VAATIMUSTEN VALIDOINTI JA VERIFIOINTI

Heli Lintula
Pro gradu -tutkielma
Tietojenkäsittelytieteen laitos
Informaatioteknologian ja
kauppatieteiden tiedekunta
Kuopion yliopisto
Elokuu 2004

KUOPION YLIOPISTO, informaatioteknologian ja kauppatieteiden tiedekunta
Tietojenkäsittelytieteen koulutusohjelma (muuntokoulutus)
Tietojenkäsittelytiede

LINTULA HELI, K.: Vaatimusten validointi ja verifiointi
Pro gradu -tutkielma, 95 s.
Pro gradu -tutkielman ohjaaja: FT Anne Eerola

Elokuu 2004

Avainsanat: Vaatimusten määrittely, toiminnalliset vaatimukset, ei-toiminnalliset vaatimukset, validointi, verifiointi, validointitekniikat, verifiointitekniikat.

Vaatimusmäärittely on ensimmäinen vaihe, kun rakennetaan uutta tietojärjestelmää tai kun korvataan vanha järjestelmä uudella. Hyvin tehty vaatimusmäärittely helpottaa kustannusten ja hyötyjen arviointia sekä tarjouskilpailun järjestämistä. Puutteellinen vaatimusmäärittely on yksi suurimmista syistä ohjelmistoprojektien keskeytymiseen tai viivästymiseen. Onnistunut vaatimusmäärittely edellyttää todellisten tarpeiden löytämistä ja vaatimusten dokumentointia eri asianosaisten ymmärtämällä tavalla.

Vaatimusten validoinnilla ja verifioinnilla taataan, että asiakkaat saavat sitä, mitä he haluavat ja että rakennettava tuote tehdään oikein. Ohjelmiston validointi- ja verifiointiprosessissa analysoidaan ja testataan ohjelmistoa kehityksen aikana, jotta se toteuttaa kaikki vaadittavat toiminnot. Validointi- ja verifiointiprosessi tuottaa myös tietoa ohjelmiston laadusta ja luotettavuudesta.

Tutkielmassa käydään läpi vaatimuksiin liittyviä asioita, kuten vaatimusten kirjoittamisen vaikeutta, vaatimusten mittaamista, vaatimusmäärittelydokumenttien sisältöä ja laadun arviointia. Tutkimuksessa käsitellään lyhyesti useita eri verifiointi- ja validointitekniikoita. Joitakin tekniikoita tutkitaan tarkemmin. Sellaisia ovat esimerkiksi tarkastus, käyttötapausten ja luokkakaavioiden yhdisteleminen, väärinkäyttötapaukset ja vaatimusten validointi siirryttäessä vanhasta järjestelmästä uuteen reaalimaailman esimerkkien avulla.

SISÄLLYSLUETTELO

1 JOHDANTO

Vaatimusten määrittelyn (requirement engineering) tarkoitus on muodostaa täydellinen, johdonmukainen ja yksiselitteinen *vaatimusmäärittely (requirement specification)* rakennettavan systeemin vaatimuksista käsitteellisellä tasolla. Useitten tutkimusten mukaan on erittäin tärkeää, että vaatimukset ovat oikeita ja ymmärrettäviä heti projektin alkuvaiheessa. Hyvin tehtyjen vaatimusmäärittelyjen ansiosta pystytään tuottamaan laadukkaampia ohjelmistoja matalammilla kustannuksilla ja lyhyemmillä aikatauluilla. Asiakkaat saavat myös sellaisia tuotteita, mitä he ovat halunneet ja mitä he tarvitsevat. Herääkin kysymys: "*Miksi vaatimusten määritteleminen epäonnistuu tai miksi vaatimukset eivät toteudu tehdyssä tuotteessa?*" [Gab99], [Gaa99]

Asiakasvaatimusten perusteella määritellään ohjelmiston ominaisuuksia ja niihin liittyviä toimintoja. Yleensä asiakasvaatimukset tarkentuvat ja uusia vaatimuksia löydetään määrittelytyön aikana ja vielä myöhäisemmissäkin vaiheissa. Ohjelmistotuotantoprosessissa varmistetaan tuotteen laatu riittävällä *testauksella*, *tarkastuksilla, verifioinnilla (todentaminen)* sekä *validoinnilla (kelpoistaminen)*. Testaus määritellään ohjelmistojen testauksen yhteydessä suunnitelmalliseksi virheiden etsimiseksi ohjelmaa tai sen osaa suorittamalla [HaM01]. Tarkastus on tiukasti määritelty prosessi, jossa tarkastettava tuote tarkastetaan tiettyjen sääntöjen ja tarkistuslistojen avulla [Ter01]. Asiantuntijat käyvät läpi dokumentteja ja koodia tarkoituksena löytää virheitä. Verifioinnissa varmistetaan, että tuote vastaa määrittelyään. Validoinnissa taas tutkitaan tuotteen sopivuutta käyttötarkoitukseensa [Som95], [HaM01]. Tässä työssä keskitytään verifiointiin ja validointiin.

Verifioinnissa ohjelmistokehitysprosessin jokaisessa vaiheessa tarkistetaan vaatimusten toteutuminen vertaamalla jokaisen vaiheen syötedokumentteja eli vaatimuksia tulosdokumentteihin. Validoinnissa pyritään osoittamaan, että toteutettava järjestelmä vastaa asiakkaan tarpeita. Projektin loppuvaiheessa vaatimusten toteutuminen voidaan varmistaa testaamalla tuotetta sen oikeassa käyttöympäristössä. Validointia voidaan tehdä myös prosessin aikaisemmissa vaiheissa esimerkiksi prototyyppien avulla. [HaM01]

Tutkielmassa käsitellään ensin vaatimusten määrittelyyn liittyviä asioita. Luvussa 2 pohditaan, miksi *vaatimusten kirjoittaminen on niin vaikeaa* ja miksi monet asiakkaat ja tuottajat *kieltävät vaatimusten määrittelemisen ja analysoinnin tarpeen*. Luvussa 3 käydään läpi *vaatimusmäärittelydokumentin sisältöä* ja luvussa 4 tutkitaan vaatimusten *mitattavuutta*. Tutkielmassa käydään luvussa 5 läpi vaatimusten ja vaatimusmäärittelydokumentin *laatuattribuutteja* ja niiden arviointimahdollisuuksia.

Tutkielman luvussa 6 tutustutaan *tarkastus*tekniikkaan. Luvussa 7 pohditaan, millaisia *validointi- ja verifiointiryhmiä* voitaisiin käyttää ja luvussa 8 tutustutaan lyhyesti ohjelmistojen *validointi- ja verifiointitekniikoihin*. Luvussa 9 *yhdistellään käyttö-tapauksia* ja *luokkakaavioita* vaatimusmäärittelyjen verifioinnin ja validoinnin apukeinoina. Luvussa 10 käsitellään *väärinkäyttötapauksia* varsinkin turvallisuusvaatimusten takaamiseksi ja luvussa 11 siirrytään vanhasta järjestelmästä uuteen järjestelmään käyttämällä *reaalimaailman esimerkkejä* vaatimusten määrittelemiseen ja validointiin. Tutkielman lopussa on pohdintaosuus.

2 VAATIMUSTEN MÄÄRITTELEMISEN VAIKEUDET

On yleisesti todettu, että hyvin tehdyt vaatimusmäärittelyt olennaisesti vaikuttavat suunnitellun tuotteen laatuun, hintaan ja aikatauluun. Vaatimuksiin liittyviä ongelmia ovat esimerkiksi seuraavat [Hoo90], [KoS98]:

- Vaatimukset on *määritelty huonosti* kirjoitusvaiheessa ja se vaikuttaa koko ohjelman elinkaaren ajan.
- Vaatimusmäärittelijä *ei osaa kirjoittaa* vaatimuksia.
- Vain *harva*t ihmiset *ymmärtävät vaatimusprosessia*.
- *Johto* ei ole kiinnittänyt tarpeeksi huomiota ongelmaan.
- Huonosti kirjattuihin ja määriteltyihin vaatimuksiin *vaaditaan muutoksia*.
- *Ulkoisiin tekijöihin*, joihin ohjelmalla ei ole hallintamahdollisuutta, vaaditaan vastetta.
- Valittu *standardi* ei ole yhdenmukainen vaatimusten kanssa.
- Systeemin *mallintamisessa* tai *ongelman ratkaisussa on virheitä*.
- Vaatimusten *ristiriitaisuuksia* ei ole huomattu analysointivaiheessa.

Vaatimukset on siis usein jo kirjattu huonosti. Vaatimusten määrittelijä ei ehkä tiedä tarkkaan, *mitä asiakas haluaa*. Vaatimusten määrittelyyn *ei uhrata tarpeeksi aikaa ja voimavaroja.* Vaatimusten määrittelijällä voi olla myös *asenneongelma* ja hän haluaisi tehdä mieluummin jotakin muuta. Vaatimusten määrittelijä *ei* ehkä *ymmärrä*, mitä hänen tulisi tehdä tai miten hänen pitäisi yhdistellä tarvitsemiaan tietoja. Vaatimustenmäärittelijällä ei ehkä ole *kokemusta* hyvien vaatimusten kirjoittamisesta. Hän voi kirjoittaa huonoja oletuksia, toteutuksen vaatimusten sijasta, käyttää vääriä termejä, huonoa kielioppia ja lauserakenteita sekä ylimääritellä vaatimuksia.

Perusvaatimuksia hyville vaatimusten määrittelyille ovat muun muassa selkeys, lyhyys ja yksinkertaisuus. Vaatimusmäärittelijöiden täytyy siis harjoittaa kirjoittamistaitojaan. He tarvitsevat myös esimerkkejä sekä hyvistä että huonoista vaatimusmäärittelyistä. Huonoihin esimerkkeihin ei kuitenkaan kannata paneutua liikaa, etteivät ne jäisi muistiin niin, että kohdatessaan vastaavanlaisen ongelman, vaatimusten määrittelijä käyttäisikin niitä apukeinoina määritellessään uuden systeemin vaatimuksia. [Hoo00], [BaK00]

Monet asiakkaat ja tuottajat valitettavasti vieläkin kieltävät vaatimusten määrittelemisen, analysoinnin ja validoinnin tarpeellisuuden ja tärkeyden, vaikka on selvästi osoitettu, että vaatimuksiin pohjautuvat virheet ovat yksi pääsyy siihen, että monimutkaiset projektit ovat epäonnistuneet. Seuraavassa on lueteltu 12 veruketta, miksi vaatimuksiin on suhtauduttu kielteisesti. Tekosyitä on tarkasteltu sekä asiakkaan, tuottajan että hankkijan kannalta [Gab99], [Dav99], [Tho99] :

1) *Asiakas ei ole kiinnostunut*. Asiakkaat uskovat, että he ovat kertoneet haluamansa tuotteen ominaisuudet tarpeeksi hyvin, eivätkä halua lisäkeskusteluja.
2) *Tuottajat ja hankkijat luulevat, ettei asiakas tiedä mitä haluaa*. He tietävät mielestään paremmin tuotteen idean ja yksityiskohdat.
3) *Asiakas ei välitä siitä, ovatko hänen vaatimuksensa epärealistisia*. Asiakkaat haluavat tuotteelle esimerkiksi sellaisia ominaisuuksia, joita ei nykytekniikalla voida toteuttaa.
4) *Asiakas haluaa tuotteen eikä vaatimuksia.*
5) *Aika ei riitä vaatimusten riittävään määrittelemiseen.* Projektien aikataulut saattavat esimerkiksi olla niin tiukkoja, ettei vaatimusten määrittelemiseen ja validointiin paneuduta tarpeeksi.
6) *Asiakkaan mielestä vaatimusten määritteleminen on liian vaikeaa.* Asiakkaalla ei ole kokemusta eikä vaadittavaa tietotaitoa, jotta hän kykenisi määrittelemään vaatimuksia.
7) *Uudella systeemillä ei ole ollut vielä käyttäjiä.* Asiakkaat ja tuottajat vetoavat siihen, ettei vaatimuksia voida määritellä, koska rakennettavaa systeemiä ei ole kukaan vielä kokeillut.
8) *Vaatimukset vaihtuvat aina.* Toimittajat ja hankkijat eivät halua tuhlata aikaa muuttuviin vaatimuksiin.
9) *Prototyyppi korvaa vaatimukset*. Asiakas ja tuottaja ovat sitä mieltä, että vaatimukset saadaan prototyypistä, eikä niitä tarvitse määritellä aikaisemmin.
10) *Käytetään hyllytavaraa (commercial off the shelf)*. Asiakas ja tuottaja uskovat, että käyttämällä valmiskomponentteja, vaatimuksiin ei tarvitse syventyä tarkemmin.
11) *Vedotaan sopimukseen*. Tuottaja ja toimittaja tietävät, että vaatimukset eivät ole hyviä, mutta vetoavat sovittuihin aikatauluihin ja kustannuksiin.
12) *Vaatimukset eivät ole pääasia*. Tuottajat ja toimittajat välittävät vain siitä, saavatko he tuotteen kaupatuksi, eivätkä vaatimuksista.

3 VAATIMUSMÄÄRITTELYJEN SISÄLTÖ

Vaatimusmäärittelyssä on täydellinen silti lyhyt kuvaus systeemin ulkoisesta vuorovaikutuksesta ympäristönsä kanssa. Tähän sisältyy muut ohjelmistot, kommunikointiportit, laitteistot ja käyttäjät. Vaatimuksia on kahden tyyppisiä: *toiminnallisia* ja *ei-toiminnallisia* vaatimuksia. [RoR99]

Toiminnalliset vaatimukset määrittelevät sen, mitä systeemi tekee. Niissä kuvataan syötteet ja tulosteet sekä niiden yhteydet toisiinsa. Toiminnalliset vaatimukset liittyvät tuotteen toimintaan eli ne kertovat, mitä tuotteen pitää tehdä. *Sopivuuskriteerit (fit criterions)* määrittelevät, onko tuote onnistuneesti tehnyt jonkin toiminnan. Toiminnallisia vaatimuksia ei tarvitse mitata, toiminta joko tapahtuu tai ei. [RoR99]

Ei-toiminnalliset vaatimukset määrittelevät systeemin laatuattribuutit ja miten systeemi suorittaa työnsä. Niissä on kuvaus esimerkiksi systeemin tehokkuustasosta, luotettavuudesta, turvallisuudesta, näkyvyydestä, kapasiteetista ja siirrettävyydestä. Jotkut ei-toiminnalliset vaatimukset saattavat näyttää olevan vaikeasti määriteltävissä. Jos vaatimusta ei pystytä määrittelemään tai mittaamaan, niin silloin vaatimus ei ehkä ole mikään oikea vaatimus. Se voi olla usean vaatimuksen yhdistelmä tai se voi olla vaatimus, jota ei ole ajateltu loppuun asti. Jos se ei ole vaatimus, pitää se poistaa määrittelyistä. [RoR99]

Ohjelmiston vaatimusmäärittely muodostuu ohjelmistosysteemin ulkoisen käyttäytymisen kuvailuista. Vaatimusmäärittelyn sisältöön vaikuttaa määrittelijä. Jos määrittelijänä on systeemin käyttäjä tai asiakas, on vaatimusmäärittelyn tarkoitus määritellä asiakkaan tarve. Vaatimusmäärittelydokumenttia voidaan käyttää kilpailullisena tarjouspyyntönä eri yhtiöille, jotka ovat halukkaita tekemään ohjelmiston. Asiakas saa tällöin parhaan tuotteen mahdollisimman pienillä kustannuksilla. Toisaalta jos kirjoittajana on systeemin kehittäjä, on vaatimusmäärittelyn sisältö tarkempi. Tarkoituksena on kuitenkin se, että tehtyjen dokumenttien avulla asiakas, käyttäjä, analysoija sekä suunnittelija voivat *kommunikoida* keskenään.

Hyvin kirjoitettu vaatimusmäärittely pienentää riskiä, että asiakas on pettynyt lopulliseen tuotteeseen. Vaatimusmäärittelyjen pitää siis olla ristiriidattomia ja väärintulkinta

mahdollisuuksia ei saisi olla. Jos asiakkaan ja kehittäjän välillä on erimielisyyttä, pitää ristiriidat käsitellä tässä vaiheessa, koska hyväksymistestausvaiheessa virheiden korjaus on paljon kalliimpaa. Valitettavasti jotkut kehittäjät pitävät monitulkintaisista vaatimusmäärittelyistä, koska se lisää heidän mielestään joustomahdollisuuksia suunnitteluvaiheessa. Tällainen joustomahdollisuus kasvattaa kuitenkin asiakkaaseen kohdistuvaa riskiä merkittävästi. Vaatimusmäärittelyn pitäisi olla tarkka kuvaillessaan systeemiä ulkoisesti. Jos vaatimusten kirjoittaja ei pysty kuvailemaan ulkoista käyttäytymistä ilman suunnittelua, pitäisi vaatimusmäärittelydokumenttiin laittaa varoitus, että suunnittelua käytetään vain apuna ulkoisen käyttäytymisen ymmärtämisekseen. [Dav90]

Vaatimusmäärittelydokumenttia käytetään myös testauksessa ja verifioinnissa. Systeemin testauksen tarkoituksenahan on osoittaa, että rakennettu systeemi vastaa vaatimuksia. Jos vaatimusmäärittely on ristiriitainen ja moniselitteinen tai siinä on vaatimuksia, joita ei voi testata, niin koko testaus on mahdotonta. Vaatimusmäärittelydokumentti on siis systeemin testaussuunnittelun ja luontiprosessin ensisijainen syöte. [Dav90]

Vaatimusmäärittelyä käytetään myös *ohjelmistosysteemin kehittymisen valvontaan*. Jos asiakas esimerkiksi haluaa ohjelman tekevän jotakin, tarkistetaan vaatimusmäärittelydokumentista, onko vaatimus uusi vai vanha. Jos se on uusi, päivitetään vaatimusmäärittely. [Dav90]

Vaatimusmäärittelyihin ei kirjoiteta projektin vaatimuksia, esimerkiksi aikataulua, kustannuksia, vaiheita ja toimintaa. Siihen ei myöskään kirjoiteta suunnittelua eikä testaus-, verifiointi- ja validointisuunnitelmia. [RoR99], [MaM00]

4 SOPIVUUSKRITEERIT

Ratkaisun pitää tyydyttää täysin vaatimukset, toisin sanoen ratkaisu tekee tarkasti sen, mitä vaatimukset sanovat sen tekevän, ei enempää eikä vähempää. Ennen kuin voidaan tietää onko ratkaisu oikea vai ei, pitää vaatimukset ensin *määrittää (quantify)*. Kun vaatimukset on määritetty, voidaan toteutus arvioida. Vaatimusten määrittämistä voidaan pitää vaatimuksen sopivuuskriteerinä. Sopivuuskriteerit voivat määrittää käyttäytymistä, suoritusta tai muita vaatimusten piirteitä. Sopivuuskriteereitä käytetään sekä toiminnallisiin että ei-toiminnallisiin vaatimuksiin. Kun ensin on kirjoitettu vaatimuksia eli käyttäjän ajatuksia vaatimuksista, vaatimukset pitää määrittää jotenkin, jotta tiedetään tarkalleen, mitä on tavoiteltu. Jos vaatimukset pystytään esittämään numeroilla, on väärinymmärrysten mahdollisuus pieni. [RoR99], [Mac96]

4.1 Vaatimusten mitattavuus

Kun tuotteen vaatimuksen täytyy toteuttaa joku toiminto tai sillä pitää olla jokin ominaisuus, niin testauksella pitää osoittaa, että tuote täyttää vaaditut toiminnot tai ominaisuudet. Testaajien täytyy verrata toimitettua tuotetta alkuperäisiin vaatimuksiin jonkin *mittapuun* mukaan. Mittapuu on sopivuuskriteeri - jokin tuotteen määrittelyt täyttävät *standardi*, jotka tuotteella on. [RoR99]

Tuotteen *rakentajien* näkemyksiä pitää myös tarkastella. Jos he tietävät, mitä tuotteelta odotetaan, he yrittävät rakentaa sellaisen tuotteen. Esimerkiksi, jos hyväksymiskriteeri on, että tuotteen pitää toimia 24 tuntia veden alla 15 metrin syvyydessä, toteuttaja tuskin rakentaa tuotetta pahvista. [RoR99]

Testauksessa voi ongelmaksi muodostua oikean mittaustavan määritteleminen. Esimerkiksi, jos asiakas haluaa tuotteen, joka on käyttäjäystävällinen, niin pitäisi löytää jokin menetelmä, jolla käyttäjäystävällisyyttä mitataan. Asiakkaan täytyy hyväksyä valittu *mittausmenetelmä*, jotta hänen näkemyksensä toteutuu. Tuotteen rakentajien on myös hyvä tietää, miten käyttäjäystävällisyyttä mitataan, että he etenevät oikeaan suuntaan tehdessään tuotetta. Oikea mittausmenetelmä löytyy usein *keskustelemalla* asiakkaan kanssa vähän tarkemmin. Asiakas voi esimerkiksi ajatella, että käyttäjäystävällinen tuo-

te on sellainen, jota henkilökunta käyttää mielellään. Lisätutkinta saattaa paljastaa, että tuote, jota henkilökunta käyttää mielellään, tarkoittaa sitä, että he ottavat tuotteen vaistomaisesti käyttöönsä. Tällöin käyttäjäystävällisyyttä voidaan mitata arvioimalla henkilökunnan oppimisaikaa ennen käyttöönottoa tai tutkimalla kuinka kauan henkilökunnalta menee aikaa tuotteen lopulliseen hyväksymiseen. Käyttäjäystävällisyyttä voitaisiin mitata myös tutkimalla henkilökunnan tyytyväisyyttä tuotteeseen heidän käytettyään sitä jonkin aikaa. Asiakkaan kanssa on tietysti keskusteltava ensin siitä, että ehdotettu mittausmenetelmä tosiaankin mittaa sitä, mitä asiakas haluaa. Kun mittausmenetelmä valitaan lopullisesti, se toimii käyttäjäystävällisen tuotteen sopivuuskriteerinä. [RoR99]

4.2 Mitta-asteikko

Tuotteen sopivuutta mitataan jollakin *mitta-asteikolla*. Esimerkiksi jos vaatimukseen liittyy nopeus, niin silloin mittausyksikkö on aika. Käyttökelpoisuusvaatimuksia mitattaessa, voidaan mitata esimerkiksi tuotteen käyttämisen opetteluun kulunut aika tai aika, joka menee ennen kuin sovittu tyytyväisyystaso saavutetaan. Värejä voidaan mitata komponenttivärien pitoisuuksilla. Äänen hiljaisuutta ja pehmyttä mitataan desibeleillä. Valon määrää ilmoitetaan lumeina. Kirjasinlajit mitataan nimen ja koon mukaan. Todellisuudessa melkein kaikelle löytyy jokin mittausmenetelmä. Sopivan mittausmenetelmän löytää *analysoimalla vaatimuksen kuvausta ja perustelua*. [RoR99]

5 VAATIMUSMÄÄRITTELYDOKUMENTIN OMINAISUUKSIA

Kirjoitettaessa vaatimusmäärittelyjä, on panostettava yksittäisten vaatimusten lisäksi myös itse dokumentin laatuun. Vaatimusmäärittelydokumenttia voidaan arvioida eri laatuattribuuttien avulla kuten vaatimuksiakin.

5.1 Vaatimusmäärittelydokumentin vaatimusten validointi ja verifiointi

Vaatimusten validoinnissa varmistetaan, että vaatimusmäärittelydokumentit ovat *johdonmukaisia ja tarkkoja.* Vaatimusten validointi on vaatimusmäärittelyn viimeinen vaihe. Vaatimusten validoinnissa tarkistetaan että vaatimukset toteuttavat systeemin hyväksytyn kuvauksen. *Validointiprosessiin* osallistuvat systeemin asianosaiset, vaatimusten määrittelijät ja suunnittelijat, jotka analysoivat, onko vaatimuksissa ongelmia, puutteita tai moniselitteisyyksiä. Dokumentin ja vaatimusten pitäisi noudattaa valittuja standardeja. Validointiprosessissa mietitään, millä tavoin vaatimuksia kuvataan eli vastataan kysymykseen, ovatko vaatimukset oikein. [KoS98]

Vaatimusten verifioinnissa käsitellään systeemin asianosaisilta saatuja vaatimuksia. Vaatimukset ovat usein epätäydellisiä ja epämuodollisesti kuvattuja. Niitä on kirjattu useammilla eri merkintätavoilla. Tässä vaiheessa pitäisi varmistaa, että vaatimukset ovat todellakin sitä, mitä asianosaiset tarvitsevat. Verifioinnissa painotetaan asianosaisten tarvetta ennen kuin yksityiskohtainen dokumentti tehdään. Vaatimustenmäärittelydokumentissa pitäisi olla vain asianosaisten hyväksymiä vaatimuksia. [KoS98]

Vaatimusmäärittelydokumentista täytyy poistaa siinä olevat virheet ja puutteellisuudet. Joskus puutteet voivat johtua yksinkertaisesti dokumentointiongelmista ja ne voidaan korjata parantamalla vaatimusten kuvauksia. Muissa tapauksissa ongelmat johtuvat siitä, että vaatimusten kalastamisessa, analysoinnissa ja mallintamisessa on ollut vikaa. [KoS98]

Vaatimusten validointiprosessin syötteitä ovat *vaatimusdokumentti*, *organisatorinen tieto* ja *organisatoriset standardit*. Vaatimusdokumentista pitäisi olla tässä vaiheessa lopullinen versio eikä vain aloitettu luonnos. Sen pitäisi noudattaa organisatorisia stan-

dardeja. Organisatorinen tieto ei ole todellinen syöte, mutta käytännössä sillä on tärkeä rooli. Vaatimusmäärittelyjen kanssa olevat ihmiset saattavat tuntea organisaation, sen oman termistön, käytännöt ja henkilöstön taidot. Tällainen *ennalta tunnettu (implicit) tieto* on tärkeää, kun vaatimuksia yhdistetään organisaation rakenteeseen, standardiin ja kulttuuriin. Kuvassa 1 näkyvät vaatimusten validoinnin syötteet ja tulosteet. [KoS98]

Kuva 1: Vaatimusten validoinnin syötteet ja tulosteet [KoS98]

Prosessin tulosteita ovat listat ongelmista ja hyväksytyistä korjaustoimenpiteistä. Ongelmalistaan kerätään vaatimusdokumentissa havaitut ongelmat. Kun validointiprosessiin kuuluvat henkilöt ovat hyväksyneet tarvittavat korjaustoimenpiteet, ne listataan myös. Jotkut ongelmat aiheuttavat useita korjaustoimenpiteitä ja jotkut ongelmat ainoastaan maininnan ilman erillisiä toimenpiteitä. [KoS98]

Vaatimusten validointi on prosessi, jossa henkilöt joutuvat lukemaan ja analysoimaan pitkiä dokumentteja. Tarvitaan tapaamisia ja asiantuntijoiden tekemiä prototyyppejä. Monimutkaisen systeemin validointi saattaa viedä useita viikkoja jopa kuukausia. Näin käy varsinkin silloin, kun asianosaiset ovat eri organisaatioista. Jos systeemin kehitysprosessilla on tiukka aikataulu, niin validointikin on tehtävä nopeasti. Se saattaa aiheuttaa lisätyötä, jos vaatimusten määrittelyvirheitä löytyy ohjelmiston kehityksen myöhemmissä vaiheissa. Virheiden ja ongelmien löytäminen jo määrittelyvaiheessa vähentää uudelleen tehtävän työn määrää ja kustannuksia. [KoS98]

5.2 Vaatimusmäärittelydokumentin attribuutteja

Taulukossa 1 on annettu 24 attribuuttia, joilla voidaan mitata vaatimusmäärittelyssä esiintyvien vaatimusten ja vaatimusmäärittelydokumentin laatua.

Taulukko 1: Vaatimusmäärittelydokumentin attribuutteja [DoT90]

1. yksiselitteinen
2. täydellinen
3. virheetön
4. ymmärrettävä
5. varmistettavissa
6. sisäisesti johdonmukainen
7. ulkoisesti johdonmukainen
8. toteutettava
9. ytimekäs
10. suunnittelusta riippumaton
11. jäljitettävä
12. muunneltava
13. elektronisesti tallennettu
14. suorituskelpoinen/tulkattava/protoiltava
15. suhtellinen tärkeys selitettynä
16. suhteellinen pysyvyys selitettynä
17. varustettu versiomerkinnöin
18. ei-redundanttinen
19. yksityiskohtien oikeat tasot
20. tarkka
21. uudelleenkäytettävä
22. jäljitetty
23. järjestetty
24. ristiviitattu

5.3 Vaatimusmäärittelydokumentin laadun mittaaminen

Laatuvaatimukset on usein kirjoitettu epämääräisesti vaatimusmäärittelydokumenttiin. Jos laatuvaatimukset ohitetaan, niin lopullinen ohjelmisto ei ehkä tyydytä käyttäjän tarpeita. Asiakkaan ja toteuttajan erilaisista tulkinnoista syntyy erimielisyyksiä. Ohjelmistoa voi olla mahdotonta testata perusteellisesti ja saatetaan rakentaa jopa väärä systeemi. Jotta tällaisilta ongelmilta vältyttäisiin, olisi hyvä, jos vaatimusmäärittelyssä esiintyviä laatuattribuutteja sekä niiden välisiä suhteita voitaisiin mitata *matemaattisesti.* Tässä kappaleessa esitetään Dorfmanin ja Thayerin tuloksia tutkimuksesta, jossa

lasketaan kaikille taulukossa 1 esiintyvälle 24 laatuattribuutille matemaattinen arvo [DoT90]. Lasketuista arvoista muodostetaan *vektori* Q_1, Q_2, …Q_N. Vektoria tutkimalla selvitetään, miten hyvin kyseiset attribuutit toteutuvat vaatimusmäärittelyssä. Esitetty menetelmä on kuitenkin haastava ja vaatii asiantuntijan onnistuakseen, sillä esitettyjä kaavoja on sovellettava kohteen mukaisesti.

5.3.1 Yksiselitteinen

Oletetaan, että vaatimusmäärittelyssä on n_k kappaletta vaatimuksia, joista n_t kappaletta on toiminnallisia vaatimuksia ja n_{et} kappaletta ei-toiminnallisia vaatimuksia eli $n_k = n_t + n_{et}$.

Yksiselitteinen (unambigious) tarkoittaa sitä, että jokaisella vaatimuksella on yksi ja vain yksi tulkinta [DoT90]. Koska monissa kielissä on synnynnäistä moniselitteisyyttä ja eri ihmiset ymmärtävät asiat eri tavalla, voidaan yksiselitteisyyttä laskea kaavalla

$$Q_1 = \frac{n_{st}}{n_k},$$

missä n_k on vaatimusten kokonaismäärä ja n_{st} sellaisten vaatimusten määrää, joista asianosaisilla on samanlainen tulkinta. Luku Q_1 saa arvoja nollan ja ykkösen väliltä. Mitä lähempänä luku on ykköstä, sitä parempi on yksiselitteisyys. *Luonnollisen* kielen voi kyllä korvata *formaalilla* kielellä, mutta silloin yleensä ymmärrettävyys pienenee. Parempi tapa olisikin käyttää molempia kieliä yhdessä.

5.3.2 Täydellinen

Vaatimusmäärittelydokumentin laatuvaatimus *täydellinen (complete)* voidaan määritellä viidellä eri tavalla [DoT90]:

1. Kaikki, mitä ohjelman oletetaan tekevän, löytyy vaatimusmärittelystä.
2. Ohjelmiston vastuut kaikkiin erilaisiin syötetietoihin kaikissa toteuttamiskelpoisissa tilanteissa löytyvät vaatimusmäärittelystä.
3. Kaikki sivut, kuvat ja taulukot on numeroitu ja termit määritelty.
4. Kaikki mittayksiköt on annettu ja kaikki liitemateriaali on mukana.
5. Sellaisia huomautuksia, kuin esimerkiksi ”selvitettävä” ei löydy vaatimusmäärittelystä.

Käyttäjät eivät ole tyytyväisiä systeemiin, jos ensimmäinen täydellisyyden määrittely ei toteudu. Jos jälkimmäiset määrittelyt eivät toteudu, suunnittelijat tekevät oletuksia mahdollisesta käyttäytymisestä ja nämä oletukset voivat olla vääriä. Ensimmäisen määrittelyn mukaista oikeellisuutta on vaikea mitata. Yleensä mitä enemmän vaatimuksia vaatimusmäärittelyssä on, sitä enemmän siihen halutaan uusia vaatimuksia [DoT90].

Käyttämällä täydellisyyden toista määritelmää, jotakin järkevää mittausta voidaan suorittaa. Esimerkiksi täydellisyys voisi merkitä sitä, että toiminto *f(tila, heräte)→(tila, vastuu)* on määritelty kaikilla tulon (tila x heräte) arvoilla. Lasketaan vaatimusmäärittelyssä olevien syötteiden määrät eli herätteet n_h sekä tilojen määrät n_t. Toimintojen kokonaismäärä saadaan tulosta (n_h x n_t). Jos nyt lasketaan vaatimusmäärittelystä löytyvät todelliset määritellyt toiminnot (n_m), voidaan täydellisyys laskea kaavalla

$$Q_2 = \frac{n_m}{n_h \text{ x } n_t}.$$

Täydellisyyden Q_2 arvo kertoo määriteltyjen toimintojen prosenttiosuuden kaikista mahdollisista toiminnoista. Tätä laskutapaa voidaan käyttää hyvin ymmärretyssä rajatussa ongelma-alueessa. Kaavassa ei huomioida ei-toiminnallisia vaatimuksia. [DoT90]

Täydellisyyden laskemiseen voidaan kokeilla myös *Aleksanderin ideaa*. Siinä oletetaan, että tulevaisuus voidaan ennustaa, eli kaikki vaatimukset, mitä asiakas voi tulevaisuudessa tarvita, taataan. Kuvan 2 lohko A edustaa vaatimuksia, jotka tunnetaan ja joiden tiedetään olevan sopivia tähän ongelmaan eli nämä ovat niitä vaatimuksia, jotka tyypillisesti saadaan vaatimusmäärittelyn yhteydessä. Lohko B edustaa sellaisia vaatimuksia, jotka tunnetaan, mutta niitä ei ole vielä ilmaistu sanoin tai todella ajateltu. Tällaiset vaatimukset löytyvät yleensä *haastattelujen* ja *aivoriihien* yhteydessä. Kun ne määritellään, ne siirtyvät lohkoon A. Lohko C edustaa vaatimuksia, jotka tiedetään tarpeellisiksi. Niitä ei kuitenkaan ymmärretä niin hyvin, että niitä voitaisiin kuvailla. Tällaiset vaatimukset paljastuvat yleensä *protoilun* aikana. Kun ne löydetään, siirtyvät nekin lohkoon A. Lohko D edustaa vaatimuksia, joita ei tiedetä, ja joista ei edes tiedetä, ettei niitä tiedetä. Protomallin tekeminen auttaa näittenkin löytämisessä, koska joskus yhden lisäpiirteen löytäminen paljastaa muita piirteitä. Kun tällaisia vaatimuksia löydetään, ne pyr-

tään siirtämään lohkoon B. Nuolet kuvassa 3 edustavat vaatimusten liikkeiden kulkusuuntia. Yleisenä pyrkimyksenä on, että kaikki vaatimukset liikkuisivat lohkoon A. Lohkoon A kuuluvien vaatimusten prosentuaalinen määrä voisi olla tehokas täydellisyyden mittari.

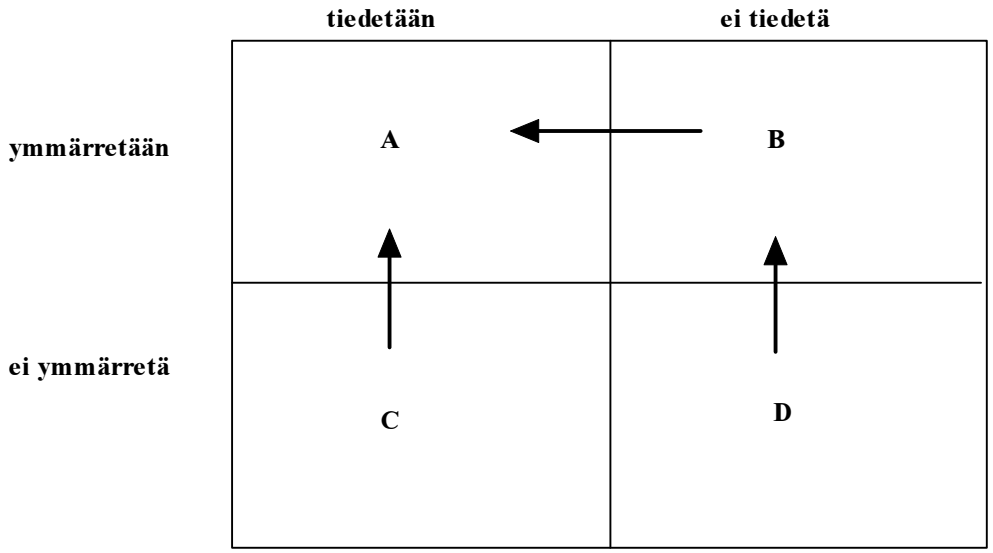
Kaavasta

$$Q_{2'} = \frac{n_A}{n_A + n_B + n_C + n_D}$$

voidaan laskea täydellisyydelle arvo välille 0 ja 1, kun n_A, n_B, n_C, ja n_D ovat lohkojen A, B, C ja D vaatimusten määrät. Koska lohkojen C ja D vaatimusten määriä ei tiedetä, vaihtoehtoinen kaava voisi olla

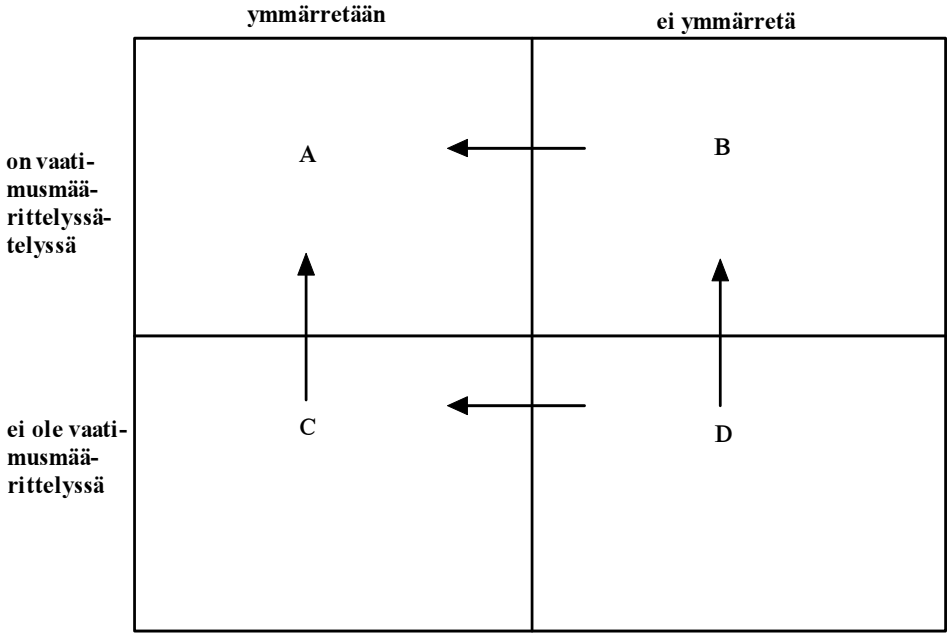
$$Q_{2''} = \frac{n_A}{n_A + n_B}.$$

Tässäkin kaavassa arvot vaihtelevat välillä 0 (täysin epätäydellinen) ja 1 (täydellinen). [DoT90]



Kuva 2: Alexanderin vaatimusten täydellisyysmalli [DoT90]

Toinen tapa mitata *paikallista täydellisyyttä (local completeness)* on laskea, kuinka monta prosenttia vaatimusmäärittelydokumentissa esiintyvistä vaatimuksista on ymmärretty. Kuva 3 on kuvan 2 muunnelma, jossa pystysuoralla akselilla on vaatimukset, jotka joko löytyvät vaatimusmäärittelystä tai eivät löydy. Vaakasuora akseli edustaa vaatimusten ymmärrettävyyttä. Lohkosta A löytyvät vaatimukset, jotka tunnetaan ja jotka ovat vaatimusmäärittelydokumentissa. Lohkosta B löytyvät vaatimukset, jotka on dokumentoitu, mutta huonosti määritelty tai niitä ei ole validoitu. Kun ne määritellään tai validoidaan, ne siirretään lohkoon A. Lohkosta C löytyy vaatimuksia, jotka tiedetään tarpeellisiksi, mutta niitä ei ole vielä määritelty. Kun ne dokumentoidaan, siirretään nekin lohkoon A. Lohko D sisältää vaatimuksia, joita ei vielä ymmärretä riittävän hyvin, jotta niitä voisi dokumentoida. Jos lohkon D vaatimuksia määritellään käsitteellisesti, siirretään ne lohkoon B. Jos lohkon D vaatimusten kelpoisuutta tarkastellaan esimerkiksi protojen avulla ja vaatimuksia ymmärretään näin paremmin, siirretään ne lohkoon C. Nuolet kuvassa osoittavat vaatimusten siirtoliikkeitä lohkojen välillä. Nytkin pyrkimyksenä on, että vaatimukset voitaisiin siirtää lohkoon A.



Kuva 3: Paikallinen vaatimusten täydellisyysmalli [DoT90]

Tätä mallia käytettäessä, täydellisyydelle voidaan laskea arvo kahdella eri vaihtoehtoisella kaavalla. Kaava on joko muotoa

$$Q_{2'''} = \frac{n_A}{n_k}$$

tai muotoa

$$Q_{2''''} = \frac{n_k}{n_A + n_B + n_C + n_D},$$

missä n_A, n_B, n_C ja n_D ovat lohkojen A, B, C ja D vaatimusten määrät. Molemmissa tapauksissa $n_k = n_A + n_B$. Q:n arvot vaihtelevat välillä 0 ja 1. Määritelläänpä täydellisyys miten tahansa, *katselmukset* asiakkaan kanssa ovat erittäin tärkeitä. Protomallit helpottavat uusien vaatimusten huomaamista ja auttavat ymmärtämään paremmin huonosti ja abstraktisesti kirjoitettuja vaatimuksia. [DoT90]

5.3.3 Virheetön

Vaatimusmäärittely on *virheetön (correct)*, jos ja vain jos jokainen vaatimus edustaa jotakin rakennettavan systeemin vaatimusta eli jokainen vaatimusmäärittelyn vaatimus johtaa jonkun tarpeen tyydyttämiseen [DoT90]. Koska termi virheetön liitetään sekä itsenäisiin vaatimuksiin, että koko vaatimusmäärittelyyn, niin eräs tapa arvioida virheettömyyttä on laskea, kuinka monta prosenttia kaikista vaatimuksista on virheettömiä. Virheettömyys voitaisiin silloin laskea kaavalla

$$Q_3 = \frac{n_o}{n_o + n_v},$$

missä n_o on virheettömien ja n_v virheellisten vaatimusten määrä. Kaavalla saatava virheettömyys vaihtelee välillä 0 ja 1. Jotta tällä kaavalla voitaisiin laskea virheettömyyttä, niin silloin pitäisi tietää, mitkä vaatimukset ovat oikeita ja mitkä vääriä. Väärät vaatimukset poistamalla päästäisiin aina 100 % virheettömyyteen. Käytännöllisempää, joskaan ei teoreettisesti yhtä tyydyttävää, on laskea, kuinka monta prosenttia vaatimuksista on validoitu. Virheettömyys voidaan silloin laskea kaavasta

$$Q_{3'} = \frac{n_{va}}{n_{va} + n_{eva}} = \frac{n_{va}}{n_k},$$

missä n_{va} on validoitujen, n_{eva} validoimattomien ja n_k kaikkien vaatimusten määrä. [DoT90]

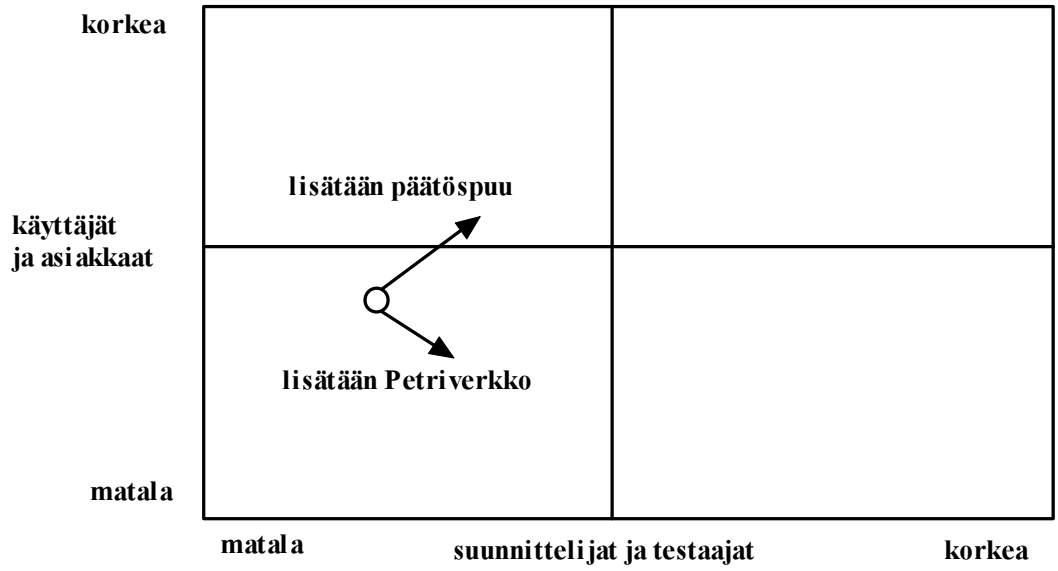
5.3.4 Ymmärrettävä

Vaatimusmäärittely on *ymmärrettävä (understandable)*, jos jokainen vaatimusmäärittelyn lukija ymmärtää kaikkien vaatimusten tarkoituksen mahdollisimman pienellä selittelyllä. Lukijoihin kuuluvat *asiakkaat*, *käyttäjät, projektin johtajat*, *ohjelmiston suunnittelijat* ja t*estaajat*. Yleensä asiakkaat, käyttäjät ja projektin johtajat haluavat helppotajuista luettavaa ja siksi luonnollisen kielen käyttö on suotavaa. Jos käyttäjät ja asiakkaat eivät ymmärrä vaatimusmäärittelyä, he eivät voi hyväksyä sitä. Toisaalta ohjelmiston suunnittelijat ja testaajat haluavat tarkempaa tietoa systeemistä, joten formaalimpi kieli olisi paikallaan. Jos suunnittelijat ja testaajat eivät ymmärrä vaatimusmäärittelyä, on mahdotonta rakentaa tai testata systeemiä. Vastuu vaatimusmäärittelyjen ymmärrettävyydestä lankeaa vaatimusmäärittelyjen kirjoittajille, koska *lukijan velvollisuus ei ole oppia kaikkea, mitä kirjoittaja tietää* laatiessaan vaatimusmäärittelydokumenttia. [DoT90]

Ymmärrettävyyden mittaaminen on vaikeaa. Jos ymmärrettävyyden määrää voitaisiin mitata asteikolla, saataisiin kuvan 4 kaltainen kaavio. Kaavion pieni ympyrä kuvaa kahden lukijaryhmän ymmärtämisen tasoa. Erilaisten tekniikoiden käyttö siirtää kaavion ympyrää. Esimerkiksi päätöspuun lisääminen vaatimusmäärittelydokumenttiin voisi liikuttaa ympyrää koilliseen eli molempien joukkojen ymmärtäminen paranisi. Jos dokumenttiin lisättäisiin petriverkkoja, liikkuisi ympyrä kaakkoon eli suunnittelijat ja testaajat ymmärtäisivät vaatimusmäärittelyjä paremmin, mutta asiakkaiden ja käyttäjien ymmärtämistaso pienenisi. Ymmärrettävyyttä voidaan laskea kaavalla

$$Q_4 = \frac{n_{yk}}{n_k},$$

missä n_{yk} on sellaisten vaatimusten määrä, jotka kaikki luulevat ymmärtävänsä. Jos Q_4 saa arvon 0, yhtään vaatimusta ei ymmärretä ja jos se saa arvon 1, kaikki vaatimukset ymmärretään. Ymmärrettävyys on niin tärkeää, että olisi suositeltavaa saada aina ymmärrettävyyden arvo mahdollisimman lähelle arvoa 1. [DoT90]



Kuva 4: Ymmärrettävyyden mittaaminen [DoT90]

Erilaisten tekniikoiden käyttö vaikuttaa dokumentin ymmärrettävyyteen. Kaikkien asianosaisten pitäisi saada lukea ja kommentoida vaatimusmäärittelyä toistuvasti eri tekniikoiden lisäysten välissä. Protoilu nostaa ymmärrettävyystasoa, koska on helpompaa nähdä miten joku toimii, kuin lukea siitä pelkästään dokumenteista. [DoT90]

5.3.5 Varmistettavissa

Vaatimusmäärittely on *varmistettavissa (verifiable)*, jos on olemassa *kustannustehokkaita tekniikoita*, joita voidaan käyttää tarkistamaan, että jokainen vaatimusmäärittelyn vaatimus täyttyy systeemiä rakennettaessa [DoT909]. Jos vaatimukset eivät ole yksiselitteisiä tai ratkaistavissa olevia, varmistettavuus huononee. Joitakin vaatimuksia ei edes kannata testata joko taloudellisista syistä tai ihmishenkien menetyksen uhan vuoksi.

Varmistettavuuden mittaaminen on vaikeaa. Jos tarkastellaan varmistamiseen kuluvaa aikaa ja kustannuksia, niin seuraavalla kaavalla voidaan laskea *kustannukseen ja aikaan sidottua varmistettavuutta*:

$$Q_5 = \frac{n_k}{n_k + \sum_i k(v_i) + \sum_i a(v_i)} .$$

Kaavassa kirjainyhdistelmä $k(v_i)$ tarkoittaa vaatimuksen v_i varmistettavuuden aiheuttamaa kustannusta ja kirjainyhdistelmä $a(v_i)$ vaatimuksen v_i varmistettavuuteen kuluvaa aikaa. Jos Q_5 on lähellä arvoa 0, vaatimusmäärittelylista ei ole varmistettavissa. Jos Q_5 on lähellä arvoa 1, silloin löytyy sellaisia tekniikoita, joita voidaan käyttää tarkistamaan, että jokainen vaatimusmäärittelyn vaatimus täyttyy systeemiä rakennettaessa [DoT90]. Tätä kaavaa tarkasteltaessa, pitäisi kuitenkin miettiä, onko mitään *järkeä* laskea yhteen aikaa ja kustannuksia. Mitä tällainen yhteenlasku todellisuudessa ilmaisee?

5.3.6 Sisäisesti johdonmukainen

Vaatimusmäärittely on *sisäisesti johdonmukainen (internally consistent)*, jos ja vain jos mikään yksittäisten vaatimusten alijoukko ei ole ristiriitainen [DoT90]. Sisäisen johdonmukaisuuden mittaaminen on helppoa, jos vaatimusmäärittelyä kuvataan *tilakoneena*. Oletetaan, että määriteltyjä herätteitä on n_h kappaletta ja kaikkia tiloja on n_t kappaletta. Tällöin kaikkien toimintojen määrä saadaan tulosta (n_h x n_t). Kaikkien toimintojen pitää olla täydellisiä, johdonmukaisia ja tarpeellisia. Lasketaan vaatimusmäärittelyssä määriteltyjen todellisten toimintojen määrä n_m ja katsotaan kuinka monta kappaletta n_e niistä on epädeterministisiä. Sisäinen johdonmukaisuus voidaan laskea silloin kaavalla:

$$Q_6 = \frac{n_m - n_e}{n_m} .$$

Kaavasta saadut arvot vaihtelevat välillä 0 ja 1. Sisäinen johdonmukaisuus on niin tärkeä asia, että lasketun arvon pitäisi olla mielellään 1. [DoT90]

5.3.7 Ulkoisesti johdonmukainen

Vaatimusmäärittely on *ulkoisesti johdonmukainen (externally consistent)*, jos ja vain jos mikään vaatimusmäärittelyn vaatimus ei ole ristiriidassa minkään jo vertailukohteeksi valitun projektin dokumentaation kanssa [DoT90]. Tällaisia vertailudokumentteja voisi olla esimerkiksi systeemitason vaatimusmäärittely tai jo aiemmat versiot vaatimusmäärittelystä.

Ulkoisen johdonmukaisuuden mittaaminen on vaikeampaa kuin sisäisen johdonmukaisuuden mittaaminen. Sitä voidaan kuitenkin yrittää laskea seuraavalla kaavalla:

$$Q_7 = \frac{n_{dj}}{n_{dj} + n_{de}} = \frac{n_{dj}}{n_k},$$

missä n_{dj} on niiden vaatimusten määrä, jotka ovat johdonmukaisia kaikkien muiden dokumenttien kanssa ja n_{de} vastaavasti niinen vaatimusten määrä, jotka eivät ole. Ulkoinen johdonmukaisuus on myös tärkeä laatuominaisuus ja siksi kaavan arvon pitäisi olla 1. [DoT90]

5.3.8 Toteutettava

Vaatimusmäärittelydokumentti on *toteutettavissa (achievable)*, jos ja vain jos on olemassa *edes yksi* systeeminsuunnittelu ja sen toteutus niin, että kaikki määritellyt vaatimukset toteutuvat oikein [DoT90]. Toteutettavuus Q_8 saa siis joko arvon 0 tai 1. Paras tapa todeta, että kaikki vaatimukset toteutuvat, on rakentaa systeemin osista prototyyppejä.

5.3.9 Ytimekäs

Vaatimusmäärittely on *ytimekäs (concise*), jos se on mahdollisimman lyhyt niin, ettei lyhyys vaikuta vaatimusmäärittelyn muihin laatutekijöihin [DoT90]. Jos on olemassa

useampi kuvaus samasta systeemistä laatuattribuuttien samoilla mittausarvoilla, valitaan lyhyempi.

Yksi tapa arvioida ytimekkyyttä olisi laskea käytettyjen paperien määrä vaatimusmäärittelyssä, kun kuvataan varmasti samaa systeemiä yhtäläisesti. Silloin ytimekkyys voitaisiin laskea kaavalla

$$Q_9 = \frac{1}{\text{sivu}},$$

missä sivu on käytettyjen sivujen lukumäärä. Arvot vaihtelisivat välillä 0 ja 1. Yleensä vaatimusmäärittelyn koon pienentäminen kuitenkin vaikuttaa epäsuotuisasti muihin laatutekijöihin. [DoT90]

5.3.10 Suunnittelusta riippumaton

Vaatimusmäärittely on *suunnittelusta riippumaton (design-independent)*, jos ja vain jos on olemassa *enemmän kuin yksi* systeeminsuunnittelu ja sen toteutus niin, että kaikki vaatimusmäärittelyssä olevat vaatimukset toteutuvat oikein [DoT90]. Vaatimusmäärittelyn tarkoitus on ilmaista sellaista ulkoisen käyttäytymisen tasoa, johon käyttäjät ovat varmasti tyytyväisiä. Ratkaisun ulkoista käyttäytymistä voidaan kuvata tilakoneen avulla niin kauan kuin ratkaisusysteemi käyttäytyy ulkoisesti samalla tavalla kuin tilakone, mutta ei ole oikein vaatia, että ratkaisusysteemi pitäisi suunnitella tilakoneeksi.

Oletetaan, että vaatimusmäärittelyn vaatimuksista r_u kappaletta on puhtaasti ulkoisen käyttäytymisen kuvauksia ja r_s kappaletta ratkaisun arkkitehtuuriin tai algoritmeihin liittyviä vaatimuksia. Tällöin kaikki vaatimukset saadaan unionista $r = r_u \cup r_s$. Oletetaan, että on olemassa joitakin todellisia systeeminsuunnitteluja, jotka toteuttavat kaikki vaatimukset. Merkitään niiden määrää kaavalla $(S(r_u \cup r_s))$. Vastaavasti merkitään kaavalla $(S(r_u))$ sellaisten todellisten systeeminsuunnittelujen määrää, jotka toteuttavat vain ulkoiset käyttäytymisvaatimukset. Suunnittelusta riippumattomuus voidaan nyt laskea kaavasta:

$$Q_{10} = \frac{S(r_u \cup r_s)}{S(r_u)} .$$

Arvot vaihtelevat välillä 0 ja 1. Jos riippuvuutta on paljon, lähestyy Q_{10} arvoa 0 ja mitä riippumattomampi vaatimusmäärittely on, sitä lähempänä Q_{10} on arvoa 1. Yksi tehokas tapa parantaa riippumattomuutta on antaa suunnittelijoiden tarkastella ja arvioida vaatimusmäärittelyjä. [DoT90]

5.3.11 Jäljitettävä

Vaatimusmäärittely on *jäljitettävissä (traceable*) jos ja vain jos se on kirjoitettu tavalla, joka helpottaa jokaiseen yksittäiseen vaatimukseen viittaamista [DoT90]. Suunnittelun ja testauksen aikana on välttämätöntä tietää, mitkä vaatimukset ovat komponenttien tukemia tai testauksen avulla todennettuja.

Vaatimusmäärittely on joko jäljitettävissä tai ei. Jos se sisältää vain joitakin jäljitettävissä olevia vaatimuksia, niin silloin koko vaatimusmäärittely on ei-jäljitettävissä. Jäljitettävyys Q_{11} saa arvon 1, jos jokin seuraavista ominaisuuksista täyttyy ja arvon 0 jos mikään niistä ei täyty [DoT90]:

1. *Numeroidaan kaikki kappaleet hierarkkisesti*. Tällöin voidaan viitata, mihin kappaleeseen ja lauseeseen tahansa jälkeenpäin. Esimerkiksi vaatimus 2.3.2.4L3 viittaa kappaleen 2.3.2.4 lauseen 3 vaatimukseen.
2. *Numeroidaan kaikki kappaleet hierarkkisesti ja sisällytetään jokaiseen kappaleeseen vain yksi vaatimus*. Tällöin viittaukseen tarvitaan vain kappaleen numero.
3. *Merkitään jokaisen vaatimuksen jälkeen sulkuihin yksikäsitteinen numero*.
4. *Käytetään jotakin muuta tapaa osoittamaan vaatimusta*. Esimerkiksi jokaiseen lauseeseen, joka sisältää vaatimuksen, lisätään jokin tietty sana. Sen jälkeen voidaan jollakin työkalulla etsiä ja numeroida kaikki tämän tietyn sanan sisältävät lauseet.

5.3.12 Muunneltava

Vaatimustenmäärittely on *muunneltava (modifiable*), jos sen rakenne ja tyyli voidaan muuttaa helposti, täysin ja johdonmukaisesti [DoT90]. Muunneltavuuteen vaikuttaa kaksi pääsyytä: jatkuva tarve kehittyä ja vaatimusmäärittelyssä esiintyvät virheet. Kehitysvaiheessa vaatimusmäärittely muuttuu uusien vaatimusten, vanhojen vaatimusten muutoksien ja poistettujen vanhentuneiden vaatimusten takia. Muunneltavuutta parantaa, jos vaatimusmäärittely on myös jäljitettävissä, koneella luettavassa muodossa, hyvin järjestetty ja ristiviitattu. Muunneltavuutta parannetaan myös sisällysluetteloiden ja hakemistojen avulla. Koska suurin osa mainituista ominaisuuksista sisällytetään muihin laatuattribuutteihin, muunneltavuutta voitaisiin mitata esimerkiksi seuraavasti: muunneltavuus Q_{12} on 1, jos sisällysluettelo ja hakemisto ovat mukana, muuten se on 0 [DoT90]. Painoarvo riippuu hyvin paljon käyttösovelluksesta. Vaikka muunneltavuuteen liittyviä ominaisuuksia tarkastellaankin muiden laatuattribuuttien yhteydessä, pitäisi miettiä, voiko pelkän sisällysluettelon ja hakemiston perusteella mitata muunneltavuutta. Kertooko näin saatu muunneltavuuden arvo mitään todellisesta muunneltavuudesta?

5.3.13 Elektronisesti tallennettu

Vaatimustenmäärittely on *elektronisesti tallennettu (electronically stored),* jos ja vain jos koko vaatimusmäärittely on tekstinkäsittelyohjelmassa, se on tuotettu vaatimustietokannasta tai muuten yhdistetty joistakin muista lomakkeista [DoT90]. Jos vain osa vaatimusmäärittelystä on tallennettu elektronisesti, voidaan laskea, kuinka monta prosenttia vaatimusmäärittelystä on tallennettu siten.

5.3.14 Suorituskelpoinen, tulkattava tai protoiltava

Vaatimusmäärittely on *suorituskelpoinen (executable)*, *tulkattava (interpretable*) tai *protoiltava (prototypable*), jos ja vain jos on olemassa ohjelmistotyökalu, joka kykenee ottamaan syötteeksi vaatimusmäärittelyn ja tuottamaan siitä dynaamisen käyttäytymismallin [DoT90]. Käyttäytymismallin tekeminen voisi onnistua, jos vaatimustenmäärit-

tely on kirjoitettu kielellä, jonka tietokone ymmärtää suoraan tai käännetty tietokoneen ymmärtämään muotoon tai jos se voidaan tulkata jollakin ohjelmistotyökalulla ja sitten simuloida.

Vaatimustenmäärittely voi olla osittain kirjoitettu suorituskelpoisella, tulkattavalla tai protoiltavalla kielellä. Siksi Q_{14} saa arvoja välillä 0 ja 1. Jos painoarvo on 1, niin vaatimusmäärittely on täysin suorituskelpoinen.

5.3.15 Varustettu huomautuksin vaatimusten suhteellisesta tärkeydestä

Vaatimusmäärittelystä löytyy *huomautuksia vaatimusten suhteellisesta tärkeydestä (annotated by relative importance)*, jos lukija voi helposti havaita, mitkä vaatimukset ovat tärkeimmät asiakkaalle, mitkä seuraavaksi tärkeimmät ja niin edelleen. Tällaista tietoa tarvitaan, jotta rahoitus ja resurssointi voitaisiin hoitaa järkevästi, varsinkin silloin kun budjetti on tiukka. Jos voidaan laskea, kuinka moneen prosenttiin vaatimusmäärittelyjen vaatimuksista on lisätty tietoa suhteellisesta tärkeydestä, saadaan ominaisuudelle Q_{15} arvo. Eräs tapa olisi lisätä jokaiseen vaatimukseen lisätieto P, H tai V sen mukaan onko vaatimus pakollinen, haluttava vai valinnainen. [DoT90]

5.3.16 Varustettu huomautuksin vaatimusten suhteellisesta pysyvyydestä

Vaatimustenmäärittely on *varustettu huomautuksin vaatimusten suhteellisesta pysyvyydestä (annotated by relative stability)*, jos lukija voi helposti huomata, mitkä vaatimukset todennäköisimmin muuttuvat, mitkä seuraavaksi ja niin edelleen. Yleensä vaatimusmäärittely on joko varustettu pysyvyyshuomautuksin tai ei. Jos voidaan selvittää, kuinka moneen prosenttiin vaatimusmäärittelyjen vaatimuksista on lisätty tietoa suhteellisesta pysyvyydestä, saadaan laskettua arvo Q_{16}. Eräs tapa olisi merkitä jokaiseen vaatimukseen muuttumisen todennäköisyys eli onko se korkea, keskitasoinen vai matala. [DoT90]

5.3.17 Varustettu versiomerkinnöin

Vaatimustenmäärittely on *varustettu versiomerkinnöin (annotated by version)*, jos lukijan on helppo huomata, mitkä vaatimukset toteutuvat missäkin versiossa. Jos voidaan laskea kuinka monta prosenttia vaatimuksista on merkitty versiomerkinnöillä, saadaan arvo Q_{17}. Yleisin tapa merkitä versio, on lisätä sarake vaatimusmäärittelydokumentin reunaan, jokaista ohjelmiston versiota varten. Vaatimusten kohdalle merkitään x, kun se on mukana versiossa. [DoT90]

5.3.18 Ei-redundanttinen

Vaatimusmäärittely on *redundanttinen (redundant)*, jos sama vaatimus esiintyy siinä useammin kuin kerran. Redundanssi ei välttämättä ole aina huono asia. Usein toistettavuutta käytetään luettavuuden lisäämiseksi. Ongelmia syntyy vasta sitten, kun vaatimusmäärittelyä muutetaan. Jos ei muuteta kaikkia kohtia, missä vaatimus esiintyy, niin vaatimusmäärittelystä tulee ristiriitainen. Ei-reduntanttisuutta voidaan arvioida esimerkiksi laskemalla kuinka monta prosenttia toiminnoista ei toistu. [DoT90]

5.3.19 Abstraktion ja yksityiskohtien oikea taso

Vaatimuksia voidaan kuvata monella tasolla. Yksityiskohtien oikea taso määräytyy siitä, kuinka vaatimusmäärittelyä käytetään. Yleensä vaatimusmäärittelyn pitäisi olla tarpeeksi tarkka, niin että mikä tahansa systeemintoteutus, joka täyttää vaatimukset, täyttää myös kaikki käyttäjän tarpeet ja on toisaalta tarpeeksi abstrakti, niin että kaikki systeemit, jotka tyydyttävät kaikki käyttäjän tarpeet, täyttävät myös kaikki vaatimukset. Näin ollen vaatimusmäärittelyn, jota käytetään asiakkaan ja suunnittelijan välisenä sopimuksena, pitää olla täsmällinen, jotta asiakas tietää mitä saa ja odottamattomia yllätyksiä tulee mahdollisimman vähän. Koska tarkoituksena on mitata abstraktiotason sopivuutta, ei vaatimusmäärittelyn abstraktiotasoa, niin mittaaminen on mahdotonta. [DoT90]

5.3.20 Tarkka

Vaatimusmäärittely on *tarkka (precise)*, jos ja vain jos numerosuureita käytetään aina, kun se on mahdollista ja numerosuureiden tarkkuustaso on tarkoituksenmukainen [DoT90]. Esimerkiksi vaatimus "systeemin on vastattava nopeasti pyyntöihin", ei ole niin tarkka kuin vaatimus "systeemin on vastattava kaikkiin pyyntöihin 2 sekunnissa".

5.3.21 Uudelleen käytettävä

Vaatimusmäärittely on *uudelleen käytettävä (reusable)*, jos ja vain jos sen lauseita, kappaleita ja osia voidaan helposti käyttää myöhemmissä vaatimusmäärittelyissä. Uudelleen käytettävyyttä tutkitaan paljon suunnittelun ja koodauksen yhteydessä, valitettavasti vain vähän vaatimusten alueella. [DoT90]

Ihannetapauksessa, jolloin koko vaatimusmäärittelyn sisältö on käytetty myöhemmissä vaatimusmäärittelyissä, saa uudelleen käytettävyys arvon 1. Arvon 0 se saa, kun yhtään osaa ei ole käytetty uudestaan. Uudelleen käytettävyyden mittaamisella ei valitettavasti ole käyttöarvoa, jos mittaamista ei voi tehdä samanaikaisesti kuin vaatimusmäärittelyäkin. Vaihtoehtoisesti tätä ominaisuutta voitaisiin mitata arvioimalla mahdollisuuksia käyttää vaatimusmäärittelyn osia uudestaan [DoT90].

Uudelleen käytettävyyttä voitaisiin lisätä käyttämällä vaatimusmäärittelyjen osissa ”*symbolisia vakioita*”. Esimerkiksi suorituskykyä käsittelevässä kohdassa voitaisiin vasteaika korvata tekstinkäsittelyohjelmassa symbolisella vakiolla. Silloin myöhemmissä toiminnaltaan samantyyppisissä sovelluksissa, voitaisiin vain vaihtaa eri vasteaika symboliseen vakioon. Toinen keino olisi käyttää *formaaleja malleja*. Vaikka esimerkiksi petriverkot, tilakoneet tai päätöspuut eivät olekaan sellaisinaan uudelleenkäytettäviä, saattaisi seuraava vaatimusmäärittelijä saada idean käyttää samoja malleja. Kolmas keino olisi luoda *kirjasto* abstrakteista vaatimustyypeistä [DoT90].

5.3.22 Jäljitetty

Vaatimusmäärittely on *jäljitetty (traced)*, jos ja vain jos jokaisen vaatimuksen alkuperä on selvä. Jos vaatimuksesta on tietoa esimerkiksi systeemitason vaatimusmäärittelyssä, sopimuksissa, systeemitason suunnitteludokumenteissa, kannanotoissa, tutkimusraporteissa ja muissa mahdollisissa dokumenteissa, niin ristiviittaukset kaikkiin dokumentteihin pitäisi löytyä. [DoT90]

Jäljitettävyysverkon tason mittaaminen on mahdotonta. Ideaalisesti voitaisiin laskea niiden vaatimusten määrä, jotka on jäljitetty alkuperiinsä ja jakaa se sellaisten vaatimusten määrällä, joilla on alkuperä. Nimittäjä voidaan kuitenkin laskea vain ristiviittausten avulla, joten jakolasku antaisi aina tulokseksi arvon 1. [DoT90]

Jäljitettävyystietojen tallentamiseen voidaan käyttää kahta erilaista tekniikkaa. Vaatimusten perään voidaan laittaa tarkat ristiviittaukset sulkuihin tai kaikki vaatimukset voidaan tallentaa tietokantaan omiksi tietueiksi, joissa ristiviittauksille olisi omat tietokentät.

5.3.23 Järjestetty

Vaatimusmäärittely on *järjestetty (organized*), jos ja vain jos sen sisältö on jäsennetty niin, että lukijoiden on helppo paikallistaa tietoa [DoT90]. Myös vierekkäisten kappaleiden loogisten suhteiden on oltava ilmeisiä. Vaatimusmäärittely voidaan järjestää käyttämällä jotakin vaatimusmäärittelyn standardia. Eri standardien pääjaot ovat samankaltaisia. Suurimmat erot niissä johtuvat yksittäisten vaatimusten järjestelemisestä. Yksittäisiä vaatimuksia voidaan ryhmitellä monella eri tavalla.

Ryhmitellään toiminnalliset vaatimukset käyttäjien mukaan. Esimerkiksi hissin kontrollisysteemin vaatimusmäärittelyssä voisi olla eriteltyinä matkustajien vaatimukset, palokunnan vaatimukset ja ylläpidon vaatimukset. [DoT90]

Ryhmitellään toiminnalliset vaatimukset olosuhteiden mukaan. Esimerkiksi helikopterin automaattinen laskeutumissysteemi voisi sisältää eriteltyinä vaatimukset, kun laskeudu-

taan puuskaiseen tuuleen, vaatimukset, kun polttoaine loppuu ja vaatimukset, kun laskupyörä rikkoutuu. [DoT90]

Ryhmitellään vaatimukset vastuun mukaan. Esimerkiksi palkanmaksusysteemi voisi sisältää omissa ryhmissään palkanmaksun kehittämiseen liittyvät vaatimukset ja kuukausittain työntekijöille maksettavien palkkojen raportteihin liittyvät vaatimukset. [DoT90]

Ryhmitellään toiminnalliset vaatimukset piirteiden mukaan. Esimerkiksi palkanmaksusysteemi voisi sisältää paikallisiin puheluihin liittyvät vaatimukset, kaukopuheluihin liittyvät vaatimukset ja konferenssipuheluihin liittyvät vaatimukset. [DoT90]

Ryhmitellään toiminnalliset vaatimukset kohteen mukaan. Esimerkiksi lennonvarausjärjestelmän vaatimusmäärittelyssä voisi olla eriteltyinä paikkoihin liittyvät vaatimukset, lennonedustajaan liittyvät vaatimukset ja lippuihin liittyvät vaatimukset. [DoT90]

Vaatimusmäärittelyn järjestäminen on hyvin *subjektiivista*, joten sitä ei voi mitata. Käyttämällä jotakin standardia ja vaatimusten ryhmittelyä, vaatimusmäärittelystä saadaan kuitenkin helposti ymmärrettävä. [DoT90]

5.3.24 Ristiviitattu

Vaatimusmäärittely on *ristiviitattu (cross-referenced)*, jos ja vain jos ristiviittauksia käytetään vaatimusmäärittelyssä viittaamaan vaatimuksia sisältävistä kappaleista muiden kappaleiden sisältämiin vaatimuksiin [DoT90]. Muiden kappaleiden vaatimukset voivat olla samoja toistettuja vaatimuksia tai saman vaatimuksen abstraktisempia tai yksityiskohtaisempia kuvauksia. Ristiviittaukset on löydyttävä myös vaatimuksista, jotka riippuvat muiden kappaleiden vaatimuksista tai mistä muiden kappaleiden vaatimukset ovat riippuvaisia.

Hyvin kirjoitetussa vaatimusmäärittelyssä kuvataan vaatimuksia eri tasoilla, abstraktilta tasolta hyvinkin yksityiskohtaisiin tasoihin. Monista vaatimusmäärittelyistä löytyy toistoa, jotta ymmärrettävyys paranisi. Sen tähden vaatimusmäärittelyissä pitäisi käyttää

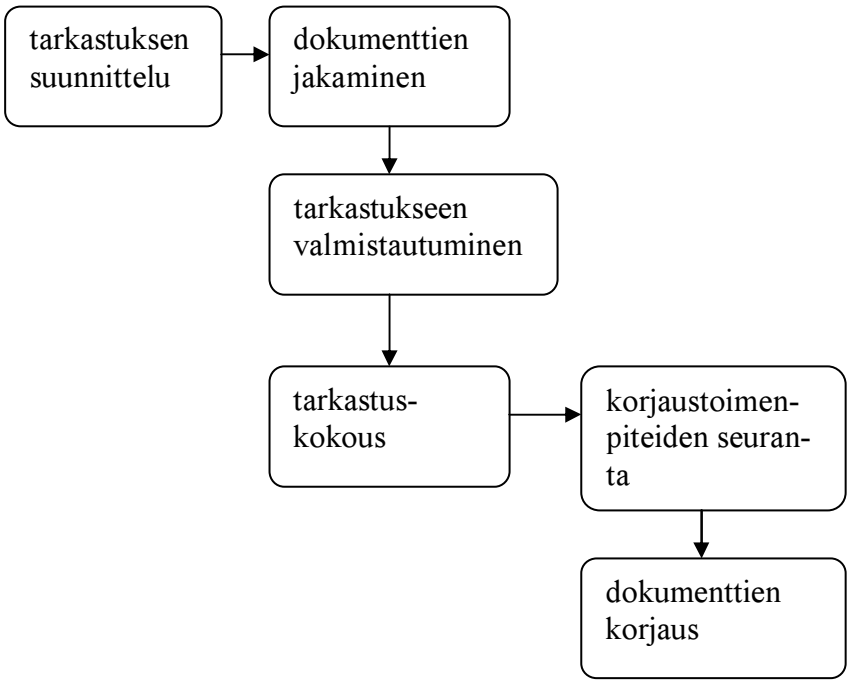
ristiviittauksia. On vaikeaa määritellä, kuinka paljon ristiviittauksia vaatimusmäärittelyssä pitäisi olla ja siksi yritys mitata ristiviittauksia on turhaa. [DoT90]

5.4 Yhteenveto ja arviointi

Täydellinen vaatimusmäärittely on mahdoton toteuttaa, sillä monet attribuutit vaikuttavat toisiinsa eli toisen ominaisuuden parantaminen heikentää toista. Laatuattribuuttien mittaaminen ei edes onnistu kaikille laatutekijöille yhtäläisesti. Väkisin tehty matemaattinen kaava ei ehkä mittaa sitä ominaisuutta, mitä sen pitäisi mitata. Tämän vuoksi tehtäessä attribuutteihin liittyvää vektoria, pitää miettiä tarkkaan, mitä kaavoja käytetään ja mihin vaatimusmäärittelyn laatutekijöihin voidaan käyttää matemaattista lähestymistapaa. Jos saatu vektori on asiallisesti tehty, niin *asiantuntijat* voivat tarkastella laatuattribuuteista mitattuja arvoja. Tutkimalla vektoria Q_1, Q_2, …Q_N he voivat huomata, jos jotakin attribuutin arvoa on jollakin keinolla nostettava tai laskettava. Vaatimusmäärittelyssä pyritään aina sellaiseen lopputulokseen, että eri asianomaiset olisivat tyytyväisiä siihen.

6 TARKASTUS

Tarkastus (Inspection) on laajalti käytetty validointitekniikka. Ryhmä ihmisiä lukee ja analysoi vaatimuksia, etsii niistä ongelmia, kokoontuu keskustelemaan ongelmista ja hyväksyy tarvittavat korjaustoimenpiteet. Kuvassa 5 on yksinkertaistettu esimerkki vaatimusten tarkastusprosessista. [KoS98]



Kuva 5: Tarkastusprosessi [KoS98]

6.1 Tarkastusprosessi

Tarkastuksen päävaiheet ovat tarkastuksen suunnittelu, dokumenttien jakaminen, tarkastukseen valmistautuminen, tarkastus, korjaustoimenpiteiden seuranta ja dokumenttien korjaus. Tarkastuksen suunnitteluvaiheessa valitaan *tarkastusryhmä* ja sovitaan tarkastuskokouksen aika ja paikka. Tämän jälkeen jaetaan tarvittavat dokumentit ryhmän jäsenille. Jokainen ryhmän jäsen tutustuu vaatimusmäärittelydokumenttiin kunnolla ja etsii siitä ristiriitoja, puutteita, epäjohdonmukaisuuksia, valitun standardin vastaisuuksia ja muita ongelmia. Tarkastuskokouksessa keskustellaan löydetyistä virheistä ja hyväksytään vastaavat korjaustoimenpiteet. Tarkastuskokouksen puheenjohtaja huolehtii siitä,

että tarvittavat korjaustoimenpiteet myös suoritetaan myöhemmin. Vaatimusmäärittelydokumenttiin korjataan tehdyt muutokset, minkä jälkeen määrittelydokumentti joko hyväksytään tai tarkastetaan uudelleen. [Fel01]

Tarkastus on muodollinen kokous. Sitä johtaa puheenjohtaja, joka ei itse ole ollut vaatimusmäärittelyjen tuottamisessa mukana. Kokouksen aikana vaatimusten määrittelijä esittelee jokaisen vaatimuksen vuorollaan ryhmän kommentoitavaksi ja löydetyt ongelmat kirjaa sihteeri myöhäisempää keskustelua varten.

Korjaavia toimenpiteitä vaativat esimerkiksi vaatimukset, jotka on kirjoitettu huonosti. Kirjoittajan täytyy parantaa vaatimuksia kirjoittamalla ne uudestaan. Vaatimusmäärittelydokumentista saattaa puuttua tietoja. Vaatimusten määrittelijöiden täytyy etsiä puuttuvaa tietoa systeemin asianosaisilta tai muista vaatimusten lähteistä. Jos vaatimukset ovat ristiriitaisia, niin asianosaisten on neuvoteltava ratkaistakseen ristiriidat. Epärealistiset vaatimukset on poistettava tai ne on muutettava realistisiksi. [Fel01]

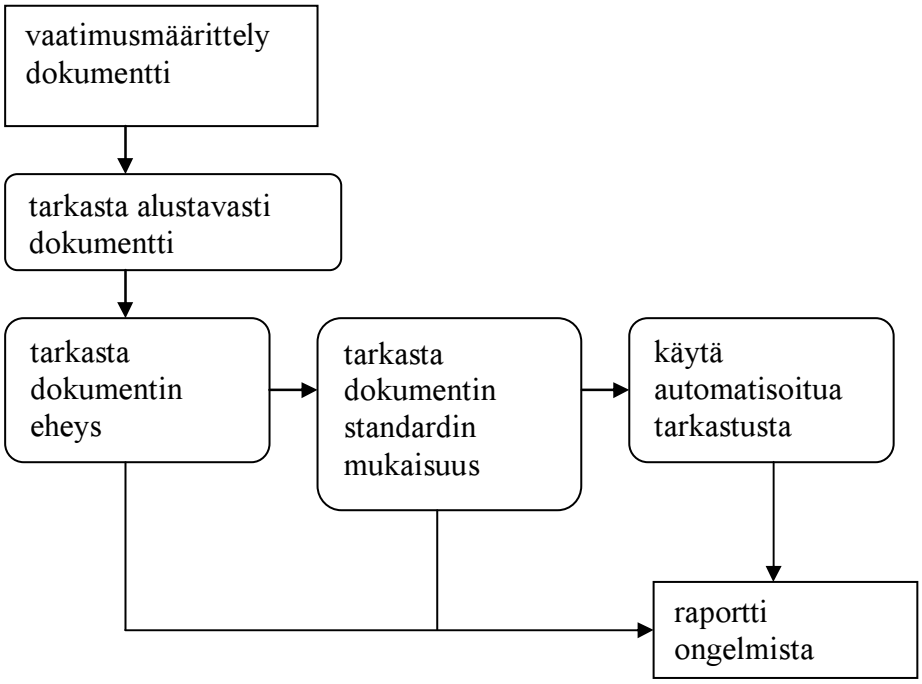
Tarkastustilaisuuden kesto riippuu tietenkin vaatimusmäärittelydokumentin koosta, mutta maksimi kestoaika saisi olla vain noin 2 tuntia eli saman verran kuin varataan aikaa valmistautumiseen. Jos arvioidaan, että tunnissa voidaan käsitellä noin 4-8 sivua dokumentaatiota ja 300 - 600 koodiriviä, niin tyypillistä vaatimusmäärittelyä ei pystytä tarkistamaan yhdellä kerralla. Dokumentti on jaettava loogisiin paloihin, joita voidaan tarkastaa eri tarkastustilaisuuksissa.

6.2 Tarkastuksen jäsenet

Tarkastukseen osallistuu yleensä 3 – 6 henkilöä, joiden valinta on tärkeää. Tarkastajien pitäisi edustaa taustoiltaan erilaisia ihmisiä. Jos mahdollista, ryhmään pitäisi kuulua systeemin loppukäyttäjä tai loppukäyttäjän edustaja, asiakkaan edustaja, yksi tai useampi alan asiantuntija, systeemin suunnittelija, toteuttaja sekä vaatimusten määrittelijä. Erilaiset taustat tuovat tarkastukseen *kokemusta*, erilaisia *taitoja* ja *tietoj*a. [KoS98], [SoS97]

6.3 Esitarkastus

Tarkastukseen kuluu paljon aikaa ja kustannuksia, joten tarkastajien työn määrää kannattaa vähentää. Virheet, jotka voidaan löytää ilman, että koko ryhmä kokoontuu, poistetaan vaatimusmäärittelydokumentista, ennen kuin se jaetaan ryhmän jäsenille. Tällaisia virheitä ovat esimerkiksi oikeinkirjoitusvirheet. Ennen dokumenttien jakamista yksi henkilö tarkastaa nopeasti dokumentin ja varmistaa, että se on rakenteeltaan oikein ja vastaa valittuja standardeja. Tarkastuksen automatisointia hyödynnetään, jos se on mahdollista. Kuvassa 6 esitellään esitarkastuksen vaiheita. [KoS98], [SoS97]



Kuva 6: Esitarkastuksen vaiheet [KoS98]

Esitarkastus on nopeaa ja edullista, koska esitarkastuksen tekemiseen tarvitaan vain yksi ihminen. Tehtävään valitaan henkilö, joka ei ole ollut tekemässä vaatimusmäärittelydokumenttia, mutta joka tuntee kuitenkin vaatimusmäärittelyjen standardeja. Hän vertaa vaatimusmäärittelydokumentin rakennetta määriteltyyn standardiin ja tuo esille puuttuvia tai epätäydellisiä kohtia. Esitarkastuksessa katsotaan myös, että dokumentin kaikki sivut on numeroitu, diagrammit nimetty ja keskeneräisiä vaatimusten määrittelyjä ei esiinny dokumentissa. Jos esitarkastuksen jälkeen aikaa on tarpeeksi, dokumentti voidaan palauttaa vaatimusten määrittelijöille takaisin korjattavaksi. Jos aikaa ei ole,

niin esitarkastuksessa löydetyt poikkeamat kirjataan ja jaetaan suoraan tarkastusryhmän jäsenille. [KoS98], [GoM99]

7 RIIPPUMATTOMAT VALIDOINTI- JA VERIFIOINTIRYHMÄT

Ohjelmistojen validoinnissa ja verifioinnissa tarkistetaan, että ohjelmiston vaatimukset toteutetaan oikein ja täysin ja että ne ovat jäljitettävissä asiakkaan vaatimuksiin. *Validointi- ja verifiointiprosessissa* tuotetaan *validointi- ja verifiointisuunnitelmat*, jotka sisältävät yksittäisiä suunnitelmia, tehtäväraportteja, yhteenvetoraportteja, poikkeamaraportteja ja lopullisen ohjelmiston verifiointi- ja validointiraportin. Suunnitelmia päivitetään tarvittaessa, jos prosessin aikana tulee vaatimuksiin lisäyksiä tai muutoksia. [CuI96]

Validointia ja verifiointia suunnittelevien henkilöiden *objektiivisuus* tuleviin tehtäviin pitäisi pystyä arvioimaan. Eräs keino olisi käyttää suunnitteluun kahta eri ryhmää. Tällaista eri organisaatioiden käyttämistä (muuta kuin ohjelmiston kehittämisryhmää), kutsutaan *riippumattomaksi verifioinniksi ja validoinniksi*. *Tekninen riippumattomuus (technical independence)* tarkoittaa sitä, että validointi- ja verifiointiryhmä ei ole henkilökohtaisesti mukana ohjelmiston kehittämistyössä. Ryhmällä pitää olla jonkin verran tietoa systeemisuunnittelusta tai heillä täytyy olla kokemusta alalta, jotta he voisivat ymmärtää systeemiä. Verifiointi- ja validointiryhmään ei ohjelmiston kehittämisryhmä saa vaikuttaa, kun tutkitaan systeemin vaatimuksia, rakennettavan systeemin ratkaisuehdotuksia ja ongelmien kohtaamista. Tekninen riippumattomuus on ratkaisevaa, kun on kyse ryhmän kyvystä havaita pieniä ohjelman vaatimuksia, ohjelman suunnittelun puutteita ja koodausvirheitä, jotka eivät tule esiin testauksessa tai katselmuksissa. Tekninen ryhmä saattaa joutua jakamaan työkaluja muun ympäristön kanssa, mutta silloin yhteiset työkalut pitäisi testata, että ne eivät itse peitä analysoitavan ja testattavan ohjelmiston virheitä. Tekninen ryhmä käyttää ja kehittää oman testijoukon ja työkalut erillään kehittäjien työkaluista aina, kun se vain on mahdollista. [CuI96]

Työnjohdollinen riippumattomuus (managerial independence) merkitsee, että vastuu riippumattomasta validoinnista ja verifioinnista kuuluu urakoitsijan tai ohjelmiston kehittäjien ulkopuoliselle organisaatiolle. Validointi- ja verifiointiryhmä päättää itsenäisesti ohjelmiston tai systeemin testattavista ja analysoitavista alueista, käytettävistä tekniikoista, tehtävien aikatauluista ja toimitettavista teknisistä asioista. Validointi- ja veri-

fiointiryhmä antaa tuloksensa samaan aikaan sekä ohjelman kehittäjille että systeemin johdolle, jotka toimivat raportoitujen tulosten ja poikkeamien mukaan. [CuI96]

Taloudellinen riippumattomuus (financial independence) merkitsee, että validoinnin ja verifioinnin talousarvion kontrolli kuuluu urakoitsijan tai ohjelmiston kehittäjien ulkopuoliselle organisaatiolle. Tällainen riippumattomuus takaa, että rahavaroja ei ohjata muualle. Se suojelee lisäksi epäsuotuisilta taloudellisilta paineilta ja vaikutuksilta, jotka saattaisivat viivyttää tai jopa pysäyttää verifiointi- ja validointianalyysin. [CuI96]

8 VERIFIOINNIN JA VALIDOINNIN TEKNIIKOITA

Validointi- ja verifiointitekniikat voidaan jakaa kolmeen pääluokkaan: *staattiseen (static), dynaamiseen (dynamic*) ja *formaaliin (formal) analyysiin* [CuI96]. Staattisen analyysin tekniikat *analysoivat tuotteen muotoa ja rakennetta* toteuttamatta itse tuotetta. Katselmukset, tarkastukset, arvioinnit ja tietovirta-analyysit ovat staattisen analyysin tekniikoita. Tämän tyyppisiä tekniikoita käytetään yleensä ohjelmiston vaatimusten, suunnittelun ja lähdekoodiin validointiin ja verifioitiin, mutta niitä voidaan käyttää myös testidokumentaatioissa. [CuI96], [DoT90]

Dynaamisen analyysin tekniikoiden avulla *tarkastellaan tuotteen toteuttamista tai simulointia*. Näillä tekniikoilla etsitään virheitä tutkimalla, kuinka erilaiset syöttötiedot vaikuttavat ohjelmistoon. Tulostetiedot tai arvojen vaihteluvälit on tiedettävä ennen kuin dynaamisen analyysin tekniikoita voidaan käyttää. Testaus ja prototyyppi kuuluvat dynaamisen analyysin tekniikoihin.

Formaalissa analyysissä *käytetään tarkkoja matemaattisia tekniikoita ratkaisujen algoritmien analysointiin*. Joskus ohjelmiston vaatimukset on kirjoitettu formaalilla määrittelykielellä (esimerkiksi VDM, Z), jolloin niitä voidaan verifioida esimerkiksi formaalin analyysin tekniikoihin kuuluvilla *todistustekniikoilla (proof-of-correctness)*. Termiä formaalinen käytetään usein myös puhuttaessa hyvin suunnitellusta, johdetusta, dokumentoidusta ja toistettavasta prosessista. Tässä merkityksessä kaikki validointi- ja verifiointitekniikat ovat formaaleja, vaikka ne eivät välttämättä toteuta matemaattisia tekniikoita. Taulukoissa 2 ja 3 on lueteltu joitakin validointi- ja verifiointitekniikoita. Harmaalla väritetyt ruudut kertovat, mihin ohjelmiston elinkaaren eri vaiheiden verifiointiin tai validointiin tekniikka sopii. [CuI96], [DoT90]

Taulukko 2: Ohjelmiston verifiointi- ja validointitekniikoita [CuI96].

TEKNIIKKA	VAATIMUK-SET	SUUNNITTE-LU	KOODI	YKSIKKÖTES-TAUS	INTEGROINTI-TESTAUS	SYSTEEMI-TESTAUS	ASENNUS	KÄYTTÖ	YLLÄPITO
algoritmi-analyysi									
analyyttinen mallinnus									
vertailu-testi									
raja-arvo-analyysi									
koodin lukeminen									
kontrollivirta-analyysi									
kattavuus-analyysi									
kriittinen ajoitus/virta-analyysi									
tietokanta-analyysi									
tietovirta-analyysi									
päätöstaulut									
pöytätar-kastus									
virheiden kylvämi-nen									
tapahtumapuu-analyysi									
tilakone									
toiminnallisuu-den testaus									
tarkastus									
raja-pinta testaus									
rajapinta-analyysi									
muutos-analyysi									
suorituskyky-testaus									
Petri-ver-kot									
oikeelli-suuden todistami-nen									
proto-tyyppi									
regressioana-lyysi ja testaus									
vaatimus-ten jäsentäminen									
katselmus									
tarkkuus-testaus									
simulointi									

Taulukko 3: Ohjelmiston verifiointi- ja validointitekniikoita (jatkuu) [CuI96].

TEKNIIKKA	VAATIMUK-SET	SUUNNITTE-LU	KOODI	YKSIKKÖTES-TAUS	INTEGROINTI-TESTAUS	SYSTEEMI-TESTAUS	ASENNUS	KÄYTTÖ	YLLÄPITO
koko- ja aika-analyysi		■	■	■	■				■
hajottaminen			■						
Vika-, vaikutus- ja kriittisyys-analyysi	■	■	■						■
Ohjelmiston vikapuu-analyysi	■	■	■						
rasitetestaus				■	■	■			
rakenne-testaus				■					
symbolinen toteutus			■						
Testin varmenta-minen				■	■	■	■		
läpikäynti	■	■	■	■	■	■	■		
UUDELLEENKÄYTTÖ (REUSE)									
Ristiriidatto-muustesti	■								
TIETOPOHJAISET SYSTEEMIT (KNOWLEDGE-BASED SYSTEMS)									
vaihtoehto-malli	■							■	
kontrolli-ryhmät				■					
uskottavuus-testaus				■					
kenttätestaus								■	
väärien attribuuttien testaus			■						
looginen verifiointi	■								
metamallit				■					
osittamis-testaus	■	■	■						
sääntöjen verifiointi	■								
tilastollinen validointi				■					
turingin testaus				■					
painoanalyysi				■					

8.1 Tekniikoiden kuvauksia

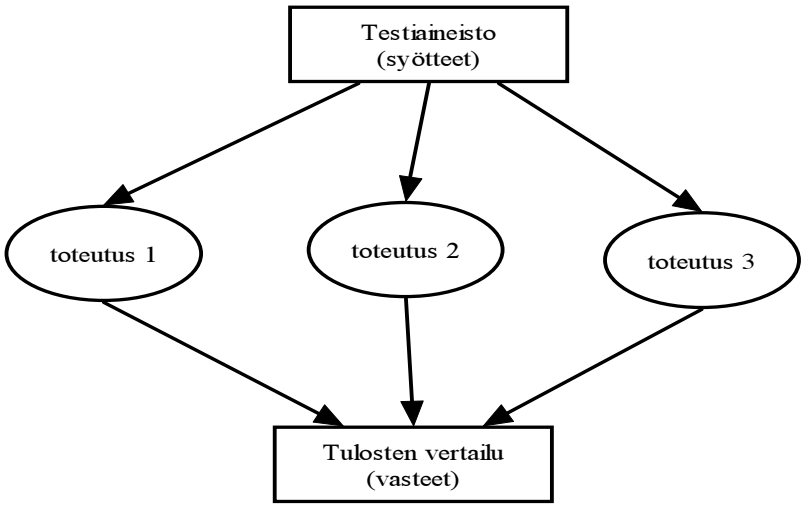
Erilaisia ohjelmiston validointi- ja verifiointitekniikoita on paljon. Seuraavissa tekniikoiden lyhyissä kuvauksissa on kuvausten perässä lueteltu, minkä tyyppisiä virheitä tekniikalla löydetään ja mitä ja millaisia tehtäviä tekniikka tukee. [CuI96], [DoT90]

Algoritmianalyysi (algorithm analysis) tutkii ohjelmiston vaatimusten loogisuutta ja paikkansa pitävyyttä kääntämällä algoritmit samalle kielelle tai samanlaiseen rakennemuotoon. Algoritmianalyysi tarkistaa, että algoritmit ovat sopivia, tarkoituksenmukaisia ja stabiileja ja että kaikki tarkkuus-, ajoitus- ja mitoitusvaatimukset täyttyvät. Algorit-

mianalyysi tarkistaa yhtälöiden ja numeeristen tekniikoiden oikeellisuutta, lyhennysten ja pyöristysten vaikutuksia ja muuttujien ja sanojen taltioinnin tarkkuuksia. *Käyttö:* tarkkuus, algoritmin tehokkuus, oikeellisuus, laskennan paikkansapitävyys, virheiden leviäminen, numeroiden pyöristäminen, numeerinen pysyvyys, tilan hyväksikäytön arviointi, systeemin suorituskyvyn ennustaminen, ajoitus.

Analyyttinen mallinnus (analytic modelling) arvioi suorituskykyä ja antaa tietoa kapasiteettisuunnittelusta. Se kuvaa ohjelman logiikkaa ja mallien prosessointia ja analysoi niiden riittävyyttä. *Käyttö:* tarkkuus, algoritmin tehokkuus, pullonkaulat, virheiden leviäminen, toteutettavuus, mallinnus, numeroiden pyöristäminen, numeerinen pysyvyys, prosessoinnin toteutettavuus, systeemin suorituskyvyn ennustaminen.

Vertailutestaus (back-to-back testing tai comparison testing) selvittää testivirheitä vertaamalla kahden tai useamman samalla määrittelyllä toteutettujen ohjelmien tulosteita. Ohjelmaversioille annetaan samat syöttötiedot ja saatuja tulosteita verrataan poikkeamien löytämiseksi. Tästä on esimerkki kuvassa 7. Testausaineisto voidaan valita millä testausstrategialla tahansa, mutta satunnaistestaus sopii hyvin vertailutestaukseen. *Käyttö:* poikkeamat, versioiden väliset eroavuudet.



Kuva 7: Vertailutestauksen idea [Koi00]

Raja-arvoanalyysi (boundary value analysis) etsii ja poistaa virheitä parametrien rajoilta tai raja-arvoista. Ohjelman syöttöalue jaetaan luokkiin, joista valituilla testitapauksilla testataan ohjelmiston toimintaa. Valittujen testitapausten pitää kattaa luokkien rajat ja ääripäät. Testaamalla tarkistetaan, että määrittelyn syöttöalueen rajat vastaavat ohjelmassa olevia rajoja. Esimerkiksi nollan käyttöön on kiinnitettävä tarkkaa huomiota (nollalla jakaminen, nollamatriisi). Yleensä syöttötietojen raja-arvot tuottavat tulostietojen raja-arvot. Testitapaukset pitää suunnitella niin, että tulostietojen rajatapaukset saavutetaan. *Käyttö:* algoritmianalyysi, joukkojen määrä, rajojen ristiriidat, määrittelyvirheet.

Koodin lukeminen (code reading) on tekniikka, missä asiantuntija lukee toisen ohjelmoijan koodia löytääkseen virheitä. Suositeltavaa olisi, että jo pseudokoodia tarkasteltaisiin ennen varsinaista koodausta. *Käyttö:* oikeellisuus, muuttujien väärinkäyttö, poisjätetyt funktiot, parametrien tarkistus, huono ohjelmointikäytäntö, tarpeeton koodi.

Kontrollivirta-analyysissä (control flow analysis) muutetaan tekstimuodossa olevat ohjelmistovaatimukset graafiseen muotoon ja saatujen kaavioiden avulla tutkitaan vaatimusten oikeellisuutta eli ehdotetun kontrollivirran kulun pitää olla ongelmatonta. Kontrollivirta-analyysiä käytetään selventämään ohjelmiston päärutiinien ja alifunktioiden hierarkiaa. Sen avulla huomataan huonot ja mahdollisesti väärät ohjelman rakenteet. *Käyttö:* väittämätestaus (assertion testing), pullonkaulat, rajatestitapaukset, haarojen ja polkujen tunnistaminen, päätöstestaus, yksiköiden solurakenne, oikeellisuus, ohjelmiston suunnittelun arviointi, virheiden leviäminen, odotetut/todelliset tulokset, peräkkäistiedostojen virheet, formaalin määrittelyn arviointi, globaali tietovirta ja johdonmukaisuus, yksiköiden hierarkkiset suhteet, integrointitestaus, yksikön sisäinen rakenne, silmukan muuttujat, polkutestaus, prosessoinnin tehokkuus, muutosten jälkeinen testaus, suorituskyvyn ennustaminen, testitapausten suunnittelu, yksikkötestaus.

Kattavuusanalyysi (coverage analysis) mittaa, kuinka kattavasti systeemi tai sen rakenneyksiköt on testattu annetuilla testitapauksilla. Systeemitason kattavuus kertoo, kuinka monessa systeemin yksityisessä osassa on käyty valitulla testiaineistolla. Koodikattavuus ilmaisee, kuinka monta prosenttia ohjelman haaroista, lauseista tai koodiriveistä on testattu. *Käyttö:* yksikkötestaus, integrointitestaus, järjestelmätestaus.

Kriittinen ajoitus/virta- analyysi (critical timing/flow analysis) tarkistaa, että prosessin ja kontrollin ajoitusvaatimukset täyttyvät mallintamalla ohjelmiston näitä piirteitä. *Käyttö:* mallinnus, tahdistaminen, ajoitus.

Tietokanta-analyysi (database analysis) on tekniikka, jonka avulla tutkitaan, että tietokannan rakenne ja pääsymetodit sopivat loogiseen suunnitteluun. Tietokanta-analyysiä käytetään, jos ohjelmilla on merkittävä tietovarasto. Tietokanta-analyysin avulla taataan, että yleistä dataa ja muuttuja-alueita käytetään johdonmukaisesti kaikissa kutsurutiineissa. Uutta tietoa ei pitäisi voida kirjoittaa vahingossa vanhan päälle eikä ylitäyttää taulukoita. *Käyttö:* käytön suojaus, datan ominaispiirteet ja tyypit, ohjelmiston suunnittelun arviointi, peräkkäistiedostojen virheet, globaali tietovirta, prosessoinnin tehokkuus, tilan hyväksikäytön arviointi, yksikkötestaus.

Tietovirta-analyysin (*dataflow analysis)* käyttö on tärkeää silloin, kun suunnitellaan korkean tason sovellusten arkkitehtuuria. Sen avulla voidaan tarkistaa erilaisia muuttujia. Esimerkiksi tutkitaan, onko muuttujat luettu ennen kuin ne on kirjoitettu tai onko niitä luettu koskaan. *Käyttö:* väittämätestaus, pullonkaulat, rajatestitapaukset, haarojen ja polkujen tunnistaminen, päätöstestaus, yksiköiden solurakenne, datan ominaispiirteet, ympäristön vuorovaikutus, virheiden leviäminen, ohjelman polkujen arviointi, odotetut/todelliset tulokset, peräkkäistiedostojen virheet, globaali tietovirta ja johdonmukaisuus, yksiköiden hierarkkiset suhteet, yksikön sisäinen rakenne, silmukan muuttujat, prosessoinnin tehokkuus, muutosten jälkeinen testaus, ohjelmiston suunnittelun arviointi, suorituskyvyn ennustaminen, testitapausten suunnittelu, alustamattomat muuttujat, käyttämättömät muuttujat, muuttujien viittaukset.

Päätöstaulujen (decision tables) avulla tutkitaan monimutkaisia loogisia yhdistelmiä ja niiden suhteita. Kyseisessä menetelmässä käytetään kaksidimensionaalisia tauluja kuvaamaan ohjelmiston Boolean-tyyppisten muuttujien välisiä suhteita. *Käyttö:* loogiset virheet.

Pöytätestaus (desk checking) on menetelmä, missä yleensä joku asiantuntija, mutta ei itse tekijä, tarkastelee ohjelmiston suunnittelua ja koodia löytääkseen virheitä. Tarkastaja etsii koodista ilmeisiä virheitä, tarkistaa vuorovaikutuksia ja lukee kommentteja. Hän yrittää selvittää, mitä koodi tekee ja vertaa sitten sitä ulkoiseen määrittelyyn. Vertaa-

malla kommentteja ohjelman suunnittelu-dokumenttiin, tarkastaja käy läpi kaikki eri polkujen syöttöehdot ja tarkastaa myös, että ohjelmoinnin standardeja ja käytäntöjä noudatetaan. *Käyttö*: vanhentunut data, olemattomien aliohjelmien kutsut, rajoittamattomat kentät, virhe suunnittelun toteutuksessa, virhe tallennettaessa tai palautettaessa rekistereitä, virheelliset sisäkkäiset silmukat ja haarat, väärät ohjelman linkitykset, väärät prosessijonot, vaillinaiset predikaatit, luvaton pääsy taulukkomuuttujiin, tehoton datan siirto, päättymättömät silmukat, alustusvirheet, syöttö-/tulostusvirheet, ohjeitten muunnokset, käännetyt predikaatit, sopimaton parametrilista, puuttuvat nimiöt tai puuttuva koodi, puuttuvat validointitestit, muuttujien väärinkäyttö, tuhlaava ohjelmointi, luvaton toistuminen, määrittelemättömät muuttujat, saavuttamaton koodi, viittaamattomat nimiöt.

Virheiden kylväminen (error seeding) on menetelmä, missä laitetaan tahallaan virheitä ohjelmistoon ennen testausta. Jos testausta suoritettaessa löytyy vain osa kylvetyistä virheistä, niin testitapausten joukko ei ole riittävä. Seuraavan kaavan avulla voidaan arvioida, kuinka paljon todellisia virheitä ohjelmistossa on [DoT90]:

$$\frac{\text{löydettyjen kylvettyjen virheiden määrä}}{\text{kylvettyjen virheiden kokonaismäärä}} = \frac{\text{löydettyjen kylvämättömien virheiden määrä}}{\text{kylvämättömien virheiden kokonaismäärä}}$$

Kaavassa on kolme muuttujaa (löydettyjen kylvettyjen virheiden määrä, kylvettyjen virheiden kokonaismäärä, löydettyjen kylvämättömien virheiden määrä) tunnettuja, joten neljäs voidaan arvioida niiden avulla. Jos kaikki tahalliset virheet löytyvät, niin se osoittaa, että testiaineisto on riittävä tai että tahalliset virheet olivat liian helppoja. *Käyttö:* testitapausten riittävyys.

Tapahtumapuu-analyysiä (event tree analysis) käytetään mallinnettaessa tapahtuman jälkiseurausten vaikutuksia. Alulle paneva tapahtuma on puun juurena. Juuresta lähtee kaksi haaraa, joista toiseen kuvataan tapahtuman positiiviset ja toiseen negatiiviset seuraukset. Menetelmää jatketaan niin kauan, että kaikki seuraukset on käsitelty. *Käyttö:* vikatiheyden analysointi, turvallisuus, käsittelyanalyysi, ajoitus.

Tilakoneilla (finite state machines) tarkistetaan ohjelmistovaatimusten epäyhtenäisyydet ja puutteet mallintamalla ohjelmistoa syöttöjen ja toimintojen avulla. Esimerkiksi kun systeemi on tilassa S1 ja saa syötteen I, niin se tuottaa toiminnon A. Sen jälkeen tila muuttuu tilaksi S2. Tilakone-analyysillä voidaan tarkistaa, että jokaisen syötteen jälkeen tapahtuu toiminto ja tilanvaihto ja että vain yksi tilanvaihto on määritelty jokaiselle syöte ja toimintoparille. *Käyttö:* epätäydellinen vaatimusten määrittely, ristiriitaiset ohjelmistovaatimukset, mallinnus.

Toiminnallisen testauksen (functional testing) avulla joko osa tai koko systeemi testataan, kun halutaan tarkastaa, ovatko käyttäjän antamat vaatimukset täyttyneet. *Käyttö:* peräkkäistiedostojen virheet, polkutestaus, ohjelman toteutuksen tunnuspiirteet, muutosten jälkeinen testaus, väittämän kattavuustestaus, systeemin suorituskyvyn ennustaminen, järjestelmätestaus, testitapausten suunnittelu, testauksen perusteellisuus, yksikkötestaus, alustamattomat muuttujat, käyttämättömät muuttujat, muuttujien viitteet, muuttujakuvaukset/seurannat.

Rajapinta-analyysi (interface analysis) on staattinen analysointitekniikka. Sen avulla demonstroidaan, että aliohjelmien rajapinnat eivät sisällä sellaisia virheitä, jotka aiheuttaisivat virheitä ohjelmiston johonkin sovellukseen. Rajapinta-analyysiä tarvitaan varsinkin silloin, kun rajapinnat eivät sisällä ominaisuuksia, jotka havaitsisivat vääriä parametrin arvoja. Tätä tekniikkaa tarvitaan myös, jos tehdään uusia kokoonpanoja jo olemassa olevista aliohjelmista. *Käyttö:* todellisten ja formaalien parametrien yhteensopimattomuus, kutsuttujen ja käytettävissä olevien aliohjelmien väliset ristiriidat, globaalimuuttujien attribuuttien epäjohdonmukaisuudet, ristiriidat COTS -parametrien käytössä suhteessa systeemin muihin parametreihin, väärät oletukset staattisten ja dynaamisten arvojen tallennuksista, väärien funktioiden käyttö tai väärä aliohjelman kutsu, syöte- ja tulostekuvausten virheet.

Rajapintatestaus (interface testing) on dynaaminen analysointitekniikka. Se muistuttaa rajapinta-analyysiä. Erona rajapinta-analyysiin rajapintatestauksessa testitapaukset valitaan niin, että niillä testataan kaikki rajapinnat. Rajapinta-testaukseen voi kuulua esimerkiksi seuraavanlaisia tapauksia: testataan kaikki rajapintamuuttujat niiden äärirajoilla tai testataan rajapintamuuttujat yksi kerrallaan äärirajoilla muiden muuttujien saadessa normaaliarvoja. *Käyttö:* todellisten ja formaalien parametrien yhteensopimattomuus,

väärien funktioiden käyttö tai väärä aliohjelman kutsu, syöte- ja tulostekuvausten virheet.

Muutosanalyysissä (mutation analysis) selvitetään, onko ohjelmaa testattu riittävästi. Tällä menetelmällä tehdään useita alkuperäisen ohjelman eri muunnelmia eli versioita muuttamalla ohjelman yksittäisiä elementtejä. Jokainen versio testataan valitulla testiaineistolla. Koska jokainen versio on erilainen kuin alkuperäinen, testauksen pitää osoittaa, että näin todellakin on. Jos jokaisen version tulosteet eroavat alkuperäisen ohjelman ja toistensa tulosteista, niin ohjelma voidaan katsoa riittävästi testatuksi. *Käyttö:* rajatestitapaukset, haarojen ja polkujen määrittely, päätöstestaus, muutosten jälkeinen testaus, testitapausten esikäsittely.

Suorituskykytestausella (performance testing) mittataan kuinka hyvin ohjelmistosysteemissä toteutuvat vasteaika, keskusyksikön käyttö ja muut toiminnan määrällisesti ilmaistavat piirteet [CuI96]. *Käyttö:* muistin jakaminen, tahdistus, ajoitus.

Petri-verkojen (Petri-nets) avulla varmistetaan, että ohjelmistosuunnittelussa on huomioitu toimintahäiriöt ja muut turvallisuusongelmat. Petri-verkot koostuvat *paikoista (places)* ja paikkojen välisistä *nuolista (arrows*) sekä resursseja esittävistä *täplistä (token)*. Miten täplät sijoittuvat paikoille, kuvaa yksikäsitteisesti verkon tilan. Verkkojen avulla voidaan nähdä, kuinka ohjelmistosuunnittelu toimii tiettyjen olosuhteiden vallitessa. Petri-verkoilla voidaan selvittää kaikki systeemin saavutettavat tilat antamalla erilaisia alkuehtoja. *Käyttö:* virheanalyysi, mallinnus, turvallisuus, uhka-analyysi, ajoitus.

Oikeellisuuden todistaminen (proof of correctness) on menetelmä, jossa käytetään teoreettisia ja matemaattisia malleja ohjelman oikeellisuuden osoittamiseen ilman ohjelman toteuttamista. Ohjelma esitetään teoreemana ja sen oikeellisuus todistetaan predikaattilaskennan avulla. *Käyttö:* oikeellisuus, kriittisten osien näyttö.

Prototyypit (protyping) helpottavat toteutettavan ohjelmiston vaatimusten tutkimista. Niiden avulla voidaan huomata epätäydelliset ja virheelliset vaatimukset. Jos jokin vaatimus aiheuttaa ei-toivotun systeemin käyttäytymisen, prototyyppi voi paljastaa sellaisenkin. Tätä menetelmää voidaan käyttää myös apuna suunniteltaessa ohjelmiston toi-

mintopohjaista arkkitehtuuria. Monimutkaisissa systeemeissä prototyypit voivat estää huonon, kalliin ja hukkaan menevän ohjelmiston lopullisen toteutuksen. *Käyttö:* toiminta, poisjätetyt toiminnot, epätäydellinen ohjelmiston vaatimusmäärittely, käyttöliittymä.

Regressioanalyysiä ja testausta (regression analysis and testing) käytetään uudelleenarviointiin, kun ohjelmiston koodiin on tehty muutoksia. Muunnellun ohjelmiston pitää toteuttaa määritellyt vaatimukset muutosten jälkeenkin. Regressioanalyysi ja testaus tehdään siis jokaisen muutoksen jälkeen, koska on varmistettava, että kaikki vaatimukset toteutuvat ja ettei ohjelman muita osia ole vioitettu. *Käyttö:* integraatiotestaus, muutosten jälkeinen testaus, järjestelmätestaus.

Vaatimusten jäsentäminen (requirements parsing) takaa sen, että kaikki ohjelmiston vaatimukset on määritelty yksiselitteisesti. *Käyttö:* virheettömyys, väittämätestaus, tarkistuslistat, eheys, johdonmukaisuus, ympäristön vuorovaikutus, formaalin määrittelyn arviointi, yksiköiden hierarkkiset suhteet, tietovirran eheys, ohjelmiston integraatiotestaus, sisäisten yksikköjen rakenne, polkutestaus, oikeellisuuden todistaminen, ohjelmiston vaatimusten arviointi, ohjelmiston vaatimusten luettelointi, vaatimusten ja suunnittelun vastaavuus, muutosten jälkeinen testaus, standardien tarkistus, kattavuustestaus, järjestelmätestaus, yksikkötestaus.

Katselmus (reviews) on kokous, missä ohjelmiston vaatimukset, ohjelmiston suunnittelu, koodi tai muut tuotteet esitellään käyttäjille, sponsoreille tai muille kiinnostuneille. He saavat kommentoida esitystä tai hyväksyä esityksen [Ter01]. *Käyttö:* tehokas menetelmä ennen testausta, loogiset virheet, syntaksivirheet.

Herkkyysanalyysillä (sensitivity analysis) arvioidaan todennäköisyyksiä, miten hyvin ohjelmiston testaussuunnitelma havaitsee ohjelmiston virheet testauksen aikana. Herkkyysanalyysiä käytetään myös arvioitaessa, mitkä koodin alueet todennäköisimmin vaikuttavat ylläpitoon (koodin muutokset). *Käyttö:* oikeellisuus, loogiset virheet, luotettavuus, testitapausten riittävyys.

Simulaation (simulation) avulla arvioidaan suurien monimutkaisten systeemien vuorovaikutuksia, kun laitteita, käyttäjiä tai muita ohjelmiston rajapintayksiköitä on paljon. Simulaatiossa tutkitaan ohjelmiston käyttäytymistä. *Käyttö:* väittämätestaus, käyttäy-

tyminen, raja-arvotestitapaukset, haarojen ja polkujen tunnistaminen, päätöstestaus, ympäristön vuorovaikutus, toteutuksen valvonta, toteutettavuus, peräkkäistiedostojen virheet, polkutestaus, ohjelman toteutuksen piirteet, muutosten jälkeinen testaus, kattavuustestaus, yksikön sisäinen rakenne, ohjelmiston suunnittelun arviointi, suorituskyvyn ennustaminen, järjestelmätestaus, alustamattomat muuttujat, käyttämättömät muuttujat, muuttujien viittaukset, muuttujakuvaukset/seurannat.

Koko- ja aika-analyysi (sizing and timing analysis) auttaa selvittämään, onko laitteisto- ja ohjelmistojako asianmukaista ohjelmiston arkkitehtuurin kannalta. Analyysiä käytetään koodin kasvukehityksen aikana. Ohjelmiston koko- ja toteutusaika-arvoista ilmenee, toteuttaako ohjelmisto sille annetut prosessorin koko- ja suorituskykyvaatimukset. Merkittävät poikkeukset todellisten ja ennakoitujen arvojen välillä kertovat täydentävän tutkimisen tarpeesta. *Käyttö:* algoritmin tehokkuus, pullonkaulat, raja-arvotestitapaukset, haarojen ja polkujen tunnistaminen, päätöstestaus, integraatiotestaus, prosessin tehokkuus, ohjelman toteutuksen piirteet, muutosten jälkeinen testaus, tilan hyväksikäytön arviointi, järjestelmätestaus, ajoitus, yksikkötestaus.

Siivuttaminen (slicing) on hajotustekniikka, missä tulostemuuttujien alkuperää jäljitetään koodin läpi, jotta tunnistettaisiin kaikki koodilauseet, joilla on merkitystä ohjelman laskentatoimenpiteille. Tätä tekniikkaa voidaan käyttää myös, kun demonstroidaan toiminnallista monimuotoisuutta. *Käyttö:* yleinen koodi, tietovirran johdonmukaisuus, ohjelman hajottaminen, muuttujien viitteet.

Vika-, vaikutus- ja kriittisyysanalyysillä (software failure mode effects and criticality analysis) paljastetaan ohjelmiston huonot tai puuttuvat vaatimukset käyttämällä induktiivista järkeilyä. Sen avulla selvitetään erilaisten toimintahäiriöiden vaikutuksia systeemiin osiin. Jokainen osa tutkitaan erityyppisillä häiriöillä ja tuloksista tehdään matriisi. Matriisista löytyy häiriöntyyppi, häiriön syyt, häiriön seuraukset, kriittisyys, muutosvaatimukset ja estämis- ja suojatoimet. Kriittisyystekijä ilmoittaa häiriön vaikutuksen vakavuuden ja sen perusteella voidaan arvioida, missä tarvitaan lisää analysointeja ja testausta. *Käyttö:* vika-analyysi, turvallisuus, epätäydellinen ohjelmiston vaatimusten määrittely, uhka-analyysi.

Ohjelmiston vikapuuanalyysi (Software fault tree analysis) on looginen kaavio, joka esittää kohteen kriittisen vikautumisen riippuvuuden kohteen osien vioista tai ulkoisista tapahtumista [Har99]. Sitä käytetään määriteltäessä tunnettujen vikojen mahdollisia syitä. Sen tarkoituksena on demonstroida, että ohjelmisto ei aiheuta systeemille epäluotettavaa tilaa, ja löytää sellaiset ympäristön olosuhteet, jotka voisivat aiheuttaa epäluotettavan tilan. Analyysissä oletetaan, että tunnistettu vika on jo tapahtunut ja yritetään takautuvasti löytää vian mahdolliset syyt. Tämä tehdään tekemällä vikapuu, jonka juureksi laitetaan tutkittava vika. Systeemin vikapuuta kasvatetaan kunnes se sisältää niin matalan tason perustapahtumia, ettei niitä voi analysoida enempää. *Käyttö:* vika-analyysi, turvallisuus, uhka-analyysi.

Rasitustestaus (stress testing) testaa systeemin vastetta äärimmäisissä olotiloissa tunnistaakseen ohjelmiston haavoittuvat paikat. Sen avulla osoitetaan, että systeemi voi kestää normaaleissa työmäärissä. *Käyttö:* suunnitteluvirheet, ylikuormitetut ohjelmat.

Rakennetestausella (structural testing) tutkitaan ohjelmiston osien loogisuutta. Rakennetestausta voidaan käyttää myös ohjelmiston vaatimusten kattavuustestauksen tukemiseen. *Käyttö:* pullonkaulat, virheiden lisääntyminen, ohjelman polkujen arviointi, parametritarkistus, ohjelman toteutuksen piirteet, muutosten jälkeinen testaus.

Symbolisen toteutuksen (symbolic execution) avulla osoitetaan, että lähdekoodi ja ohjelmiston vaatimusmäärittely ovat yhtäpitäviä. Tässä arviointitekniikassa ohjelmaa simuloidaan käyttäen symboleita syöttötiedon todellisten arvojen sijasta. Ohjelman tulosteet esitetään myös loogisella tai matemaattisella tavalla käyttäen symboleja. *Käyttö:* väittämätestaus, ohjelman toteutuksen piirteet, oikeellisuuden todistaminen, muutosten jälkeinen testaus.

Testin varmentamisella (test sertification) taataan, että raportoidut testitulokset ovat todellakin testistä saatuja tuloksia. Testaukseen liittyvät työkalut, media ja dokumentaatio todistetaan sellaisiksi, että ne takaavat testien ylläpidettävyyden ja toistettavuuden. Tätä tekniikkaa käytetään myös osoittamaan, että luovutettu ohjelmistotuote on testattu määritellyillä validointi- ja verifiointitekniikoilla. Varsinkin kriittisissä ohjelmistosysteemeissä menetelmää käytetään tarkistettaessa, että kaikki vaadittavat testaukset on

tehty. *Käyttö:* väärän tuoteversion lähetys, väärät testitulokset, tekemättä jätettyjen testitapausten raportit.

Läpikäynti (walkthrough) on epävirallinen tarkastustilaisuus. Läpikäynti on arviointitekniikka, missä suunnittelija tai ohjelmoija pyytää yhtä tai useampaa tiimin jäsentä tarkastelemaan ohjelman osia tai koodia, kun muut kyselevät tai tekevät kommentteja tekniikasta, tyylistä, mahdollisista virheistä, standardien rikkomisista ja muista ongelmista [Ter01]. *Käyttö:* tehokas testauksen edeltäjä, formaalin määrittelyn arviointi, loogiset virheet, manuaalinen simulointi, parametritarkistus, muutosten jälkeinen testaus, virhealttiit suunnittelun tai koodauksen alueet, statuskatselmus, syntaksivirheet, järjestelmätestaus, tekniset katselmukset.

8.2 Uudelleenkäytettävien ohjelmien tekniikat

Suurin osa validointi- ja verifiointi tekniikoista sopii myös uudelleenkäytettävien ohjelmien validointiin ja verifiointiin. Seuraavat kaksi tekniikkaa ovat tärkeitä tällä alueella:

Ristiriidattomuustestillä (consistency analysis) vertaillaan minkä tahansa olemassa olevan ohjelmiston vaatimuksia uuden ohjelmiston vaatimuksiin, jotta voitaisiin taata ristiriidattomuus [CuI96]. *Käyttö: ristiriidattomuus.*

Rajapinta analyysillä (interface analysis) tutkitaan uudelleenkäytettävän ohjelman liittämistä jonkin uuden systeemin osaksi. Uudelleenkäytettävän koodin pitää mukautua alkuperäisen käytön ja uuden systeemin ohjelmistoarkkitehtuurin, suoritusympäristön ja sovellusalueen eroihin.

8.3 Tietopohjaisten systeemien tekniikat

Termi *tietopohjainen (knowledge-based) systeemi* viittaa systeemiin, jossa käytetään tai käsitellään monimutkaista tietoa tai tietorakenteita. Taulukossa 3 on tietopohjaisten systeemien validointiin ja verifiointiin liitettyviä tekniikoita.

*Vaihtoehtoisella mallilla (alternative mod*el) vertaillaan kohdealuemallia toiseen vaihtoehtoiseen malliin tutkittaessa eheyttä ja virheettömyyttä. *Kontrolliryhmiä (control groups)* käytetään testauksen aikana vertaamaan suorituskykyä tai jonkin tehtävän suoritusta. *Uskottavuusanalyysiä (credibility analysis)* käytetään, kun vertaillaan systeemin tuloksia asiantuntijoiden antamiin arvioihin systeemin uskottavuuden mittaamiseksi. *Kenttätestausta (field testing)* käytetään vain matala riskisissä sovelluksissa. Silloin tietopohjaista systeemiä testataan todellisessa ympäristössä.

Testaamalla vääriä attribuutteja (illegal attribute testing) tarkistetaan sääntöjä ja rajoitteita. Menetelmä on tehokas, kun eliminoidaan virheitä, toteutettaessa tietopohjaisen systeemin kehitysprosessia. *Loogisessa tarkistuksessa (logical verification)* käytetään asiantuntijoiden tietoja täydellisyyden, eheyden ja muiden laatuattribuuttien tarkistamiseen, sen jälkeen kun kohdealuemalli on rakennettu. *Metamallilla (meta model)* vertaillaan tietoja ja sääntöjä kohdealueen metamalleihin.

Ositustestauksessa (partition testing) valitaan testitapaukset käyttämällä syöteavaruuden ja tulosavaruuden ositusta valintakriteereinä. Testauksessa tarkastetaan testitapausten määrittelyjä. Ositustestaus on tehokas tekniikka virheiden eliminointiin tietopohjaisten systeemien vaatimus-, suunnittelu- ja toteutusprosessin aikana. *Sääntöjen vahvistuksessa (rule verification)* tarkistetaan erilaisten sääntöjen paikkansapitävyyttä. *Tilastollisessa validoinnissa (statistical validation)* lasketaan kuinka usein tietopohjainen systeemi käyttää sääntöjä tai sääntökokoelmia tietämyskannassa. *Turingin testauksessa (Turing test)* vertaillaan systeemin suorituskykyä asiantuntijan sokkokokeiluihin. *Painoanalyysissä (weight analysis)* vertaillaan tilastollisia tietoja jo aiemmin kohdealueesta saatuihin tilastollisiin tietoihin ja sääntöihin.

9 KÄYTTÖTAPAUSTEN JA LUOKKAKAAVIOIDEN YHDISTELEMINEN

Monissa sovelluksissa varsinkin liikealalla vaatimusmäärittelyissä pääasiassa käsitellään käyttötapauksia ja luokkamalleja. Valitettavasti nämä mallit pohjautuvat usein erilaisiin mallinnustekniikoihin ja ovat myös abstraktiotasoltaan erilaisia, mikä aiheuttaa ongelmia määrittelyjen johdonmukaisuuteen ja eheyteen [DzF98]. Yksi tapa ratkaista tällaisia ongelmia on yhdistää käyttötapauksia sekä aktiviteettikaavioiden että luokkakaavioiden kanssa. [KöS01]

9.1 Kaavioiden yhteiskäyttö

Vaatimusmäärittelyssä se, kuvataanko systeemin *toiminnallisuutta*, *rakennetta* vai *käyttäytymistä*, määrää mitä kaavioita käytetään. Esimerkiksi *UML:ssä (Unified Modelling Language)* rakennettavan systeemin toiminnallisuutta yleensä kuvataan käyttötapausten avulla ja rakenteellista puolta luokkakaavioilla. Systeemin käyttäytymistä kuvailtaessa käytetään usein tilakoneita. Monissa sovelluksissa vaatimusmäärittely keskittyy systeemin toiminnallisuuden ja rakenteellisuuden kuvaamiseen ja tilakoneilla mallinnettu käyttäytyminen jää usein toisarvoiseen asemaan. Vaatimusten kehittämisprosessi ja vaatimusmäärittelyjen validointi aloitetaan yleensä määrittelemällä systeemin toiminnallisuutta. Luokkakaavioita tarvitaan, kun kehitysprosessia jatketaan eteenpäin. Luokkakaaviot sisältävät tärkeimmät *luokkatyypit*, jotka edustavat olemassa olevia asioita tai systeemin toiminnassa sattuvia tapahtumia. [KöS01]

Luokkakaaviot ovat hienorakenteisempia ja tarkempia kuin käyttötapauskaaviot. Tarkennetaan ensin käyttötapauksia, jotta niistä tulisi täsmällisemmin määriteltyjä. Yhdistetään sen jälkeen käyttötapaukset aktiviteettikaavioiden kautta luokkakaavioihin. Tällainen lähestymistapa ei ainoastaan tue *johdonmukaisuus- ja eheystarkistuksia*, vaan myös luokkakaavion verifiointia käyttötapauskaavion pohjalta. Verifiointi varmistaa sen, että luokkakaaviot voidaan yhdistää käyttötapausten kanssa. Käyttäjäkeskeiset *skenaariot* parantavat käyttötapausten aktiviteettikaavioihin perustuvia määrittelyjä ja kohdealueen avulla tehtävää käyttötapauskaavioiden validointia. Tästä johtuen luokkakaaviot todellisuudessa verifioidaan validoitujen käyttötapauskaavioiden pohjalta. [KöS01]

Vuorovaikutuskaavioilla (interaction diagrams) kuvataan usein käyttötapauksiin liittyviä *kontrollivirtoja*. Käyttötapaukset kuvataan ensin vuorovaikutuskaavioilla ja sitten niihin lisätään hierarkkisesti muodostettuja rakenteita, kuten esimerkiksi jonoja, valintoja tai toistoja. Kaikki yhden toimijan käyttötapaukset yhdistetään käyttötapauskaavioksi. Tällainen kaavio käsittelee vain vuorovaikutustietoja, mutta ei vuorovaikutusyhteyksien yksityiskohtia. Yhden toimijan näkökulman vuoksi tällainen lähestymistapa sopii ihminen–tietokone tai tietokone-tietokone vuorovaikutusten määrittelemiseen. [KöS01]

Aktiviteettikaaviot sopivat käyttötapausten kontrollivirtojen mallintamiseen. Aktiviteettikaavio on *tilakoneen variaatio*, missä tilat edustavat toimintojen tai alitoimintojen suoritusta. Aktiviteettikaavioilla ei pystytä ilmaisemaan vuorovaikutustietoa, jos vuorovaikutuselementtejä on enemmän kuin yksi. Aktiviteettikaaviolla ei myöskään pystytä kuvaamaan toimijoiden yhteyksiä eikä käyttötapausten välisiä *käyttää ja laajentaa – yhteyksiä (uses and extend relationships*). [KöS01]

9.2 Toimijat, käyttötapaukset ja toiminnot

Toimijat (acrtors) edustavat systeemin kanssa yhteyksissä olevia tahoja. Toimijalla on yksiselitteinen nimi ja toimijan roolista löytyy tietokuvaus. *Käyttötapaus (use case)* kertoo, mikä tehtävä on suoritettava ja mikä toimija tai mitkä toimijat sen suorittavat. Käyttötapauksilla on yksikäsitteinen nimi ja niiden sanallisessa kuvauksessa kerrotaan esi- ja jälkiehdot, toimijat, tapahtumienkulku ja poikkeustapaukset. Käyttötapausten yhteydessä voidaan käyttää määritettä *laajennus (extends)*, jos käyttötapaus on samankaltainen jonkun toisen käyttötapauksen kanssa, mutta tukee laajempaa toimintomäärää. Laajennusyhteydessä tapauksen toiminnallisuutta laajennetaan toisessa käyttötapauksessa olevalla tietyllä erityistoiminnolla. Jos käyttötapaus kuvataan jonkun toisen käyttötapauksen avulla, niin tällöin sanotaan, että käyttötapaus *käyttää (uses)* toista käyttötapausta. [Eer02], [FoS00]

Aktiviteettikaavion peruskomponentteja ovat *toiminnot*. Toiminnoille on annettu yksikäsitteinen nimi, tietokuvaus, esi- ja jälkiehdot sekä lopputulos. Kaaviossa on myös siirtymisiä toiminnosta toiseen. Aloitustoiminnasta päästään erilaisia polkuja pitkin

kaikkiin muihin toimintoihin. Aktiviteettikaaviossa voi olla myös useita lopputiloja. *Skenaariolla* tarkoitetaan aktiviteettikaavioiden yhteydessä yhtä polkua aloitustilasta lopputilaan, niin että siirtymisten esi- ja jälkiehdot toteutuvat. UML- notaatioissa skenaarioita voidaan kuvata *peräkkäiskaaviolla (sequence diagram)*. [KöS01]

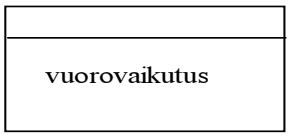
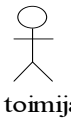
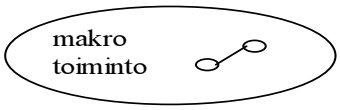
9.3 Käyttötapausten mallintaminen aktiviteettikaavioiden avulla

Käyttötapauksia muutetaan ensin sellaisiksi, että ne sopivat luokkakaavioiden rakenteeseen ja tarkkuuteen. Käyttötapauskaavioista täytyy löytyä kolmea erilaista tietoa: *systeemin sisäistä tietoa (system internal information)*, *vuorovaikutustietoa (interaction information*) ja *kontekstuaalista tietoa (contextual information)*. Systeemin sisäinen tieto kuvaa itse systeemiä. Vuorovaikutustieto kuvaa puolestaan systeemin ja sen ympäristön vuorovaikutusta ja kontekstuaalinen tieto systeemin ympäristöä. [KöS01]

Lisätään seuraavaksi aktiviteettikaavioon kaikkia kolmea erilaista tietotyyppiä. Määritellään uudet käsitteet *kontekstuaalinen toiminta (contextual action)*, *vuorovaikutustoiminta (interaction)*, *toimija toiminnassa (actor in action)* ja *makrotoiminta (macro action)*. Kontekstuaalisella toiminnalla tarkoitetaan sellaista toimintaa, johon ei tarvita systeemin apua, vaan sen toteuttaa itse toimija. Vuorovaikutustoiminnossa systeemi toteuttaa toiminnon tai auttaa sen toteuttamisessa. Toimija toiminnassa kertoo, mikä toimija toteuttaa toiminnon, on vuorovaikutuksessa toiminnon kanssa tai toiminnon laukaisijana. Makrotoiminto kuvaa puolestaan käyttötapauksiin liittyviä käyttää- ja laajentaa-yhteyksiä.

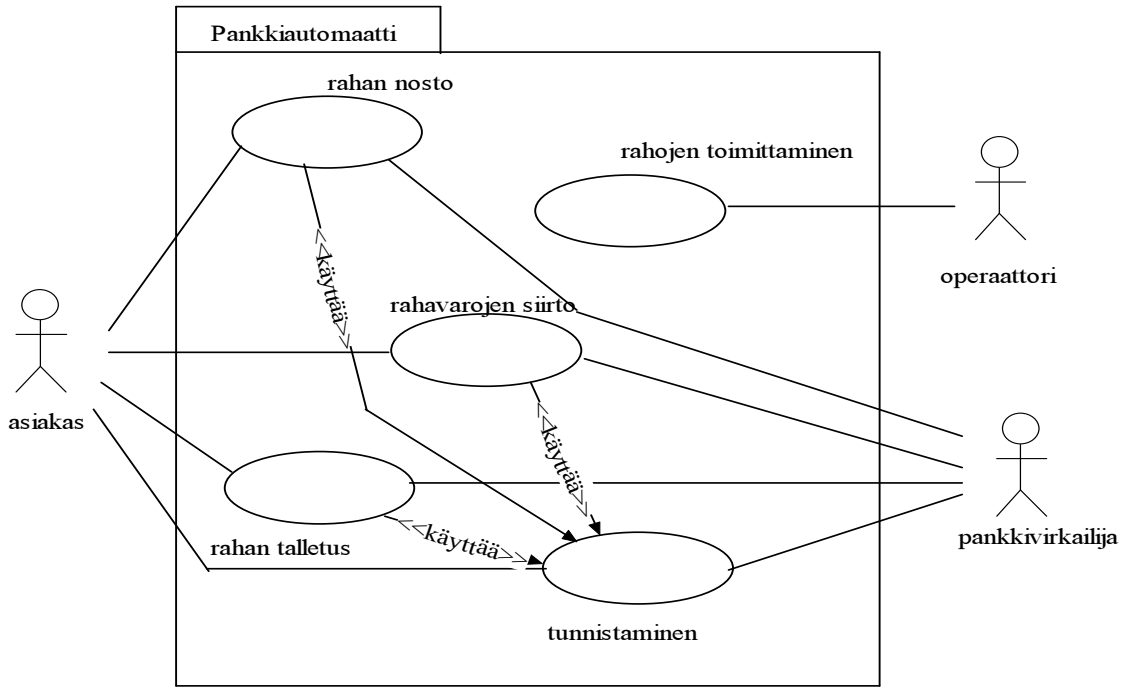
Jokainen toiminto aktiviteettikaavioissa merkitään joko kontekstuaaliseksi-, vuorovaikutus- tai makrotoiminnoksi. Taulukossa 4 ovat aktiviteettikaavion *stereotyyppien (stereo types)* graafiset symbolit. [KöS01]

Taulukko 4: Aktiviteettikaavion stereotyypit [KöS01]

Stereotyyppi	graafinen symboli
kontekstuaalinen toiminto	
vuorovaikutus	
toimija toiminnossa	
makrotoiminto	

9.4 Esimerkki pankkimaailmasta

Tarkastellaan pankkiautomaattia. Toimijoita on kolme: asiakas, pankkivirkailija ja operaattori. Käyttötapauksia on viisi: tunnistaminen, rahan nosto, rahan talletus, rahavarojen siirto ja rahojen toimittaminen. Kuvassa 8 on pankkiautomaatin käyttötapauskaavio. Asiakas voi siis nostaa rahaa, tallettaa rahaa ja tehdä tilisiirtoja. Operaattori laittaa rahat automaattiin ja pankkivirkailija varmistaa, että kaikki asiakkaan haluamat toiminnot tapahtuvat oikein sekä asiakkaan että pankin kannalta. [KöS01]



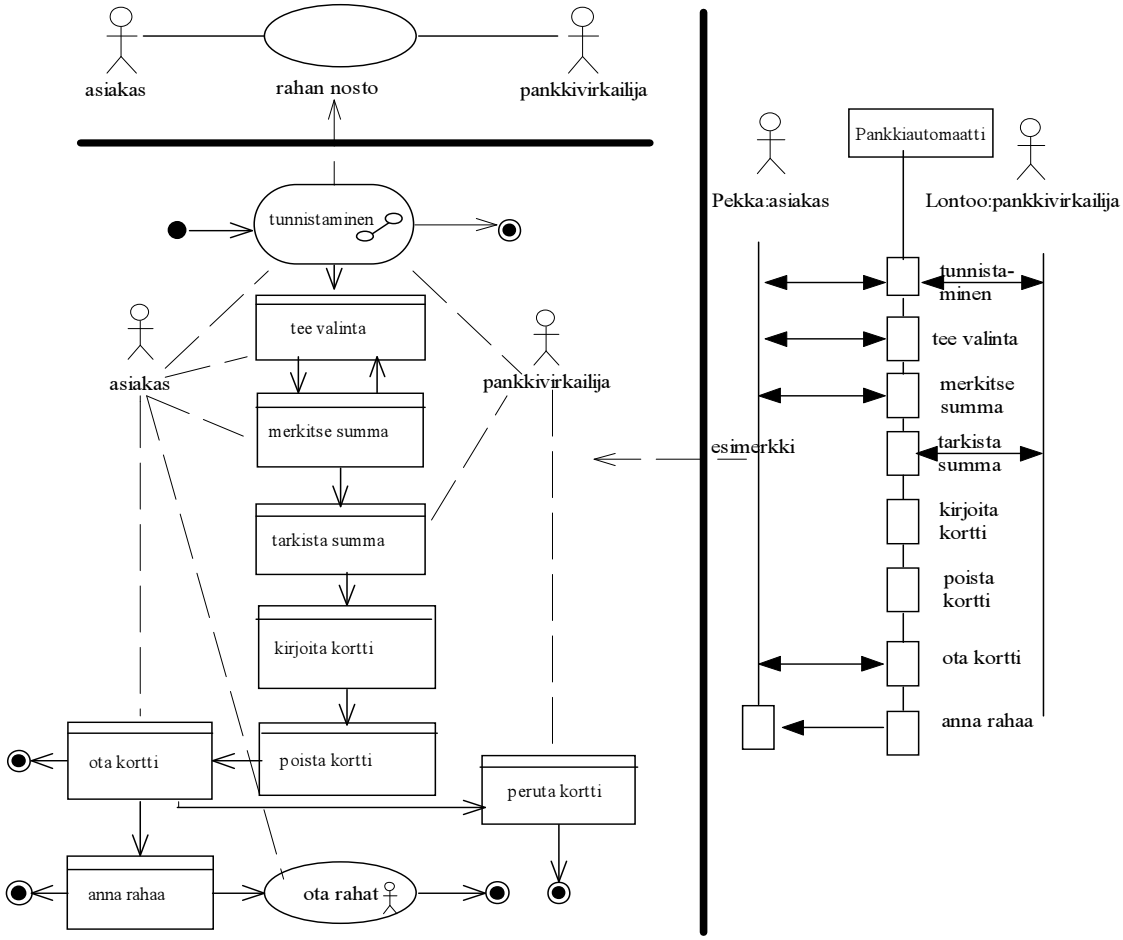
Kuva 8: Pankkiautomaatin käyttötapauskaavio [KöS01]

Kuvassa 9 on käyttötapauskaavion käyttötapauksen "tunnistaminen" sanallinen kuvaus. Siinä kerrotaan käyttötapauksen nimi, toimijat, lyhyt kuvaus, esi- ja jälkiehdot ja lopputila.

Käyttötapaus "Tunnistaminen pankkiautomaatissa"
Kuvaus: Kun kortti laitetaan automaattiin, pankkiautomaatin kortinlukija lukee koodin ja tarkistaa tunnusluvun ja kortin. (Tarkistamisen tekee pankkivirkailija tai koneessa oleva algoritmi.)
Jos tunnusluku annetaan kolmesti väärin, kone merkkaa kortin lukituksi ja poistaa kortin. Jos tunnusluku on oikein, kone näyttää valikon ja odottaa asiakkaan valintaa.
Toimijat: asiakas, pankkivirkailija
Esiehdot: kortinlukija valmiina, näytöllä logo
Jälkiehdot: kortti kortinlukijassa, kortinlukija lukittu, kortti ok, tunnusluku ok, asiakas tunnistettu, näytöllä valikko
Poikkeustoiminta: korttia ei voi lukea, kortti on peruutettu, kortinlukija valmis, näytöllä logo
tai tunnusluku kolmesti väärin, kortti lukittu, kortti peruttu, kortinlukija valmis, näytöllä logo
Lopeta tunnistaminen

Kuva 9: Käyttötapauksen "tunnistaminen" sanallinen kuvaus [KöS01]

Kuvan 10 oikealla puolella on peräkkäiskaaviona skenaario käyttötapauksesta "rahan nosto", kun nosto onnistuu. Skenaario yhdistetään kuvan 10 vasemmalla puolella olevan käyttötapauksen "rahan nosto" aktiviteettikaavion kanssa. Skenaario on siis yksi vaihtoehto erilaisista rahannostossa tapahtuvista mahdollisuuksista. Aktiviteettikaavioon on lisätty toimintojen stereotyypit. Esimerkiksi aktiviteettikaaviossa, toiminto tunnistaminen on makrotoiminto, koska käyttötapaus "rahan nosto" käyttää sitä. [KöS01]

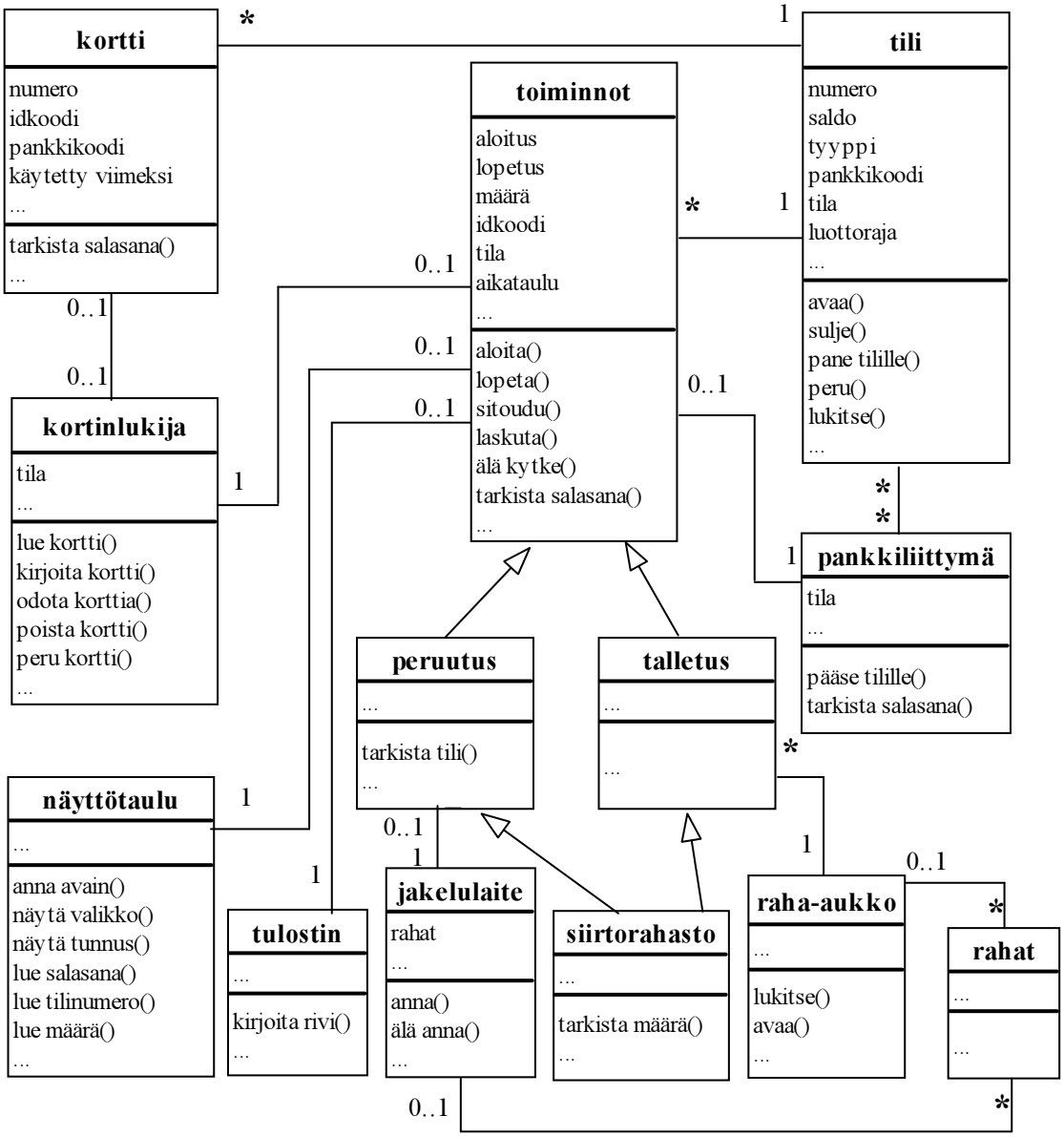


Kuva 10: Käyttötapauksen "rahan nosto" aktiviteettikaavio ja skenaario onnistuneesta rahan nostosta [KöS01]

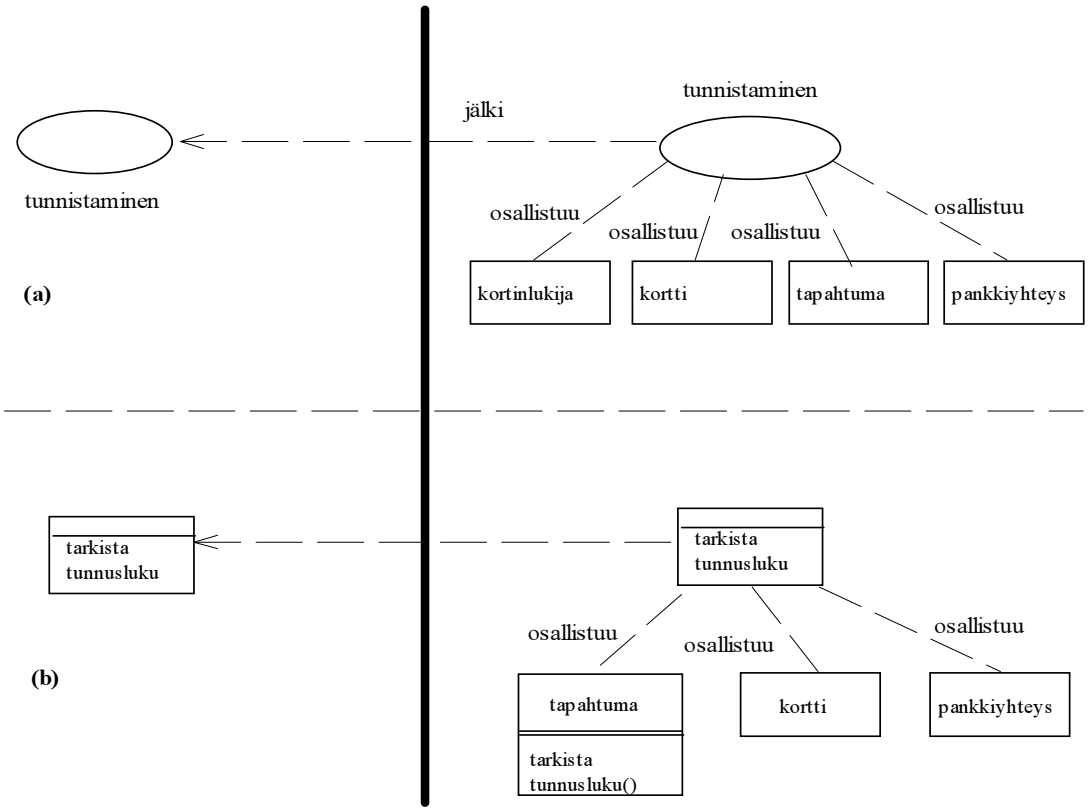
9.5 Luokkakattavuus

Yhdistetään seuraavaksi luokkakaavio ja lisätiedoin laajennettu aktiviteettikaavio toisiinsa. Jokaiselle käyttötapaukselle määritellään luokkia, joissa näkyy jo jonkin verran toteutustakin, esimerkiksi luontiaste, kutsuminen, päivittäminen ja poistaminen. Näitä luokkia kutsutaan käyttötapauksen *luokkakattavuudeksi (class scope)*. Luokkakattavuuteen kuuluvat luokat on saatu käyttötapauksen kuvauksesta ja sen esi- ja jälkiehdoista ja niiden pitää noudattaa luokkakaavion sääntöjä. Jokaiselle aktiviteettikaavion toiminnalle pystytään määrittelemään luokkakattavuus, esimerkiksi joukosta luokkia, jotka ovat toiminnan toteutuksessa mukana. Yhtenäistetään aktiviteettikaavioita ja luokkakaavioita niin, että jos aktiviteettikaaviossa on vuorovaikutustoimintaa, niin löytyy sellainen luokan *operaatio* o, joka saa aikaan saman vuorovaikutuksen luokkakaaviossa. Operaatio o kuuluu siis luokkaan C, joka on osana vuorovaikutuksen luokkakattavuutta. Luokkaa C kutsutaan *juuriluokaksi* ja operaatiota o vuorovaikutuksen *juurioperaatioksi*. [KöS01]

Tarkastellaan kuvassa 11 olevaa pankkiautomaatin luokkakaaviota. Siinä luokat tili, rahat, ja kortti edustavat liiketalouden kohteita. Luokat toiminnot, peruutus, talletus ja siirtorahasto kuvaavat pankkiautomaatin tapahtumia. Muut luokat edustavat laitekomponentteja. Kuvassa 12 on määritelty käyttötapaukselle ja aktiviteettikaavion vuorovaikutustoiminnalle vastaavia luokkia. Kuvan 12a- kohdassa havainnollistetaan käyttötapaus "tunnistamisen" luokkakattavuutta ja b-kohdassa vuorovaikutuksen "tarkista salasana" luokkakattavuutta. [KöS01]



Kuva 11: Pankkiautomaatin luokkakaavio [KöS01]



kuva 12: a) Käyttötapauksen "tunnistaminen" luokkakattavuus b) Vuorovaikutustoiminnan" tarkista tunnusluku" luokkakattavuus [KöS01]

9.6 Mallintaminen ja validointi

Käyttötapauskaavioita, luokkakaavioita ja niitä yhdistäviä luokkakattavuuksia käytetään vaatimusten mallintamiseen, validointiin ja verifiointiin. Taulukossa 5 kuvataan mitä kaavioita käytetään *pääpiirteiden* (ulkoinen näkyvä käyttäytyminen, sisäinen havaittava käyttäytyminen ja rakenne) ja *päätoimintojen* (kalastaminen, määritteleminen ja validointi) välisiä suhteita mallinnettaessa. Esimerkiksi ulkoisen käyttäytymisen määrittelyyn voidaan käyttää aktiviteetti- tai käyttötapauskaavioita. [KöS01]

Taulukko 5: Kohdealueen mallinnus: piirteet liitettyinä toimintoihin [KöS01]

piirre/toiminto	kalastaminen ja määrittely	validointi
ulkoinen näkyvä käyttäytyminen	käyttötapaukset, aktiviteettikaaviot	liiketalouden skenaariot
rakenne	kohdealueen luokat, attribuutit, assosiaatiot	kohderyhmät
sisäinen havaittava käyttäytyminen	luokka-alueet, operaatiot	työtehtävät

9.6.1 Käyttötapausten määrittely

Analysoijat ja alan erikoistuntijat tutkivat ja luonnostelevat liiketalouden skenaarioita ja tunnistavat toimijoita. Tästä informaatiosta kehitellään käyttötapauskuvauksia. Käyttötapauksia parannellaan määrittelemällä toimintoja tarkemmin. *Toiminnoista* sekä *liiketalouden tehtäväkuvauksista* johdetuista *peruskontrollivirtojen tiedoista* tehdään aktiviteettikaavio. Jos tehtävä on useiden muiden tehtävien osa, mallinnetaan se omaksi käyttötapaukseksi ja merkitään makrotoiminnaksi aktiviteettikaavioon.

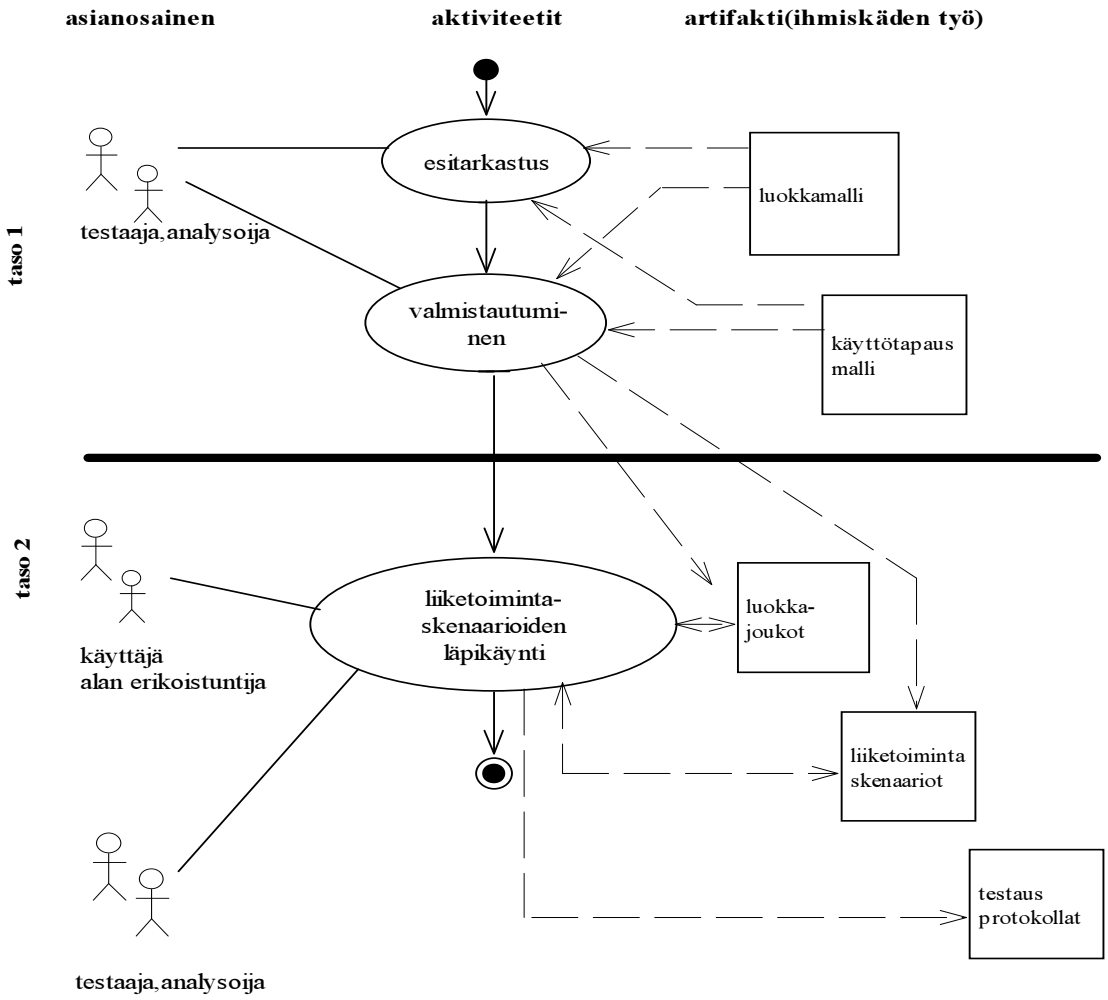
Käyttötapausten määrittelyssä voidaan käyttää neljän askeleen *iteratiivista* prosessia. Ensimmäisessä askeleessa kuvaillaan toimintojen *tavallisimmat menettelytavat* käyttämällä esimerkiksi peräkkäiskaavioita. Seuraavaksi tunnistetaan *vaihtoehtoisia toimintatapoja*. Kolmannessa askeleessa tehdään *aktiviteettikaavio aikaisempien askelten tuloksista*. Neljännessä askeleessa *tarkastetaan toimintatapojen solmut ja haarat*. Ensimmäiseen tai toisen askeleeseen palataan niin kauan, kun *katekriteeri (coverage criteria)* ei täyty ja yritetään löytää lisää erilaisia toimintatapoja.

Tietovaatimukset johdetaan usein alan avainkäsitteistä. Analysoijat tekevät ensin luokkamallin hahmotelman luokista ja niiden yhteyksistä. Jos toiminnon kuvauksessa mainitaan jokin luokka, niin luokka lisätään toiminnon luokkakatteeseen. Luokan vastuut, alustava joukko operaatioita, attribuutteja ja assosiaatioita määritellään. Jokaiselle vuorovaikutustoiminnolle valitaan sen luokkakatteesta juuriluokka ja juurioperaatiot. [KöS01]

9.6.2 Validointi

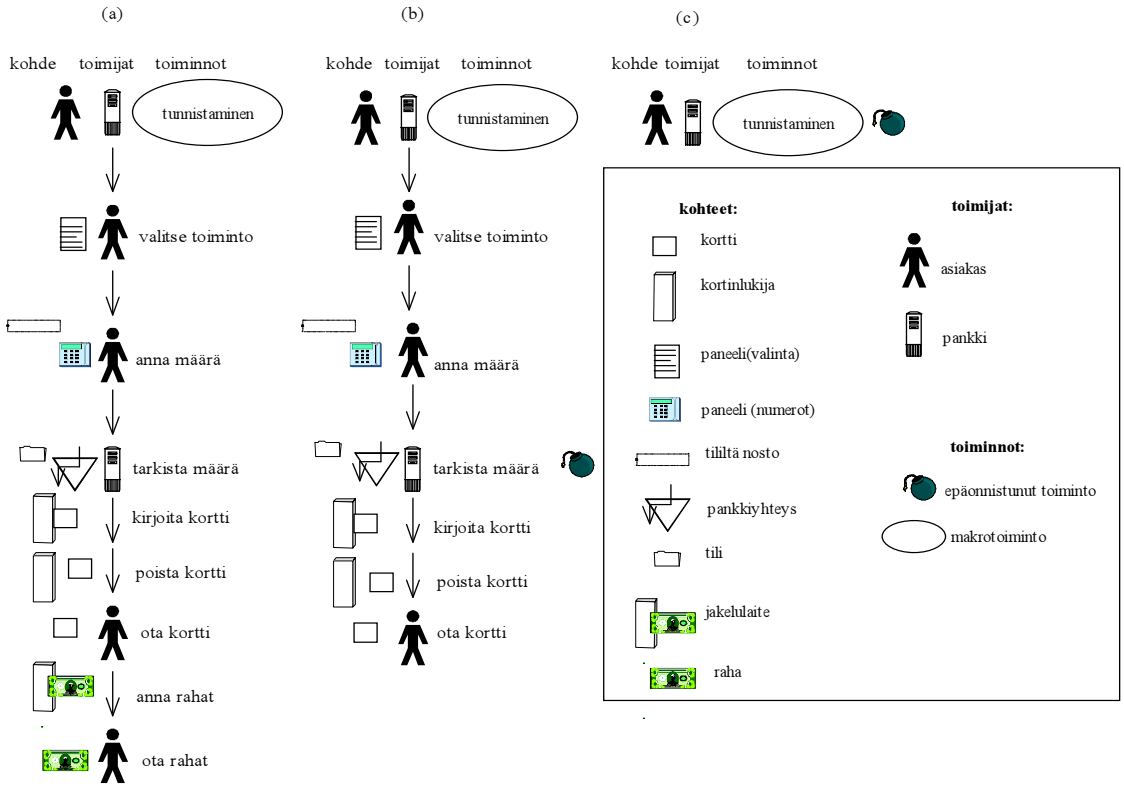
Vaatimusmäärittelyjen validointiin voidaan harvoin käyttää lyhyitä ja ytimekkäitä lähdedokumentteja. Usein joudutaan tyytymään käyttäjien epämääräisiin mielikuviin. Siksi analysoijien ja testaajien lisäksi validointiprosessiin täytyy osallistua myös muita alan erikoisasiantuntijoita ja käyttäjiä. Näiden kaikkien asianosaisten täytyy toimia yhdessä, jotta lopullisen määrittelyn ymmärtävät oikein kaikki osapuolet. [KöS01]

Sopivia validointitekniikoita ovat esimerkiksi tarkastukset, läpikäynnit ja prototyypin tekeminen. Tarkastusten ja läpikäyntien on havaittu olevan tehokkaita menetelmiä dokumenttipohjaisessa validoinnissa, joten ne on valittu kaksitasoiseen tarkastusprosessiin. Kuvassa 13 on esitelty tällaista prosessia.



Kuva 13: Kaksitasoinen validointiprosessi [KöS01]

Ensimmäisellä tasolla analysoijat ja testaajat esitarkastavat käyttötapauskaavion ja luokkakaavion huomatakseen UML-syntaksin ja käytetyn menetelmän vastaiset virheet. Koska liiketoimintaskenaarioita voidaan käyttää validoinnissa, tehdään jokaiselle käyttötapaukselle liiketoimintaskenaarioita esiehtojen ja aktiviteettikaavion pohjalta. Jokaiselle skenaariolle kootaan aloitustoiminnon luokkakatteesta luokkajoukkoja. Analysoija voi myös tehdä käyttäjäkeskeisiä skenaarioita ja luokkajoukkoja parantaakseen ymmärrettävyyttä. Lisäksi tällä tasolla voidaan laatia kuvan 14 kaltaisia kohdealuekohtaisia reaalimaailman esimerkkejä. Kuvassa on kolme erilaista skenaariota rahojen nostamisesta. Ensimmäisessä tapauksessa rahan nosto onnistuu, keskimmäisessä rahan nosto epäonnistuu, koska tilillä ei ole tarpeeksi rahaa. Kolmannessa tapauksessa jo kortin tunnistaminen epäonnistuu. [KöS01]



Kuva 14: Käyttötapaus "rahan nosto" kolme käyttäjäkeskeistä skenaariota [KöS01]

Validointiprosessin toisella tasolla liiketoiminta skenaariota käydään läpi yhdessä kaikkien asianosaisten kanssa. Ensin alan erikoistuntijat ja käyttäjät tarkistavat alkutilanteen luokkajoukkoja ja skenaarioiden esiehtojen tekstikuvauksia varmistuakseen, että kaikki tarvittava tieto on saatavilla. Läpikäyntien aikana alan erikoistuntijat ja käyttäjät keskit-

tyvät toimintojen tekstikuvauksiin, tarkistavat niiden oikeellisuuden ja niihin liittyvät luokkajoukot. Kun luokkajoukkoja validoidaan, niin samalla validoidaan osittain luokkakaaviota. [KöS01]

Liiketoimintaskenaarioiden läpikäynnit paljastavat huonosti muodostetut luokkajoukot, väärät tai puuttuvat toiminnot, väärät tai puuttuvat muutokset sekä varmistavat toimintakuvausten asianmukaiset sisällöt. Analysoijat ja testaajat dokumentoivat löydetyt virheet korjatakseen ja jopa uudelleen validoidakseen määrittelyt. Jos käyttäjä tai asiantuntija löytää puuttuvan skenaarion, niin se lisätään käyttötapauskaavioon ja validoidaan. Kuvassa 15 on esimerkkinä eheyteen ja oikeellisuuteen liittyviä kysymyksiä validoinnin tarkistuslistasta. Tällaiset listat ovat kohdealueriippumattomia ja niitä pitäisi räätälöidä jokaiseen organisaatioon ja projektiin. [KöS01]

Eheys
- Löytyvätkö kaikki liiketoimintaskenaariot käyttötapauksista?
- Löytyykö aktiviteettikaaviosta puutteita, esimerkiksi puuttuvia toimintoja ja muutoksia?
- Sisältyvätkö kaikki tarvittavat luokat toiminnon luokkakatteeseen?

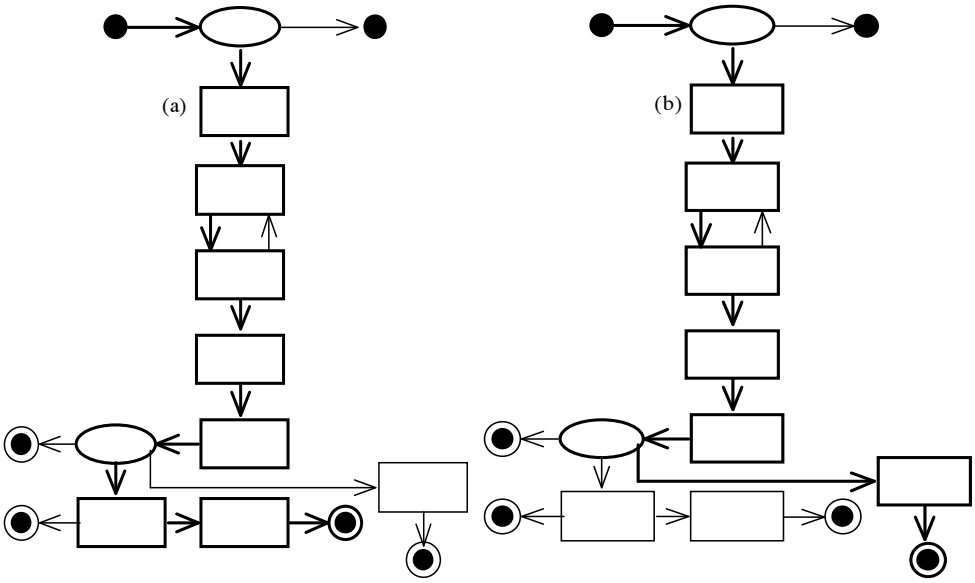
Oikeellisuus
- Onko kaikilla käyttötapausten esi- ja jälkiehdoilla laillinen liiketoiminta- asema?
- Noudattavatko kaikki käyttötapausten määrittelyt liiketoiminnan sääntöjä?
- Heijastaako skenaarion odotettu lopputulos jälkiehtoja ja lopputoimintoja?

Kuva 15: Eheyteen ja oikeellisuuteen liittyviä kysymyksiä validoinnin tarkistuslistasta [KöS01]

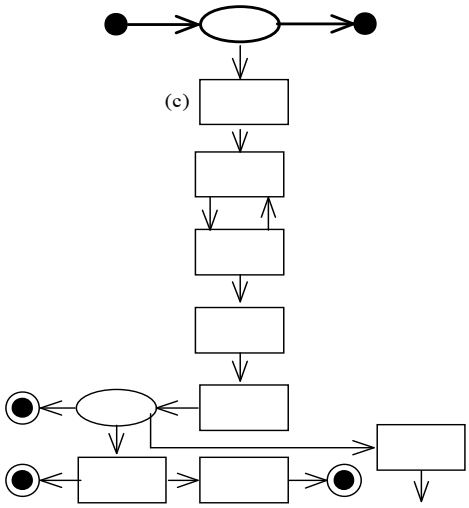
Validoinnista, verifioinnista ja ohjelmien testauksesta löytyy paljon samoja ongelmia, esimerkiksi, milloin voi lopettaa. Monia testaukseen liittyviä kattavuuskriteereitä on ehdotettu käytettäväksi apuna myös validoinnissa ja verifioinnissa oikean ajankohdan määräämiseksi. Jotta vaatimusmäärittelyjen validoinnissa ja verifioinnissa voitaisiin käyttää samanlaisia määrällisiä mittauksia kuin testaustauksessa, toiminnot pitäisi suunnitella ja kontrolloida paljon tarkemmin. Koska ohjelmien kontrollivirtakaaviot ja aktiviteettikaaviot ovat rakenteeltaan samankaltaisia, ohjelmistojen testauksen kattavuusperusteet voitaisiin ulottaa käyttötapauskaavioiden validointiin. Esimerkiksi aktiviteettikaavioissa olisi suositeltavaa, jos saavutettaisiin 100 % *solmukattavuu*s eli jokai-

sessa toiminnossa käytäisiin vähintään kerran läpikäyntien yhteydessä. Kriittisissä käyttötapauksissa pitäisi päästä 100 % *kaarikattavuuteen* eli aktiviteettikaavion kaikki kaaret kuljettaisiin läpi vähintään kerran. Jokaiseen makrotoimintaan olisi kiinnitettävä erityistä huomiota, koska niistä alkaa useampia mahdollisia polkuja. [KöS01]

Käyttäjien ja alan asiantuntijoiden pitäisi validoida kuvan 14 käyttötapaus "rahan noston" kolme eri käyttäjäkeskeistä skenaariota. Kuvassa 16 ja 17 on käyttötapauksen "rahan nosto" aktiviteettikaaviossa paksunnettu ne solmut ja kaaret, joissa käydään. Kuvan 16 a-kohdan skenaarion "onnistunut rahannosto" läpikäynti kattaa luokat kortti, kortinlukija, paneeli, pankkiliittymä, tili, nosto, jakelulaite ja käteinen sekä niiden yhtymät. Skenaario saavuttaa 69 % solmukattavuuden ja 63 % kaarikattavuuden, koska siinä käydään 11 solmussa 16 mahdollisesta solmusta ja 10 kaaressa 16 mahdollisesta kaaresta. Skenaariossa "epäonnistunut rahan nosto-tilin ylitys" kuvan 16 b-kohdassa päästään 56 % solmukattavuuteen ja 50 % kaarikattavuuteen. Skenaario "epäonnistunut rahannosto-tunnistaminen epäonnistunut" kuvassa 17 saavuttaa vain 19 % solmukattavuuden, koska siinä päästään vain kolmeen solmuun 16 mahdollisesta solmusta. Kaarikattavuudessa päästään 12,5 % kattavuuteen, koska käytetään vain kahta kaarta 16 kaaresta. Nämä kolme skenaariota saavuttavat yhdessä 87,5 % solmukattavuuden ja 81,3 % kaarikattavuuden. [KöS01]



Kuva 16: Aktiviteettikaavion "rahan nosto" kaksi erilaista skenaariota [KöS01]



Kuva 17: Aktiviteettikaavion "rahan nosto" kolmas skenaario [KöS01]

9.7 Analyysi

Kohdealueen mallinnus ja validointi suoritetaan käyttäjien avulla. Vaatimusmäärittely täytyy kuvata niin, että käyttäjät voivat ymmärtää sen. Rakentajien kannalta katsottuna nämä määrittelyt eivät ole tarpeeksi yksityiskohtaisia. Analyysin aikana menetelmä tukee käyttötapauskaavion, luokkakaavion ja verifiointitoimintojen parantamista. Taulukossa 6 kerrotaan pääpiirteiden ja päätoimintojen välisistä suhteista. [KöS01]

Taulukko 6: Analyysi: piirteet liitettynä toimintoihin [KöS01]

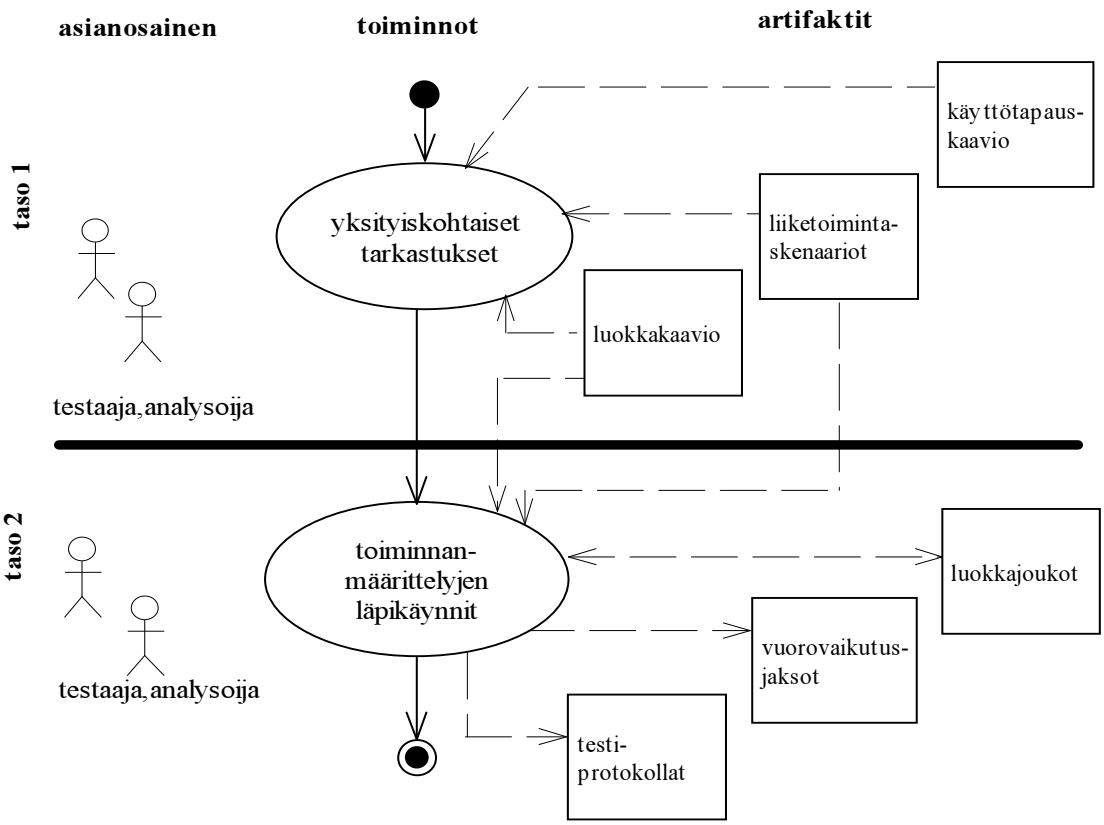
Piirre/toiminto	jalostus	validointi
Ulkoinen havaittava käyttäytyminen	käyttötapaukset, aktiviteettikaviot	verifiointiskenaariot
Rakenne	alueen luokat, attribuutit yhteydet	luokkajoukot
Sisäinen havaittava käyttäytyminen	luokkakattavuudet, operaatiot	jaksot

9.7.1 Jalostaminen

Kohdealueen mallintamisesta johdettu luokkakaavio sisältää kohdealueen kaikki tärkeimmät luokat sekä niiden vastuut, juurioperaatiot, alkuattribuutit ja yhteydet muihin luokkiin. Kasvavassa ja iteratiivisessa prosessissa, analysoija tutkii aktiviteettikaaviota yksityiskohtaisemmin saadakseen kaikki mallinnuselementit lisättyä vastaaviin käyttötapauksiin. Käyttötapausten vuorovaikutuksien kartoittamiseksi, tutkitaan luokkien vastuut luokkakatteessa ja niiden juurioperaatiot. Kartoittamalla luokan vastuut vastaaviin operaatioihin ja määrittelemällä juurioperaatioita tarkemmiksi (esimerkiksi peräkkäiskaavioilla) löydetään usein lisää operaatioita, puuttuvia attribuutteja, yhteyksiä, riippuvuuksia ja olioiden tiloja. Luokkia käytetään myös kontrollin, entiteetti- ja rajaluokkien määrittelyyn. Tällaisten määrittelyjen avulla pystytään erottamaan yleinen käyttäytyminen yleisiin luokkiin ja löytämään ryhmittymiä, jotka jakavat monimutkaiset luokat sopiviin kokonaisuuksiin ja osiin. Ryhmittymiä uudelleen tutkimalla voidaan mallia muokata sopivammaksi. [KöS01]

9.7.2 Verifiointi

Luokkakaavioita voidaan verifioida käyttötapauksia vastaan, kun jalostusprosessi on saavuttanut riittävän vakaan tilan. Vaatimusmäärittelyn validointi ja verifiointi sisältävät samankaltaisuuksia, mutta jälkimmäinen keskittyy olennaisesti yksityiskohtaisemmalle ja muodollisemmalle tasolle. Käyttäjät ja alan asiantuntijat eivät yleensä kykene vaikuttamaan verifiointitoimintoihin niin kuin analysoijat ja testaajat ja siksi verifiointiprosessiin on otettu vain jälkimmäiset mukaan. Verifioinnissa käytetään kaksitasoista prosessia vastaavasti kuin kuvan 13 validoinnissakin. Kuvassa 18 kuvataan kaksitasoista verifiointiprosessia. [KöS01]



Kuva 18: Kaksitasoinen verifiointiprosessi [KöS01]

Ensimmäisellä tasolla verifiointi alkaa käyttötapauskaavion formaalien osien yksityiskohtaisella tarkastelulla. Sen jälkeen tarkastetaan luokkakaavio, jotta löydettäisiin epätäydellisiä, epäjohdonmukaisia ja moniselitteisiä määrittelyjä tai puuttuvia asioita. Viimeiseksi tutkitaan jokainen aktiviteettikaavion vuorovaikutustoiminnon määrittely, jotta saataisiin selville, noudattaako se luokkakattavuutta ja juurioperaatioiden määrittelyjä. Kun yksityiskohtaiset tarkastukset on tehty loppuun, on käyttötapauskaavio verifioitu. Luokkakaavio on nyt johdonmukainen ja yksiselitteinen ja enää on tarkastettava, että se on oikea ja täydellinen käyttötapauskaavion suhteen. [KöS01]

Toisella verifiointitasolla varmistetaan, että luokkakaaviosta löytyy jokaiselle aktiviteettikaavion vuorovaikutukselle tarvittavat luokat. Analysoijat ja testaajat käyvät läpi skenaariot ja tutkivat vuorovaikutusten laukaisemien operaatioiden toteutusmahdollisuuksia luokkakaaviossa. Selvästikin sellainen operaatio, joka herätetään ensimmäiseksi jossakin toteutuksessa, on juurioperaatio. Se toimii eräänlaisena *pääsypisteenä* luokkakaavion muihin operaatioihin. (Tässä kohdassa olisi hyvä miettiä eheyteen liittyviä asi-

oita. Olisiko yhteisen rajapinnan käyttö järkevämpää, kuin se, että yksittäisiin luokkiin päästäisiin mistä tahansa?) Koska jokaisen vuorovaikutuksen alkuperäinen ja lopullinen luokkakokoelma tiedetään, voidaan luokkakokoelmia käyttää uudelleen, jos solmukattavuus on saavutettu kaikissa validoiduissa aktiviteettikaavioissa. Luokkakaavion vuorovaikutustoiminnasta saatujen toteutuspolkujen simulointia sanotaan *vuorovaikutussarjaksi (interaction sequence)*. Vuorovaikutussarjoja voidaan visualisoida peräkkäiskaavioilla. Vuorovaikutussarja riippuu asiayhteydestä, laukaisevasta vuorovaikutuksesta ja vastaavasta luokkajoukosta. Koska luokkajoukko on jo validoitu, se tyydyttää juurioperaatioiden esiehdot, niin että vuorovaikutussarja voi käynnistyä. [KöS01]

Käytäessä läpi vuorovaikutussarjaa, luokkajoukko muuttuu. Luokkakaaviosta löydetään virhe, jos nykyinen luokkajoukko ei kohtaa seuraavaksi toteutettavan operaation esiehtoja. Vuorovaikutussarja päättyy onnistuneesti, jos sellaisia virheitä ei tapahdu ja lopullinen luokkajoukko toteuttaa vuorovaikutuksen jälkiehdot. Jos luokkakaaviosta löytyy osia, jotka eivät ole osallistuneet johonkin vuorovaikutussarjaan on luokkakaavio ylimääritelty tai käyttötapauskaavio on yhä vaillinainen. Kuvassa 19 on esimerkkinä eheyteen ja oikellisuuteen liittyviä kysymyksiä verifioinnin tarkistuslistasta. [KöS01]

Eheys
- Onko jokainen luokka vähintään yhdessä luokkakatteessa?
- Onko jokainen vuorovaikutus yhdistetty juurioperaation?
- Ovatko kaikki juurioperaatioon liittyvät luokat myös vastaavien vuorovaikutusten luokkakatteessa?

Oikeellisuus
- Tarkoittavatko vuorovaikutuksen esiehdot juurioperaatioiden esiehtoja?
- Tarkoittavatko juurioperaatioiden jälkiehdot vuorovaikutuksen esiehtoja?
- Onko makrotoiminnon luokkakattavuus vastaavan aktiviteettikaavion luokkakattavuuden alijoukko?

Kuva 19: Eheyteen ja oikeellisuuteen liittyviä kysymyksiä verifioinnin tarkistuslistasta [KöS01]

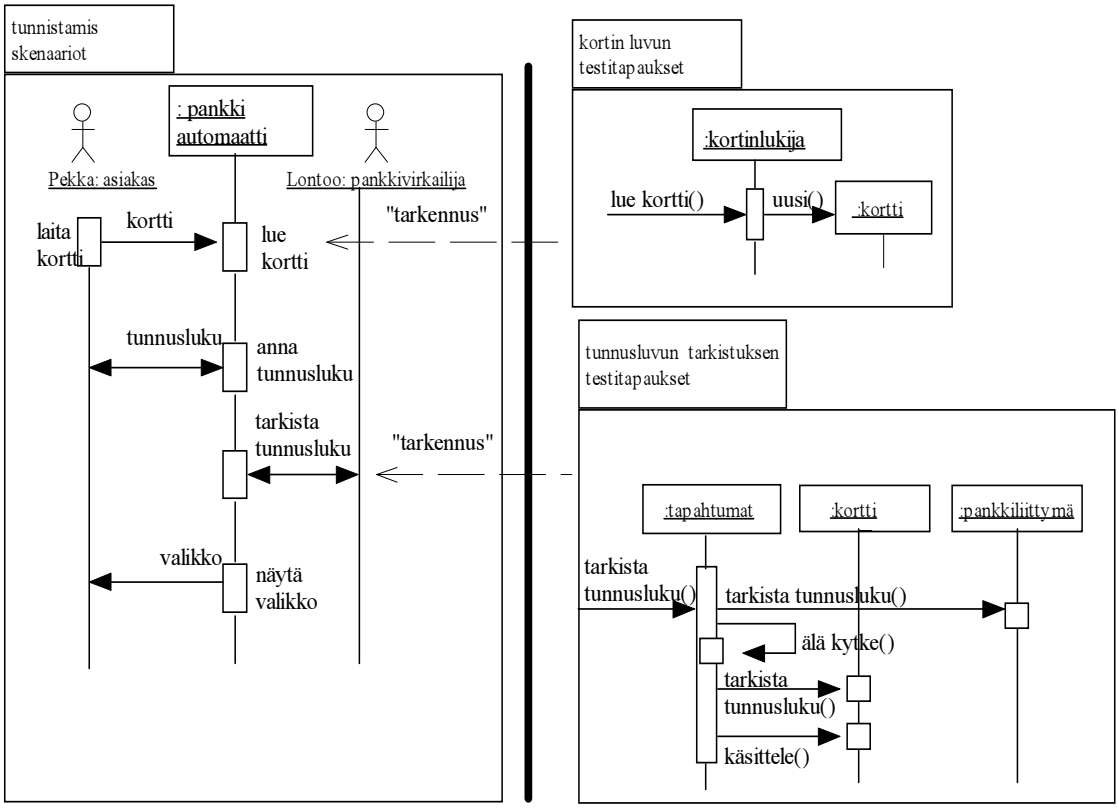
Liiketoimintaskenaariota, joita käytetään uudelleen verifioinnin aikana ja joihin on lisätty vuorovaikutussarjoja, kutsutaan *verifiointiskenaarioiksi*. Verifiointiskenaarioilla

testataan luokkakaavioiden ja aktiviteettikaavioiden vastaavuuksia. Myös systeemi- ja hyväksymistestauksen testiskenaariot voidaan johtaa verifiointiskenaarioista. [KöS01]

Validoinnissa käytettyjä kattavuuskriteereitä voidaan käyttää myös verifioinnin yhteydessä. Verifiointia voidaan kontrolloida esimerkiksi *luokka- tai operaatiokattavuudella* eli jokaisen luokan ja jokaisen operaation on liityttävä vuorovaikutussarjaan. Jos käyttötapaus edustaa kriittisiä tehtäviä, pitää *ehtokattavuudessa* saavuttaa kaikki ehdot. Ehtokattavuus tarkoittaa sitä, että jokainen toiminnon esi- ja jälkiehto pitää arvioida kaikissa mahdollisissa yhdistelmissä. Se voidaan tehdä vaihtelemalla syötteitä ja olioiden tilaa vastaavissa skenaarioissa. [KöS01]

9.7.3 Esimerkki verifiointiskenaariosta

Tarkastellaan kuvan 20 verifiointiskenaariota "onnistunut offline – tilassa tunnistaminen". Offline - tila tarkoittaa sitä, että pankkivirkailijaan ei saada yhteyttä. Kuvan 20 oikealla puolella on kaksi vuorovaikutuskaaviota. Alemmassa, jonka laukaisee toiminto "tarkista tunnusluku", pankkiautomaatti muuttaa tilansa offline-tilaan, aktivoi operaation "tarkista tunnusluku", joka on määritelty luokassa kortti ja ohjailee korttiilmentymän attribuutteja. Ylemmässä vuorovaikutuskaaviossa toiminto "lue kortti" toimii laukaisijana. Käyttötapausta "tunnistaminen" käyttää pankkiautomaatin kaikki muut käyttötapaukset. Ehtokattavuudessa kaikki sen jälkiehtojen kolme lausetta on arvioitava todeksi jollakin sopivalla skenaariolla. Jälkiehdot löytyvät kuvasta 9. Koska esiehto on *atominen (atomic)*, lisäarvioita ei tarvita. [KöS01]



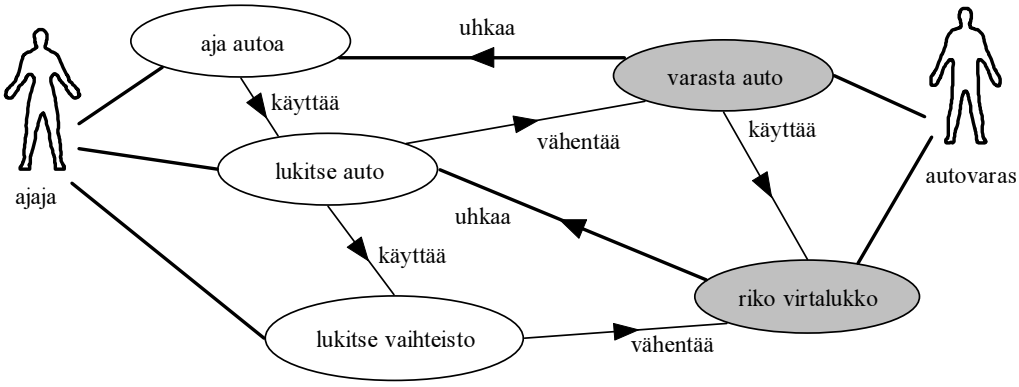
Kuva 20: Verifiointiskenaario "onnistunut offline-tilassa tunnistaminen" ja kaksi vuorovaikutuskaaviota [KöS01]

10 VÄÄRINKÄYTTÖTAPAUKSET

Käyttö- ja väärinkäyttötapauksia voidaan käyttää vaatimusten kalastamisessa, analyysissä, suunnittelun vaihdoksissa ja verifioinnissa. *Väärinkäyttötapaus* on käyttötapaus, jossa toimija haluaa käyttää systeemiä tahallaan väärin. Nykyään systeemisuunnittelijat joutuvat ottamaan huomioon myös väärinkäyttötapaukset analysoidessaan eri skenaarioita ja tehdessään dokumentteja. Varsinkin *turvallisuusvaatimuksia* pohdittaessa, väärinkäyttömahdollisuudet tulisi havaita mahdollisimman pian. [Ale03]

Osa väärinkäytöistä sattuu erityistilanteissa ja osa taas voi olla jatkuvaa toimintaa. Esimerkiksi auto varastetaan todennäköisimmin silloin, kun se on pysäköity ja vartioimaton, kun taas Web-palvelin saattaa kärsiä hyökkäyksistä milloin tahansa. Käyttötapauksia ja väärinkäyttötapauksia voidaan kehittää rekursiivisesti, menemällä systeemistä alisysteemeihin ja niin alhaiselle tasolle kuin on tarpeellista. Tällainen lähestymistapa helpotta vaatimusten etsimistä, ymmärtämistä, validointia ja verifiointia.

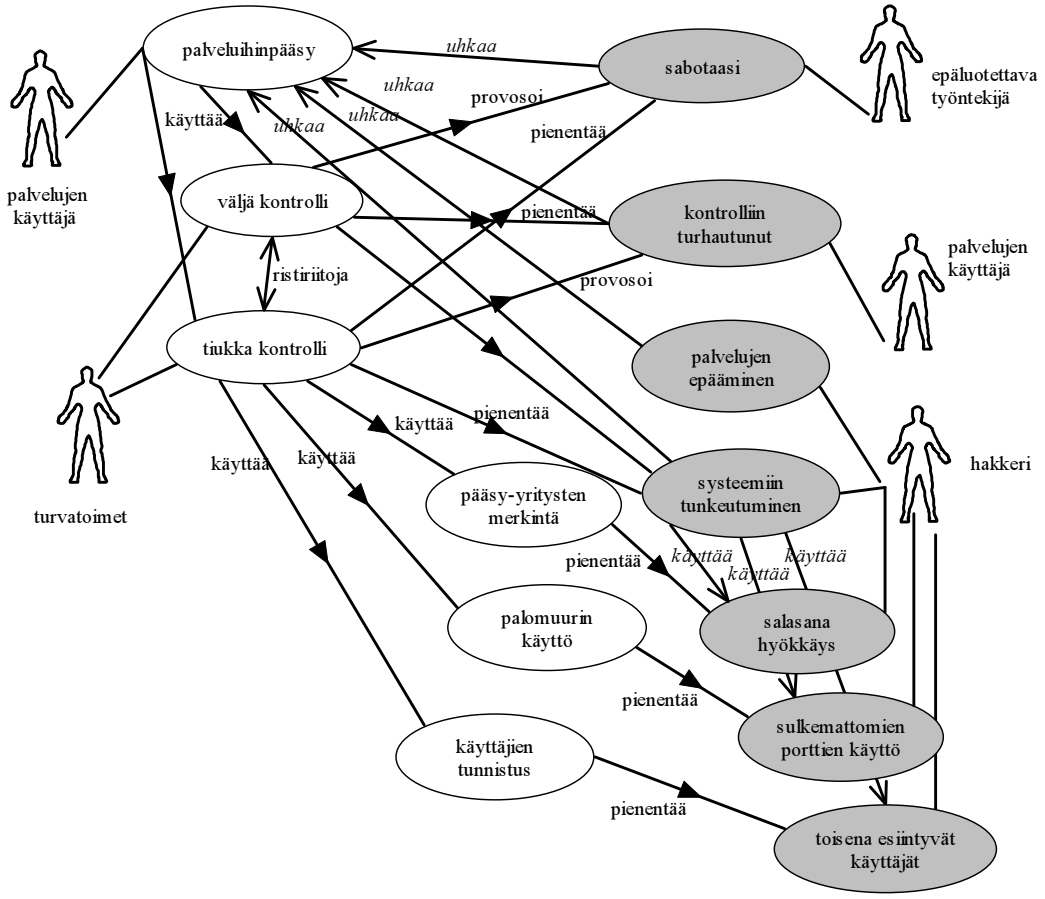
Väärinkäyttötapausten ja käyttötapausten suunnittelua voidaan verrata *pelin pelaamiseen.* Pelaajan paras strategia on pystyä ajattelemaan etukäteen vastapuolen liikkeet ja estää ne. Kuvassa 21 käyttötapaukset on piirretty kuvan vasemmalle ja väärinkäyttötapaukset kuvan oikealle puolelle. Väärinkäyttötapauspelaaja on autovaras ja käyttötapauspelaaja on auton laillinen omistaja. Omistajan liikkumisvapautta rajoittaa riski auton varastamisesta. Omistajan täytyy pystyä lukitsemaan auto pienentääkseen uhkaa. Näin saadaan uusi vaatimus: auto on pystyttävä lukitsemaan. Seuraavalla tasolla mietitään varkaan vastetta lukitsemiselle. Jos hän särkee oven ja käsittelee virtalukkoa, saadaan taas uusi vaatimus, kuten esimerkiksi vaihteiston lukitseminen. Tällä tavoin ”pelaamalla” jatketaan eteenpäin. Tarkastelemalla kuvaa 21 huomataan, että uhkaaminen ja uhkan pienentäminen muodostavat *siksak-kuvion* pelaajan ja vastapelaajan välille [Ale03].



Kuva 21: Auton varmuusvaatimusten käyttö- ja väärinkäyttötapauskaavio [Ale03]

Vaatimuksia voidaan kehittää ja muuttaa, kun löydetään uhkien vähentämiskeinoja. Kaikkien uhkien poistaminen on toiveajattelua ja sellaista ei voida edes vaatia. Esimerkiksi jotkut autoilijat jättävät auton käymään, kun poistuvat lyhyeksi aikaa autostaan. Nyt jää vain arvioitavaksi, voidaanko auto suojata tällaisissa tilanteissa. Uhkien kartoittaminen on hyödyllistä, niin kauan kuin se on toteutettavissa järkevin kustannuksin. [Ale03]

Systeemisuunnittelun yksi tärkeä osatekijä on käyttäjien ristiriitaisten vaatimusten tyydyttäminen. Tilannetta monimutkaistaa se, että jokainen suunnitteluvalinta avaa uusia mahdollisuuksia sekä oikealle että väärälle käytölle. Suunnittelijoiden täytyykin siksi vaihtaa vaihtoehtoja toisiin. Esimerkiksi Web-portaalin käyttäjien täytyy päästä tuotettuihin palveluihin. Pääsyä voivat uhata monet turvallisuushyökkäykset. Niitä voivat tehdä hakkerit tai sabotoijat. Itse turvallisuuskin voi uhata systeeminkäyttöä, jos turvallisuusvaatimukset ovat niin tiukasti asetettuja, että ne turhauttavat lailliset käyttäjät ja saavat heidät etsimään vaihtoehtoisia palveluja. Toisaalta löysä turvallisuuskontrolli lisää väärinkäyttöä. Kuva 22 havainnollistaa tällaisia ongelmia, sillä siihen on lisätty tapausten välille myös ristiriitaisia ja provosoivia suhteita. Kuvassa vasemmalla puolella ovat systeemiä oikein käyttävät ja oikealla väärinkäyttäjät. Kuvaan on merkitty, miten joitakin väärinkäyttötapauksiin liittyviä uhkia voitaisiin pienentää ja mitkä käyttötapaukset käyttävät toisia käyttötapauksia. Väärinkäyttötapaukset on väritetty harmaiksi. [Ale03]



kuva 22: Web-portaalin turvallisuuteen vaikuttavat käyttö- ja väärinkäyttötapaukset [Ale03]

Väärinkäyttötapauksia etsittäessä voidaan järjestää esimerkiksi pieni *työpaja*, jossa asianosaiset tunnistavat negatiivisia toimijoita, jotka saattavat uhata rakennettavaa systeemiä. Jokaiselle negatiiviselle toimijalle listataan mahdollisia väärinkäyttötapauksia. Jos systeemiin suunniteltuja laillisia käyttötapauksia on jo valmiina, etsitään niistä kohtia, missä niitä voitaisiin uhata. Tästä saattaa seurata uusien väärinkäyttötapausten löytäminen. Sen jälkeen mietitään, miten väärinkäyttötapauksia voitaisiin vähentää tai poistaa kokonaan. Käyttötapauksiin lisätään tarvittaessa uusia toimintoja. *Ristiriita-analyysiä* kannattaa käyttää avuksi. Eri suunnitelmien hyötyjä ja kustannuksia harkitsemalla, valitaan paras mahdollinen ratkaisu. [Ale03]

Samalla tavalla voisi analysoida mahdollisia *vahingossa* tapahtuvia käyttövirheitä, koska suurin osa virheistä ei ole tahallisia. Jos halutaan parantaa esimerkiksi käytettävyyttä, niin tätä menetelmää kannattaisi kokeilla.

11 VANHASTA SYSTEEMISTÄ UUTEEN SYSTEEMIIN

Vaatimusmäärittelyssä kuvataan uuden systeemin vaatimuksia käsitteellisellä tasolla. Skenaariot edustavat puolestaan *konkreettisia* esimerkkejä nykyisen tai uuden systeemin käytöstä [HaH98]. Vaatimusten määrittelyn alkuvaiheissa skenaarioita käytetään tukemaan uuden systeemin korkean tason vaatimusmäärittelyä. Usein vaatimukset on saatu nykyistä systeemiä tarkkailemalla, lukemalla dokumentteja ja analysoimalla nykyisen systeemin käyttöskenaarioita. Uuden systeemin täytyy toteuttaa usein monia olemassa olevan systeemin toiminnallisia ja ei-toiminnallisia vaatimuksia. [HaP98], [AnC01]

Vaatimusmäärittelyjen laatuun vaikuttaa asianosaisilta saatu tieto. Asianosaisten täytyy siis olla vaatimusmäärittelyprosessissa *aktiivisesti* mukana. Kun nykyisen systeemin käyttäjiä tarkkaillaan, saadaan systeemin käytöstä materiaalia. Materiaali lajitellaan ja lajiteltua materiaalia hyödynnetään uusien vaatimusten kalastamisessa ja validoinnissa. [HaP98]

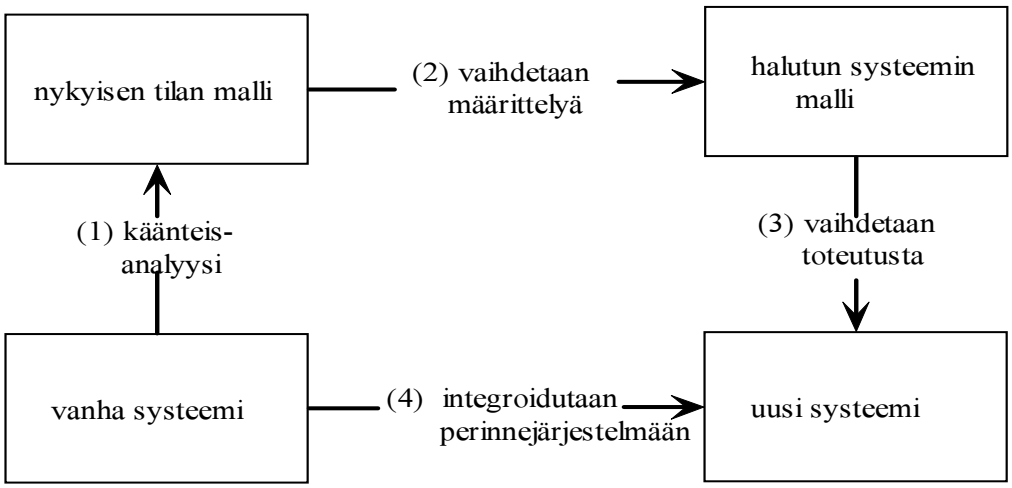
Perinteisesti vaatimusmäärittelyprosessissa on pyritty tekemään uudesta systeemistä johdonmukainen ja ristiriidaton vaatimusmäärittely käsitteellisellä tasolla. On tärkeää tarkastella olemassa olevan systeemin historiaa ja toiminnallisuutta, kun määritellään uuden systeemin vaatimuksia. Tähän on olemassa kaksi pääsyytä: uuden systeemin on toteutettava monia vanhan systeemin toimintoja ja samoja virheitä ei saisi tehdä kahta kertaa. [HaP98]

11.1 Nykyisen tilan ja halutun tilan mallit

Nykyisen tilan malli (*current-state model)* määrittää olemassa olevan systeemin toiminnallisuutta ja historiaa ja *halutun tilan malli* (*desired-state model)* määrittelee tulevan systeemin vaatimuksia. Kuvassa 23 uuden systeemin toteuttamista vanhasta systeemistä liikkeelle lähtien kuvataan neljällä askeleella [HaP98]:

1) *Käänteisanalyysi (reverse analysis)*: Todellisuutta abstrahoimalla määritellään nykyisen tilan malli, koska useissa tapauksissa todellisen systeemin käsitteellistä mallia ei ole enää olemassa tai se ei ole ajan tasalla.

2) *Määrittelyn muuttaminen (change definition)*: Lisäämällä määrittelyn muutokset nykyisen tilan malliin, määritellään uuden systeemin malli.
3) *Toteutuksen vaihtaminen (change implementation)*: Uusi systeemi suunnitellaan, toteutetaan, testataan ja asennetaan pohjautuen halutun tilan malliin.
4) *Perintöintegraatio (legacy integration)*: Toteutusta muutettaessa on otettava huomioon olemassa oleva konteksti, jotta voidaan välttää ristiriitoja uutta systeemiä toteutettaessa ja parantaa uudelleenkäyttöä.



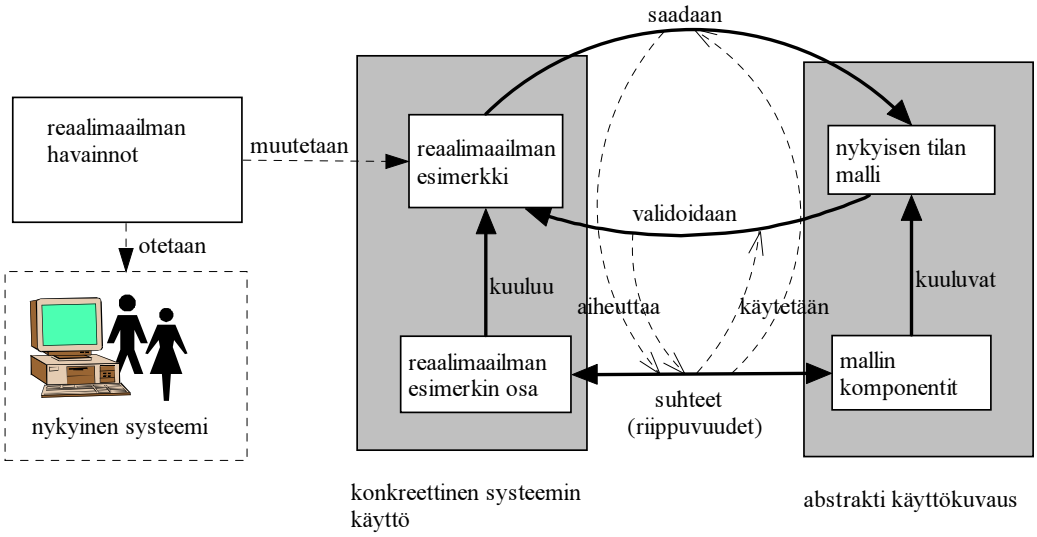
Kuva 23: Muutosprosessin neljä askelta [HaP98]

Uudelleen määritellyn tai päivitetyn nykyisen tilan ja halutun tilan mallien laatu riippuu asianomaisilta saadusta tiedosta. Mitä enemmän saadaan tietoa vaatimusmäärittelyprosessiin osallistuneilta asianosaisilta *multimedian* avulla, sitä paremmin pystytään ymmärtämään käyttöaluetta, välttämään liian yleisiä tai toisaalta liian yksityiskohtaisia abstraktioita ja toistamaan tuloksia. Äänen, videon ja datan yhdistämistä kutsutaan multimediaksi.

11.2 Havaintojen ja mallien väliset yhteydet

Olemassa olevasta systeemistä kerätään havaintoja, joita kutsutaan *reaalimaailman kuvauksiksi (real world scene)*. Kerätty materiaali sisältää tietoa useista systeemin käyttötavoista. Järjestellään materiaali uudestaan *reaalimaailman esimerkiksi (real world example)*, joka koostuu systeemin yhden käyttökerran materiaalista.

Reaalimaailman esimerkkien käyttöön on kaksi pääsyytä. Uudet käsitykset saadaan reaalimaailman esimerkeistä ja toisaalta nykytilan mallit voidaan validoida reaalimaailman esimerkkien avulla. Vaatimusten määrittelijä valitsee reaalimaailman esimerkeistä osia ja merkitsee niiden ja nykyisen tilan malliin kuuluvien komponenttien väliset suhteet. Kuvassa 24 olemassa olevasta systeemistä saadut reaalimaailman havainnot muutetaan reaalimaailman esimerkeiksi. Reaalimaailman esimerkkien pohjalta saadaan nykyisen tilan malli. Malliin kuuluu komponentteja. Reaalimaailman esimerkkien osien ja komponenttien välille merkitään riippuvuussuhteet. Nykytilan mallia ja riippuvuussuhteita voidaan käyttää vaatimusten validointiin. [HaP98]



Kuva 24: Nykyisen tilan mallin ja reaalimaailman esimerkin väliset suhteet [HaP98]

Tämän tyyppinen ratkaisumalli ei sovellu kaikkiin projekteihin. Sitä voidaan käyttää, jos vanhan systeemin toiminnallisuutta voidaan havainnoida ja sovittaa suurelta osin uuteen systeemiin.

11.3 Hyödyt

Tarkastelemalla mallin komponenttien ja reaalimaailman esimerkkien osien välisiä suhteita, saadaan alkujaan täysin suunnittelemattomista havainnoista tehtyä hienorakeinen formaali rakenne. Käsitteellisen mallin komponentit joko määritellään reaalimaailman

esimerkeistä tai validoidaan niitä vastaan. Näin syntyy kaksisuuntainen yhteys havaintojen ja käsitteellisen mallin välille. [HaP98]

Tarkastellaan neljää päähyötyä, jotka saadaan hienorakeisista suhteista. Niiden avulla voidaan selittää nykyisen tilan mallia, vertailla reaalimaailman esimerkkejä, vertailla asianosaisten erilaisia käsityksiä ja tarkastella käsitteellisiä malleja [HaP98].

1) *Nykyisen tilan mallin perustelu (Explanation of the current-state model)*: Nykyisen tilan malleja suunnittelee yksi tai useampi asianosainen. Reaalisuuden abstrahointi syntyy yleensä asianosaisten mielissä ja siksi henkilön, joka ei tunne mallisuunnittelua, on vaikea ymmärtää tehtyjä abstraktioita ja malleja. Komponenttien ja reaalimaailman esimerkkien riippuvuussuhteet selittävät mallin komponentteja. Toisin sanoen mallin määrittelyn abstrahointiprosessi on osittain jäljitettävissä. Uusien ihmisten valmentaminen projektiin helpottuu ja mallia ymmärretään paremmin koko systeemin kehityksen elinkaaren aikana.

2) *Reaalimaailman erilaisten esimerkkien vertailu (Comparison of diffrent real world examples)*: Asianosaiset voivat suorittaa saman tehtävän eri tavoin. Samoin saman vaatimuksen toteuttaminen voi poiketa toisistaan eli olennaisen piirteen toteutus saattaa johtaa erilaisiin tuloksiin. Toteutusten ytimen ymmärtäminen voi olla vaikeaa. Erilaisten havaintojen vertailua voidaan tukea käyttämällä abstraktisempia käsitteellisiä havaintojen kuvauksia. Reaalimaailman esimerkkien ja mallin komponenttien väliset suhteet tukevat tällaista vertailua.

3) *Erilaisten käsitysten määritteleminen ja vertaileminen (Definition and comparison of multiple viewpoints)*: Reaalimaailmaan pohjautuvat nykyisen tilan mallin määrittelyt ovat aina *subjektiivisia* eli ne perustuvat malleja suunnittelevien henkilöiden havaintoihin. Jokainen suunnittelija määrittelee oman näkemyksensä käsitteelliseen malliin samojen tallennettujen havaintojen pohjalta. Erilaisia näkemyksiä voidaan tukea reaalimaailman esimerkkien ja mallin komponenttien erityyppisillä yhteyksillä. Lisäksi yhteyksiä voidaan hyödyntää suunnittelijoiden keskusteluissa, jos keskustelu pohjautuu reaalimaailman havaintoihin.

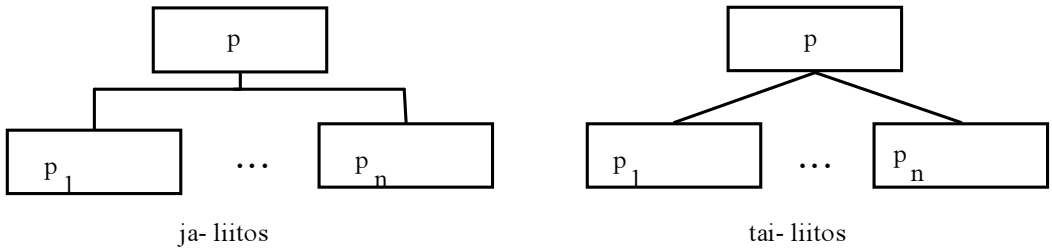
4) *Käsitteellisten mallien tarkasteleminen (reviewing of conceptual models)*: Nykyisen tilan mallien rakenteinen tarkastelu parantaa mallien laatua. Tarkastelun avulla ymmärretään helpommin reaalimaailman esimerkkien osia käyttämällä tehtyjä abstrahointeja. Tarkasteltavaa mallia on helpompi perustella ja kommentoida.

11.4 Päämäärämallinnus

Vanhaa systeemiä analysoitaessa on jo varhaisessa vaiheessa ymmärrettävä systeemin ominaisuuksiin liittyvä kysymys miksi (esimerkiksi, miksi systeemi tukee jotakin aktiviteettia), ennen kuin mennään kysymyksiin mitä ja miten.

Päämääriä (korkean tason vaatimuksia) käytetään vaatimusten kalastamiseen ja kehittämiseen. Asianosaiset kyselevät siis miksi, miten ja kuinka muuten kysymyksiä. Päämäärien avulla systeemi yhdistetään organisaation ja liiketoiminnan käsitteisiin. Päämäärillä selvennetään vaatimuksia ja asianosaisten tavoitteita tarvitsematta mennä liian tarkkoihin yksityiskohtiin. Niitä käytetään myös ristiriitojen käsittelyssä. [HaP98]

Päämäärät voidaan järjestää erilaisiksi *renkaattomiksi graafeiksi*, riippuen siitä onko kysymyksessä *ja-liitos* vai *tai-liitos*. Jokainen päämäärä on graafin solmu. Ja-liitoksessa päämäärä p jaetaan alipäämääriin $p_1, p_2, ..., p_n$. Piirtämisessä päämäärien ja niiden alipäämäärien välillä käytetään suorakulmaisia viivoja. Ja-liitos tarkoittaa sitä, että jos päämäärä p toteutuu, niin kaikkien alipäämäärien $p_1, p_2, ..., p_n$ täytyy myös toteutua. Tai-graafi piirretään käyttämällä suoria viivoja päämäärien ja niiden alipäämäärien välillä. Tai-liitos tarkoittaa sitä, että jos jokin päämäärän p alipäämääristä toteutuu, niin päämäärä p toteutuu. Jokainen alipäämäärä voidaan jakaa edelleen omiin alipäämääriin rekursiivisesti molemmissa tapauksissa. Kuvassa 25 on piirretty ja- ja tai-liitokset. [HaP98]



Kuva 25: Päämäärien mallinnuksessa käytettäviä liitoksia [HaP98]

Päämäärämallit ovat abstraktisempia kuvauksia kuin *tieto-*, *käyttäytymis-* ja *toiminnalliset mallit*. Keskittymällä päämääriin, havaitaan paremmin eri havaintojen välisiä yhtäläisyyksiä. Jos tarvitaan yksityiskohtaisempaa tietoa siitä, miten päämäärä saavutetaan, voidaan tehdä yllämainittuja tarkempia kuvauksia. Päämääriin perustuvassa lähestymistavassa ei vaadita tiukkaa ylhäältä alaspäin suuntautuvaa käsitteellistämisprosessia. Malli tukee myös alhaalta ylöspäin käsitteellistämistä, jossa ensin määritellään havainnoidut matalan tason toiminnalliset päämäärät ja sen jälkeen abstrahoidaan korkeamman tason toiminnallisuutta. Myös molempien yhdistelmää voidaan käyttää. [HaP98]

11.5 Nykyisen tilan mallin määritteleminen

Tyypillisessä analyysityöskentelyssä vaatimusten validointi ja kalastaminen ovat usein tiukasti yhteydessä toisiinsa. Kun päämäärämallintamista kehitetään reaalimaailman esimerkkien pohjalta, vaatimusten kalastaminen ja validointi voidaan *erottaa kahdeksi käsitteellisesti eri objektiksi*. Kalastaminen keskittyy päämäärämallin rakentamiseen saatujen havaintojen pohjalta, kun taas validoinnissa kerätään todisteita reaalimaailman esimerkeistä mallin yksittäisille päämäärille. Analysoija vahvistaa oikeaksi, että päämäärämalli ja reaalimaailman esimerkit vastaavat toisiaan. Jos vahvistusvaiheessa tulee ongelmia, löytyy usein uusia päämääriä. Löydetyt päämäärät lisätään malliin joko uusina tai vaihtoehtoisina päämäärinä. [HaP98]

Jokaisen reaalimaailman esimerkin havaintomateriaali järjestellään jonkin kriteerin perusteella (esimerkiksi tapahtumien ajallisen sekvenssin mukaan), minkä jälkeen ne analysoidaan kauttaaltaan. Jos vaatimusten määrittelijä huomaa, että jostakin reaalimaailman esimerkistä saavutetaan päämäärä, joka on jo määritelty nykytilan mallissa, hän

erottaa reaalimaailman esimerkistä erillisen osan ja yhdistää sen päämäärään riippuvuus linkillä. Jos päämäärää ei ole vielä määritelty mallin hierarkiassa, analysoija on löytänyt uuden päämäärän. Se lisätään päämäärämalliin ja linkitetään reaalimaailman esimerkin osaan. [HaP98]

11.6 Riippuvuuslinkit

Reaalimaailman esimerkin osan ja mallin päämäärän välinen riippuvuus voidaan tulkita kahdella eri tavalla. Merkitään reaalimaailman esimerkin osaa kirjaimella o ja päämäärää kirjaimella p. Osa o voi olla esimerkki siitä, kuinka nykyisestä systeemistä päästään päämäärään p. Osa o merkitään reaalimaailman esimerkin erilliseksi osaksi ja linkitetään päämäärään p. Riippuvuuslinkkiin kirjoitetaan sana *päästään (attain)*. Osaa o kutsutaan päämäärän p *saavuttamistodisteeksi (attainment evidence)*. Jos osa o on esimerkki huonosta, ei-toivotusta tai vältettävästä yrityksestä päästä päämäärään p, se merkitään reaalimaailman esimerkin erilliseksi osaksi ja linkitetään vaatimukseen p riippuvuuslinkillä *ei päästä (fail)*. Tällaisissa tapauksissa osa o on päämäärän p *epäonnistumistodiste (failure evidence)*. Jos osan o ja päämäärän p välillä ei ole vastaavuuksia, niin ne ovat erillisiä, eikä edellä mainittuja linkkejä tarvita. [HaP98]

11.7 Päämäärien kalastaminen

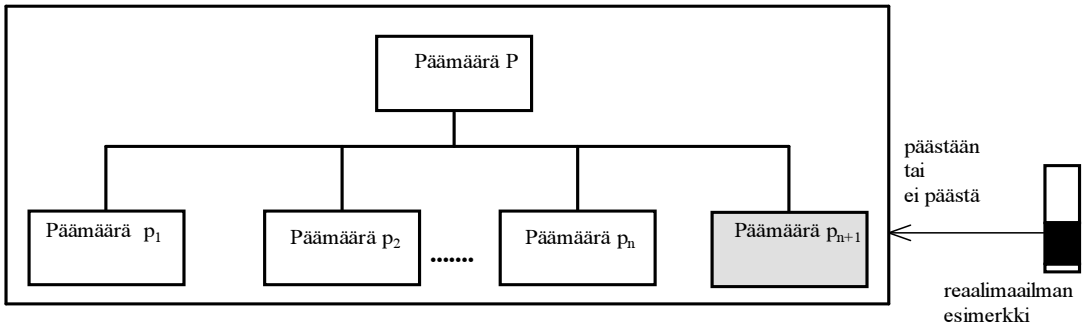
Reaalimaailman esimerkkien analysointi voi johtaa uusien päämäärien kalastamiseen. Kun asianosaiset (vaatimustenmäärittelijät, alan asiantuntijat, käyttäjät) tulkitsevat, että reaalimaailman esimerkkien osasta o voi päästä sellaiseen päämäärään p, jota ei vielä ole päämäärämallissa, voidaan erottaa neljä eri tapausta. Kuvissa 26 ja 27 on kuvattu tapaukset 1 ja 4. Uusi päämäärä on niissä väritetty harmaalla ja reaalimaailman esimerkin osat ovat joko mustia tai valkoisia. [HaP98]

1) Asianosaisten mielestä uutta päämäärää p tarvitaan, jotta päästäisiin ylempään päämäärään P. Tällöin uusi päämäärä on päämäärän P alipäämäärä ja se lisätään vaatimuksiin $p_1, p_2, \ldots, p_n$ ja-liitoksella.

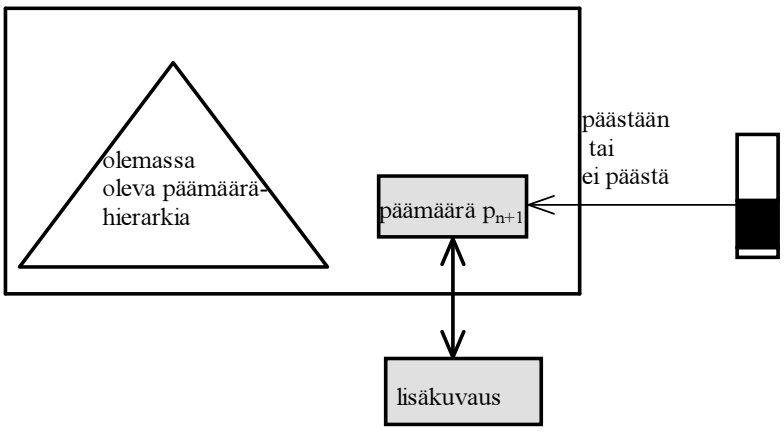
2) Asianosaisten mielestä päämäärä p on uusi vaihtoehto päämäärän P alipäämääräksi. Päämäärää P ei ole vielä ilmaistu olemassa olevilla vaihtoehdoilla $p_1, p_2, \ldots, p_n$. Tällöin uusi päämäärä lisätään päämäärän P alipäämäärävaihtoehtoihin tai-liitoksella.

3) Jos asianosaiset eivät pysty päättämään, mikä on uuden päämäärän suhde muihin päämääriin jo olemassa olevassa päämäärähierarkiassa, niin silloin uusi päämäärä lisätään hierarkiaan ilman liitoksia muihin päämääriin.

4) Asianosaiset eivät tiedä mihin päämäärään reaalimaailman esimerkin osa vaikuttaa. Päämäärä merkitään määriteltäväksi päämääräksi ja liitetään reaalimaailman esimerkkiin. Päämäärään voidaan lisätä ylimääräinen kuvaus. Kun päämäärä myöhemmin kiteytyy, voidaan lisäkuvausta ja yhteyttä reaalimaailman esimerkin osaan käyttää lisäinformaationa.



Kuva 26: Päämäärän kalastamisen tapaus 1: reaalimaailman esimerkin osa aiheuttaa ja-liitoksen [HaP98]



Kuva 27: Päämäärän kalastamisen tapaus 4: reaalimaailman esimerkin osa aiheuttaa määriteltävän vaatimuksen lisäkuvauksineen [HaP98]

Kaikissa neljässä tapauksessa reaalimaailman esimerkkien osat linkitetään uuteen päämäärään. Jos päämäärä todella saavutetaan, kirjoitetaan linkkiin päästään. Jos päämäärää ei saavuteta, merkitään linkkiin ei päästä. [HaP98]

11.8 Positiivia ja negatiivisia esimerkkejä

Päämäärään voidaan päästä monella eri tavalla, kun eri asianosaiset yhdistävät päämäärämalliin useita reaalimaailman esimerkkejä. Olkoon o päämäärän p saavuttamistodiste, eikä välitetä siitä, keneltä todiste on saatu. Erotetaan reaalimaailman esimerkin osista alijoukko $\{o_{i1},\ldots o_{im}\}$, niin että tähän joukkoon otetaan sellaiset saavuttamistodisteet eli osat, joista päästään parhaiten päämäärään p. Nyt riippuvuus kaikkien alijoukon osien o_i ja päämäärän p välillä on tyypiltään positiivista. Saavuttamistodiste o_i on siis päämäärän p *positiivinen todiste*. Vastaavasti voidaan tehdä epäonnistumistodisteista alijoukko $\{o_{j1},\ldots o_{jm}\}$, joista ei päästä selvästikään päämäärään p. Tällöin o_j on päämäärän p *negatiivinen todiste*. Positiivisten ja negatiivisten riippuvuuksien avulla ymmärretään paremmin päämäärämallin yksilöityjä päämääriä. [HaP98]

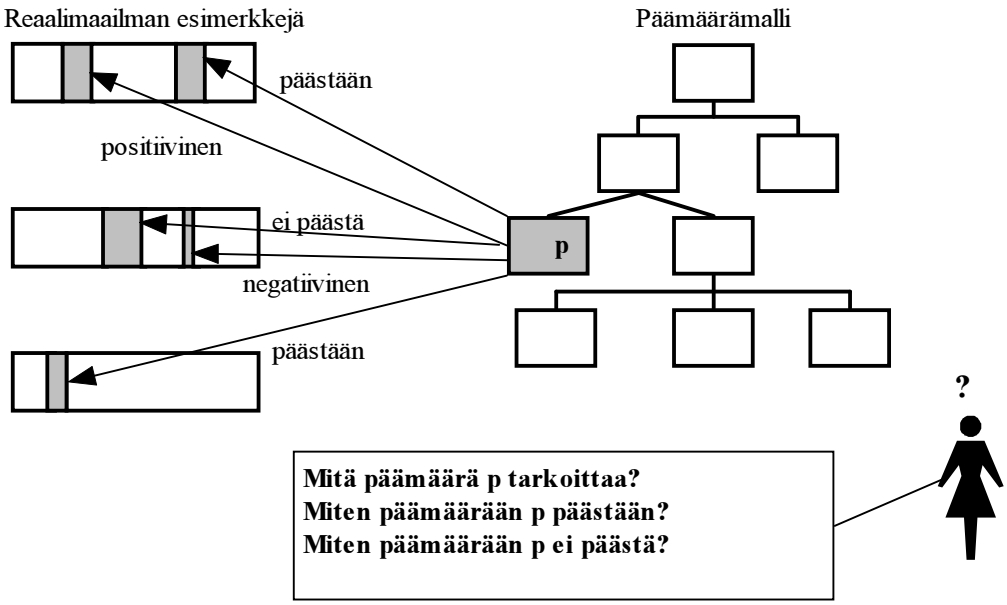
11.9 Riippuvuuksien hyödyntäminen

Riippuvuuksista voidaan muodostaa *kolmikko* K(o,lt,p), missä o edustaa reaalimaailman esimerkin jotakin osaa, kirjaimet lt linkin tyyppiä ja p päämäärää. Näin saadaan alun perin täysin suunnittelemattomasta multimedia materiaalista linkitetty rakenne. Päämäärämalliin on lisätty saavuttamistodisteita, jotka on saatu asianosaisten havainnoista. Kyselyillä kolmikosta K saadaan päämäärämallin ja reaalimaailman esimerkkien välisiä polkuja, joita voidaan hyödyntää erilaisissa tilanteissa. [HaP98]

11.9.1 Päämäärien perusteleminen

Päämääriä suunnitellaan usein liian abstraktisesti, jolloin eri asianosaisten on vaikea ymmärtää päämäärän tarkoitusta yhdenmukaisesti. Kolmikon K jäljitettävyyslinkeistä saadaan tietoa, päästäänkö johonkin päämäärään p vai ei. Varsinkin käyttämällä positiivisia ja negatiivisia linkkejä, voidaan etsiä tietoa päämäärien saavutettavuudesta. Posi-

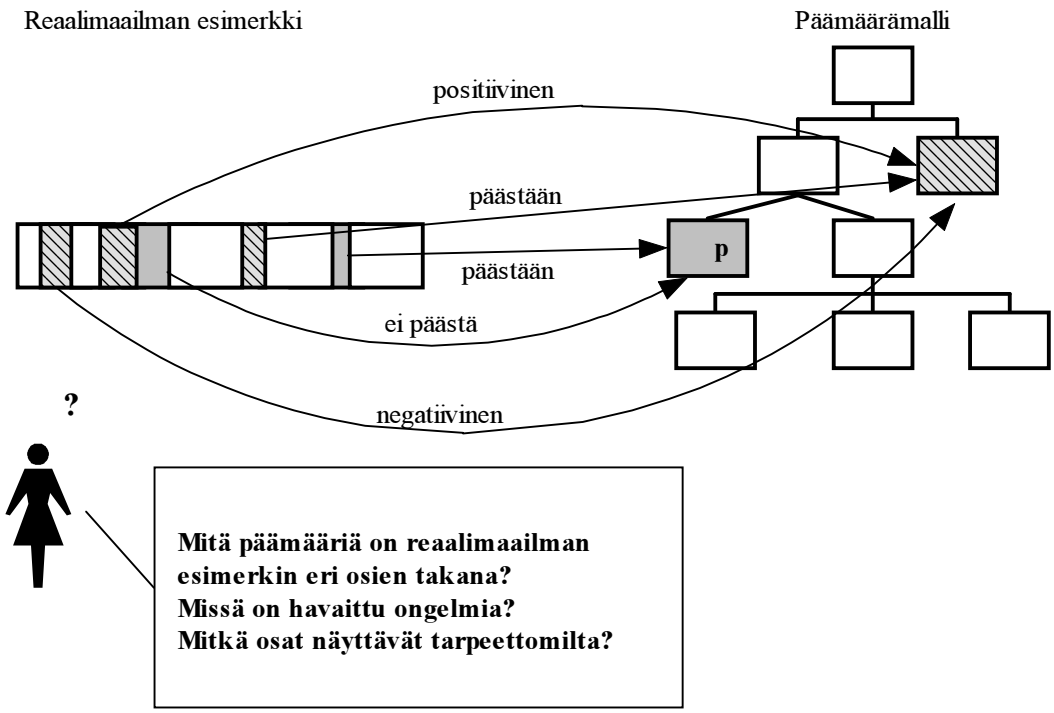
tiivisten ja negatiivisten linkkien avulla uudet ryhmän jäsenet ja asianosaiset, jotka eivät tunne päämäärämallinnuksen *notaatioita*, voidaan tutustuttaa helpommin ja nopeammin tutkittavaan aiheeseen. Kuvassa 28 on esimerkki jäljitettävyyslinkeistä, joita voidaan käyttää, selitettäessä päämäärämallin päämääriä. Päämäärästä p piirretään nuolet niihin reaalimaailman esimerkkien osiin, joista päästään tai ei päästä päämäärään p tai jotka edustavat positiivisia tai negatiivisia saavutettavuustodisteita. [HaP98]



Kuva 28: Päämäärän selittäminen reaalimaailman esimerkkien osien avulla [HaP98]

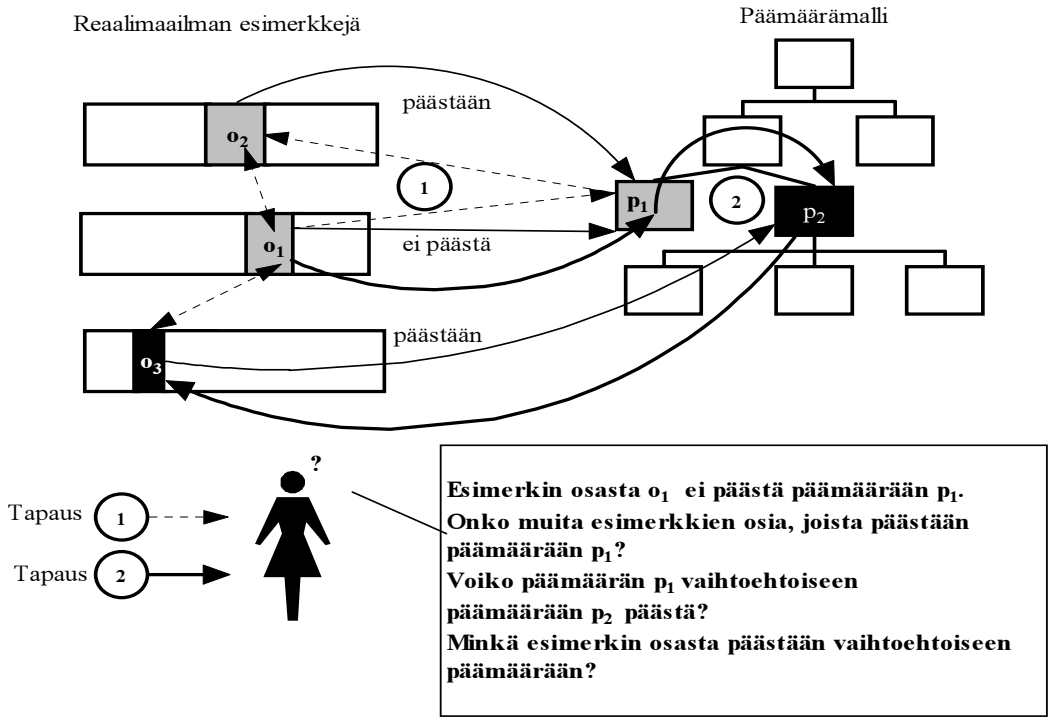
11.9.2 Reaalimaailman esimerkkien perusteleminen

Aloituskohdaksi voidaan valita myös jokin reaalimaailman esimerkki ja tarkastella asioita toiseen suuntaan. Kuvassa 29 on kuvattu, mitkä reaalimaailman esimerkkien osat on linkitetty päämäärämallin päämääriin saavutettavuustietoineen. Kaikista reaalimaailman esimerkkien osista ei lähde nuolta tarkasteltaviin päämääriin, koska niiden ja päämäärien välille ei ole löydetty riippuvuuksia. [HaP98]



Kuva 29 Reaalimaailman esimerkkien takana olevien päämäärien perusteleminen [HaP98]

Jäljitettävyystietoja voidaan käyttää myös etsittäessä reaalimaailman esimerkin osan o *jäljitettävyysvaihtoehtoja*. Kuvassa 30 on alkutilanteena sellainen tapaus, että reaalimaailman esimerkin osasta o_1 ei päästä päämäärään p_1. Tarkastellaan kahta erilaista tapausta. Tapauksessa 1 etsitään esimerkeistä osia, josta päästään päämäärään p_1. Sellainen on kuvan 30 esimerkissä oleva osa o_2. Tapauksessa 2 tutkitaan päämäärän p_1 vaihtoehtoista päämäärää p_2. Linkkien avulla etsitään reaalimaailman esimerkeistä sellaisia osia, joista päästäisiin päämäärään p_2. Sellainen on osa o_3. [HaP98]



Kuva 30: Vaihtoehtoisten tapojen löytäminen päämäärien käsittelyyn [HaP98]

11.9.3 Päämäärämallin tarkasteleminen

Päämäärämallin ja muidenkin käsitemallien kehittäminen riippuu vaatimusten määrittelijän *subjektiivisista* reaalimaailman tulkinnoista. Päämäärämallin tarkastelu on erittäin tärkeää, jos halutaan *tasapainottaa* eri asianosaisten havaintoja ja mielipiteitä. Reaalimaailman esimerkkien osien ja päämäärämallin komponenttien välisten suhteitten avulla voidaan havainnollistaa mallin kehitykseen vaikuttavia päätöksiä ja huomioita tarkasteluprosessin aikana. *Tarkastaja* ymmärtää alkuperäisen mallin rakentajan tulkintoja, vaikkei olisikaan niistä samaa mieltä. Tarkastaja voi lisätä reaalimaailman esimerkkien osiin liittyviin linkkeihin mielipiteensä siitä, onko hän samaa mieltä kuin mallin rakentaja. Tarkastaja voi myös lisätä uusia linkkejä päämäärien ja reaalimaailman esimerkkien välille tai jopa lisätä malliin uusia päämääriä. Tarkastaja dokumentoi havaintonsa ja muutoksensa samalla tavoin kuin alkuperäisen mallin rakentaja. Näin toimimalla huomataan eri asianosaisten havaintojen erilaisuudet ja myös tarkastajan mahdolliset virheet, varsinkin jos hän on alkuperäistä rakentajaa tietämättömämpi asiasta. [HaP98]

11.10 Lisätiedoin varustettu päämääräpuu

Muodostettuja riippuvuuksia voidaan käyttää päämäärien perustelemisten lisäksi myös kahden tai useamman esimerkin vertailemiseen sekä kahden tai useamman asianosaisen näkemysten tutkimiseen. [HaP98]

11.10.1 Reaalimaailman esimerkkien vertaileminen

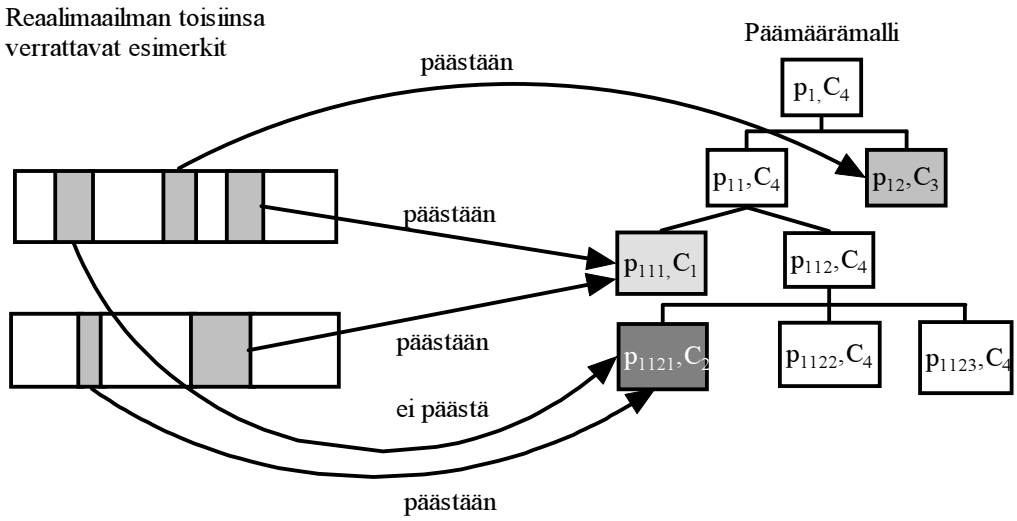
Analysoija haluaa *vertailla kahta reaalimaailman esimerkkiä* löytääkseen niiden välisiä yhtäläisyyksiä tai eroja. Yleensä tällainen vertailu on vaikeaa, koska esimerkkitasolla vastaavuudet on hahmoteltu epäselvästi. Tutkimalla mallin ja esimerkkien välisiä suhteita, voidaan löytää mahdollisia vastaavuuksia esimerkkien välillä tarkastelemalla niitä päämääriä, joihin esimerkit liittyvät. Esimerkkien kaikki osat voidaan liittää mallin päämääriin jollakin seuraavista kolmesta *riippuvuustyypistä*: päästään, ei päästä ja irrallinen. Näin saadaan yhdeksän erilaista riippuvuusyhdistelmää, jotka on kuvattu taulukossa 7. [HaP98]

Taulukko 7: Reaalimaailman esimerkkien e_1 ja e_2 ja päämäärän p väliset riippuvuudet [HaP98]

e_1/e_2	päästään	ei päästä	irrallinen
päästään	C1	C2	C3
ei päästä	C2	C1	C3
irrallinen	C3	C3	C4

Molempien esimerkkien riippuvuudet päämäärästä p voidaan ilmaista käyttäen neljää kategoriaa C1, C2, C3 ja C4. C1 tarkoittaa sitä, että molemmat esimerkit on linkitetty päämäärään p samantyyppisellä linkillä. C2 tarkoittaa puolestaan sitä, että linkit ovat erityyppisiä. Tapauksessa C3 vain toinen esimerkeistä on linkitetty päämäärään p ja tapauksessa C4 kumpaakaan esimerkeistä ei ole linkitetty päämäärään p. Kuvassa 31 nämä neljä kategoriaa on sijoitettu eri päämääriin samoilla harmaan sävyillä kuin taulukossa 7. Tarkastelemalla kuvaa huomataan, että esimerkiksi päämäärän p_{1121} kohdalla esimerkit eroavat toisistaan. Vastaavasti tutkimalla kaikkia esimerkkien osien linkkejä

päämäärämalliin, voidaan tutkia esimerkkien eroja toisiinsa nähden, jos linkitykset on tehty oikein [HaP98].



Kuva 31: Reaalimaailman kahden esimerkin erojen hahmottaminen [HaP98]

11.10.2 Asianosaisten erilaiset näkökulmat

Tarkastuksen aikana tarkastaja laittaa omat näkemyksensä päämäärämalliin sekä päämäärien ja reaalimaailman esimerkkien osien välisiin suhteisiin. Lisäämällä päämäärämallin päämääriin kaksi värikenttää, joista toinen edustaa itse päämäärää ja toinen reaalimaailmanesimerkin osaan o liittyvää linkkiä, voidaan samaan kuvaan saada kahden eri asianosaisen näkemykset. Taulukoissa 8 ja 9 käy ilmi harmaan eri sävyjen avulla, minkä tyyppisiä muutoksia tarkastaja tekisi vaadittuun malliin ja suhteisiin. [HaP98]

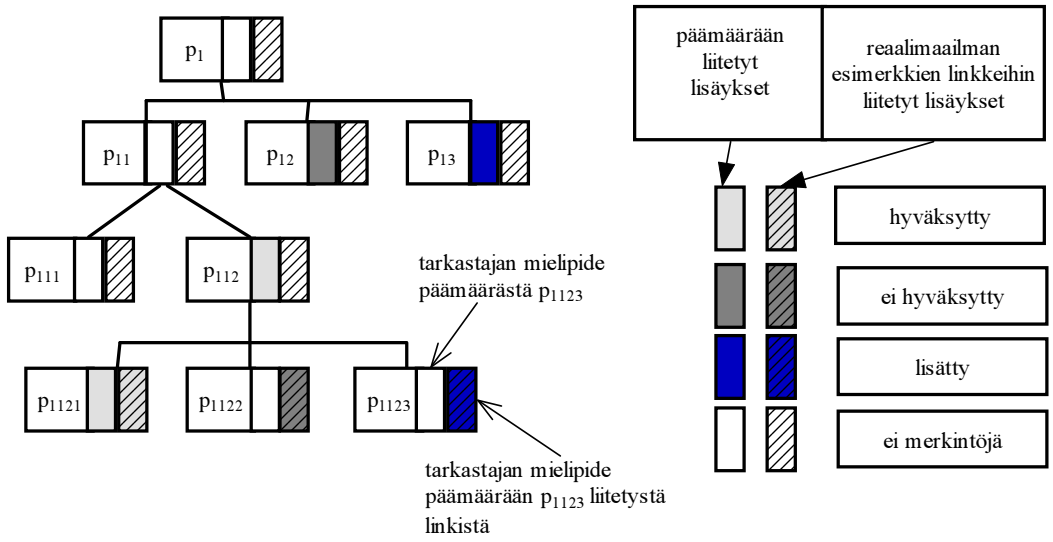
Taulukko 8: Asianosaisten erilaiset näkökulmat päämäärästä p [HaP98]

Tarkastajan tekemät muutokset	**mielikuva päämäärästä p**
Tarkastaja on hyväksynyt päämäärän p	C1
Tarkastaja ei ole hyväksynyt päämäärää p	C2
Tarkastaja on lisännyt päämäärän p	C3
Tarkastaja ei ole tehnyt merkintöjä päämäärään p	C4

Taulukko 9: Asianosaisten erilaiset näkökulmat mallin päämäärän p ja reaalimaailman esimerkin o välisistä suhteista

Tarkastajan tekemät muutokset	**mielikuva päämäärästä p**
tarkastaja on hyväksynyt päämäärän p ja esimerkin osan o välisen riippuvuustyypin lt	C1
tarkastaja ei ole hyväksynyt päämäärän p ja esimerkin osan o välistä riippuvuustyyppiä lt	C2
tarkastaja on lisännyt päämäärän p ja esimerkin osan o välille riippuvuustyypin lt	C3
tarkastaja ei ole tehnyt merkintöjä päämäärän p ja esimerkin osan o välille	C4

Kuva 32 esittää tarkastajan lopputulosta. Siinä jokaiseen päämäärään on lisätty kaksi suorakulmiota, toinen itse päämäärää varten ja toinen sen ja esimerkin osien välisille linkeille. Suorakulmion väri kertoo mallin alkuperäisen suunnittelijan ja tarkastajan välisistä näkemyseroista. Tarkastelemalla kuvaa, huomataan esimerkiksi, että tarkastaja ei ole samaa mieltä päämäärästä p_{12} kuin alkuperäinen mallin tekijä. Tarkastaja on myös lisännyt malliin uuden päämäärän p_{13}. Päämäärästä p_{1121} ja siihen liittyvästä linkistä tarkastaja on saamaa mieltä mallin tekijän kanssa, mutta päämäärään p_{1123} hän lisäisi uuden linkin reaalimaailman esimerkin osaan o. Lisätiedoin varustettua päämääräpuuta voidaan käyttää eri asianosaisten näkemysten välillä olevien ristiriitojen etsimiseen. [HaP98]



Kuva 32: Päämääräpuu hahmottaa asianosaisten näkökantoja. Kuvassa on käytetty taulukkojen 8 ja 9 väritystä [HaP98]

11.10.3 Vaatimusten merkityksellisyys- ja onnistumisindeksit

Riippuvuussuhteita määriteltäessä ei päämäärien riippuvuuksia kaikkiin reaalimaailman esimerkkeihin tarvitse tarkastella. Varsinkin jos päämäärä on päämäärähierarkiassa jonkin haaran alla olevassa tai-liitoksessa. Jos esimerkit liittyvät vain yhteen päämäärähierarkian haaraan, ei niitä oteta huomioon tarkasteltaessa muita haaroja. [HaP98]

Määritellään seuraavaksi *merkityksellisyysindeksi (relevance index)* ja *onnistumisindeksi (success index)* päämäärille, joihin liittyy useamman kuin kahden asianosaisen näkemykset tai joihin liittyy enemmän kuin kaksi reaalimaailman esimerkkiä. Määritellään ensin kolme joukkoa S_p , E_p ja K_p . Olkoon joukossa S_p kaikki sellaiset reaalimaailman esimerkit, joista päästään päämäärään p ja vastaavasti joukossa E_p kaikki sellaiset esimerkit, joista ei päästä päämäärään p. Nämä joukot eivät ole välttämättä erillisiä. Joistakin reaalimaailman esimerkin osista löytyy sellaisia osia, joista päästään päämäärään p, sekä sellaisia osia, joista ei päästä tähän päämäärään. Otetaan joukkoon K_p kaikki sellaiset reaalimaailman esimerkit, joita on käytetty mallinnuksessa. Merkityksellisyysindeksi saadaan laskemalla kaavalla:

$$M_p = (| S_p \cup E_p |)/ | K_p |,$$

missä itseisarvot tarkoittavat sisältöjen kokonaislukumäärää mallinnuksessa. Merkityksellisyysindeksi saa arvoja nollan ja ykkösen väliltä. Jos M_p on 0, niin se tarkoittaa sitä, että mistään reaalimaailman esimerkistä ei päästä päämäärään p. Päämäärä p olisi siis merkityksetön päämäärä. Mitä suurempi merkityksellisyysindeksin arvo on, sitä merkityksellisempi päämääräkin on. [HaP98]

Onnistumisindeksi lasketaan kaavasta:

$$O_p = | S_p | /(| S_p | + | E_p |).$$

Jos onnistumisindeksi on 0, niin se tarkoittaa sitä, että mallista löytyy vain epäonnistumistodisteita. Jos onnistumisindeksi on 1, niin se tarkoittaa puolestaan sitä, että kaikista reaalimaailman esimerkeistä päästään päämäärään p. Jos onnistumisindeksi on 0.5, niin silloin päämäärän päästään yhtä monesta esimerkistä kuin ei päästä.

Merkityksellisyysindeksejä laskemalla voidaan löytää päämäärät, joihin päästään vain harvoin, jolloin indeksin arvo olisi esimerkiksi pienempi kuin 0.2. Lisäanalysoinnilla voidaan etsiä syitä, miksi päämääriin ei päästä. Vastaavasti jos onnistumisindeksit olisivat matalia, niin nykyinen systeemi ei tukisi tällaisia päämääriä. Jos ja-liitoksessa päämäärän onnistumisindeksi on matalampi kuin sen alipäämäärien keskimääräinen onnistumisindeksi, niin se osoittaisi, että jokin alipäämäärä on jäänyt huomaamatta tai jokin alipäämäärä on kriittinen. [HaP98]

Asianosaisten näkemysten eroja voidaan myös tutkia indeksien avulla. Olkoon M_p jonkun asianosaisen näkemysten mukainen merkityksellisyysindeksi ja M_p' toisen asianosaisen näkemysten mukainen merkityksellisyysindeksi. Vastaavasti onnistumisindeksit olisivat O_p ja O_p'. Laskemalla erotukset saadaan *merkityksellisyys- ja onnistumiseroavuudet (relevance and success difference)*:

$$\triangle M_p = M_p - M_p'$$
$$\triangle O_p = O_p - O_p'.$$

Tarkastelemalla näitä erotuksia, huomataan esimerkiksi, että jos merkityksellisyyseroavuus $\triangle M_p > 0$, niin silloin jälkimmäinen asianosainen on löytänyt vähemmän sellaisia reaalimaailman esimerkkien osia, mistä päästään päämäärään p, kuin edellinen asianosainen. Jos $\triangle M_p < 0$, niin silloin jälkimmäinen asianosainen on löytänyt uusia reaalimaailman esimerkkien osia, joista päästään päämäärään p. Vastaavasti, jos $\triangle O_p > 0$, niin jälkimmäinen asianosainen ei ole samaa mieltä edellisen kanssa päämäärään p pääsemisestä tai hän on löytänyt lisää sellaisia osia, mistä ei päästä päämäärään p. Päämääriä, joihin liittyy suuria indeksien eroavuuksia täytyy tarkastella lähemmin ja niistä on hyvä keskustella asianosaisten kesken. [HaP98]

11.1 Kokeilun tulokset ja arviointi

Kokeiltuaan menetelmää, Peter Haumer, Klaus Pohl ja Klaus Weidenhaupt tulivat siihen tulokseen, että reaalimaailman esimerkkien ja päämäärien hienorakeiset suhteet helpottivat päämäärämallien ymmärtämistä. Vaikka asianosaiset tarkkailivat vanhaa systeemiä eri aikoina ja eri paikoissa, päästiin tarkkailun tuloksista parempaan yhteisymmärrykseen. Uusien ryhmän jäsenien perehdyttäminen ongelma-alueeseen ja mallinmäärittelyyn helpottui. [HaP98]

Menetelmän käyttäminen lisäsi selvästi uuden systeemin vaatimusten kalastamisesta ja validoinnista vastuussa olevien asianosaisten työmäärää. Aikaa kului havaintojen keräämiseen eri paikoista ja saadun materiaalin järjestämiseen reaalimaailman esimerkeiksi ja niistä johdetuiksi päämääriksi. Materiaalin digitointiin ja leikkaamiseen kului myös suhteellisen paljon aikaa, koska tutkimuksessa käytettiin sellaista videotekniikkaa, joka oli uutta havaintojen kerääjille ja analysoijille. [HaP98]

Menetelmä ei sovi, jos vain muutama tiedoiltaan ja kokemuksiltaan samantaustainen asianosainen neuvottelee nykytilan mallista, koska he pääsevät todennäköisesti yhteisymmärrykseen mallista ilman reaalimaailman esimerkkien osiin jakoakin. Reaalimaailman esimerkeistä johdettavat päämäärät eivät selviä aina vain tutkimalla esimerkkejä, vaan tarvitaan asiantuntijoiden apua ja lisäinformaatiota. Menetelmä vaatii myös, että on vanha systeemi, minkä toiminnallisuudesta suuri osa viedään uuteen systeemiin. Videointi tarkkailutilanteissa on hyvä menetelmä silloin, kun tapahtuman toistaminen

on kallista tai vaikeaa. Myös asianosaiset, jotka eivät itse pääse tarkkailemaan vanhaa systeemiä, voivat katsoa videomateriaalia ja tehdä omat huomionsa.

12 POHDINTA

Vaatimusten kalastaminen, validointi ja verifiointi ovat tärkeitä asioita, kun pohditaan jonkin ohjelmointiprojektin tai tietojärjestelmän onnistumista. Valitettavasti näihin toimenpiteisiin *ei panosteta tarpeeksi*, koska niin useat projektit epäonnistuvat. Vaikka myönnetäänkin niin sanottu *lumipalloilmiö* eli virheiden aiheuttamat kustannukset kasvavat, mitä myöhemmin ne löydetään, ei *aikaa* tai *varoja* myönnetä tarpeeksi näihin toimenpiteisiin. Ohjelmistotuotannossa ollaan siirtymässä komponenttien käyttöön, mikä korostaa hyvän arkkitehtuurin tarvetta. Ohjelmiston vaatimukset heijastuvat tiiviisti arkkitehtuuriin ja jollei vaatimusmäärittelyä tehdä kunnolla, epäonnistuu arkkitehtuurikin.

Organisaatioiden *johdon ja esimiesten pitäisi sisäistää* ajatus vaatimusten kirjaamisen, validoinnin ja verifioinnin tärkeydestä. Heidän pitäisi *kouluttaa* työntekijöitään paremmiksi vaatimusmäärittelijöiksi, vaikka koulutuksesta saatavat hyödyt eivät näkyisikään heti. Johdon pitäisi myös kiinnittää huomiota *motivaatio-ongelmiin*, jotta vaatimusten määrittelijä tuntisi tekevänsä tärkeää työtä. Onnistumiset pitää palkita. Koulutus voisi kattaa kaikkea mahdollista, mitä hyvän vaatimusmäärittelijän on osattava. Sellaisia voisi olla esimerkiksi kirjoittamisen opettelu, tiedonhankinta, keskustelutaito, ongelmien kartoittaminen ja rajaaminen sekä erilaisten validointi- ja verifiointitekniikoiden opetteleminen.

Vaatimuksia on pystyttävä mittaamaan ja arvioimaan. Huonosti määritellyt vaatimukset aiheuttavat ongelmia ohjelmistoa tai systeemiä tuotettaessa. Asiakas ei ehkä saakaan sellaista lopputuotetta, kuin hän on halunnut ja kustannukset ja aikataulut ovat saattaneet pettää. Tutkielmassa onkin käsitelty erilaisia laatuattribuutteja, jotka vaatimusmäärittelyn vaatimusten ja dokumentin pitäisi toteuttaa.

Validoinnissa ja verifioinnissa voidaan käyttää apuna useita erilaisia tekniikoita. Tutkielmassa paneuduttiin tarkemmin muutamiin niistä. Yhtenä valintakriteerinä oli se, että onko tekniikassa *jotakin uutta*. Vaatimusten kaksitasoisessa validoinnissa ja verifioinnissa esimerkiksi yhdisteltiin tuttuja kaavioita, kuten käyttötapaus-, luokka-, peräkkäis- ja aktiviteettikaavioita. Väärinkäyttötapaukset ovat puolestaan esimerkki sellaisesta tek-

niikasta, missä asioita tarkasteltiin hieman *eri näkökulmasta* kuin tavallisesti. Siinä turvallisuusvaatimuksia kalastetaan ja validoidaan pelaamalla eräänlaista "peliä", missä ovat vastakkain laillinen käyttäjä ja väärinkäyttäjä. Oli myös kiinnostavaa tutkia validointiin ja verifiointiin liittyviä ongelmia siirryttäessä vanhasta systeemistä uuteen *päämääräkeskeisen* vaatimusten kehittelyn kautta.

Lisätutkimusta kannattaisi tehdä siitä, löytyykö markkinoilta uusien tekniikoiden kehittäjiä tai valmiita työkaluja vaatimusten validointiin ja verifiointiin, ja kuinka paljon tällaisia työkaluja käytetään eri organisaatioissa. Mielenkiintoista olisi myös tietää, löytyykö organisaatioista tahtoa ja rahoitusta uusien työkalujen ja tekniikoiden kokeiluun.

LÄHDELUETTELO:

[Ale03] Alexander, I.: *Misuse Cases: Use Cases with Hostile Intent,* IEEE Computer Society, Software, Vol. 20, No.1, January/February, 2003.

[AnC01] Anton, A., Carter, R., Dagnino, A., Dempster, J., Siege, D.: *Deriving Goals from Use-Case Based Requirements Specification,* Requirements Engineering 6: 63-73, Springer-Verlag London Limited, 2001.

[BaK00] Bailey, M., Kar, P.: *Characteristics of Good Requirements,* Compliance Automation, Inc., Saatavana www-muodossa: <URL: http/www complianceautomation.com/papers/incose.htm> viitattu 22.11.2003.

[CuI96] Cuthill, B., Ippolito, L., Wallace, D.: *Reference Information for the Software Verification and Validation Process,* NIST Special Publication, 1996

[Dav90] Davis, Alan: *Software Requirements Analysis and Specification,* Prentice-Hall, Inc., 1990.

[Dav99] Davies, P.: *Requirements in Context – Finding a Common Culture with Overseas Customer,* Incose insight, Saatavana www- muodossa: <URL:http/www.incose.org/insight-archive/posted/vol-2-issue-4a.pdf>, viitattu 10.11.2003.

[DoT90] Dorfman, M., Thayer, R.: *Software Reguirements Engineering,* IEEE Computer Society Press, Los Alamitos, California, 1990.

[DzF98] Dzida, W., Freitag, R.: *Making Use of Scenarios for Validating Analysis and Design,* IEEE Transactions on Software Engineering, Vol. 24, No. 12, December, 1998.

[Eer02] Eerola, A.: *Tietojärjestelmien analyysi- ja suunnittelumenetelmät,* Kurssimateriaali, Tietojenkäsittelytieteenlaitos, Kuopion yliopisto, 2002.

[Fel01] Fellows,L.: *Increase Product Quality, Decrease Development Cost,* Compliance Automation, Inc., Saatavana www-muodossa: <URL: http/www.complianceautomation.com>, viitattu 22.11.2002.

[FoS00] Fowler, M., Scott, K.: *UML distilled: a Brief Guide to the Standard Object Modelling Language,* Addison-Wesley, 2000.

[Gaa99] Van Gaasbeek, J., *Before Requirements: What, Who, Where, When, Why and How ?,* Incose insight, Saatavana www- muodossa: <URL:http/www.incose.org/insight-archive/posted/vol-2-issue-4a.pdf>, viitattu 10.6.2003.

[Gab99] Gabb, A.: *Denying Requirements – Twelwe Excuses,* Incose insight, 1999, Saatavana www- muodossa: <URL:http/www.incose.org/insight-archive/posted/vol-2-issue-4a.pdf>, viitattu 15.3.2004.

[GoM99] Gopnzales, R., Montoya, V.: *Model-based Standards: Wouldn't That be Nice?,* Incose insight, Saatavana www- muodossa: <URL:http/www.incose.org/insight-archive/posted/vol-2-issue-4a.pdf>, viitattu 15.3.2003.

[HaH98] Haumer, P., Heymans, P., Pohl, K.: *An Integration of Scenario-Based Requirements Elicitation and Validation Techniques,* Crews report, 1998.

[HaM01] Haikala, Ilkka & Märijärvi, Jukka: *Ohjelmistotuotanto*, Tummavuoren Kirjapaino Oy, Vantaa, 2001

[Har99] Harju, H.: *Ohjelmiston luotettavuusarvioinnit*, Sytyke ry, Saatavana www- muodossa:<URL: http://www.pcuf.fi/sytyke/ lehti/kirj/ st19992/ 15.pdf>, viitattu 11.1.2004.

[HaP98] Haumer, P., Pohl, K., Weidenhaupt, K.: *Reuirements Elicitation and Validation with Real World Scenes,* IEEE Transactions on Software Engineering, Vol. 24, No. 12, Special Issue on Scenario Management, December, 1998.

[Hoo00] Hooks, I.: *Why Johnny Can't Write Requirements,* Saatavana www-muodossa: <URL: http://www.complianceautomation.com/papers/ whyjohnny.htm>, viitattu 27.4.2004.

[Koi00] Koikkalainen, P.:*Vertailutestaus,* Saatavana www- muodossa: <URL:http://erin.mit.jyu.fi/pako/kurssit/ot2000/112/lect12.html>, viitattu 21.4.2004.

[KoS98] Kotonya, G., Sommerville, I.: *Requirements Engineering Processes and Techniques,* John Wiley & Sons, England, 1998.

[KöS01] Kösters, G., Six, H., Winter, M.: *Coupling Use Cases and Class Models as a Mean for Validation and Verification of Requirements Specifications,* Requirements Engineering 6: 3-17, Springer-Verlag London Limited, 2001.

[Mac96] Macaulay, L.: *Requirements Engineering,* Springer-Verlag London Limited, 1996.

[MaM00] Mathiassen, L., Munk-Madsen, A., Nielsen, P.A., Stage, J.:*Object-oriented analysis & design,* Marko Publishing Aps, Aalbporg, Denmark, 2000.

[RoR99] Robertsson, S., Robertson, J., *Mastering the Requirements Process*, Inc., ACM Press, 1999

[Som95] Sommerville, Ian: *Software Engineering*, Addison-Wesley Publishers Ltd., Edinburgh Gate, 1995.

[SoS97] Sommerville, I., Sawyer, P.: *Requirements Engineering, a Good Practice Guide,* John Wiley & Sons, 1997.

[Ter01] Tervonen I.: *Katselmointi ja testaus,* Kurssimateriaali, Tietojenkäsittely-tieteenlaitos, Oulun yliopisto, 2001.

[Tho99] Thomas, B.: *Meeting the Challenges of Requirements Engineering,* SEI Interactive, Saatavana www-muodossa: <URL: http://interactive.sei.cmu.edu/Features/1999/March/ Spotlight/ Spotlight.mar99.htm>, viitattu 20.11.2003.