä, jossa testikäyttäjä kokeilee testattavaa ohjelmaa rauhassa [Sin02]. Se siis eroaa perinteisestä käytettävyystestistä siinä, että vapaassa läpikäynnissä ei käytetä ennalta suunniteltuja tehtäviä. Vapaaseen läpikäyntiin tarvitaan joko toimiva prototyyppi tai valmis ohjelma. Testin ohjaajan täytyy hallita järjestelmä läpikotaisin, jotta hän pystyy tarvittaessa auttamaan testikäyttäjää tilanteessa kuin tilanteessa [Rii02b].

7.9.4 Tilannesidonnainen läpikäynti

Tilannesidonnaisessa läpikäynnissä yksi tai useampi tarkkailijaa seuraa käyttäjän työtä ja tekee siitä muistiinpanoja. Useimmiten käytetään apuna myös videointia. Menetelmää sovelletaan siten todellisessa käyttöympäristössä ja -tilanteissa. Sopivissa väleissä käyttäjä voi kertoa, mitä juuri päättynyt työ piti sisällään. Tarkkailijat voivat myös esittää muistiinpanojensa

pohjalta kysymyksiä käyttäjälle. Tilannesidonnaiseen läpikäyntiin ei voida laatia testitehtäviä etukäteen, eikä todellisessa tilanteessa tehtävää voi testin takia keskeyttää tai häiritä. Tilannesidonnainen läpikäynti sopii erityisesti tehokkuuden arviointiin [Rii02b].

7.9.5 Testausmenetelmän valinta

Taulukossa 2 on kuvattu kunkin edellisissä kappaleissa kuvatun testausmenetelmän ominaisuuksia, sillä ne vaikuttavat kuhunkin tilanteeseen sopivimman käytettävyystestausmenetelmän valintaan. Tärkeimmät valintakriteerit ovat ensinnäkin se, mitä halutaan testata ja toisaalta se, miten pitkällä tuotteen rakentamisessa ollaan. Esimerkiksi opittavuutta voidaan testata jo hyvinkin aikaisessa vaiheessa ohjelmistotuotantoprosessia, kun vasta näyttökuvat ovat valmiina. Toisaalta järjestelmän soveltuvuutta käyttäjien työtehtäviin voidaan tutkia vasta siinä vaiheessa, kun toimiva prototyyppi on valmis. Käyttöliittymän suunnitteluvaiheessa määritellyt käytettävyystavoitteet (ks. kappale 7.1.7) vaikuttavat siihen, mitä kunkin järjestelmän kohdalla testataan ja mikä on sopivin testausmenetelmä. Testausmenetelmän valintaan vaikuttaa myös käytettävissä olevien resurssien määrä ja se, tehdäänkö järjestelmä mittatilaustyönä jollekin tietylle käyttäjäkunnalle, vai onko kysymyksessä massavalmistukseen tuleva tuote. Olivatpa järjestelmän tulevat käyttäjät ketä tahansa, käytettävyystestien tekeminen on erittäin tärkeää.

Menetelmä	**Osallistujat**	**Mitä ominaisuuksia testataan?**	**Tarvittava materiaali**	**Tehtävät**
Käytettävyystesti	1 testaaja kerrallaan, yhteensä 3-8 hlöä, ohjaaja	Opittavuus, muistettavuus, virhetilanteiden määrä, käyttäjien työn tukeminen	Toimiva prototyyppi tai valmis ohjelma	Ennalta suunniteltu
Paritestaus	2 käyttäjää, ohjaaja	Opittavuus, muistettavuus, virhetilanteiden määrä	Toimiva prototyyppi tai valmis ohjelma	Ennalta suunniteltu
Ryhmäläpikäynti	2-4 käyttäjää, 1-2 suunnittelijaa 2 ohjaajaa	Opittavuus	Näyttökuvat, paperiprotot tai toimiva prototyyppi	Ennalta suunniteltu
Vapaa läpikäynti	1 testaaja kerrallaan, ohjaaja	Järjestelmän toiminnan ja käyttötilanteiden viestiminen, intuitiivisuus, opittavuus	Toimiva prototyyppi tai valmis ohjelma	Ei ennalta suunniteltu
Tilannesidonnainen läpikäynti	1 testaaja kerrallaan, ohjaaja	Tehokkuus, työhön soveltuvuus	Toimiva prototyyppi tai valmis ohjelma	Ei ennalta suunniteltu

Taulukko 2: Käytettävyystestausmenetelmien ominaisuuksia

7.9.6 Tulevaisuuden järjestelmien käytettävyystestaus

Nykyisin käytössä olevia käytettävyystestausmenetelmiä on varmasti tulevaisuudessa kehitettävä ja monipuolistettava, sillä myös järjestelmät monipuolistuvat ja kehittyvät. Nykyäänkin yhä useammat järjestelmät toimivat hajautetusti ja niitä käytetään samanaikaisesti useammasta eri paikasta. Myös työskentely-ympäristöt ja työskentelytavat muuttuvat, sillä tietoverkot ja erilaiset kannettavat päätelaitteet, kuten kannettavat tietokoneet ja matkapuhelimet, mahdollistavat työskentelyn jopa toisella puolella maapalloa. Pelkillä laboratoriotesteillä ei päästä kuin alkuun näissä muuttuvissa olosuhteissa. On varmasti kehiteltävä esimerkiksi siirreltäviä käytettävyyslaboratorioita, jotka voidaan viedä käyttäjien luokse sen sijaan, että käyttäjät pyydetään laboratorioihin testaamaan.

7.10 Ylläpito ja seuranta

Käytettävyyttä ei tulisi unohtaa varsinaisen tuotekehityksen päättymisen jälkeenkään, sillä käytettävyysongelmat tulevat usein esille vasta todellisissa käyttötilanteissa. Seuraamalla tuotteen tai ohjelmiston elämää sen kehitystyön jälkeen, voidaan saada kootuksi tärkeää tietoa todellisten käyttötilanteiden käytettävyysvaatimuksista ylläpitoa ja mahdollista seuraavaa versiota tai tuotekehityshanketta varten. Seuraavassa projektissa ei tarvitsekaan enää lähteä liikkeelle ihan alusta, vaan lähtötietona voidaan käyttää jo olemassa olevia käytännön tilanteiden kuvauksia [Vuo96].

Tuotteen käytön seurannassa voidaan käyttää passiivista tai aktiivista lähestymistapaa [Vuo96]. *Passiivisessa lähestymistavassa* tuotteen kehittäneellä yrityksellä on passiivinen asenne tuotteen käytön seurantaan. Palautetta ja tietoa tuotteen todellisesta käytöstä saadaan tällöin lähinnä reklamaatioiden ja esimerkiksi messujen tai muiden asiakaskontaktien kautta. Tälläkin tavalla saadaan selville tuotteen käyttöön liittyviä ongelmia, jos yrityksellä on ennalta sovitut menettelytavat asiakaspalautteen käsittelyyn, välittämiseen ja tallentamiseen. *Aktiivisen tuotteiden käytön seuraamisen* avulla yritys voi syventää kehittäjäorganisaation tietämystä sekä omista että kilpailijoiden tuotteista. Saatu palaute voi palvella tuotekehityksen lisäksi myös markkinointia. Aktiivinen toiminta vaatii luonnollisesti paljon jäsentyneempää toimintaa kuin passiivinen toimintatapa.

8 KÄYTETTÄVYYDEN HYÖDYT JA MENETELMÄT

Käytettävyyteen panostamalla voidaan saavuttaa monenlaisia hyötyjä. Käytettävyyden avulla voidaan lisätä yksittäisten käyttäjien tehokkuutta, vähentää järjestelmävirheiden vaikutuksia sekä lisätä käyttäjien luottamusta ja mukavuudentunnetta käytettävää tuotetta kohtaan (ks. kappale 3). Käytettävyyden hyödyt voidaan saavuttaa toteuttamalla kussakin tilanteessa merkityksellisimmät käytettävyysskenaariot (ks. Liite 1). Jokaisen käytettävyysskenaarion toteuttamiseksi on olemassa apukeinoja sekä perinteisten ohjelmistotuotantoprosessin menetelmien että arkkitehtuuri- ja suunnittelumallien puolella. On siis tärkeää muistaa, että käytettävyysskenaariot harvoin toteutuvat ilman, että niihin kiinnitetään erityistä huomiota. Useimmiten skenaarioiden, ja sitä kautta käytettävyyden hyötyjen saavuttamiseksi, on tehtävä päätöksiä ja paljon työtä koko ohjelmistotuotantoprosessin ajan. Käytettävyys on siis myös rakennettava jossakin vaiheessa tuotteeseen samalla tavalla kuin muukin toiminnallisuus.

Taulukossa 3 on yhteenvedonomaisesti kuvattuna käytettävyyden hyödyt, käytettävyysskenaariot sekä arkkitehtuuri- ja ohjelmistotuotantoprosessin ratkaisut Bass et.al [BaB01] ajatuksia muunnellen. Taulukossa olevat numerot viittaavat käytettävyysskenaarioihin, jotka on kuvattu numeroiden mukaisessa järjestyksessä liitteessä 1. Jatkotutkimusaiheeksi on jätetty, mitkä käytettävyysskenaariot toteutuvat x:llä merkityissä kohdissa. Kysymysmerkillä on merkattu kohdat, joihin voisi myös olla positiivista vaikutusta, kun menetelmää hyödynnetään. Eri arkkitehtuuriratkaisuja voidaan käyttää hyödyksi hieman eri vaiheissa ohjelmistotuotantoprosessia. Vaatimusmäärittelyvaiheessa (ks. kappale 7.1) kuvataan käyttäjien tehtävämallit sekä kuvataan järjestelmän tulevat käyttäjät. Arkkitehtuurisuunnittelun (ks. kappale 7.3) tuloksena syntyy puolestaan arkkitehtuurikuvaus, jota voidaan kutsua myös järjestelmämalliksi. Komponenttien suunnittelu -vaiheessa (ks. kappale 7.4) voidaan käyttää hyväksi erittely- ja välillistämistekniikoita ja toimintosuunnitteluvaiheessa (ks. kappale 7.5) keskeytyksen sallivan ajoituksen suomia mahdollisuuksia. Toteutusvaiheessa (ks. kappale 7.8) voidaan puolestaan hyödyntää tallentamista ja erilaisia toistamistekniikoita.

Käytettävyyden hyödyt → / Arkkitehtuuri-ratkaisu ↓		Lisää yksittäisten käyttäjien tehokkuutta						Vähentää järjestelmä-virheiden vaikutuksia		Lisää luottamusta ja mukavuutta
		Nopeuttaa rutiinisuorituksia		Parantaa ei-rutiinisuorituksia		Vähentää virheiden vaikutuksia				
		Nopeuttaa virheetöntä osaa	Vähentää lipsahdusten vaikutuksia	Tukee ongelman-ratkaisua	Helpottaa oppimista	Estää virheitä	Sietää virheitä	Ehkäisee järjestelmä-virheitä	Sietää järjestelmä-virheitä	
Mallit	Tehtävä	18, 19	5, 17, 18	5, 10, 17	5, 10, 17	5, 17, 19	5, 17			17, 18
	Käyttäjä	12, 18	5, 12, 17, 18	5, 10, 12, 17, 22	5, 10, 12, 17	5, 12, 17, 22	5, 12, 17			12, 17, 18
	Järjestelmä	4, 6, 19, 23	3, 5, 17	3, 4, 5, 6, 17	4, 5, 17	4, 5, 17, 19	3, 5, 17	3	6, 23	17
Erittely	Funktioiden kapselointi	4, 13, 14, 15, 20, 23		4, 13, 20	4, 13, 20	4, 13, 20	9, 14		23	
	Tieto sen näky-mästä	12, 13, 24, 25	12	12, 13, 22, 24, 25, 26	12, 13, 24	12, 13, 22 24	12			12
	Tieto komennoista	1, 24, 25	5, 17	5, 17, 24, 25, 26	5, 17, 24	1, 5, 17, 24	1, 5, 17			17
	Toiminnan valtuutus suorituksesta	1, 2	2			1, 2	1, 2			
Välillistä-minen	Tieto	7, 11 14	11	7, 11			14			x
	Funktiot	6, 14, 20		6, 20	20	20	14		6	
	Palvelut	x	x	x	x				?	x
Keskeytyksen salliva ajoitus		15, 18, 19	3, 5, 17, 18	3, 5, 10, 17	5, 10, 17	5, 17, 19	3, 5, 17	3		17, 18
Tallentaminen		2, 7	2, 3, 21	3, 7, 21		2	2, 3, 21	3, 8		
Toista-minen	Tieto	16				?	?	x	x	x
	Komennot	2	2	22		2, 22	2			
	Komponentit	x				x	x	x	x	

Taulukko 3: Menetelmät käytettävyyshyötyjen saavuttamiseksi

9 KÄYTETTÄVYYSONGELMIA

Tutkimukseen liittyvässä kokeellisessa osiossa kerättiin ihmisten kohtaamia todellisia käytettävyysongelmia. Käytettävyysongelmien kerääminen tapahtui sähköpostin välityksellä siten, että erilaisten valmiiden sähköpostilistojen välityksellä ihmisille kerrottiin asiasta ja halukkaat lähettivät tai kertoivat kohtaamiansa käytettävyysongelmia. Ongelmien keräämisen jälkeen ongelmat luokiteltiin ja pohdittiin, mitkä käytettävyysskenaariot ja heuristiset säännöt ovat jääneet kyseisessä tuotteessa huomioimatta. Lisäksi arvioitiin, miten ongelma voitaisiin korjata.

Havaitut ongelmat voidaan jaotella ongelman aiheuttajan perusteella käyttöliittymästä johtuviin ongelmiin, ohjelman toimintatavasta johtuviin ongelmiin, käyttäjän toimintaprosessia vaikeuttaviin ominaisuuksiin, käyttäjästä johtuviin ongelmiin ja muihin ohjelmasta tai tuotteesta riippumattomien tekijöiden aiheuttamiin ongelmiin. Kuhunkin ryhmään kuuluvien ongelmien tyypillisiä piirteitä kuvataan tarkemmin seuraavissa kappaleissa.

9.1 Käyttöliittymästä johtuvat ongelmat

Kootuista käytettävyysongelmista yleisimpiä ovat jollakin tavalla käyttöliittymästä aiheutuvat ongelmat (Taulukko 4). Syynä tähän voi olla ensinnäkin se, että ihmiset monesti mieltävät, että käytettävyys liittyy pelkästään käyttöliittymään. Toisaalta esimerkiksi käyttäjän muita tehtäviä ja toimintaprosessia vaikeuttavat ohjelmiston piirteet unohtuvat helposti, kun käyttäjä mukautuu ja mukauttaa toimintansa järjestelmän toiminnallisuuteen, eikä enää huomaa kaikkia puutteita tai ongelmia. Lisäksi käyttöliittymä on ohjelman konkreettisin osa, jonka kanssa ihminen on vuorovaikutuksessa.

Vaikuttaa huolestuttavalta, että monissa vielä nykypäivänäkin valmistuvissa ohjelmistojen versioissa käyttöliittymässä on paljon puutteita, vaikka käyttöliittymien käytettävyyttä on tutkittu varsin paljon. Suurin osa taulukossa 4 esitetyistä ongelmista ratkeaisi joko käyttöliittymien arviointimenetelmiä tai käytettävyystestausta hyödyntämällä.

Ongelman kuvaus	Toteutumattomat skenaariot	Huomioimattomat heuristiset säännöt	Ehdotus ongelman korjaamiseksi
Pikanäppäinkomennot eivät ole samoja saman tuotteen eri näytöillä saati saman tuoteperheen tuotteissa.	13. Aikaisempien tietojen hyväksikäyttö 24. Yhdenmukainen toiminta	3. Muistikuorman minimointi 4. Käyttöliittymän yhdenmukaisuus	Näppäinyhdistelmien standardointi.
Käyttöliittymä ei mahdu kerralla näytölle.	6. Laiteriippumattomuus 14. Käyttöliittymän muuntelu 16. Näytöllä navigointi 25. Näyttöjen helppo pääsyisyys 26. Mielikuvien tukeminen	3. Muistikuorman minimointi	Huomioidaan, että käyttäjillä voi olla erikokoisia näyttöjä. Mukautuminen niihin.
Käyttöliittymän harhaanjohtava tai jopa virheellinen terminologia.	10. Hyvän avun tarjoaminen 13. Aikaisempien tietojen hyväksikäyttö	1. Yksinkertainen ja luonnollinen kieli 2. Käyttäjän kieli 9. Vältä virhetilanteita 10. Riittävä ja selkeä apu	Käyttäjän sanaston huomioiminen ja selvittäminen esim. käsiteanalyysin avulla. Vakiintuneiden ilmaisujen käyttäminen.
Käyttäjältä pyydetään liikaa vahvistuksia. Sama asia saatetaan varmistaa käyttämällä erilaisia sanakäänteitä		9. Vältä virhetilanteita 5. Palautteen antaminen toiminnoista	Varmistusten pyytämien harkiten. Liiat varmistukset turruttavat käyttäjän.
Jos jokin kenttä on väärin täytetty, kaikki muutkin kentät tyhjenevät ja ne on täytettävä uudelleen.	5. Virheettömyyden tarkistaminen 8. Virheistä toipuminen 10. Hyvän avun tarjoaminen	1. Yksinkertainen ja luonnollinen kieli 2. Käyttäjän kieli 8. Selkeät virheilmoitukset 9. Vältä virhetilanteita 10. Riittävä ja selkeä apu	Hyvät täyttöohjeet käyttäjälle. Ohjeet virheellisen kohdan korjaamiseksi, muiden kohtien tietojen tulisi säilyä. Tyypillisten virheiden ennakointi.
Täytettävät kentät liian lyhyitä, kaikki tarvittava tieto ei mahdu niihin.			Täytettävän tietomäärän tarkempi arviointi. Mieluummin liikaa tilaa.
Ohjelman käyttäminen vaatii paljon hiiren käyttöä, sillä tabulaattori-näppäimen käyttämistä ei ole mahdollistettu.	13. Aikaisempien tietojen hyväksikäyttö 16. Näytöllä navigointi	7. Oikopolkujen käytön mahdollistaminen	Yleisesti käytössä olevia periaatteita (esim. tabulaattorilla siirtyminen) olisi syytä noudattaa.
Käyttäjän itse tekemän virheen jälkeen on vaikea palata edelliseen tilaan.	5. Virheettömyyden tarkistaminen 17. Järjestelmän tilan tarkkailu 21. Perumisen tukeminen	5. Palautteen antaminen toiminnoista 6. Selkeät poistumistavat 8. Selkeät virheilmoitukset 9. Vältä virhetilanteita	Käyttäjä ei saa tuntea olevansa ansassa. Selkeän paluun osoittaminen virhettä edeltäneeseen tilaan.
Esim. ohjelmointi-työkalut ovat aloittelijalle hankalia käyttää, koska niissä on paljon alkuvaiheessa turhaa toiminnallisuutta	1. Tiedon koostaminen 10. Hyvän avun tarjoaminen 14. Käyttöliittymien muuntelu	1. Yksinkertainen ja luonnollinen kieli	Mahdollisuus valita käyttäjän taitotasoa vastaava toiminnallisuus ja käyttöliittymä.
Käyttäjä yrittää suorittaa toimintoa, joka voi aiheuttaa vahinkoa tai ongelmia. Varoitus-/ohjeteksti on hämäävä, eikä mahdollisuutta toiminnon perumiseen ole.	10. Hyvän avun tarjoaminen 21. Perumisen tukeminen	6. Selkeät poistumistavat 8. Selkeät virheilmoitukset	Tarpeeksi havainnolliset toimintaohjeet. Pelottelun välttäminen ohjeissa ja varoituksissa.
Toiminnan epäloogisuus. Esim. toisella näytöllä tallennukselle on oma painikkeensa, toisella näytöllä tallentaminen tapahtuu ruudun ylälaidan pikakuvakkeesta.	13. Aikaisempien tietojen hyväksikäyttö 16. Näytöllä navigointi 24. Yhdenmukainen toiminta	3. Muistikuorman minimointi 4. Käyttöliittymän yhdenmukaisuus	Looginen toiminta läpi koko ohjelman. Saman toiminnon löydyttävä samasta paikasta joka näytöltä.
Käyttäjän antaessa vääränlaisen syötteen johonkin kenttään, välillä kentän väri muuttuu punaiseksi, välillä kenttä tyhjenee.	13. Aikaisempien tietojen hyväksikäyttö 24. Yhdenmukainen toiminta	4. Käyttöliittymän yhdenmukaisuus	Toiminnan täytyy olla loogista näytöstä ja kentästä toiseen.

Tallennus levykkeelle jatkuu, vaikka näytöstä voisi päätellä, että tehtävä on suoritettu. Jos levykkeen ottaa liian aikaisin levykeasemasta pois ja laittaa tilalle toisen levykkeen, kone menee jumiin.	17. Järjestelmän tilan tarkkailu 19. Tehtävän keston arviointi	5. Palautteen antaminen toiminnoista 8. Selkeät virheilmoitukset 9. Vältä virhetilanteita	Käyttäjää tulisi tiedottaa selvästi, milloin tallennus on vielä kesken ja estää levykkeen poistaminen liian aikaisin.

Taulukko 4: Käyttöliittymästä aiheutuvat ongelmat

9.2 Ohjelman toimintatavasta johtuvat ongelmat

Ohjelman toimintatavasta johtuvia ongelmia (Taulukko 5) voitaisiin luonnehtia myös ohjelmistovirheiksi, jotka olisi voitu huomata tavanomaisen testauksen yhteydessä. Näiden ongelmien kohdalla käyttöliittymässä ei välttämättä ole mitään ongelmia. Käyttäjä on osannut käyttää tuotetta tai ohjelmaa oikein, mutta silti ohjelman toiminta ei ole ollut toivotunlaista. Syyt ongelmiin löytyvät syvemmältä ohjelmistosta. Esimerkiksi se, että sama junapaikka voidaan myydä kahdelle ihmiselle, voi johtua siitä, että tietokannassa ei ole huolehdittu samanaikaisuuden hallinnasta, vaan hajautetussa järjestelmässä useampi voi muuttaa samaa tietoa samaan aikaan.

Ongelman kuvaus	**Toteutumattomat skenaariot**	**Huomioimattomat heuristiset säännöt**	**Ehdotus ongelman korjaamiseksi**
Sama junapaikka myyty useaan kertaan		9. Vältä virhetilanteita	Samanaikaisen käytön huomioiminen
Perheenjäsen haluaa ottaa oman vakuutuksen muutettuaan pois kotoa. Vakuutusvirkailija tekee virheen ja muuttaa vahingossa koko perheen osoitteen. Hän huomaa virheen, mutta uudet maksulaput lähetetään silti seuraavana yönä automaattisesti. Virhe on korjattava "muuttamalla" perhe uudelleen ja taas lähtee uudet maksulaput perheelle.	3. Komentojen peruminen 21. Perumisen tukeminen	8. Selkeät virheilmoitukset	Toimintaprosessi on ehkä liiankin automatisoitu. Tekemistään virheet pitäisi pystyä korjaamaan ja perumaan toiminto.
Kahviautomaatti laskee kahvit maahan, jos kupit ovat loppuneet tai antaa pelkkää vettä, jos jauhe on loppunut.	17. Järjestelmän tilan tarkkailu 23. Resurssien tarkistaminen	9. Vältä virhetilanteita	Poikkeustilanteet tulisi huomata ja jos jokin resurssi on loppunut, toimintoa ei pitäisi pystyä suorittamaan.
Kuva pomppii tekstinkäsittelyohjelmassa sivulta toiselle ennalta arvaamattomasti.	23. Resurssien tarkistaminen		Ohjelman tulisi tarkastaa, mahtuuko kuva sivulle vai ei ja jos kuva ei mahdu sivulle, siirtää se seuraavalle sivulle.

Uusi automalli lämmittää itse itsensä, kun ulkona on tarpeeksi kylmä. Auto ei kuitenkaan lämpene, jos ulkolämpötila on -25 astetta tai alle, koska oletetaan, että kukaan ei halua ajaa autolla niin pakkasella. Suomessa on kuitenkin usein talvella pakkasta enemmän kuin tuo lukema.	12. Kansainvälisen käytön tukeminen 22. Epätavallisten käyttöympäristöjen huomioiminen		Suunnittelijat ovat tehneet käyttäjistä ja heidän toimintatavoistaan liian pitkälle meneviä päätelmiä. Erikoistilanteet ja eri maiden olosuhteet olisi huomioitava. Käyttäjille mahdollisuus itse määritellä lämpötila, jonka jälkeen autoa ei lämmitetä.
Kun erillisissä tekstitiedostoissa olevista kappaleista on koottu kirja ja määritelty kirjalle sivunumerot, erillisissä tiedostoissa olevilla kappaleilla on sama sivunumerointi kuin kirjassa, eikä niitä voi enää muuttaa			Eri tiedostoissa olevat kappaleet tulisi pystyä numeroimaan myös erikseen.

Taulukko 5: Ohjelman toimintatavasta johtuvat ongelmat

9.3 Käyttäjän toimintaprosessia vaikeuttavat ohjelman ominaisuudet

Monissa ohjelmissa on myös ominaisuuksia, jotka tavalla tai toisella vaikeuttavat käyttäjien toimintaprosessia (Taulukko 6). Ohjelma ei siis mukaudu ihmisten toimintaan, vaan ihmisten on mukautettava toimintansa ohjelman toimintatapaan soveltuvaksi. Suurin osa tämän tyyppisistä ongelmista olisi voitu välttää seuraamalla todellisten loppukäyttäjien työskentelyä ja toimintaprosessien suorittamista. Syynä ongelmiin on osaltaan varmasti myös se, että kaikki käyttäjäryhmät ja heidän vaatimuksensa ohjelman suhteen on pyritty ottamaan huomioon, mutta ohjelmaan ei ole kuitenkaan rakennettu erilaisia näkyvyystasoja tms. erilaisille käyttäjille. Tästä syystä monet käyttäjäryhmät joutuvat käsittelemään myös sellaista tietoa ja välivaiheita, joita he eivät tehtäviä suorittaessaan tarvitse.

On kuitenkin syytä muistaa, että syy siihen, että erilaisia näkyvyystasoja tai muita ratkaisutapoja ei ole hyödynnetty, ei ole välttämättä ohjelmiston rakentajassa ja toimittajassa. Syy voi olla myös siinä, että ohjelmiston maksaja on usein eri kuin ohjelmiston käyttäjä, ja maksaja tahtoo luonnollisesti saada tuotteen mahdollisimman edullisesti. Usein unohtuu, että käytettävyydeltään hyvä järjestelmä tai tuote tehostaa työntekoa ja tekee työskentelystä työntekijöilleen mukavampaa (ks. kappale 3). Käyttäjän toimintaprosessia vaikeuttavien ongelmien kohdalla Nielsenin määrittelemistä heuristisista säännöistä (Liite 2) ei löytynyt useaan kohtaan sopivaa sääntöä, joten tulevaisuudessa olisi mielenkiintoista miettiä, voisiko toimintoprosessien toteutumisen arviointiin löytyä omat heuristiset sääntönsä.

Ongelman kuvaus	Toteutumattomat skenaariot	Huomioimattomat heuristiset säännöt	Ehdotus ongelman korjaamiseksi
Pakotetaan käyttäjä tekemään aloitettu tehtävä loppuun ennen uuden aloittamista, vaikka työn kannalta olisi tarpeen joskus tehdä tehtäviä rinnakkain.	15. Samanaikaisten toimintojen tukeminen		Käyttäjien toimintatapojen ja toimintaprosessien huomioiminen tarpeeksi ajoissa.
Eri ohjelmien välillä ei voi "pomppia", vaan on käytettävä yhtä ohjelmaa kerrallaan.	4. Sovellusten samanaikainen käyttö 15. Samanaikaisten toimintojen tukeminen		Käyttäjien toimintatapojen ja toimintaprosessien huomioiminen tarpeeksi ajoissa.
Jonkin toiminto-kokonaisuuden suorittamisessa on paljon turhia välivaiheita, koska ohjelmalla on erilaisia käyttäjiä, joilla on omat vaatimuksensa ohjelman toiminnallisuuteen.	1. Tiedon koostaminen 2. Komentojen koostaminen 14. Käyttöliittymien muuntelu 17. Järjestelmän tilan tarkkailu	7. Oikopolkujen käytön mahdollistaminen	Erilaiset käyttöliittymät erilaisia toimintoja tarvitseville käyttäjäryhmille tai käyttäjille mahdollisuus itse valita haluamansa toiminnot ja tiedot.
Näytöllä paljon turhaa ja toistuvaa tietoa. Ei tukea erilaisille käyttäjäryhmille	1. Tiedon koostaminen 14. Käyttöliittymien muuntelu		Erilaiset näytöt erilaisia tietoja tarvitseville. Käyttäjälle mahdollisuus valita tiedot, joita hän kulloinkin haluaa tarkastella.
Luentosalin piirtoheitin ja valot on kytkettävä ja sammutettava tietokoneesta. Piirtoheittimessä on virtakytkin, mutta sitä ei saa käyttää.	13. Aikaisempien tietojen hyväksikäyttö		Mahdollistetaan sekä manuaaliset että tietokoneohjatut laitteiden käytöt.
Käyttäjä joutuu syöttämään samat tiedot useaan eri ohjelmaan.	11. Tietojen uudelleenkäyttö 15. Monien samanaikaisten toimintojen tukeminen		Ohjelmista tulisi tehdä toisiinsa integroituvia, tai olisi muuten tehtävä ns. työpöytäintegraatiota.
Tehtäessä ohjelmalla kalvoja, tekstin koko tai fontti muuttuu automaattisesti. Määrittelyjä voi muuttaa, mutta sen tekeminen on hankalaa.	1. Tiedon koostaminen 10. Hyvän avun tarjoaminen 25. Näyttöjen tekeminen helppopääsyisiksi	5. Anna käyttäjälle palautetta toiminnoista 10. Riittävä ja selkeä apu	Ohjelma olettaa liikaa, mitä käyttäjä haluaa tehdä. Omien määritysten tekemisen täytyisi olla helpompaa kokemattomallekin käyttäjälle.
Lasku on niin epäselvä, että siitä ei saa tarvittavia tietoja selville, eikä aiheutumisperustetta voi todentaa.	10. Hyvän avun tarjoaminen 13. Aikaisempien tietojen hyväksikäyttö	1. Yksinkertainen ja luonnollinen kieli 2. Käyttäjien kieli 10. Riittävä ja selkeä apu	Laskun saajia on hyvin monenlaisia. Alaan tarkemmin perehtymättömien on hankala ymmärtää alan omaa sanastoa. Laskutietojen yksinkertaistaminen. Lukuohjeet.
Budjettiohjelmassa ei tukea "entä-jos"-kokeiluille. Kokeilut on tehtävä ja laskettava käsin.	21. Perumisen tukeminen	6. Käyttäjille selkeät poistumistavat	Mahdollistetaan kokeilut siten, että tietoja on mahdollista muuttaa ja lisäillä ennen lopullisen budjetin kirjaamista.
Käyttäjä joutuu laskemaan lukuarvot laskimella, koska ohjelma ei osaa tehdä laskutoimituksia.			Jos ohjelman kenttiin syötetään arvoja ja niistä pitää laskea jotain, ohjelman tulisi tehdä se itse.

Taulukko 6: Käyttäjän toimintaprosessia vaikeuttavat ominaisuudet

9.4 Käyttäjästä johtuvat ongelmat

On olemassa myös ongelmatilanteita, jotka johtuvat käyttäjän tietämättömyydestä tai inhimillisestä erehdyksestä (Taulukko 7). Vaikka voidaankin todeta, että nämä ongelmat johtuvat käyttäjästä, ei syy kuitenkaan ole käyttäjän. Ohjelman tai tuotteen käytettävyydessä on parantamisen varaa, jos ihmiset eivät sitä osaa käyttää. Tällaisissa tilanteissa on syytä miettiä, onko ohjeistus ja eritasoiset käyttöohjeet tarpeeksi kattavia (ks. kappale 7.8.4).

On myös muistettava, että joitakin virheitä voi olla suunnittelun aikana hankala ennustaa ja toisaalta korjata, jos sellainen kaikesta huolimatta tapahtuu. Virheen korjaamisen mahdollistaminen voisi esimerkiksi altistaa muille virheille tai jopa väärinkäytöksille. Esimerkiksi tilanteessa, jossa laskunmaksaja on tehnyt tilinumeroa syöttäessään virheen ja rahat menevät väärälle tilille, voi virheen korjaaminen olla erittäinkin hankalaa, jos rahat saanut tilin omistaja ei tahdo palauttaa saamiaan rahoja. Virhettä voidaan tuskin korjata ohjelmallisesti sen jälkeen, kun se on ehtinyt tapahtua, huomiota tulisikin kiinnittää enemmän siihen, miten tällaisia virheitä ennaltaehkäistään.

Ongelman kuvaus	**Toteutumattomat skenaariot**	**Huomioimattomat heuristiset säännöt**	**Ehdotus ongelman korjaamiseksi**
Kannettavan tietokoneen sulkeminen (painettava virtanappia tarpeeksi kauan)	10. Hyvän avun tarjoaminen	10. Riittävä ja selkeä apu	Riittävä apu myös niihin toimintoihin, jotka voivat vaikuttaa yleisiltä, kaikkien tiedossa olevilta asioilta.
Maksettaessa lasku automaatilla tai Internetissä, rahat voivat mennä väärälle tilille, jos tilinumero on väärä. Rahoja ei välttämättä saa enää takaisin, jos rahat saanut ei tahdo niitä palauttaa.	10. Hyvän avun tarjoaminen	9. Vältä virhetilanteita 10. Riittävä ja selkeä apu	Tarpeeksi hyvät ohjeet ja muistutukset siitä, että tilinumeron on oltava oikein.
Ravintoloiden yksittäispakatut sokeripalat on hankala avata (taitettava keskeltä kahtia).	10. Hyvän avun tarjoaminen	10. Hyvä ja riittävä apu	Idea on hyvä ja toimiva, mutta ihmiset eivät osaa avata pakkausta. Riittävä mainostus voisi ratkaista ongelman.

Taulukko 7: Käyttäjästä johtuvat ongelmat

9.5 Muista syistä aiheutuvat käytettävyysongelmat

Muista syistä aiheutuvat käytettävyysongelmat ovat ongelmia, jotka johtuvat ohjelmistosta tai tuotteesta riippumattomista syistä (Taulukko 8). Esimerkiksi Internet-yhteyden kaatuminen aiheuttaa sen, että jonkin ohjelman lataaminen keskeytyy. Toisaalta esimerkiksi sähköt voivat yllättäen katketa aiheuttaen luonnollisesti käytettävyysongelmia. Tällaisissa tilanteissa huomiota tulee kiinnittää erityisesti virheistä toipumiseen, jotta käyttäjälle aiheutuvat vauriot jäisivät mahdollisimman pieniksi. Toisaalta kilpailu ostajista pakottaa erottumaan joukosta jollakin tavalla ja tämä aiheuttaa monesti käytettävyysongelmia loppukäyttäjälle, kun halutut tiedot ja toiminnot löytyvät jokaisesta tuotteesta hieman eri paikoista (ks. kappale 7.5).

Ongelman kuvaus	**Toteutumattomat skenaariot**	**Huomioimattomat heuristiset säännöt**	**Ehdotus ongelman korjaamiseksi**
Uudessa matkapuhelinmallissa numeronäppäimet ovat ympyrän muotoon sijoitettuna. Laitetta on lähes pakko käyttää kahdella kädellä, koska sormet eivät muuten ylety näppäimille.	13. Aikaisempien tietojen hyväksikäyttö	1. Yksinkertainen ja luonnollinen kieli	Käytettävyyden asettaminen muotoilun ja yksilöllisyyden tavoittelun edelle. Käytetään hyväksi muita kilpailukeinoja.
Internet-yhteys katkeaa kesken ohjelman/musiikin lataamisen ja koko toiminta on aloitettava alusta.	8. Virheistä toipuminen		Ongelma johtuu ohjelmasta riippumattomista tahoista, joten sitä on hankalaa korjata.
Palvelin kaatuu juuri, kun käyttäjällä olisi kiire saada jokin tehtävä tehdyksi.	8. Virheistä toipuminen		Ongelma johtuu ohjelmasta riippumattomista tahoista, joten sitä on hankalaa korjata.

Taulukko 8: Muista syistä aiheutuvat ongelmat

10 POHDINTA

Käytettävyyden suunnittelu ja rakentaminen ovat haasteellisia tehtäviä ohjelmistotuotantoprosessissa. Tehtävän haasteellisuudesta viestii jo käytettävyys-termin monenlaiset ja varsin kirjavat määritelmät. Käytettävyydellä voidaan tarkoittaa käyttöliittymän käytettävyyttä, ohjelmiston tai tuotteen virheettömyyttä, järjestelmän saatavuutta ja kaikkea siltä väliltä.

Osaltaan käytettävyysongelmat johtuvat siitä, että käyttäjien todellisia toimintaprosesseja ja tehtäviä ei huomioida tarpeeksi hyvin suunnitteluprosessin alkuvaiheessa. Toisaalta, vaikka käyttäjät otettaisiinkin alussa hyvin huomioon, he saattavat unohtua prosessin aikana välillä lähes tyystin. Lisäksi käytettävyyden huomioimiseen vaikuttavat vallalla olevat asenteet suunnittelijoiden keskuudessa. Aivan liian usein käytettävyysasioita pidetään helppoina ja itsestään selvinä asioina, joihin ei kannata tuhlata sen enempää aikaa kuin muitakaan resursseja. Käytettävyys ja todelliset loppukäyttäjät tulisi kuitenkin huomioida läpi koko ohjelmistotuotantoprosessin, sillä prosessin alkuvaiheessa määritellyt käytettävyysvaatimukset on myös rakennettava jossakin vaiheessa, jotta asetetut käytettävyystavoitteet voidaan saavuttaa. On myös muistettava, että jokaisessa ohjelmistotuotantoprosessin vaiheessa tehdään merkittäviä päätöksiä, jotka vaikuttavat osaltaan myös valmistuvan järjestelmän käytettävyyteen. Esimerkiksi tilanteeseen sopivimman arkkitehtuurityylin tai sopivien komponenttien valinta ja sijoittelu vaikuttavat merkittävästi järjestelmän käytettävyyteen.

Kaikki syy käytettävyysongelmiin ei kuitenkaan ole suunnittelijoissa, vaan ongelmia aiheuttaa myös se, että tuotteen maksajat ovat usein eri tahoja kuin tuotteen todelliset käyttäjät. Tämän vuoksi todellisten käyttäjien tarpeet voivat jäädä huomioimatta, sillä lienee sanomattakin selvää, että maksaja haluaa saada tuotteensa mahdollisimman halvalla ja mahdollisimman nopeasti. Tällöin tarpeeksi kattavia käyttäjä- ja tehtäväanalyysejä sekä muita käytettävyyden rakentamisessa hyödynnettäviä menetelmiä on lähes mahdotonta toteuttaa.

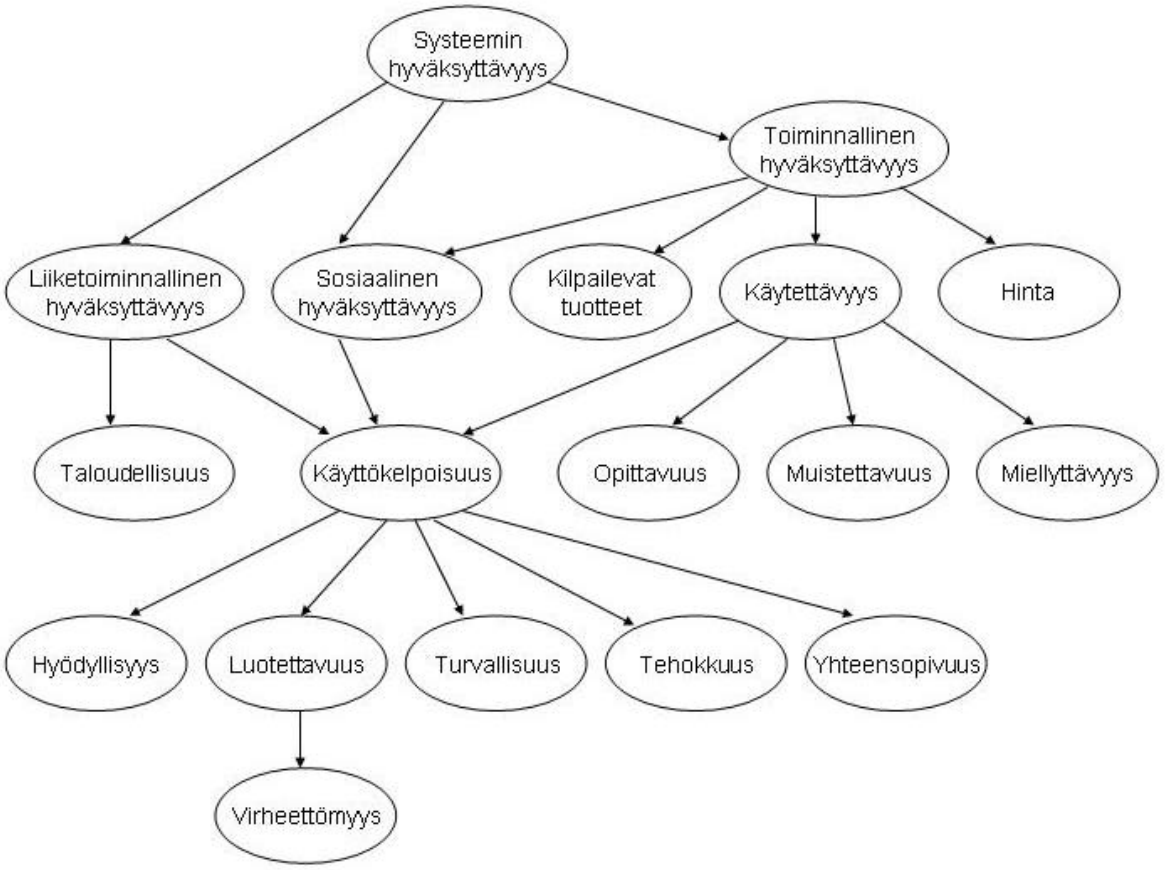
10.1 Käytettävyys käyttäjän näkökulmasta

Tutkimuksen edetessä on käynyt ilmi, että luvussa 2 esitetty Nielsenin näkemys käytettävyyden määrittelemisestä on selvästi ohjelmistokeskeinen (Kuva 2). Vaihdettaessa näkökulma ohjelmiston sijasta järjestelmän käyttäjiin, käytettävyys nousee kuvassa ylöspäin (Kuva 19).

Käyttäjien näkökulmasta systeemin hyväksyttävyys koostuu toiminnallisesta, sosiaalisesta ja liiketoiminnallisesta hyväksyttävyydestä. Liiketoiminnallisella hyväksyttävyydellä tarkoitetaan sitä, että tuotteen tulee olla välttämätön, taloudellinen ja tarpeellinen sen tehtävän suorittamiseen, johon sitä käytetään. Tuotteen täytyy siten soveltua yrityksen tavoitteisiin. Toiminnallinen hyväksyntä vaikuttaa osaltaan myös sosiaaliseen hyväksyntään, eli toiminnoiltaan hyvin työtä tukeva järjestelmä varmasti saavuttaa helpommin myös sosiaalisen hyväksynnän. Lisäksi toiminnalliseen hyväksyntään vaikuttavat muut markkinoilla olevat vastaavat kilpailevat tuotteet ja niiden toiminnallisuus sekä tuotteen käytettävyys ja hinta.

Nielsenin ajatuksesta poiketen käytettävyys nousee tässä näkökulmassa käyttökelpoisuuden yläpuolelle. Ajatuksena on, että tuote ei ole käytettävä, jos se ei ole käyttökelpoinen käyttöympäristössään, eikä sovellu niihin tehtäviin, joita sen avulla halutaan suorittaa. Käyttökelpoisuuden lisäksi käytettävyyteen vaikuttaa opittavuus, muistettavuus ja miellyttävyys. Lisäksi Nielsen erottaa käytettävyyden ja hyödyllisyyden toisistaan, mutta voidaan myös ajatella, että hyödyllisyys kuuluisi yhtenä osana tuotteen käyttökelpoisuuteen ja sitä kautta tuotteen käytettävyyteen. Voiko tuote olla hyödyllinen, eli onko siitä merkittävästi apua tai hyötyä tehtävän suorittamisessa, jos se ei ole käytettävä? Toisaalta, onko tuotteen käytettävyydellä mitään arvoa, jos tuote ei ole hyödyllinen? Jos käytettävyys ja hyödyllisyys ovat erillään toisistaan, kärjistäen voitaisiin ajatella, että esimerkiksi liiketoiminnan kannalta tarpeeksi hyödyllisen tuotteen ei edes tarvitse olla käytettävä.

Käyttökelpoisuus rakentuu hyödyllisyyden lisäksi luotettavuudesta, turvallisuudesta, tehokkuudesta ja yhteensopivuudesta. Luotettavuuteen liittyy osaltaan tuotteen virheettömyys. Turvallisuus vaikuttaa käyttökelpoisuuteen osaltaan siksi, että joissakin työskentelyympäristöissä turvallisuus on merkittävä tekijä, sillä kriittisissä tilanteissa heikko käytettävyys voi aiheuttaa työskentelyprosessin vaikeutumista ja johtaa jopa onnettomuuksiin [Saa02]. Tuotteen käyttökelpoisuuteen vaikuttava tehokkuus voi yksittäisen käyttäjän kohdalla olla lähes merkityksettömältä vaikuttava ominaisuus, mutta mitä suurempi työyhteisö, sitä suurempia tehokkuudesta saatavat hyödyt ovat. Yhteensopivuus on myös merkittävä tekijä käyttökelpoisuuden kannalta, sillä yhä useammassa työssä työntekijä joutuu käyttämään useampia ohjelmia samanaikaisesti. Luonnollisesti muiden ohjelmien kanssa yhteensopiva ohjelma on tehtävään käyttökelpoisempi ja käytettävämpi kuin muita häiritsevä tai muuten yhteensopimaton ohjelma tai järjestelmä.



Kuva 19: Käytettävyys käyttäjän näkökulmasta

10.2 Lisätutkimusta

Käytettävyyden alueella on vielä paljon asioita, joihin tulevaisuudessa tulisi kiinnittää huomiota ja joita pitäisi tutkia lisää. Tämän tutkimuksen näkökulmassa ei ole kiinnitetty juurikaan huomiota monen käyttäjän systeemeihin, vaan huomio on kiinnittynyt enemmän yksittäisen käyttäjän kohtaamiin ongelmiin. Käyttäjien välisellä vuorovaikutuksella ja yhteistoiminnallisuudella on varmasti omat vaikutuksensa ohjelmiston käytettävyyteen. Lisäksi ihmisten erilaiset tehtäväkokonaisuudet ja niiden vaikutus käytettävyyteen jäivät tässä tutkimuksessa vähälle huomiolle. Myös se, millaisia käytettävyysongelmia hajautetuissa järjestelmissä on, tai kuinka käytettävyys voitaisiin huomioida hajautettuja järjestelmiä rakennettaessa, olisivat varmasti mielenkiintoisia jatkotutkimushaasteita.

Taulukossa 3 (ks. s.71) kuvataan yhteenvedonomaisesti erilaisia arkkitehtuuriratkaisuja ja käytettävyysskenaarioita, joiden avulla voidaan saavuttaa monenlaisia käytettävyyden hyöty-

jä. Jatkotutkimusaiheeksi on jätetty, mitkä käytettävyysskenaariot toteutuvat x:llä merkatuissa kohdissa, ja toisaalta kysymysmerkillä on osoitettu ne kohdat, joihin myös voi olla positiivista vaikutusta, kun menetelmää hyödynnetään. Lisäksi olisi mielenkiintoista pohtia, miten taulukko muuttuu, jos näkökulmaa laajennetaan koskemaan hajautettuja ja monen käyttäjän systeemejä. Onko näiden systeemien käytettävyyden parantamiseksi olemassa vielä lisää menetelmiä, ovatko menetelmät toisenlaisia, tai voidaanko käytettävyyteen panostamalla saavuttaa myös muunlaisia hyötyjä kuin taulukossa mainitut?

Näkökulman ollessa monen käyttäjän systeemeissä ja hajautetuissa järjestelmissä täytyy myös miettiä, miten hyvin liitteessä 1 kuvatut käytettävyysskenaariot ja liitteessä 2 kuvatut heuristiset säännöt soveltuvat näihin yhteyksiin, vai tarvitseeko niitä muokata ja laajentaa. Lisäksi voisi miettiä, onko käyttäjien toimintoprosessien tukemisen arvioimiseksi mahdollista kehittää omat heuristiset sääntönsä, kuten käyttöliittymän arvioimiseksi on kehitetty.

LÄHTEET

[BaB01] Bass Len, Bonnie John E., Kates Jesse: *Achieving Usability Through Software Architecture.* Technical report, Carnegie Mellon Software Engineering Institute, Pittsburgh, 2001. Viitattu 24.11.2002. Saatavilla www-muodossa: <URL:http://www.sei.cmu.edu/publications/documents/01.reports/01tr005.htm>

[BaG95] Baecker Ronald M., Grudin Jonathan, Buxton William A. S., Greenberg Saul: *Readings in human-computer interaction: toward the year 2000.* Morgan Kaufmann Publishers, Inc., San Francisco 1995.

[Ber96] Berztiss Alfs, *Software Methods for Business Reengineering*. Springler-Werlag, New York, 1996.

[Bos00] Bosch Jan: *Design and Use of Software Architectures: Adopting and evolving a product-line approach.* Addison-Wesley, England, 2000.

[BuM01] Buschmann Frank, Meunier R., Rohnert H., Sommerland P., Stal M.: *Pattern-Oriented Software Architecture, A System of Patterns.* John Wiley & Sons, 2001.

[Chr00] Chrusch Marc: *Seven Great Myths of Usability.* Interactions: new visions of human-computer interaction, september+october 2000, s. 13-16.

[Eer03] Eerola Anne: *Ohjelmistotuotteeseen käytettävyyttä ohjelmistotuotantoprosessia kehittämällä.* "Käytettävyys – Kuka tekee ja mitä Centek-verkostossa" -seminaari, Kuopio 30.1.2003.

[FoS97] Fowler Martin, Scott Kendall: *UML Distilled: Applying the Standard Object Modeling Language.* Addison-Wesley, USA, 1997.

[GaH02] Gamma Erich, Helm Erich, Johnson Ralph, Vlissides John: *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley, Holland 2002.

[Gra02] Grand Mark: *Java Enterprise Design Patterns*. John Wiley & Sons, Inc., New York 2002.

[GrK91] Greenbaum Joan, Kyng Morten: *Desingn at Work: Cooperative Design of Computer Systems.* Lawrence Erlbaum Associates, USA, 1991.

[HaM01] Haikala Ilkka, Märijärvi Jukka: *Ohjelmistotuotanto.* Talentum Media Oy, Pieksämäki, 2001.

[HaR98] Hackos JoAnn T., Redish Janice C.: *User and Task Analysis for Interface Design.* Wiley, New York, 1998.

[HeC01] Heineman George T., Councill William T.: *Component-Based Software Engineering, Putting the Pieces Together.* Addison-Wesley, Boston, 2001.

[Inf02] Information & Design: *What is Scenario?* Information & Design, Melbourne, 2002. Viitattu 29.11.2002. Saatavilla www-muodossa:
<URL: http://www.infodesign.com.au/usability/scenarios.html>

[Imm02] Immonen Mirja: *Suunnittelumallit.* Erikoistyö, Kuopion yliopisto, tietojenkäsittelytieteen ja sovelletun matematiikan laitos, 2002. Viitattu 31.1.2003. Saatavilla www-muodossa:
<URL: http://www.uku.fi/atkk/plugit/julkaisut/docs/Immonen-2002.pdf>

[Iso02] Isomäki Hannakaisa: *The Prevailing Conceptions of the Human Being in Information Systems Development: Systems Designers' Reflections.* Väitöskirja, Acta Electronica Universitatis Tamperensis, Tietojenkäsittelytieteiden laitos, Tampereen yliopisto, 2002. Viitattu 27.1.2003. Saatavilla www-muodossa:
<URL:http://acta.uta.fi/pdf/951-44-5388-3.pdf>

[Jär02] Järvi Antero: *Arkkitehtuurisuunnittelu.* Luentomoniste, Turun yliopisto, informaatioteknologian laitos, 2002. Viitattu 29.1.2003. Saatavilla www-muodossa:
<URL:http://staff.cs.utu.fi/kurssit/ohjelmistotuotanto/syksy_2002/
luennot02/Arkkitehtsuun(12).pdf>

[KoN94] Koivunen Marja-Riitta, Nieminen Marko: *Käytettävyys tietojärjestelmien suunnittelussa ja toteutuksessa.* Sytyke ry – Systeemityö, 3/1994, s. 6-12.

[Kuj99] Kujala Sari: *User-centered requirements definition activities.* Reports of the Qure Project, Helsingin Teknillinen korkeakoulu, 1999. Viitattu 14.4.2003. Saatavilla www-muodossa:
<URL:http://www.soberit.hut.fi/qure/reports/user-centered.pdf>

[Kuj02] Kujala Sari: *User Studies: A Practical Approach to User Involvement for Gathering User Needs and Requirements.* Väitöskirja, Acta Polytechnica Scandinavica, Mathematics and Computing Series No. 116, Helsingin Teknillinen korkeakoulu, Tietotekniikan osasto, Espoo, 2002. Viitattu 14.12.2002. Tiivistelmä saatavilla www-muodossa:
<URL:http://lib.hut.fi/Diss/2002/isbn9512259001/>

[Käh00] Kähkipuro Pekka: *Komponenttiarkkitehtuurien vaikutus ohjelmistotuotantoon.* Sytyke ry – Systeemityö, 1/2000. Viitattu 5.2.2003. Saatavilla www-muodossa:
<URL:http://www.pcuf.fi/sytyke/lehti/kirj/st20001/st001.pdf>

[Laa02] Laakso Sari: *Käyttöliittymien arviointimenetelmät*. Luentomoniste, Helsingin yliopisto, tietojenkäsittelytieteen laitos, 2002. Viitattu 22.1.2003. Saatavilla www-muodossa:
<URL:http://www.cs.helsinki.fi/u/salaakso/kl2-2002/lahteet/arviointi.html>

[Nie93] Nielsen Jacob: *Usability Engineering*. Academic Press, London, 1993.

[Nie98] Nieminen Marko: *Käytettävyys – osa systemaattista tuotekehitystä.* Sytyke ry – Systeemityö, 4/1998.

[Nor91] Norman Donald A.: *Miten avata mahdottomia ovia? Tuotesuunnittelun salakarit*. Weilin+Göös, Jyväskylä, 1991 (suomentanut Annu Jämes).

[Ova02] Ovaska Saila: *Käytettävyyden perusteet*. Luentomoniste Tampereen yliopisto, tietojenkäsittelytieteiden laitos, 2002. Viitattu 27.1.2003. Saatavilla www-muodossa:
<URL:http://www.cs.uta.fi/~ov/usab/luennot/pdf/4-arviointi.pdf>

[Pre94] Preece Jenny: *Human-Computer Interaction*. Addison-Wesley, Wokingham, 1994.

[RaW97] Rauste-von Wright, Marja-Liisa, von Wright, Johan: *Oppiminen ja koulutus*. WSOY:n graafiset laitokset, Juva, 1997.

[Rii00] Riihiaho Sirpa: *Experiences with usability evaluation methods*. Lisensiaattityö, Helsingin Teknillinen korkeakoulu, Tietotekniikan osasto, 2000.

[Rii02a] Riihiaho Sirpa: *Käytettävyyden arviointi ilman käyttäjiä.* Teknillisen korkeakoulun käytettävyyslaboratorion julkaisuja, 2002. Viitattu 22.1.2003. Saatavilla www-muodossa:
<URL:http://www.soberit.hut.fi/T-121/T121.600/asiantuntija-arviot.pdf>

[Rii02b] Riihiaho Sirpa: *Käytettävyystestauksen muunnelmia.*. Teknillisen korkeakoulun käytettävyyslaboratorion julkaisuja, 2002. Viitattu 22.1.2003. Saatavilla www-muodossa:
<URL:http://www.soberit.hut.fi/T-121/T121.600/muunnelmat.pdf>

[Rob99] Robertson Suzanne, Robertson James: *Mastering the Requirements Process*. Addison-Wesley, UK, 1999.

[Rou02] Routio Pentti: *Tuote ja tieto: Tuotteiden tutkimuksen ja kehittämisen opas.* Taideteollinen korkeakoulu, 2002. Viitattu 22.11.2002. Saatavilla www-muodossa:
<URL:http://www.uiah.fi/projects/metodi/010.htm>

[Saa02] Saariluoma, Pertti: *Kognitiivinen käytettävyystutkimus: Käyttäjän malli.* Proposals Archive: Fuego Future Mobile and Ubiquitous Computing Research Prog-

ram, Helsingin Teknillinen korkeakoulu, 2002. Viitattu 29.11.2002. Saatavilla www-muodossa:
<URL:http://www.hut.fi/~pma/hiit/fuegoproposals/kognitio/Saariluoma.pdf>

[Sin98] Sinkkonen Irmeli: *Yrityksen käyttöliittymästandardi*. Sytyke ry – Systeemityö, 4/1998. Viitattu 23.1.2003. Saatavilla www-muodossa:
<URL:http://www.pcuf.fi/sytyke/lehti/kirj/st19984/15.pdf>

[Sin02] Sinkkonen Irmeli: *Käytettävyyssanasto.* Adage Oy, 2002.
Viitattu 23.1.2003. Saatavilla www-muodossa:
<URL:http://www.adage.fi/artikkelit/kaytettavyyssanasto.html>

[SiK02] Sinkkonen Irmeli, Kuoppala Hannu, Parkkinen Jarmo, Vastamäki Risto: *Käytettävyyden psykologia*. IT-Press, Helsinki, 2002.

[Sne03] Snellman Ritva Liisa: *Homo digitalissimus*. Helsingin Sanomat, 23.1.2003.

[Tie01] Tietotekniikan tutkimusinstituutin julkaisuja: *Sanasto*. Jyväskylän yliopisto, Jyväskylä, 2001. Viitattu 3.1.2003. Saatavilla www-muodossa:
<URL: http://www.titu.jyu.fi/julkaisut/julk09/sanasto.htm>

[Toi98] Toiminnan teorian ja kehittävän työntutkimuksen yksikön julkaisuja: *Cultural-Historical Activity Theory.* Kasvatustieteellinen tiedekunta, Helsingin yliopisto, 1998. Viitattu 11.12.2002. Saatavilla www-muodossa:
<URL: http://www.edu.helsinki.fi/activity/6a0.htm>

[Tik91] Tikkala Anneli: *Käsiteanalyysi ja relaatiotietokannan suunnittelu.* Tutkimusraportti, Lappeenrannan teknillinen korkeakoulu, Lappeenranta, 1991.

[Usa02] UsabilityNet: *International standards for HCI and usability.* UsabiltyNet, 2002. Viitattu 18.11.2002. Saatavilla www-muodossa:
<URL: http://www.usabilitynet.org/tools/r_international.htm>

[Vuo96] Vuori Matti (toim): *Käytettävyys tuotekehityksessä: innovatiivisuutta ja varmistusta*. Usability 2 -hankkeen loppuraportti, Sähkö- ja elektroniikkateollisuusliitto, Helsinki, 1996. Viitattu 16.1.2003. Saatavilla www-muodossa: URL:http://mango2.vtt.fi:84/aut/rm/val45/usabil2/loppurap/lrap.htm

Käytettävyysskenaariot [BaB01] Liite 1

1. **Tiedon koostaminen:** Järjestelmän tulisi sallia käyttäjän valita ja toimia mielivaltaisella tietojen yhdistelmällä. Esimerkiksi silmälääkäri voi haluta nähdä potilaistaan hieman eri tietoja potilastietojärjestelmästä kuin nukutuslääkäri.

2. **Komentojen koostaminen:** Järjestelmä tarjoaa sarjoja tai käyttäjälle mahdollisuuden tehdä käsky-yhdistelmiä (macros), jotta jokin pitkä toimintosarja voidaan suorittaa vähemmällä määrällä komentoja.

3. **Komentojen peruminen:** Järjestelmä sallii käyttäjän perua antamiaan komentoja.

4. **Sovellusten samanaikainen käyttö:** Järjestelmä mahdollistaa erilaisten sovellusten samanaikaisen käytön tuottamatta ongelmia käyttäjälle.

5. **Virheettömyyden tarkistaminen:** Järjestelmä huomaa ja korjaa tyypillisimmät käyttäjien tekemät virheet. Kontekstista riippuen korjaaminen voidaan toteuttaa suoraan tai ilmoittaa virheestä käyttäjälle ja kehottaa korjaamaan.

6. **Laiteriippumattomuuden ylläpitäminen:** Sovellus pitäisi voida käynnistää eri ympäristöissä. Järjestelmä tulisi suunnitella niin, että laitteistoristiriitojen vakavuus ja yleisyys pienenisivät.

7. **Järjestelmän arvioiminen:** Järjestelmissä tulisi olla testipisteitä (test points) ja tiedon koostamisen mahdollisuus, jotta arvioiminen helpottuisi.

8. **Virheistä toipuminen:** Käyttäjälle tulisi tarjota mahdollisuuksia vähentää järjestelmävirheiden takia katoavan työn määrää.

9. **Unohdettujen salasanojen saaminen takaisin:** Käyttäjällä täytyy olla jokin mahdollisuus saada takaisin unohdettu salasana.

10. **Hyvän avun tarjoaminen:** Apumenettelytapojen tulisi olla kontekstiriippuvaisia ja riittävän täydellisiä auttaakseen käyttäjää ongelmanratkaisussa.

11. **Tietojen uudelleenkäyttö:** Käyttäjä voi haluta siirtää tietoa järjestelmän osasta toiseen, tai hän voi käyttää kahta sellaista sovellusta samaan aikaan, jotka käyttävät samoja tietoja. Järjestelmän tulisi mahdollistaa tietojen syöttäminen vain yhteen kohtaan tai sovellukseen ja tiedot olisivat silti käytössä molemmissa paikoissa.

12. **Kansainvälisen käytön tukeminen:** Järjestelmän tulisi olla helposti konfiguroitavissa useisiin kulttuureihin.

13. **Ihmisen aikaisempien tietojen hyväksikäyttö**: Tuottavuus paranee ja opetteluaika pienenee, kun mahdollistetaan käyttäjien käyttää vanhoja tietoja hyväkseen uudessa versiossa. Esimerkiksi valikkojen nimet ja paikat tulisi säilyttää samoina kuin vanhemmissa versioissa.

14. **Käyttöliittymien muuntelu:** Käyttöliittymien tulisi olla helposti muokattavissa.

15. **Monien samanaikaisten toimintojen tukeminen:** Järjestelmän tai sen sovellusten tulisi sallia käyttäjän siirtyä nopeasti tehtävästä toiseen ja takaisin.

16. **Yksittäisellä näytöllä navigointi:** Käyttäjien täytyy pystyä navigoimaan näytöllä tehokkaasti ja odotusaikojen tulisi olla kohtuullisia.

17. **Järjestelmän tilan tarkkailu:** Käyttäjän tulisi olla selvillä, missä järjestelmän tilassa hän on. Suunnittelijoiden tulisi ottaa huomioon käyttäjien tarpeet ja kyvyt päättäessään, mitä näkökulmia järjestelmän tilasta näytetään ja miten ne esitetään.

18. **Käyttäjän tahdilla työskentely:** Järjestelmän tulisi sopeutua käyttäjän työskentelynopeuteen. Käyttäjällä tulisi myös olla mahdollisuus muuttaa tätä nopeutta.

19. **Tehtävän keston arvioiminen:** Järjestelmän täytyisi antaa arvio suorittamansa tehtävän kestosta, jotta käyttäjä voi esimerkiksi pitkäkestoisen tehtävän suoritusaikana tehdä jotain muuta.

20. **Monipuolisen ja kokonaisvaltaisen hakukomentojen tukeminen:** Järjestelmän tulee tukea käyttäjän monipuolisia, kokonaisvaltaisia ja spontaaneja hakukomentoja tiedonhakuun tietovarastosta.

21. **Perumisen tukeminen:** Käyttäjän on pystyttävä perumaan edellinen toiminto ja palaamaan toimintoa edeltäneeseen tilaan. Myös peruttua toimintoa edeltäneen toiminnon peruutusmahdollisuus olisi suotavaa.

22. **Epätavallisessa käyttöympäristössä työskentely:** Järjestelmän tulisi tukea erilaisissa ympäristöissä tapahtuvaa työskentelyä ja tarjota tarpeeksi apua uudessa ympäristössä työskentelevälle.

23. **Resurssien tarkistaminen**: Sovellusten tulee tarkistaa, että kaikki tarvittavat resurssit ovat saatavilla, ennen kuin operaatio aloitetaan.

24. **Yhdenmukainen toiminta näytöstä toiseen**: Komentojen tulisi perustua tiedon tyyppiin tai tiedon sisältöön, eikä nykyiseen tietonäkymään, jotta operaatiot ymmärretään myös seuraavalla näytöllä.

25. **Näyttöjen tekeminen helppopääsyisiksi:** Käyttäjät haluavat usein nähdä tiedon useista näkökulmista, esimerkiksi pitkistä dokumenteista halutaan nähdä ensin jäsennys. Tämän vuoksi näyttöjen on oltava helppokäyttöisiä ja -pääsyisiä.

26. **Mielikuvien tukeminen:** Käyttäjät haluavat nähdä tiedon eri näkökulmista. Järjestelmän tulisi tukea näitä erilaisia tehtäväsidonnaisia näkökulmia (esim. esikatselu tekstinkäsittelyohjelmassa).

Heuristiset säännöt [Nie93] Liite 2

1. **Käytä yksinkertaista ja luonnollista kieltä:** Käyttöliittymän tulee olla mahdollisimman yksinkertainen. Käyttöliittymän täytyy tukea käyttäjän tehtäviä mahdollisimman luonnollisella tavalla. Yhteenkuuluvat asiat tulisi esittää myös käyttöliittymässä lähellä toisiaan, vähintään samalla näytöllä. On syytä muistaa, että kaikki turha kilpailee käyttäjän huomiosta. Yksinkertaisuuteen ja luonnollisuuteen päästään graafisen sommittelun ja tiedon ryhmittelyn ja järjestelyn avulla.

2. **Käytä käyttäjien omaa kieltä:** Käyttöliittymissä tulisi suosia käyttäjien omaa kieltä eli ammattisanastoa, käyttöjärjestelmän vakiintunutta sanastoa ja mikäli mahdollista myös käyttäjien omaa äidinkieltä. Kielen ei pitäisi rajoittua pelkästään sanoihin, vaan käyttöliittymässä tulisi olla myös ei-kielellisiä elementtejä, kuten kuvakkeita (icons).

3. **Minimoi käyttäjien muistikuorma:** Tietokoneet pystyvät muistamaan valtavan määrän tietoa tarkasti, joten tietokoneen tulisi kantaa mahdollisimman suuri osa käyttäjän muistikuormasta. Käyttöliittymää suunniteltaessa on syytä muistaa, että ihminen tunnistaa paljon enemmän kuin muistaa asioita. Tämän vuoksi tulisi suosia esimerkiksi valintalistoja ja erilaisia muistivihjeitä.

4. **Tee käyttöliittymästä kauttaaltaan yhdenmukainen:** Komentojen on toimittava yhdenmukaisesti, lisäksi sama tieto on esitettävä samassa paikassa kaikilla näytöillä. Yhdenmukaisuus ei ole pelkästään käyttöliittymän yhdenmukaisuutta, vaan yhdenmukaisuus liittyy myös suoritettaviin tehtäviin ja systeemin muuhun toiminnallisuuteen. Yhdenmukaisuudella on eri tasoja: tuotteen sisäinen yhdenmukaisuus, eri versioiden välinen yhdenmukaisuus ja tuoteperheen yhdenmukaisuus.

5. **Anna käyttäjille palautetta toiminnoista:** Järjestelmän on jatkuvasti annettava käyttäjälle tietoa siitä, mitä systeemi tekee ja miten se tulkitsee käyttäjän antamaa syötettä. Käyttäjän siis tulisi nähdä järjestelmän toimintatila. Palautetta pitäisi antaa jo ennen virhetilanteen ilmaantumista, palaute voi olla myös positiivista. Palautteen tärkeys tulee erityisesti esille tilanteissa, joissa järjestelmällä on pitkä vasteaika tietyille toiminnoille. Jos vasteaika on 0.1–1.0 sekuntia, käyttäjä kokee saavansa vasteen välittömästi, joten erityistä palau-

tetta ei tarvita. Jos vasteaika on yli 10 sekuntia, käyttäjä useimmiten haluaa tehdä jotain muuta vastetta odottaessaan, joten palautteen antaminen on erittäin tärkeää. Toisaalta vasteaika voi olla myös liian lyhyt ja käyttäjä ei ehdi reagoida siihen tarpeeksi ajoissa.

6. **Anna käyttäjille selkeät poistumistavat:** Käyttäjä ei saa tuntea olevansa ansassa kokeiltuaan jotakin toimintoa, vaan käyttöliittymän on tarjottava keino päästä edelliseen tilaan ja mielellään myös alkutilaan. Käyttäjälle on suotava kontrollin tuntu ja mahdollisuus oppia kokeilemalla.

7. **Anna käyttäjälle mahdollisuus käyttää oikopolkuja:** Näkyvien valintavaihtoehtojen lisäksi kokeneille käyttäjille tulisi tarjota mahdollisuus käyttää erilaisia oikopolkuja toimintojen suorittamiseksi. Tällaisia ovat mm. näppäinyhdistelmät, kaksoisnapsautukset, pikakuvakkeet ja oletusarvot.

8. **Anna selkeät virheilmoitukset:** Käyttäjän on pystyttävä huomaamaan, tunnistamaan ja korjaamaan tekemänsä virhe. Virheilmoitukset täytyy sen vuoksi antaa selvällä kielellä ja tavalliselle käyttäjälle epämääräisiä koodeja tulisi välttää. Virheilmoitusten tulee myös olla tarkkoja ja auttaa käyttäjää ratkaisemaan eteen tullut ongelma. Lisäksi hyvä virheilmoitus on kohtelias, eikä pakota tai syytä käyttäjää. Virheilmoituksissa tulisi välttää esimerkiksi sanoja *tuhoisa* (fatal) tai *laiton* (illegal).

9. **Vältä virhetilanteita:** On olemassa paljon käyttäjän tekemiä virheitä, joita voidaan estää hyvällä käyttöliittymäsuunnittelulla. Esimerkiksi valintalistojen avulla vältetään kirjoitusvirheistä aiheutuvia ongelmia. Peruuttamattomiin toimintoihin on aina pyydettävä varmistus käyttäjältä, mutta toisaalta liika varmistusten antaminen voi turruttaa käyttäjän, eikä hän enää jaksa lukea kaikkia varmistuskehotteita vaan vastaa niihin automaattisesti. Erilaisia toimintatiloja tulisi välttää tai tila tulee ilmoittaa selvästi.

10. **Anna riittävä ja selkeä apu:** On syytä huomata, että yleensä käyttöohjeita luetaan vasta suuressa hädässä. Tämän vuoksi käyttöohjeissa on syytä olla selkeä sisällysluettelo, hyvä hakemisto erilaisine hakusanoineen ja ohjeet käyttäjien tehtävien mukaan edeten. Mukana on hyvä olla myös esimerkkejä erilaisista tilanteista.