CASE-välineet ohjelmistoprosessien parantamisessa

Jussi Kähkönen

13.12.2004

Joensuun yliopisto
Tietojenkäsittelytiede
Pro gradu -tutkielma

Tiivistelmä

Ohjelmistoprosessien parantaminen on yksi keino kehittää organisaation ohjelmistotuotantoprosessia. Prosessien parantamisen avulla pyritään parantamaan ohjelmistotuotteen laatua ja organisaation suorituskykyä. Ohjelmistoprosessien suorittamista voidaan tukea ja tehostaa CASE-välineiden avulla. Välineiden tehokas käyttäminen edellyttää kuitenkin myös monien muiden tekijöiden huomioimista organisaatiossa. Tässä tutkielmassa on perehdytty kirjallisuuden pohjalta CASE-välineiden vaikutukseen ohjelmistoprosessien parantamisessa sekä arvioitu Joensuun yliopistossa kehitetyn Sfrm-ohjelman ominaisuuksia ja soveltuvuutta vaatimusmäärittelyprosessin parantamisessa. Sfrm-ohjelma soveltuu erityisesti ensimmäiseksi vaatimusmäärittely- ja vaatimustenhallintaohjelmaksi lähdettäessä parantamaan määrittelemätöntä, kypsyydeltään satunnaisella tasolla olevaa prosessia kohti toistettavaa tasoa. Ohjelma on edullinen ja helppo käyttää ja se tukee monia prosesseja parantavia käytäntöjä.

ACM-luokat (ACM Computing Classification System, 1998 version): D.2.2, D.2.9

Avainsanat: ohjelmistoprosessien parantaminen, CASE, CASE-välineet, vaatimusmäärittely, vaatimustenhallinnan välineet, Sfrm

Sisältö

1 Johdanto

Erilaisilla ohjelmistoilla ja tietokoneistetuilla järjestelmillä on merkittävä rooli kaikkialla nyky-yhteiskunnassa. Olemme yhä enemmän riippuvia näiden järjestelmien tarjoamista palveluista. Samalla järjestelmissä käytetyt ohjelmistot tulevat yhä laajemmiksi ja niiden rakenteellinen monimutkaisuus kasvaa. Tämä kehitys asettaa haasteen ohjelmistoja valmistaville organisaatioille. Niiden on pystyttävä selviytymään tiukkojen kustannus- ja aikataulutavoitteiden sisällä yhä suurempien ja monimutkaisempien ohjelmistoprojektien toteutuksesta. Ongelma on yleismaailmallinen ja sen seurauksena on toteutettu monia kansallisia ja kansainvälisiä kehitys- ja tutkimushankkeita. Niiden tavoitteena on kehittää erilaisia menetelmiä helpottamaan ja tehostamaan projektien toteutusta sekä kehittää standardeja yhtenäistämään käytettyjä menettelytapoja.

Yhtenä keinona ohjelmistotuotannon kehittämisessä ja laadun parantamisessa on ohjelmistoprosessin parantaminen (software process improvement, SPI). Ohjelmistoprosessin tutkimus käynnistyi 1980-luvulla ja siitä kiinnostuttiin laajemmin 1990-luvulle siirryttäessä. Perustana on olettamus, että prosessin laadulla ja kehitetyn ohjelmiston välillä on suora riippuvuussuhde (Fuggetta 2000). Ohjelmistoprosessin parantaminen voidaan jakaa useisiin toisiaan tukeviin menetelmiin. Kuvaja et al. (1994) esittävät jaon neljään menetelmään, jotka ovat mallintaminen, arviointi, mittaaminen ja teknologian siirto. Ohjelmistoprosessin mallintamisen avulla ohjelmiston kehitystyö esitetään vaiheittaisena. Mallissa kuvataan eri vaiheiden toteuttamisen periaatteet. Ohjelmistoprosessin arvioinnin tavoitteena on selvittää organisaation nykyisen kehitysprosessin heikkouksia ja vahvuuksia. Arviointituloksia voidaan käyttää prosessien kehittämisen ja kehitystoimien toteutusjärjestyksen ohjaukseen. Prosessien mittaamisen avulla kerätään tietoa prosessista ja sen tuottamien tuotteiden ominaisuuksista. Mittaamisen tavoitteena on valvoa prosessia ja asetettujen tavoitteiden toteutumista sekä mittaustuloksia analysoimalla etsiä prosessien parantamiskohteita. Teknologian siirrolla tarkoitetaan ohjelmistotuotantoa tukevan tekniikan tai välineen siirtämistä organisaation yleiseen käyttöön auttamaan ja automatisoimaan prosessien suorittamista.

Ohjelmistotuotannon tarkastelu prosessina on auttanut tunnistamaan ohjelmistokehityksen erilaisia ulottuvuuksia ja ongelmia, jotka on huomioitava tehokkaiden käytäntöjen muodostamisessa. Ohjelmistotuotannossa on huomioitava organisatoristen, kulttuurillisten, teknologisten ja taloudellisten tekijöiden monimutkaiset suhteet sekä valittava järkevä elinkaaristrategia ja otettava käyttöön tehokkaat välineet ja ympäristö

helpottamaan ohjelmistoprosessin suorittamista (Fuggetta 2000). Kokonaisuutena ohjelmistotuotanto on kuitenkin ihmiskeskeinen prosessi, joka tuo mukanaan myös rajoitteita (Humphrey 1989). Ihmisen älyllinen kyvykkyys on rajallinen. Vaikka kyvykkyys kasvaakin asteittain ajan kuluessa, on merkittävien parannusten aikaansaamiseksi ohjelmistoprosessia kehitettävä teknologian avulla. Automatisoinnin avulla prosessien suorittamisesta tulee rutiininomainen, mekanisoitu tehtävä, joka vähentää ihmisten rutiinityön määrää ja samalla vähenee myös inhimillisten virheiden mahdollisuus. Välineiden avulla automatisoitua ohjelmistotuotantoa kutsutaan tietokoneavusteiseksi ohjelmistotuotannoksi (computer-aided software engineering, CASE) ja käytettyä teknologiaa CASE-teknologiaksi.

Tämän tutkielman tarkoituksena on kirjallisuuteen perustuen selvittää, miten CASE-teknologian avulla voidaan vaikuttaa ohjelmistoprosessien parantamiseen ja mitä muita edellytyksiä välineiden tehokas käyttö vaatii organisaatiolta. Tarkastelussa selvitetään lähemmin vaatimusmäärittelyprosessien parantamisen erityispiirteitä ja arvioidaan tämän perusteella Joensuun Yliopiston Tietojenkäsittelytieteen laitoksella kehitetyn Sfrm-ohjelman (Kähkönen 2002) ominaisuuksia ja soveltuvuutta vaatimusmäärittelyprosessien parantamisessa.

Luvun 2 alussa esitetään tutkielmassa esiintyviä keskeisimpiä käsitteitä. Loppuosa luvusta käsittelee välineiden asemaa ja merkitystä ohjelmistotuotannon kokonaisuudessa sekä esitetään välineiden ryhmittely, joka perustuu niiden käyttöön erilaisissa prosessitoiminnoissa. Luvun lopussa esitetään lyhyesti myös muita, eri perustein tehtyjä välineiden ryhmittelyjä. Luvun 3 alussa käsitellään välineiden käyttöönottamista sekä edellytyksiä niiden tehokkaalle käytölle organisaatiossa. Luvussa käsitellään välineiden käytön etuja ohjelmistoprosessien parantamisen kannalta sekä rajoitteita ja haittavaikutuksia, jotka myös on huomioitava. Luvun lopussa luodaan katsaus välineiden tulevaisuuden kehitystarpeisiin. Luvussa 4 käsitellään lähemmin vaatimusmäärittelyprosesseja, prosessien parantamista ja käytettyjen välineiden ominaisuuksia. Luvun lopussa tarkastellaan Sfrm-ohjelmaa ja sen ominaisuuksia vaatimusmäärittelyprosessien parantamisen näkökulmasta. Lopuksi luvussa 5 esitetään tutkielman yhteenveto.

2 CASE-välineet ohjelmistotuotannossa

Ohjelmistotuotannon tehtävänä on tuottaa asiakkaalle sen tarpeiden mukainen tietokoneohjelmisto tehokkaasti ja sovitun aikataulun mukaisesti. Ohjelmistotuotanto koostuu useista monimutkaisista toisiaan seuraavista vaiheista. Näiden osatehtävien toteuttamisessa voidaan käyttää apuna erilaisia tietokoneohjelmia ja ohjelmistoja, jotka on kehitetty juuri näitä erikoistehtäviä varten. Tässä luvussa tutustutaan erilaisiin välineisiin ja niiden käyttötapoihin. Ensimmäiseksi käydään läpi ohjelmistotuotantoon ja välineisiin liittyviä peruskäsitteitä, jotta tutkielmassa myöhemmin esitettävät asiat olisi helpompi ymmärtää. Kohdassa 2.2 tarkennetaan ohjelmistotuotannon kokonaisuutta tasomallin avulla sekä esitetään viitekehys, jossa tuotanto jaetaan eri työvaiheista muodostuviksi osaprosesseiksi. Seuraavaksi kohdassa 2.3 esitetään näihin osaprosesseihin perustuva CASE-välineiden luokittelu ja tutustutaan niiden tyypillisimpiin käyttötarkoituksiin. Luvun lopuksi käsitellään lyhyesti muita, eri perustein esitettyjä CASE-välineiden ryhmityksiä.

2.1 Käsitteitä

CASE (computer-aided software engineering) tarkoittaa tietokoneavusteista ohjelmistotuotantoa. Termistä esiintyy kirjallisuudessa useita rinnakkaisia määrittelyjä. Sodhin (1991) mukaan CASE käsittää joukon automatisoituja välineitä ja menetelmiä, joita käytetään apuna ohjelmistotuotannon eri kehitysvaiheissa. Forten ja McCulleyn (1991) mukaan CASE on automatisoiduilla välineillä laajennettua ohjelmistotuotantoa. Sommervillen (2000) määritelmän mukaan CASE on nimitys ohjelmistotuotannon prosesseja tukeville tietokoneohjelmistoille.

CASE-väline (CASE tool) on tietokoneohjelma, jota voidaan käyttää apuna suoritettaessa tiettyä, ohjelmistotuotannon prosessiin kuuluvaa tehtävää (Fuggetta 1993).

CASE-teknologia (CASE technology) on yleisnimitys kaikille välineille, joita käytetään tuotanto- ja metaprosessien tukena, prosessien ja tietokannan hallintaan sekä käyttöjärjestelmän palveluihin (Fuggetta 1993).

Käytäntö (practice) on ohjelmistotuotannon tai hallinnan toimenpide, joka edistää prosessin kohteena olevan työn tai tuotoksen valmistamista tai parantaa prosessin kyvykkyyttä (ISO/IEC 15504-9 1998).

Menetelmä (method) on rakenteinen lähestyminen ohjelmistokehitykseen, joka sisältää kuvaukset käytettävistä graafisista malleista, esitystavoista, säännöt mallien rajoitteista, suositukset soveltuvista suunnittelukäytännöistä sekä ohjeet prosessin suorittamisesta (Sommerville 2000).

Metaprosessi (metaprocess) on joukko toimintoja, menettelytapoja, tehtäviä ja tietokoneavusteisia apukeinoja, joita käytetään tuotantoprosessin luomiseksi, ylläpitämiseksi ja edelleen parantamiseksi (Fuggetta 1993).

Ohjelmisto (software) on joukko tietokoneohjelmia, menettelytapoja ja mahdollisesti liittyviä ohjeita sekä tietokonejärjestelmän toimintaan liittyvää tietoa (IEEE 1990).

Ohjelmistoprosessi (software process) on joukko toimintoja, menetelmiä, käytäntöjä ja transformaatioita joita ihmiset käyttävät ohjelmistojen ja niihin liittyvien tuotteiden, (esimerkiksi projektisuunnitelmien, suunnitteludokumenttien, ohjelmakoodien, testitapausten ja käyttöohjeiden) kehittämisessä ja ylläpitämisessä (Paulk et. al 1993). Toiminnot ja niiden sisältö vaihtelee riippuen organisaatiosta ja kehitettävästä järjestelmästä. Ohjelmistoprosessi on määriteltävä selkeästi, jotta sitä voidaan hallita (Sommerville 2000).

Ohjelmistotuotanto (software engineering) systemaattisten, kurinalaisten ja määrällisten työtapojen käyttäminen ohjelmiston kehittämiseksi, toimittamiseksi ja ylläpitämiseksi (IEEE 1990).

Prosessimalli (process model) on ohjelmistoprosessin yksinkertaistettu kuvaus, joka voidaan esittää useasta näkökulmasta (Sommerville 2000). Esimerkkejä ohjelmistoprosessin näkökulmista ovat työn kulku, tiedon kulku sekä toiminnot ja niiden toteuttajat. Ne esitetään usein graafisina kaavioina. Prosessien kuvaamiseen on kehitetty useita *prosessin mallinnuskieliä* (process modeling language, PML).

Prosessin parantaminen (process improvement) on organisaation prosessien muuttamista siten, että ne vastaavat paremmin organisaation liiketoimintatarpeita ja saavuttavat sen liiketoimintatavoitteet tehokkaammin (ISO/IEC 15504-9 1998).

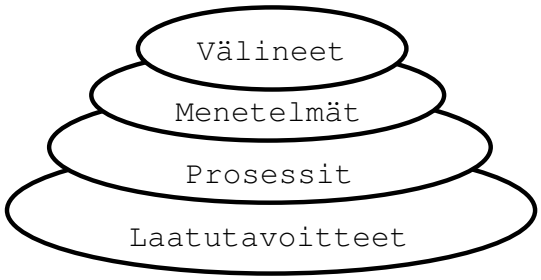
Vaatimusmäärittely (requirements engineering) on ohjelmistotuotannon alue, joka keskittyy ohjelmistojärjestelmien reaalimaailman tavoitteisiin, toiminnallisuuksiin sekä rajoitteisiin. Se sisältää myös näiden tekijöiden liittymisen ohjelmistojen tarkkoihin käyttäytymisen määritelmiin ja niiden kehittymisen ajan kuluessa sekä eri ohjelmisto-

perheissä (Zave 1997).

Vaatimustenhallinta (requirements management) on systemaattinen lähestymistapa järjestelmän vaatimusten keräämiseen, järjestämiseen ja dokumentointiin sekä prosessi, joka luo ja ylläpitää asiakkaan ja projektitiimin välistä sopimusta järjestelmän muuttuviin vaatimuksiin liittyen (Leffingwell ja Widrig 1999).

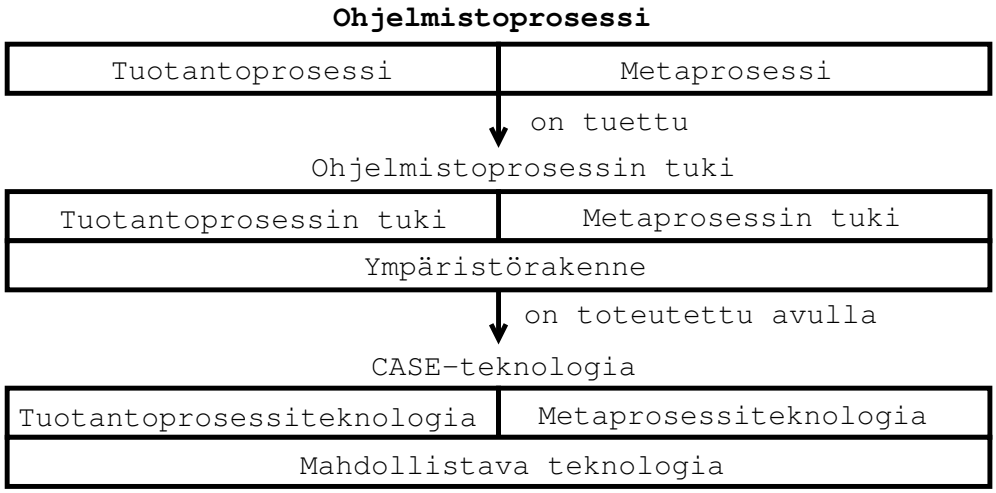
2.2 Ohjelmistoprosessin kuvaus

Pressman (2000) esittää ohjelmistotuotannon eri osa-alueet tasomaisena rakenteena. Esityksen mukaan ohjelmistotuotannon menestymisen perustana on laatuajattelu ja siihen sitoutuminen yrityksen kaikilla toiminta-alueilla (kuva 1). Tämän tavoitteen saavuttamiseksi tarvitaan *prosessitasoa*. Jotta ohjelmistotuotteen valmistamiseksi tarvittavan kokonaisuuden kaikki eri osatehtävät ja niiden erikoispiirteet voidaan ymmärtää, toteuttaa ja hallita, jaetaan prosessitaso osatehtäviin eli prosesseihin. Prosesseissa kuvataan osatehtävien tuloksena valmistuvien tuotteiden lisäksi käytettävät tekniset menetelmät, välitavoitteet, laadun varmistus sekä muutoksen hallinta. Näin prosessit muodostavat perustan projektin hallinnalle ja mahdollistavat ohjelmistotuotteen taloudellisen ja aikataulun mukaisen toteutuksen. *Menetelmätaso* tukee prosessitason toimintoja. Kun prosessitasolla määritellään se, mitä kunkin osatehtävän yhteydessä tehdään, määritellään menetelmätasolla se, miten nämä prosessit toteutetaan. Menetelmät määritellään valmistettavan ohjelmistotuotteen ominaisuuksien ja organisaation toimintaperiaatteiden mukaan valittujen perusteiden mukaisesti. Ylimpänä tasona on *välineet*, jotka tarjoavat automaattisen tai puoliautomaattisen tuen alemmilla tasoilla kuvattujen prosessien ja menetelmien toteuttamiseksi. Kun nämä välineet yhdistetään siten, että ne voivat hyödyntää toistensa tuottamaa tietoa, kutsutaan näin muodostuvaa järjestelmää Pressmanin (2000) mukaan *tietokoneavusteiseksi ohjelmistotuotannoksi*.



Kuva 1: Ohjelmistotuotannon tasot (Pressman 2000).

Edellisestä hieman poikkeava kuvaus ohjelmistotuotannon sisällöstä on Conradi et al. (1992) esittämä ohjelmistoprosessin viitekehys. Sen sisältöä on selvitetty kuvassa 2. Viitekehyksessä ohjelmistoprosessi jaetaan kahdeksi aliprosessiksi, *tuotantoprosessiksi* (production-process) ja *metaprosessiksi* (metaprocess). Tuotantoprosessi käsittää kaikki toiminnot, säännöt, menettelytavat, organisaatiorakenteet ja käytetyt välineet ohjelmistotuotteen suunnittelemiseksi, kehittämiseksi, toimittamiseksi ja ylläpitämiseksi. Tuotantoprosessi voidaan määritellä ja sitä voidaan arvioida ja kehittää systemaattisen ja jatkuvan metaprosessin avulla. Metaprosessin tarkoitus on hankkia ja käyttää hyväksi uutta teknologiaa, joka tukee tuotantotoimintoja auttamalla kehittämään ja parantamaan toimintatapoja, sääntöjä ja tuotannossa käytettyä tekniikkaa. Prosessien parantamista ja erityisesti välineiden vaikutusta tähän tarkastellaan lähemmin luvussa 3.



Kuva 2: Ohjelmistoprosessin yleiskuvaus (Conradi et al. 1992).

Tuotantoprosessia voidaan tukea ja osittain automatisoida *tuotantoprosessiteknologian* (production-process technology) avulla, joka auttaa ohjelmistokehittäjiä määrittelemään, toteuttamaan ja ylläpitämään ohjelmistotuotteita. Yrityksen käytössä olevaa tekniikkaa, siihen liittyviä toimintoja ja ohjeita kutsutaan *tuotantoprosessien tueksi* (production-process support). Vastaavalla tavalla metaprosesseja voidaan tukea ja automatisoida *metaprosessiteknologian* (metaprocess technology) avulla. Käytössä olevat metaprosessitoimintoja automatisoivat ja avustavat apuvälineet muodostavat *metaprosessien tuen* (metaprocess support).

Tuotanto- ja metaprosessien tuet perustuvat yhteiseen *ympäristörakenteeseen* (infrastructure), joka tarjoaa palveluita ja toimintoja yhdistetyssä ja yhtenäisessä ympäris-

tössä. Tuotanto- ja metaprosessien tuki sekä ympäristörakenne yhdessä muodostavat *ohjelmistoprosessin tuen* (software-process support). Ympäristörakenne voidaan toteuttaa käyttämällä käyttöjärjestelmän palveluita sekä prosessien tarkkailuun ja tietokannan hallintaan kehitettyjä välineitä. Ympäristörakenteen toteutusta tukevia välineitä kutsutaan yhteisesti nimellä *mahdollistava teknologia* (enabling technology). Tuotantoprosessi-, metaprosessi- ja mahdollistavan teknologian välineet muodostavat yhdessä *CASE-teknologian*.

2.3 Välineiden luokittelu ohjelmistoprosessien mukaan

Edellisessä kohdassa esitettyyn ohjelmistoprosessin viitekehykseen perustuen on Fuggetta (1993) esittänyt ohjelmistoprosesseja tukevien CASE-välineiden luokituksen. Tuotantoprosessi voidaan nähdä joukkona yksittäisiä *tehtäviä* (task), jotka on suoritettava ohjelmistotuotteen valmistamiseksi. Tehtävät voidaan ryhmitellä isommiksi kokonaisuuksiksi muodostamalla *toimintoja* (actvity), jotka puolestaan voidaan nähdä ohjelmistotuotantoprosessin osina. Toimintoja ei välttämättä suoriteta tietyssä järjestyksessä, eivätkä ne myöskään kuvaa ohjelmistotuotannon prosessimallina yleisesti käytetyn *vesiputousmallin* vaiheita. Edelliseen pohjautuen Fuggetta (1993) tarkentaa tuotantoprosessia tukevien välineiden luokitusta:

- *Väline* (tool) tukee ainoastaan tiettyä yksittäistä tehtävää ohjelmistoprosessissa.
- *Toimintoalustat* (workbench) tukevat ainoastaan yhtä tai vain muutamia tehtäviä ohjelmistoprosessissa.
- *Ympäristöt* (environment) tukevat laajaa osaa tai koko ohjelmistoprosessia.

Toimintoalustat ja ympäristöt on yleensä rakennettu useasta välineestä, joten väline voi olla sekä itsenäisesti toimiva ohjelma että osa toimintoalustaa tai ympäristöä. Tuotantoprosessien tuki kokonaisuudessaan voidaan muodostaa yhdistämällä yksi tai useampia välineitä, toimintoalustoja tai ympäristöjä. Seuraavissa alakohdissa käsitellään tarkemmin näihin ryhmiin kuuluvia välineitä sekä niiden tyypillisiä piirteitä ja käyttötarkoituksia. Kirjallisuudessa käytetty ja myös välinevalmistajien käyttämä terminologia ei ole vakiintunut, joten se voi aiheuttaa sekaannusta (Albizuri-Romero 2000). Selvyyden vuoksi käytetään tässä kohdassa ja muuallakin tutkielmassa soveltuvin osin Fuggettan (1993) esittämää terminologiaa.

2.3.1 Välineet

Editorit (editing tool) jakaantuvat tekstieditoreihin ja graafisiin editoreihin. Perinteisiä tekstieditoreita ja tekstinkäsittelyohjelmia käytetään tekstimuotoisten dokumenttien tuottamiseen. Graafisia editoreita käytetään graafisten määrittelyjen ja kaavioiden tuottamiseen sekä käyttöliittymien suunnitteluun. Näiden ohjelmien toiminta perustuu graafisiin symboleihin, jotka toteuttavat niihin valmiiksi määritellyt toiminnot.

Ohjelmointivälineitä (programming tool) käytetään sekä koodin tuottamiseen, että koodin muodostamiseen valmiista ohjelmista. Tähän ryhmään kuuluvat perinteiset ohjelmien kääntämiseen, suorittamiseen ja virheiden tarkastukseen tarkoitetut välineet. *Koodigeneraattoreiden* avulla voidaan tuottaa koodia automaattisesti. Ne käyttävät lähdeaineistona tuotettavasta ohjelmistosta tehtyä määrittelyä. Sen on oltava välinekohtaisesti tietyn muotoinen, jotta väline pystyy sen tulkitsemaan. Kolmannen alaryhmän muodostavat välineet, joiden avulla voidaan analysoida ja muokata valmiiden ohjelmien toimintaa sekä parantaa olemassa olevaa lähdekoodia. Välineiden avulla suoritetun analyysin perusteella koodin rakennetta voidaan muokata parantamalla alkuperäisessä koodissa tehottomasti toteutettuja osia.

Todentamisen ja kelpoistamisen välineiden (verification and validation tool) avulla tarkkaillaan sitä, että tuotettava ohjelmisto on toiminnaltaan sille ennakkoon asetettujen vaatimusten mukainen. Tähän ryhmään kuuluvat erityisesti testauksen välineet. Termillä *todentaminen* (verification) tarkoitetaan varmistumista siitä, että ohjelmisto on toteutettu vaatimusmäärittelyn mukaisesti ja *kelpoistaminen* (validation) tarkoittaa sitä, että ohjelmisto on asiakkaan tarpeiden mukainen (IEEE 1998). Ohjelman *analysointivälineillä* (analyzer) testataan ohjelman toimintaa. Staattinen analysointi tehdään tarkastelemalla koodia suorittamatta ohjelmaa ja dynaamisessa analyysissä tehdään havaintoja ohjelman suorituksen aikana. *Vertailuvälineillä* (comparator) tutkitaan kahden tiedoston eroja. Tyypillinen tällainen käyttötarkoitus on testitulosten vertaaminen odotettuihin tuloksiin. *Emulaattorit* ja *simulaattorit* jäljittelevät ohjelmiston tai sen osan toimintaa. Ne hyväksyvät samat syötteet sekä niiden toiminnallisuus ja tulosteet ovat vastaavat kuin verrattavalla ohjelmistolla. Näin voidaan testata ohjelmiston osien toimintaa ennen kuin kaikki sen osat ovat valmiita. *Testitapausgeneraattorit* (test-case generator) tuottavat automaattisesti testitapauksia. Lähtötietoina annetaan testattava ohjelma ja kokoelma ehtoja, joiden tulee toteutua. *Testauksenhallinnan* (test-management tool) välineillä ylläpidetään esimerkiksi testauksen tuloksia, tarkastuslistoja, taantuma-

testauksia ja selostuksia testauksen aikaisista mittauksista.

Konfiguraationhallinnan välineitä (configuration management tool) käytetään ohjaamaan ja valvomaan pienemmistä osista koostuvien ohjelmistojen rakennetta. Niiden avulla ylläpidetään esimerkiksi tieto siitä, mitä osia kuuluu tiettyyn ohjelmaversioon ja mikä on kunkin osan sisäinen versio. Fuggetta jakaa ryhmän välineiden suorittamien tehtävien mukaan seuraavaksi esitettäviin alaluokkiin, mutta toteaa samalla, että uudemmissa välineissä suurin osa näiden alaluokkien tehtävistä on yhdistetty yhteen välineeseen. Toimivalla *versionhallinnalla* (version management) on suuri merkitys sekä ohjelmiston kehitysvaiheessa että ylläpidossa. Jokaisesta ohjelman osasta valmistuu kehityksen aikana useita versioita ja myöhemmin muutosten ja ylläpidon yhteydessä on tärkeää tietää, että tarvittavat muutokset pystytään tekemään oikeaan versioon. Ohjelmiston *osien yksilöinti* (item identification) on tärkeää. Jokaisen osan on oltava yksilöllisesti tunnistettavissa. Kaikkien projektiin osallistuvien on pystyttävä muodostamaan kehitettävän ohjelmiston osista yhtenäinen työversio. Samoin ohjelmistosta on pystyttävä myös jälkeenpäin muodostamaan mikä tahansa sen kehitysversioista. *Konfiguraation rakentamisella* (configuration building) hallitaan koko ohjelmistotuotteen ja siihen kuuluvien osien versiotietoja. Jokainen ohjelmisto ja sen eri versiot koostuvat useista osista, joiden keskinäiset versiot voivat vaihdella. Ohjelmistoversion tuottaminen vaatii koodin muokkaamisen lisäksi myös monia muita tehtäviä, kuten koodin kääntämisen ja eri osien linkityksen. Tämän vuoksi jokaisen version yhteydessä on tiedettävä myös välineet ja niiden versiot, joilla nämä tehtävät on suoritettu. Esimerkiksi ohjelmiston ylläpitovaiheessa on tiedettävä tarkasti, mikä ohjelmistoversio kullekin asiakkaalle on toimitettu, jotta tarvittavat muutokset pystytään tekemään oikeisiin osiin. *Muutoksenvalvonnan* (change control) avulla valvotaan versionhallinnassa mukana olevien ohjelmiston osien yhtenäisyyttä useamman henkilön työskennellessä samanaikaisesti. Muutoksenvalvonnan avulla valvotaan ja estetään tietyn osan samanaikainen muokkaaminen ja tehtyjen muutosten tallentaminen tietovarastoon siten, että osaa tehdyistä muutoksista ei kadoteta. Versionhallinnasta muokattavaksi otettu elementti merkitään, jolloin muut käyttäjät voivat saada käyttöönsä vain kopion. Näitä kopioita ei voi tallentaa ennen kuin ensimmäisenä elementin käyttöönsä ottanut käyttäjä on tallentanut oman muuttamansa version takaisin versionhallintaan. Edellä esitetyt toiminnot vaativat toimiakseen *kirjaston hallintaa* (library managemet). Kaikki ohjelmistoon kuuluvat osat on tallennettava järjestelmällisesti ja toisaalta niiden hakeminen tietovarastosta myöhemmin on oltava hallittua. Versionhallinnan tulee käsittää ohjelmakoodin lisäksi myös dokumentit, kuten määrittelyt ja käyttöohjeet sekä suun-

nittelu-, testaus- ja muut projektiin liittyvät asiakirjat.

Mittausvälineet (metrics and measurement tool) on tärkeä välineryhmä ohjelmistotuotannossa. Mittaustietojen avulla voidaan valvoa prosessien suorittamista ja se on tärkeä apu johdon päätöksenteossa koko ohjelmistotuotannon elinkaaren aikana. Myös ohjelmistoprosessien parantamisessa mittaustiedoilla ja niiden analysoinnilla on keskeinen asema. Mittaustoiminnot voidaan kohdistaa tuotteeseen, prosessiin ja resursseihin (Fenton ja Neil 2000). Fuggetta käsittelee artikkelissaan ainoastaan tuotteen mittaamista ja jakaa mittausvälineet kahteen alaryhmään, lähdekoodia analysoiviin ja ohjelman suoritusta tarkkaileviin välineisiin. Koodianalyysin pohjalta saadaan tietoa esimerkiksi ohjelman rakenteesta ja ajonaikaisen mittauksen pohjalta saadaan tietoa esimerkiksi suoritusajoista ohjelmiston erilaisilla kuormitusasteilla.

Projektinhallinnan välineitä (project management tool) käytetään erilaisten koko ohjelmistoprojektin toteuttamiseen liittyvien tehtävien suorittamiseen. Tällaisia tehtäviä ovat kustannusarvioiden laadinta, projektisuunnittelu ja yhteydenpito projektiin osallistuvien kesken. Projektisuunnittelu käsittää aikataulujen laadintaa, tarvittavien resurssien arviointia ja projektin seurantaa.

Muut välineet on viimeinen Fuggettan (1993) nimeämä välineiden ryhmä. Tähän ryhmään voidaan nimetä välineitä, jotka eivät kuulu erityisesti mihinkään edellä esitetyistä ryhmistä, mutta joita voidaan käyttää avustavina välineinä eri tehtävissä.

2.3.2 Toimintoalustat

Toimintoalustassa yhdistetään useita yksittäisiä välineitä yhdessä toimivaksi ohjelmistoksi, jolla voidaan suorittaa tiettyjen ohjelmistoprosessien toimintoja. Yhdistämisen avulla saavutetaan yhtenäinen käyttöliittymän esitystapa, välineiden hallinta ja pääsy yhteiseen, keskitetysti hallittuun tietovarastoon. Seuraavassa käsitellään lähemmin Fuggettan (1993) esittämää toimintoalustojen ryhmittelyä ja niihin sisältyviä tyypillisiä välineitä.

Liiketoiminnan suunnittelun ja mallinnuksen (business planning and modeling) toimintoalustoja käytetään monimutkaisten liiketoimintamallien suunnitteluun ja määrittelyyn. Niiden avulla määritetään yleisiä vaatimuksia ja tietovirtoja pyrkien selvittämään yritykselle toteutettavan tietojärjestelmän kehityskohteiden tärkeysjärjestys. Toi-

mintoalustoihin liitettyjä välineitä ovat tyypillisesti graafiset editorit sekä raportti- ja ristiviittausgeneraattorit.

Analysointi ja suunnittelu (analysis and design) muodostavat tärkeän toimintoalustojen ryhmän. Kirjallisuudessa tämän ryhmän välineistä on käytetty yleisesti myös nimitystä *upper-CASE* kuvaamaan ohjelmistoprosessin alkuvaiheessa käytettäviä välineitä (Fuggetta 1993). Toimintoalustat sisältävät tavallisesti yhden tai useampia editoreita, joiden avulla laaditaan ja muokataan määrittelyjä. Lisäksi on välineitä määrittelyjen analysointiin, simulointiin ja muuntamiseen. Toimintoalustojen tarjoama toiminnallisuus ja automaation aste riippuu niiden esitystavan muodollisuusasteesta. Jos esitystapaa ei ole määritelty muodollisesti, voi toimintoalusta tarjota ainoastaan valmiudet editointiin ja dokumenttien tuottamiseen. Muodollisuusasteen kasvaessa on tarjolla enemmän automatisoituja toimintoja, esimerkiksi erilaisten kaavioiden tuottamiseen.

Käyttöliittymän suunnittelun (user interface development) toimintoalustat eroavat muista esitetyistä ryhmistä siten, että ne eivät palvele ainoastaan tiettyä ohjelmistoprosessin tehtävää, vaan niitä käytetään käyttöliittymän suunnittelemiseen ja toteuttamiseen. Käyttöliittymän ulkoasu luodaan graafisen editorin avulla maalaamalla ja sijoittamalla ikkunoita, dialogeja ja painikkeita kuvaavia symboleja. Käyttöliittymän toimintaa voidaan testata simulaattorin avulla ennen sen liittämistä osaksi muuta toteutettavaa ohjelmaa. Koodigeneraattorin avulla tuotetaan käyttöliittymän toiminnoista vastaava koodi käyttäen hyväksi ajonaikaisia kirjastofunktioita.

Ohjelmoinnin toimintoalustat (programming workbenches) on kehitetty perusohjelmointivälineistä yhdistämällä ne yhteiseen käyttöliittymään. Tyypillisiä välineitä ovat tekstieditorit lähdekoodin luomiseen ja muokkaukseen, kääntäjät ja linkittäjät suoritettavan ohjelmakoodin tuottamiseen sekä virheiden tarkastuksen välineet. Toimintoalustat ylläpitävät kaikkia työn aikana tuottamiansa tietoja.

Todentamisen ja kelpoistamisen (verification and validation) toimintoalustoja käytetään lähinnä moduuli- ja systeemitestaukseen. Toimintoalustaan yhdistetään usein mittaus- ja testausvälineitä. Yhdistämällä ryhmien toiminnallisuudet voidaan analysoida koodin laatua sekä tukea todentamista ja kelpoistamista. Liitettyjä välineitä ovat tyypillisesti analysointivälineet, ristiviittaus-, raportti- ja testitapausgeneraattorit sekä testauksenhallintavälineet testitietojen, tulosten ja tarkastuslistojen tuottamiseen, tallentamiseen ja hallintaan.

Ylläpito ja *käänteistekniikka* (maintenance and reverse engineering) muodostavat oman toimintoalustojen ryhmän. Ylläpidon tehtävissä käytetään samoja välineitä kuin varsinaisessa ohjelmistokehityksessäkin. Välineitä tarvitaan vaatimusmäärittelyjen muokkaamiseen, suunnitteluun, koodaamiseen ja testaukseen. Myös konfiguraationhallinnan tehtävien suorittaminen on tärkeää ylläpidon yhteydessä. Sen sijaan *käänteistekniikan* (reverse engineering) yhteydessä tarvitaan erityisesti tähän kehitettyjä ohjelmistoja, jossa automatisoidut välineet lukevat lähdekoodia syötteenään ja tulostavat ohjelmasta suunnittelutason tietoa graafisina ja tekstimuotoisina esityksinä (Hoffer et al. 2002). Käänteistekniikan toimintoalustoihin kuuluvia välineitä ovat koodin uudelleen rakentajat, vuokaavion piirto-ohjelmat ja ristiviittausgeneraattorit.

Konfiguraationhallinnan (configuration management) ja *projektinhallinnan* (project management) toimintoalustat yhdistävät edellä esitettyjen vastaavien ryhmien välineitä ja laajentavat niiden toiminnallisuutta.

2.3.3 Ympäristöt

Ympäristö on yksittäisten välineiden ja toimintoalustojen kokoelma, jolla tuetaan ohjelmistoprosessia. Seuraavassa tarkastellaan Fuggettan (1993) esittämää ympäristöjen ryhmittelyä.

Työkalupakit (toolkit) ovat kevyesti yhdistettyjä kokoelmia välineitä ja toimintoalustoja, joita voidaan helposti laajentaa. Työkalupakit eroavat toimintoalustoista siinä, että ne tukevat useiden eri ohjelmistoprosessien toimintoja. Niiden tuki on kuitenkin usein rajoittunut ohjelmointiin sekä konfiguraation- ja projektinhallintaan. Liitetyillä välineillä ei ole yhteistä käyttöliittymää, vaan ne käynnistetään käyttäjän kutsusta tai yksinkertaisten ohjauskäskyjen avulla. Yhteisten tiedostojen käyttö on rajoittunutta ja voi vaatia jopa tiedostojen konvertointia.

Ohjelmointikielikeskeiset ympäristöt (language centered environment) ovat nimensä mukaisesti keskeisesti sidoksissa niiden tukemaan ohjelmointikieleen. Ne ovat usein keskittyneet muokkaus-, käännös- ja virheentarkastustoimintoihin ja niiden tuki muiden ohjelmistoprosessien toiminnoille on hyvin vähäinen. Usein myös ympäristö itse on kirjoitettu samalla kielellä, jota ne tukevat. Tämän ansiosta käyttäjät voivat muokata ja laajentaa ympäristöä. Haittapuolena on se, että eri kielisten koodien yhdistäminen on vaikeaa tai mahdotonta. Ohjelmointikielikeskeisissä ympäristöissä välineiden esi-

tystapa ja hallinta on yhdistetty. Käyttöliittymä on yhtenäinen ja välineitä voidaan kutsua automaattisesti tai kytkettynä eri välineiden kautta. Puutteena on prosessien ja tiedon yhdistäminen. Ohjelmointikielikeskeiset ympäristöt perustuvat yleensä sisäisesti rakenteiseen esitystapaan, joka kuitenkin on käyttäjälle näkymättömissä. Tämän seurauksena ympäristön muokkaaminen ja laajentaminen ulkopuolisilla välineillä on vaikeaa.

Integroidut ympäristöt (integrated environment) muodostuvat välineistä, jotka tietyin rajoituksin toimivat yhteisten sääntöjen mukaisesti. Käyttäjät voivat liittää ympäristöön välineitä ja toimintoalustoja. Välineiden integraatiota voidaan tarkastella neljän eri tekijän perusteella (Thomas ja Nejmeh 1992). *Tietointegraatio* varmistaa, että kaikki ympäristön tieto hallitaan yhtenäisenä kokonaisuutena. *Hallintaintegraatio* sallii ympäristön joustavan toiminnan, *esitystapaintegraatio* parantaa käyttäjän vuorovaikutusta ja *prosessi-integraatio* varmistaa välineiden tehokkaan yhteistoiminnan määritellyn prosessin tukemisessa. Integroiduissa ympäristöissä esitystapaintegraatio on saatu aikaan yhtenäisen käyttöliittymän avulla. Tietointegraatio on toteutettu yhteisen, keskitetysti hallitun tietokannan avulla. Tietokanta on rakenteinen, joten se sallii tiedon saannin ja vaihdon myös muille vastaavan määrittelyn mukaisesti toimiville välineille. Hallintaintegraatio on varmistettu toiminnoilla, jotka mahdollistavat välineiden keskinäisen kutsun ympäristön muista välineistä. Integroidut ympäristöt eivät erityisesti tue prosessieheyttä. Tämä erottaa ne *prosessikeskeisistä ympäristöistä*. Integroitujen ympäristöjen toteuttaminen vaatii huomattavasti pidemmälle kehitettyä perusrakennetta kuin perinteiset käyttöjärjestelmän palvelut.

Sovelluskehitysympäristöt (fourth-generation environment) ovat integroitujen ympäristöjen edeltäjiä ja ne ovat oikeastaan sen alaluokka. Ne on tarkoitettu erityisesti laajojen liiketoimintaan suuntautuneiden tietokantaohjelmistojen kehittämiseen. Näille ohjelmistoille on tyypillistä, että operaatiot ovat yksinkertaisia, vaikka käsiteltävä tieto on rakenteeltaan monimutkaista. Kehitettävien ohjelmistojen käyttöliittymä on kriittinen. Se muodostuu useista lomakkeista, joita käytetään tietokannassa säilytettävän tiedon syöttämiseen, esittämiseen ja muokkaamiseen. Vaatimusmäärittelyt voivat olla vajaavaisia ja kehitystyötä tehdään prototyyppien avulla. Ohjelmistot valmistuvat yleensä useiden kehitysvaiheiden kautta. Sovelluskehitysympäristöjen ryhmään tyypillisesti sisältyviä välineitä ovat editorit, tulkit, kääntäjät, koodigeneraattorit, virheen tarkastajat, käyttöliittymäeditorit, simulaattorit, yksinkertaiset konfiguraationhallinnan välineet sekä välineet tietokannan ja dokumenttien käsittelyyn. Ne toimivat täysin yh-

tenäisen käyttöliittymän kautta ja tiedot tallennetaan kaikille välineille yhteiseen tietovarastoon. Välineiden käynnistys tapahtuu ympäristön sisältämien herättimien avulla tiettyjen tapahtumien yhteydessä.

Prosessikeskeiset ympäristöt (process centered environment) perustuvat ohjelmistoprosessin muodolliseen määrittelyyn. Tätä määrittelyä kutsutaan *prosessimalliksi* (process model) ja sitä tulkitaan ohjelmallisesti. Prosessimalli opastaa kehitystyötä automatisoimalla osaprosesseja, kutsumalla automaattisesti tarvittavia välineitä, pakottamalla tiettyihin toimintatapoihin sekä avustamalla projektihenkilöstön työskentelyä. Prosessimallien laatimisessa käytetään hyväksi välineitä, joilla on toiminnallisuudet mallien määrittelyyn, analysointiin ja hallintaan. Prosessikeskeiset ympäristöt koostuvat usein osista ja käsittävät kaksi toimintoa, *prosessimallin suorituksen* (process-model execution) ja *prosessimallin tuotannon* (process-model production). Prosessiajuri tulkitsee ja suorittaa prosessimallia ja toimii siten prosessikeskeisenä ympäristönä ohjelmistosuunnittelijoille. Prosessin mallintajat käyttävät välineitä mallien luomiseen ja kehittämiseen. Prosessimallin tuotanto-ominaisuuksien johdosta prosessikeskeiset ympäristöt voidaan sijoittaa myös seuraavana esitettävään metaprosessiteknologian ryhmään.

2.3.4 Metaprosessi- ja mahdollistava teknologia

Metaprosessiteknologia (metaprocess technology) ja *mahdollistava teknologia* (enabling technology) ovat tärkeitä tehokkaiden välineiden kehittämisessä. Metaprosessiteknologia mahdollistaa tuotantoprosessien tuen luomisen, käyttämisen ja parantamisen. Mahdollistava teknologia puolestaan tarjoaa perusmekanismit erilaisten tuotteiden yhdistämiseen tuotanto- ja metaprosessiteknologiassa. Metaprosessi on hyvin samankaltainen tuotantoprosessin kanssa. Metaprosessissa suoritetun työn kohteena ja tuloksena on tuotantoprosessi ja sitä tukevat välineet. Välineiden parantaminen vaatii ohjelmistoprosessien tapaan määrittelyä, suunnittelua, testausta, käyttöä, arviointia ja ylläpitoa. Työssä voidaan käyttää apuna tuotantoprosessiteknologiaa, erityisesti analysointi- ja suunnitteluohjelmistoja. Niiden antama tuki on kuitenkin rajoitettu. Ne auttavat käsittelemään prosessien sääntöjä, menettelytapoja ja dokumentteja sekä arvioimaan olemassa olevia käytäntöjä. Jotta näistä voitaisiin kehittää edistyneempiä, tuotantoprosesseja paremmin tukevia ympäristöjä, tarvitaan lisäksi metaprosessivälineitä prosessimallien arviointiin, testaukseen ja simulointiin sekä edelleen kehittämiseen.

Monimutkaisten CASE-välineiden kehitys vaatii käyttöjärjestelmätason tiedoston- ja

tehtävänhallintapalvelujen lisäksi edistyneempiä, välineiden erityistoimintojen kehittämistä tukevia välineitä. Monet CASE-välineet käyttävät tiedon varastointiin tietokantoja, joiden luomiseen ja ylläpitoon tarvitaan pitkälle kehitettyjä *tietokannanhallintajärjestelmiä* (data-base-management system, DBMS). Graafisten käyttöliittymien suunnitteluun ja kehittämiseen tarvitaan *käyttöliittymän hallintajärjestelmää*. Myös hajautettujen asiakas-palvelin järjestelmien sekä useissa käyttöjärjestelmissä toimivien sovellusten kehittäminen vaativat erityisvälineitä. Lisäksi erityisvälineitä tarvitaan esimerkiksi edistyneitä prosessinvalvontamekanismeja sekä multimediaa ja ryhmätyötä tukevien CASE-välineiden toimintojen kehittämisessä.

2.4 Muita lähestymistapoja CASE-välineiden ryhmittelyyn

Dart et al. (1987) esittivät ensimmäisen CASE-välineiden ryhmittelyn. He keskittyivät ohjelmiston kehitysympäristöjen luokituksen. Luokituksen tarkoituksena on ymmärtää niiden periaatteiden kehitys, joiden mukaan ohjelmiston kehitysympäristö on rakennettu. Luokitus sisältää neljä ryhmää. *Kielikeskeiset ympäristöt* (language-centered environment) perustuvat yhteen ohjelmointikieleen, jota ympäristö tukee. Ne voivat olla monipuolisesti vuorovaikutteisia, mutta tarjoavat kuitenkin rajoitetun tuen ohjelmoinnille laajemmassa ympäristössä. *Rakennekeskeiset ympäristöt* (structure-centered environment) perustuvat ympäristön muokattavuuteen. Ne sallivat ohjelmointikielen kieliopin muokkaamisen, jonka avulla voidaan toteuttaa rakenteeseen perustuvia välineitä. Tästä ovat esimerkkinä syntaksiin perustuvat editorit. *Työkaluympäristöt* (toolkit environment) koostuvat pienistä, suoraan koodausvaihetta tukevista välineistä. Niiden käyttötavan tarkastamiseen ei ole kuitenkaan rakennettu mitään ympäristön sisäistä toimintoa. *Menetelmään perustuvat ympäristöt* (method-based environment) keskittyvät johonkin ohjelmistokehityksen menetelmään, kuten oliosuuntautuneeseen suunnitteluun. Dart et al. eivät kuitenkaan esitä luokituksessaan tarkempaa välineiden jaottelua. Dart et al. luokitus on suppea verrattuna edellä esitettyyn Fuggettan (1993) CASE-välineiden luokitteluun, koska se keskittyy lähinnä toteutusvaiheen välineympäristöihin. Fuggettan luokituksessa lähinnä vastaava ryhmä on sovelluskehitysympäristöt.

Forte ja McCulley (1991) esittävät kaksiosaisen luokituksen, jossa välineet jaetaan ohjelmistotuotannon vaiheiden mukaisesti horisontaalisiin ja vertikaalisiin välineisiin (horizontal and vertical tools). Luokituksen mukaiset ohjelmistokehityksen tehtävät on esitetty kuvassa 3. Vertikaalitason välinettä käytetään tietyn ohjelmistokehitysvaiheen

tai tehtävän suoritukseen. Esimerkkejä tällaisista tehtävistä ovat koodaus ja testaus. Horisontaalitason välineitä käytetään kaikissa vaiheissa koko ohjelmistokehitysprosessin ajan. Tällaisia välineitä ovat esimerkiksi projektinhallinnan ja dokumentoinnin välineet. Luokituksen etuna on erilaisten luokkien runsaus, mutta se ei kuitenkaan huomioi ohjelmistoprosessien käsitteellistä rakennetta. Luokitus perustuu ohjelmistokehityksen vesiputousmalliin ja jako horisontaalisiin ja vertikaalisiin välineisiin on epäselvä muiden mallien yhteydessä.

Määrittely	Analysointi	Suunnittelu	Toteutus	Testaus	Ylläpito
Projektin hallinta					
Prosessin ja laadun hallinta					
Työryhmän ja henkilöstön tuottavuus					
Ohjelmiston kehitysympäristö					
Dokumentointi					

Kuva 3: Ohjelmistokehityksen tehtävät Forten ja McCulleyn (1991) esittämässä luokituksessa.

Pressman (2000) esittää CASE-välineiden toiminnallisuuteen perustuvan luokituksen. Se tukee jossain määrin ohjelmistokehitysprosessien rakennetta, mutta ei ota huomioon metaprosesseja. Luokitus on kattava käsittäen koko ohjelmistokehitysprosessin eri vaiheiden toiminnot. Ryhmitys noudattelee edellä esitettyä yksittäisten välineiden kuvausta, mutta siinä on mukana myös viime vuosina tapahtuneen kehityksen mukanaan tuomia uusia ryhmiä, kuten internetsovellusten kehitysvälineet ja asiakas-palvelin ympäristön testauksen välineet.

Sommervillen (2000) luokitus perustuu prosessien toimintoihin ja on hyvin samankaltainen edellä esitetyn Fuggettan (1993) luokituksen kanssa. Sommerville esittää konfiguraationhallinnan välineet omana ryhmänään, vaikka korostaakin niiden läheistä ja toimivaa suhdetta CASE-välineisiin. Tässä luokituksessa ei ole mukana metaprosessivälineitä.

3 Välineet prosessien parantamisen osana

Ohjelmistoprosessin parantaminen on hyvin monimutkainen tehtävä. Parantamisen tavoitteina voi teknologisten näkökohtien lisäksi olla myös monia muita asioita kuten organisaation hallinto ja yrityskulttuuri sekä henkilöstö (El Emam et al. 1998). Myös parantamisen keinoina voidaan käyttää useita toisiaan tukevia menetelmiä. Teknologian lisäksi voidaan hyödyntää prosessien mallintamista, arviointia ja mittaamista (Kuvaja et al. 1994). Tässä luvussa tarkastellaan ohjelmistoprosessien parantamista teknologian näkökulmasta. Pyrimme selvittämään, kuinka välineiden avulla voidaan parantaa erilaisten prosessien suorittamista ja mitä muita toimenpiteitä organisaatiolta vaaditaan haluttujen tuloksien saavuttamiseksi. Aluksi tarkastellaan normien näkökulmasta välineiden käyttöä erilaisissa prosesseissa sekä prosessien eri kypsyystasoilla. Seuraavaksi kohdassa 3.2 tarkastellaan välineiden käyttöönottoa. Pyritään selvittämään vaadittavia edellytyksiä ja tarvittavia toimenpiteitä käyttöönoton onnistumiseksi. Kohdassa 3.3 esitetään tavoitteita eduista, joita välineiden käytön avulla pyritään saavuttamaan prosessien parantamisessa. Seuraavaksi kohdassa 3.4 tarkastellaan välineiden hankintaan ja käyttöön liittyviä haittavaikutuksia ja ongelmia. Luvun lopuksi kohdassa 3.5 tarkastellaan välineiden käytön tulevia kehityssuuntia.

3.1 Käyttö prosessien eri kypsyystasoilla

Ohjelmistoprosessien kypsyyden arviointiin ja parantamiseen yleisesti käytettyjä viitekehyksiä ovat *CMM* (the capability maturity model for software) (Paulk et al. 1993) ja *ISO/IEC 15504*, joka tunnetaan myös nimellä *SPICE* (software process improvement and capability determination) (El Emam et al. 1998). Viitekehyksissä ei suoraan määritellä sitä, mitä välineitä tietyllä prosessien kypsyystasolla edellytetään käytettävän. Ne kuitenkin antavat ohjeita siitä, millaisia resursseja tietyn prosessin suorittamiseksi olisi varattava. Tässä yhteydessä viitataan myös prosessin suorittamisessa mahdollisesti käytettäviin välineisiin ja annetaan niistä esimerkkejä. Viitekehykset antavat näin organisaatioille mahdollisuuden itse määritellä ja valita omiin tarpeisiinsa sopivat ja suoritettavia prosesseja parhaiten tukevat välineet. Seuraavaksi esitetään koottuna SPICE:ssa (ISO/IEC 15504-2 1998, ISO/IEC 15504-5 1999) esiin tulevia viittauksia välineiden käyttöön eri kypsyystasoilla.

Ei toimivalla ja *toimivalla tasolla* (tasot 0 ja 1, incomplete, performed) normit eivät ota kantaa välineisiin. Pyrittäessä prosessin kypsyydessä toimivalle tasolle, keskitytään tämän yksittäisen prosessin suorittamiseen. Tämän tavoitteen saavuttamiseksi voidaan käyttää yksittäistä, juuri kyseessä olevaan tehtävään soveltuvaa välinettä. Käytetyt välineet voivat olla henkilöiden omien mieltymysten mukaan valitsemia. Prosessin suorittamista voidaan tehostaa valitsemalla projektikohtaisesti väline, joka organisaation ja tehtävän erityispiirteet huomioon ottaen parhaiten täyttää prosessin suorittamiselle asetetut vaatimukset. Kyseeseen voi tulla myös jonkin yleiskäyttöisen välineen sijasta räätälöity tai kokonaan itse kehitetty erikoistyökalu. Organisaatiossa voi täten olla käytössä joukko erilaisia välineitä, joita käytetään tiettyjen prosessien suorittamiseen. Näillä välineillä ei kuitenkaan välttämättä ole yhdistettyjä toimintoja, eivätkä ne mahdollisesti pysty hyödyntämään toistensa tuottamaa tietoa omana syötteenään. Siirryttäessä prosessista toiseen joudutaan tietoa muuntamaan tai syöttämään uudelleen toisen prosessin lähtötiedoiksi. Editorit ja ohjelmointivälineet kuuluvat tyypillisesti tähän ryhmään. Dokumentit laaditaan tekstinkäsittelyohjelmilla, ohjelmakoodi tuotetaan editoreilla ja suoritettava ohjelma saadaan kääntäjien avulla.

Hallitulla tasolla (taso 2, managed) prosessin suoritus on hallittua ja työn lopputulos voidaan tällöin saavuttaa määritellyillä resursseilla laadittujen aikataulujen mukaisesti. Pyrittäessä prosessien kypsyydessä hallitulle tasolle keskeisiä kehitettäviä prosessiominaisuuksia ovat suorittamisen ja työtulosten hallinta. Käytettävät välineet on määritelty projektitasolla ja keskeiset välineryhmät ovat projektinhallinnan ja konfiguraationhallinnan välineet. Pyrittäessä parantamaan työn suorittamista, ovat merkittäviä projektin alkuvaiheessa käytettävät suunnittelua tukevat välineet. Näitä ovat esimerkiksi ohjelmiston koon määrittelyn, työsaavutusten ja tuottavuuden arvioinnin välineet sekä aikataulun laadinnan välineet. Prosessin suorittamisen valvontaa tukevat prosessinhallintavälineet. Niitä käytetään koko projektin keston ajan ja niiden avulla voidaan seurata projektin edistymistä ja suunnitelmien mukaista toteutumista. Tehostuneen seurannan avulla poikkeamat havaitaan nopeammin ja voidaan ryhtyä ajoissa korjaaviin toimenpiteisiin. Konfiguraationhallinnan välineillä parannetaan työtulosten hallintaa. Käytettäville välineille on tärkeää, että ne sisältävät myös version- ja muutostenhallinnan sekä tukevat jäljitettävyyttä. Jos virheitä tai puutteita havaitaan, niiden syyt voidaan jäljittää. Tarvittaessa muutetaan vaatimusmäärittelyä ja korjataan siitä aiheutuvat muutokset myös virhettä edeltäneisiin prosesseihin. Hallitulla tasolla SPICE antaa myös ensimmäiset viitteet prosessien suoritusaikaisen seurantatiedon keräämisestä. Seurantatietojen tallentaminen on järkevintä toteuttaa tietokantapohjaisena. Tällöin suurenkin

tietomäärän analysointi ja tiedon hyödyntäminen myöhempää käyttöä varten voidaan toteuttaa tehokkaammin. Tietokannan käyttöönotto vaatii riittävän tehokkaiden tietokannanhallintavälineiden käyttöä. Hallitulla tasolla seurantatietoina kerätään raportoidut ongelmatilanteet ja niiden korjaamiseksi suoritetut toimenpiteet suorittamisen ja työtulosten hallinnan osalta.

Vakiintuneella tasolla (taso 3, established) prosessin suorittamisessa suurin ero verrattuna hallittuun tasoon on se, että prosessi on suunniteltu ja hallittu standardiprosessia käyttäen. Tällöin yksittäinen prosessi suoritetaan käyttäen hyväksyttyä, standardiprosessista muokattua ja dokumentoitua prosessin määrittelyä. Vakiintuneelle tasolle siirryttäessä kehitettäviä prosessiominaisuuksia ovatkin prosessin määrittely ja resurssointi. Prosessin määrittelyä tukevat prosessin mallintamisvälineet, erilaiset ohjelmiston suoritusta mittaavat välineet sekä konfiguraationhallinnan välineet. Seurantatietoina tallennetaan tietokantaan ohjelmiston mittauksesta saadut tiedot sekä yleinen, kokemusperäinen palaute prosessin suorittamisesta. Lisäksi myös ohjelmistoprosessien määrittelydokumentit tallennetaan omaan kirjastoonsa ja ne liitetään muutoksenhallintaan. Vakiintuneella tasolla korostuvat hallittua tasoa enemmän tietokannan analysointimenetelmät ja -välineet. Resurssoinnin avulla varmistetaan prosessin suorittamisessa tarvittava riittävä ja ammattitaitoinen henkilöstö sekä tarkoituksenmukainen työskentelytila ja tarvittavat välineet. Käytetyt välineet määritellään organisaatiotasolla ja SPICE edellyttää koko organisaation laajuisen ohjelmistotuotantoympäristön määrittelyä ja toteuttamista. Lisäksi edellytetään vakiintuneita vaatimuksia tietojen eheydestä, saatavuudesta ja turvallisuudesta sekä uudelleenkäytön strategiaa. Tämä voidaan toteuttaa tarkoituksenmukaisten välineiden tai yksittäiset välineet integroivan kehitysympäristön avulla.

Ennustettavalla tasolla (taso 4, predictable) prosessin kyvykkyys ymmärretään ja sen suorittamista voidaan ohjata. Tämä saadaan aikaan keräämällä prosessin suorittamisesta yksityiskohtaista mittaustietoa ja analysoimalla sitä. Tämän tavoitteen saavuttamiseksi ennustettavalla tasolla kehitettäviä prosessiominaisuuksia ovat prosessin mittaaminen sekä prosessin ohjaus ja valvonta. Toimintojen automatisointi on tärkeää, jotta mittaustiedon keruu ei sido liian paljon työvoimaresursseja suhteessa varsinaisten tehtävien suorittamiseen. Välineitä tarvitaan mittaustiedon keräämisen lisäksi tiedon analysointiin, raportointiin sekä prosessinhallintaan. Myös toimiva jäljitysjärjestelmä on tärkeä, jotta havaittuihin poikkeamiin voidaan reagoida nopeasti ja kohdistaa korjaavat toimet oikein.

Optimoivalla tasolla (taso 5, optimizing) prosessin suoritus on optimoitu ja toistettava. Prosessi on myös säännöllisesti toistettava ja sen avulla saavutetaan liiketoiminnalle asetetut tavoitteet. Mittaustietoja analysoimalla pyritään systemaattisesti ja ennakoiden tunnistamaan prosessien parantamismahdollisuudet. Prosesseja säädetään jatkuvasti ja keskitytään erityisesti heikoimmin toimivien prosessien parantamiseen. Edellä kuvatun mukaisesti optimoivan tason prosessiominaisuuksia ovatkin prosessin muuttaminen ja jatkuva parantaminen. Optimoivalla tasolla korostuvat jo ennustettavalla tasolla käytössä olevien välineiden käyttö ja niiden ominaisuudet. Mittaustieto tallennetaan keskitettyyn tietokantaan, jonka hallintaan ja analysointiin tarvitaan monipuolisia ja tehokkaita välineitä. Saadut tulokset on pystyttävä siirtämään prosessinmallinnusvälineiden käyttöön, joiden avulla tarvittavat toimenpiteet voidaan määritellä. Myös prosessien muutostoimet tallennetaan tietokantaan.

CASE-välineiden käyttöönottoa ja CMM:ää prosessien kypsyyden viitekehyksenä on käsitelty useissa artikkeleissa. Sekä Curtis (1992) että Humphrey (1989) päättelevät, että CASE-teknologiaa voidaan hyödyntää täysin vasta ohjelmistoprosessin kypsyyden saavutettua määritellyn tason. Tällöin he edellyttävät täydellisen yhdistetyn CASE-järjestelmän käyttöönottoa. Curtisin mukaan teknologian käytöllä satunnaisella tasolla on vain vähän merkitystä ja toistettavalla tasolla voidaan hyödyntää lähinnä projektinhallinnan välineitä. Tällöin tavoitteena on vakiinnuttaa prosessien hallinta. Tätä tukee myös Blackburn et al. (2000) tutkimus, jonka tuloksena he päättelevät, että tasoilla 1 ja 2 on CASE-teknologian käyttöönottoa tärkeämpää prosessinhallinnan perusteiden omaksuminen. Määriteltyä tasoa lähestyttäessä välineitä voidaan käyttää prosessien mallintamiseen. Hallitulla ja optimoivalla tasolla pääasiallinen hyöty teknologian käytöstä on saada määrältään riittävästi erilaista mittaustietoa projekteista.

Mathiassen ja Sørensen (1996) mukaan CMM:n vahvuutena on se, että CASE-teknologian käyttöönotto ei ole ainoa päämäärä, vaan se on yhtenä mahdollisena tekijänä ohjelmistoprosessin parantamisessa. CMM tarjoaa menetelmät organisaatiossa vallitsevien toimintojen arviointiin ja analysointiin. Jokaisella kypsyystasolla ehdotetaan avainkäytäntöjä toimintojen parantamiseksi, jolloin teknologian käyttöönottoa voidaan verrata ja arvioida suhteessa muihin mahdollisiin parannustoimenpiteisiin. CMM:n heikkoutena on se, että otaksutaan yksipuolisen syy-yhteys prosessin kypsyyden ja välineiden käytön välille. CMM ei anna vastausta seuraaviin kysymyksiin: Mikä on organisaation kokeilun osuus käyttöönotossa? Miten erilaiset organisaatiosta johtuvat tekijät sekä välineiden toiminnalliset ominaisuudet vaikuttavat teknologian käyt-

töönottoon? Myös organisaation kypsyys ja erityispiirteet tulee huomioida teknologian käyttöönoton yhteydessä ja ennen valintaa on tehtävä kokeiluja organisaation toimintatapoihin parhaiten sopivan teknologian löytämiseksi. Kaikkia teknologian tarjoamia toiminnallisuuksia ei tarvitse ottaa käyttöön kerralla, vaan niitä voidaan lisätä kokemusten ja prosessien kypsyyden kasvaessa. Teknologian käyttöönoton merkitys ei rajoitu pelkästään ohjelmistoprosessiin, vaan sillä on vaikutuksia myös koko organisaatiossa. Seuraavassa kohdassa tutustutaan lähemmin CASE-teknologian käyttöönottoon liittyviin tekijöihin.

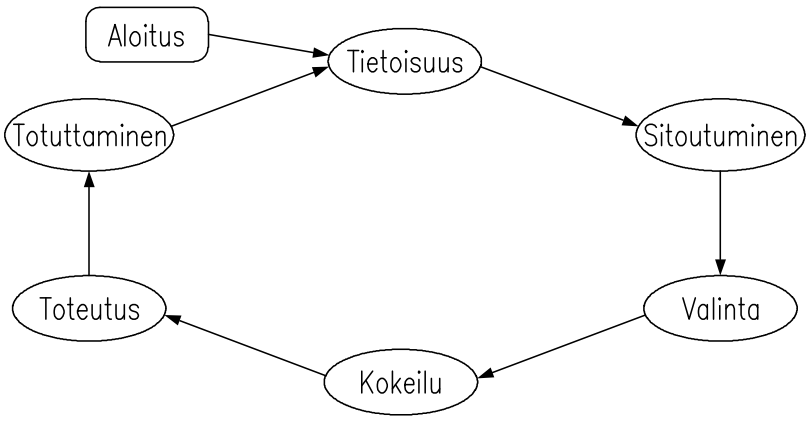
3.2 Valinta ja käyttöönotto

Välineiden tai kokonaisen CASE-järjestelmän lisääminen organisaation ohjelmistoprosessiin on monimutkainen ja vaativa tehtävä, joka onnistuakseen vaatii huolellista suunnittelua, koulutusta ja jatkuvaa seurantaa myös varsinaisen käyttöönoton jälkeen. Ohjelmistoprosesseja tarkasteltaessa on teknologisten tekijöiden lisäksi huomioitava myös yhteisölliset ja sosiaaliset tekijät. Usein kiinnitetään liiaksi huomiota vain teknologisiin näkökohtiin (Perry et al. 1994). Nämä eri tekijät on otettava huomioon myös teknologian käyttöönoton yhteydessä. Välineiden käyttöönottoprosessin ja ohjelmistotuotantoprosessin välillä voidaankin nähdä paljon yhtäläisyyksiä (Oakes et al. 1992). Tässä kohdassa tarkastelemme lähemmin CASE-teknologian käyttöönottoon liittyviä erityispiirteitä.

3.2.1 Käyttöönottoprosessin vaiheet ja suunnittelu

CASE-teknologian käyttöönotto voidaan esittää ohjelmistoprosessin tapaan vaiheittaisena prosessina, jossa edellisen vaiheen huolellinen toteuttaminen on edellytyksenä seuraavan tason toimintojen onnistumiselle. Varsinkin alkuvaiheen suunnittelu ja vaatimusten kerääminen ovat tärkeitä ja jo tässä vaiheessa voidaan ehkäistä monien myöhemmin esille tulevien ongelmien syntyminen. Jos jonkin tason toteuttaminen laiminlyödään, se vaarantaa koko prosessin onnistumisen. Oakes et al. (1992) esittävät vaiheittaisen *CASE-teknologian mukauttamismallin* (kuva 4). Siinä esitetään tarkemmin, mitä toimintoja käyttöönoton eri vaiheissa suoritetaan.

Tietoisuusvaiheessa (awarness) organisaatiossa todetaan CASE-teknologian tarve. Vaiheelle on tyypillistä organisaation nykytilan arvioiminen. Tähän liittyen selvitetään



Kuva 4: CASE-teknologian mukauttamisen vaiheet (Oakes et al. 1992).

henkilöstön luonnetta, käytössä olevaa teknologiaa, vakiintuneita prosesseja, toteutettavien projektien luonnetta sekä liikkeenjohdon poliittisia näkemyksiä ja tarpeita. Tämän avulla saadaan tarkka käsitys organisaation nykytilasta ja vaatimukset hankittavalle teknologialle. Arvioinnin jälkeen seuraa *sitoutuminen* (commitment), joka tarkoittaa organisaation päätöstä hankkia CASE-teknologiaa. On ehdottoman tärkeää, että kaikki osapuolet sitoutuvat tehtyyn päätökseen. Tämä koskee sekä organisaation johtoa, käyttäjiä että tukihenkilöitä.

Valintavaiheessa (selection) suoritetaan varsinainen hankittavan teknologian valinta. Laatimalla edellä suoritetun organisaation arvioinnin pohjalta erillinen välineiden hankintastrategia, voidaan parantaa onnistuneen valinnan edellytyksiä. Hankintastrategian tulee huomioida organisaation lyhyen ja pitkän ajan tarpeet perustuen yleisiin prosessien ja teknologian parannussuuntauksiin. Seuraavaksi arvioidaan saatavissa olevaa teknologiaa. Siihen sisältyy tiedon keruuta ja vertailua eri välineistä ja järjestelmistä sekä tutustuminen myös toimittajien taustoihin kuten luotettavuuteen, maineeseen, koulutustarjontaan ja tukipalveluihin. Kolmantena tekijänä teknologian arvioinnissa on muiden käyttäjien kokemukset. Vastaavatko kokemukset toimittajien lupauksia, millaisia käytönaikaisia ongelmia on ollut ja kuinka toimittaja on näissä tilanteissa toiminut. Ennen varsinaista valintaa arvioidaan välineiden sopivuus organisaation tarpeisiin edellä suoritettujen organisaation ja teknologian tarkastelujen pohjalta. Lisäksi selvitetään uusien välineiden yhteistoiminta järjestelmässä jo olevien muiden välineiden kanssa ja analysoidaan vaihtoehtoja teknisten ja muiden kriteerien perusteella. Samalla voidaan arvioida myös välineiden vaikutuksia sekä käyttöönoton aikana että

pitemmällä aikajaksolla. Tarkastelun perusteella karsitaan vaihtoehtoisten välineiden lukumäärä muutamaan. Ennen lopullista valintaa on välineitä myös testattava.

Kokeiluvaiheessa (trial implementation) valittua välinettä testataan käyttämällä sitä todellisen projektin suorittamisessa. Kokeiluun valittu projekti ei saa olla organisaation toiminnalle kriittisen tärkeä, mutta sen tulisi kuitenkin olla jatkossa toteutettavien projektien kaltainen ja kooltaan sen verran suuri, että kaikki välineiden käytössä myöhemminkin tarvittavat toiminnot tulevat testatuiksi. Välineen toimintaa seurataan ja tuloksia analysoidaan koko projektin ajalta. Havaittujen ongelmien perusteella voidaan suunnitella tarvittavat korjaustoimenpiteet. Saatujen kokemusten perusteella voidaan päättää välineen ottamisesta yleiseen käyttöön ja laaditaan lopullinen toteutussuunnitelma.

Toteutusvaiheessa (implementation strategy) suoritetaan varsinainen teknologian käyttöönotto toteutussuunnitelmaa hyväksi käyttäen. Käyttöönottovaiheessa on tärkeää minimoida uuden välineen tuomat haittavaikutukset olemassa olevaan ympäristöön. Vaikutukset voivat kohdistua sekä henkilöstöön että olemassa oleviin prosesseihin ja menetelmiin. Haittavaikutusten minimoimiseksi tarvitaan koulutusta, olemassa olevien prosessien ja menetelmien muuttamista sekä toisaalta myös välineen toimintojen mukauttamista organisaation tarpeita vastaaviksi. Haittavaikutukset pyritään minimoimaan myös käyttöönoton jälkeen laatimalla sopeuttamissuunnitelma, jossa edellisten lisäksi voidaan esittää projektiohjeet sekä menetelmät välineiden käytön tehokkuuden mittaamiseksi.

Totuttamisvaihe (routinization) on käyttöönoton jälkeinen välineen ylläpitovaihe. Tässä vaiheessa on tärkeää taata edellytykset välineen jatkuvalle käytölle. Tämä edellyttää riittävää koulutusta niin uudelle henkilöstölle kuin tarvittavaa lisäkoulutusta muulle henkilöstölle. Myös välineen ylläpitoa ja tukipalveluja varten on varattava riittävä ja perehtynyt henkilöstö. Ympäristön on oltava joustava ohjelmistopäivityksille ja siihen liittyville toimenpiteille.

Riippuen organisaation kypsyydestä, voivat ensimmäiset vaiheet, kuten organisaation nykytilan arvioiminen, olla jo suoritettu. Tällöin voidaan siirtyä suoraan seuraavaan vaiheeseen. Organisaation kypsyyden kasvaessa tulisi CASE-teknologian käyttöönottoprosessi ottaa yhdeksi jatkuvan parannuksen kohteeksi. Välineiden käytöstä kehitetään sopivat mittarit ja saatuja mittaustuloksia voidaan käyttää hyväksi, kun toimintojen parantamiseksi käyttöönottoprosessi aloitetaan uudelleen organisaation tilan ja

siihen mahdollisesti sopivan uuden tekniikan tarkasteluilla.

CASE-teknologian käyttöönoton moniulotteisuutta lähinnä prosessien kypsyyden näkökulmasta, mutta myös edellä esitettyä osittain täydentäen kuvaa Humphreyn (1989) esittämät ohjeet. Ensimmäiseksi kaoottinen prosessi on otettava hallintaan ennen kuin yritetään asentaa CASE-järjestelmää. Niiden organisaatioiden, joiden ohjelmistoprosessi on satunnaisella tasolla, tärkein tavoite on saada ohjelmistoprosessi hallintaan ja päästä toistettavalle tasolle. Jos hallitsemattomaan prosessiin yritetään lisätä edistynyttä CASE-teknologiaa, se aiheuttaa vain lisää ongelmia. Yhtenä mahdollisuutena on käyttää yksittäisten välineiden kokoelmaa. Tämän pitäisi kuitenkin olla välivaihe ja tavoitteena on oltava siirtyä myöhemmin käyttämään yhtenäistä sovelluskehitysympäristöä. Toiseksi prosessia on kehitettävä ennen CASE-järjestelmän asennusta tai sen aikana, mutta ei enää sen jälkeen. Prosessimenetelmät on standardoitava ja niille on saatava organisaatiossa kaikkien hyväksyntä, jotta CASE-järjestelmän käyttöönotto on helpompaa (Albizuri-Romero 2000). Jos organisaatio on lähellä toistettavaa tasoa, ovat konfiguraationhallinnan ja vaatimustenhallinnan toteuttaminen tärkeitä CASE-järjestelmää toteutettaessa. Lisäksi organisaatioon on perustettava *ohjelmistoprosessiryhmä* (software engineering process group, SEPG), joka keskittyy prosessien parantamiseen. Ryhmä toimii yhteistyössä teknologian käyttöönottoa suunnittelevan ja valintoja toteuttavan ryhmän kanssa. Humphreyn kolmannen ohjeen mukaan järjestelmän muuntaminen on kriittinen, mutta kaikkein hankalin tehtävä on henkilöstön muuttaminen. Henkilöstöä koskevissa asioissa koulutus on tärkein ja se on usein myös laiminlyöty. Toistettavalla tasolla oleville organisaatioille se on yleisin yksittäinen heikkous niiden toiminnassa. Henkilöstö ei usein halua tai ei pysty käyttämään edistynyttä teknologiaa ilman riittävää koulutusta. Neljäs ohje muistuttaa, että CASE-järjestelmän asennus ei ole koskaan täysin valmis. Ihmiset ja heidän kokemuksensa, saatavilla oleva teknologia, prosessimenetelmät, liiketoimintaympäristö ja projektin tarpeet muuttuvat ajan kuluessa. Sen vuoksi on tärkeää ylläpitää toimintaympäristön vakautta CASE-järjestelmä mukaan lukien. Tämä vaatii osaavaa henkilöstöä, raportointimenetelmiä, ongelmien jäljitysmekanismeja, prosessien arviointimenetelmiä ja kykyä testata muutoksia ennen niiden varsinaista käyttöä. Humphreyn viides ohje muistuttaa, että ajattelua ei saa kokonaan unohtaa. Tehokkaiden välineiden käyttöönottoon liittyy riski, että niistä saatava hyöty menetetään väärinkäyttöön. Välineisiin ei tule luottaa liikaa, vaan edelleen suunnittelussa tarvitaan luovaa ajattelua. Hyvien suunnitelmien pohjaksi tarvitaan toiminnallinen ja rakenteellinen käsitteistö, jota välineet eivät pysty yksin tuottamaan. Käytettävät suunnittelumenetelmät on valittava huolellisesti omien tarpeiden

pohjalta, eikä sen mukaan, mitä välineet tarjoavat. Valittuja menetelmiä on noudatettava koko järjestelmän arkkitehtonisen suunnittelun ajan ja se on perustana kaikille sitä seuraaville toimille.

3.2.2 Edellytykset organisaatiolta

CASE-teknologian käyttöönottoprosessi koskettaa koko organisaatiota ja vaatii mukautumista. Suunnittelun yhteydessä ja päätöksiä tehtäessä on otettava huomioon lukuisia eri näkökohtia ja selvitettävä suoritettavien toimenpiteiden erilaisia vaikutuksia organisaatiossa. Tarkasteltavat kysymykset voidaan jakaa merkitykseltään lyhytkestoisiin ja pitkäkestoisiin (Oakes et al. 1992). Lyhyellä aikavälillä organisaatiossa on otettava huomioon mahdollisuus:

- tuottavuuden tilapäiseen heikkenemiseen,
- henkilöstön tyytymättömyyteen uuden teknologian omaksumiseen ja käyttöönottoon,
- muutoksiin totutuissa prosesseissa ja menettelytavoissa,
- laajamittaisen koulutuksen tarpeeseen ja
- korkeisiin kustannuksiin.

Pitkän aikavälin kysymyksiä tarkasteltaessa on huomioitava:

- Mahdollisuus pitkäkestoisiin ylläpitokustannuksiin, jopa järjestelmän avulla toteutettujen ohjelmistojen koko elinkaaren ajalta.
- Hankittavan teknologian toistuvat päivitykset ja uudet versiot.
- Mahdollisuus, että teknologia kehittyy edelleen ja kokonaan uusien ominaisuuksien lisääminen nyt hankittavaan järjestelmään aiheuttaa mittavia muutoksia ja prosessien mukauttamista.
- Jatkuvat kustannukset uuden henkilöstön kouluttamisesta sekä muun henkilöstön täydennyskoulutuksesta. Karkeasti laskien koulutukseen on varauduttava sijoittamaan yhtä paljon kuin itse hankintaan (Daneva 1999).

Mendoza et al. (2001) ovat esittäneet mittariston, jonka avulla voidaan arvioida hankittavaksi suunnitellun CASE-teknologian soveltuvuutta organisaation käyttöön. Mittaristossa teknologian käyttöönottoon liittyvät organisaation sisäiset tekijät jaetaan kahteen pääryhmään. Hallinnollinen osa kuvaa organisaation sosiaaliseen järjestelmään liittyviä tekijöitä ja toiminnallinen osa teknisiä näkökohtia. Hallinnolliseen ryhmään kuuluvia tekijöitä ovat johdon tuki, teknologisten uudistusten toteuttaminen, ohjelmistojen ja laitteiden ajanmukaistaminen, koulutussuunnitelma, organisaation rakenne ja mukautumiskyky sekä projektinhallinnan toteuttaminen. Toiminnallisen ryhmän tekijöitä ovat sovelluskehittäjien osallistuminen teknologian valintaan ja käyttöönottoprosessiin, välineiden soveltuvuus vakiintuneisiin prosesseihin ja niihin haluttuihin muutoksiin sekä henkilöstön pätevyys ja kyky omaksua välineiden käyttö.

CASE-teknologialla on suuri vaikutus myös organisaation johdolle. Mitä strategiasuuntautuneempi organisaatio on, sitä suurempi ja myönteisempi on välineiden vaikutus. Tällöin myös teknologia kyetään omaksumaan nopeammin ja tehokkaammin sekä välineitä käytetään laajemmalla tasolla. Liikkeenjohdollinen kypsyys on tärkeä teknologian mukauttamisessa. Tärkeimpiä ominaisuuksia ovat strateginen tietous ja suunnittelu, objektiivisuus ja kyky nähdä organisaatio kokonaisuutena ja pitkän ajan visiona. Teknologian tärkeys organisaatiolle ja sen oma osuus mukauttamisprosessissa on arvioitava reaalisesti. Edellä esitetyt tekijät merkitsevät henkilöstön keskuudessa parempaa motivaatiota ja hyväksymistä sekä vähentävät teknologisia rajoitteita kohtaan tunnettuja suhtautumisongelmia ja kielteisiä reaktioita. (Siltanen 1992)

3.2.3 Edellytykset yksilötasolla

Prosessin kypsyys voidaan nähdä myös henkilöstön työtapojen kehittymisenä. Satunnaisella tasolla organisaatiossa työskentelee laaja koordinoimaton ryhmä yksilöitä. Vaikka he voivatkin ajoittain työskennellä yhdessä, se on enemmän satunnaista kuin organisaation tai prosessin aikaansaamaa. Toistettavalla tasolla henkilöstö muodostaa pieniä ammattilaistiimejä, joissa henkilöt työskentelevät hyvin ryhmänä, mutta ryhmien välinen koordinointi ei ole tehokasta. Määritellyltä tasolta alkaen organisaatiossa on laajemman tason viiteryhmiä, jotka toimivat yhteistyössä toisiaan tukien. Prosessin kypsyys ei siten ole pelkkä hallintakysymys, vaan yksi henkilöstöympäristöä määrittävä pyrkimys laaja-alaiseen yhteistyöhön (Humphrey 1989). Yksittäisille työntekijöille on luotava yhteinen työskentelyetiikka ja toimintatavat. Nämä ovat avaintekijöitä

siirryttäessä satunnaiselta toistettavalle tasolle sekä omaksuttaessa CASE-järjestelmän yleisiä menetelmiä ja välineitä. Jos henkilöstö ei ole valmis hyväksymään CASE-järjestelmää, he kokevat sen rajoittavan työskentelyä. Toisaalta järjestelmä ei saa asettaa liian tiukkoja toimintaehtoja, vaan henkilöstöllä on oltava mahdollisuus tarvittaessa vaihtoehtoisiin menettelytapoihin. Prosessin kypsyyden lisääntyessä ja kokemusten kasvaessa myös henkilöstö oppii ymmärtämään, kuinka eri tehtävät tulee suorittaa ja milloin on tarpeen poiketa määritellyistä toimintatavoista.

3.3 Tavoitteelliset edut ohjelmistoprosessien parantamisessa

Kirjallisuudessa esitetään monia välineiden käytöstä saatavia etuja valmistettavan ohjelmiston kannalta. Esimerkiksi Low ja Leenanuraksa (1999) käyttävät tutkimuksessaan viittä laatutekijää: luotettavuus, ylläpidettävyys, siirrettävyys, tehokkuus ja käytettävyys. Välineiden avulla parannetaan monin tavoin myös ohjelmistoprosessien suorittamista, jotka omalta osaltaan parantavat myös työn lopputuloksena olevaa ohjelmistoa. Tässä kohdassa tarkastellaan välineiden avulla saavutettavia etuja ohjelmistoprosessien parantamisen näkökulmasta.

3.3.1 Prosessien automatisointi

Ohjelmistotuotanto on pohjimmiltaan ihmisistä lähtevä, luovaa ajattelua vaativa prosessi (Humphrey 1989). Ihmisten kyvykkyyttä voidaan lisätä tietyn ajan kuluessa koulutuksen ja kokemuksen avulla, mutta se on kuitenkin rajallista. Tehokkuuden lisäämiseksi tarvitaan automaatiota ja tehokkaita välineitä. Automatisointi on keskeisin peruste teknologian käyttöönottamiselle ohjelmistoprosessien suorittamisessa. CASE-välineiden avulla voidaan automatisoida projektinhallinnan toimintoja, kaikkien prosessien aikana tuotettujen työtulosten hallintaa sekä avustaa analysoinnin, suunnittelun, koodauksen ja testauksen suorittamisessa (Pressman 2000). Jo prosessien kypsyyden ollessa toistettavalla tasolla voidaan yksittäisten välineiden avulla automatisoida niiden tukemien tehtävien suorittamista. Tällä tavalla saadaan parannettua esimerkiksi prosessien toistettavuutta, tarkkuutta ja suoritusnopeutta. Automaatiota voidaan käyttää myös tukemaan harjoittelua sekä opastamaan käyttäjiä varsinaisten tehtävien suorittamisessa (Christie et al. 1996). Opastus voi olla eritasoista riippuen käyttäjän kokemuksesta. Välineiden automaattisen tuen tarve kasvaa kehitettävien ohjelmisto-

jen kasvaessa. Samalla ohjelmistot tulevat monimutkaisemmiksi ja myös kehitystyöhön osallistuvien henkilöiden lukumäärä kasvaa. Välineiden avulla voidaan automatisoida esimerkiksi työtulosten hallintaa, yhteistyön koordinointia ja välineiden keskinäistä yhteistoimintaa. Prosessien parantamisen kannalta automaatio on välttämättömyys prosessien kypsyyden kasvaessa. Tarvitaan yhä enemmän ja monipuolisempia mittaustietoja, joita analysoimalla tarkkaillaan prosessien suoritusta ja etsitään heikkouksia, joita voidaan kehittää. Mittaustiedon keruun ja analysoinnin suorittaminen ilman automaattisten välineiden tukea on hyvin vaikeaa ja työlästä. Automaation avulla voidaan varmistaa mitattujen tietojen yhtenäisyys ja riittävä otanta analyysejä varten sekä tehokkaat ja nopeat analyysit. Kokonaisuutena CASE-välineiden avulla voidaan automatisoida ohjelmistotuotannon käsin tehtäviä vaiheita ja parantaa teknistä näkemystä toteutettavasta ohjelmistosta siirtämällä ohjelmistosuunnittelijoiden vapautuva työpanos analysointiin ja suunnitteluun (Sitol 2002). Lisäksi voidaan varmistaa, että ohjelmiston laatu on suunniteltu jo ennen sen toteuttamista (Pressman 2000).

3.3.2 Keskitetty tiedon varastointi ja tiedon siirrettävyys

Keskitettyä tietovarastoa on pidetty CASE-teknologian keskeisenä elementtinä. Jokaisen välineen käyttöönoton yhteydessä tulee aina suunnitella myös tietovaraston rakentaminen ja hallinta (Purvis et al. 2000). Tietovarasto on tärkeä pääoma, joka CASE-teknologian käytön avulla saavutetaan. Tietovarasto sisältää kaikki tiedot, dokumentit ja työtulokset, joita ohjelmistotuotantoprosessin aikana syntyy. Varastoituja tietoja voidaan hyödyntää monin tavoin sekä projektin eri vaiheissa että myöhemmin muiden projektien yhteydessä esimerkiksi analysointiin ja uudelleenkäyttöön. Ilman välineitä ja tietovarastoa näin laajojen projektitietojen kerääminen on mahdotonta. Tietovarasto on keskeinen tiedon tallennuspaikka, josta tiedot jaetaan eri välineiden käyttöön (Hoffer et al. 2002). Tämä edellyttää tietojen tallentamista sellaisessa muodossa, että kaikki ympäristöön liitetyt välineet pystyvät käsittelemään tietoja tarvitsemallaan tavalla. Tietovarastot toteutetaan yleensä relaatio-, olio- tai olio-relaatiotietokantana (Dittrich et al. 2000). Tietovaraston tehokas hyödyntäminen vaatii keskenään yhteensopivien välineiden lisäksi normaaleja tietokannanhallintatoimintoja sekä CASE-välineille tyypillisiä toimintoja esimerkiksi tietovaraston hallintaan ja tallennettujen tietojen analysointiin.

Keskitetyllä tiedon varastoinnilla on prosessien parantamisen kannalta monia etuja. Yksittäiset välineet ohjaavat käyttäjiä määrittämään ja tallentamaan kunkin proses-

sin suorittamisessa tarvittavat tiedot ja työn lopputulokset yhtenäisellä tavalla. Tällöin myös prosessien eri suorituskerroilla tallennettavat tiedot ovat keskenään samanmuotoisia. Keskitetyn tietovaraston ansiosta on helppo valvoa, että kaikki tarvittavat tiedot tallennetaan. Myös olemassa olevan tiedon hakeminen yhdestä paikasta on helppoa ja nopeaa verrattuna yksittäisten prosessien omiin varastoihinsa tallennetun, mahdollisesti keskenään epäyhtenäisen tiedon hakemiseen. Tiedon hakeminen ja oikean tiedon löytyminen helpottuu edelleen, kun käytössä on kaikkien eri ohjelmistoprosessialueiden kattava versionhallintajärjestelmä. Dittrich et al. (2000) luettelevat myös muita vaatimuksia, joita tietokantamuotoiselta tietovarastolta edellytetään. Näitä ovat versionhallinta, työtulosten eheyden ja yhtenäisyyden varmistaminen, työtulosten hakeminen ja jäljittäminen, tuki ryhmätyöskentelyn tapahtumien ja työtilojen hallinnalle, ohjelmistoprosessien esitystavan ja sääntöjen määrittely, mittaustiedon keruu ja arviointi sekä välineiden yhteensovittaminen ja siirrettävyys.

Käytettäessä yksittäisiä, vain tiettyjä työvaiheita tukevia välineitä, voi vaikeutena olla eri työvaiheiden välinen tiedon siirrettävyys. Esimerkkeinä tällaisista ohjelmistotuotannon eri työvaiheista on siirtyminen vaatimusmäärittelystä suunnitteluun sekä suunnittelusta toteutukseen. Eri valmistajien välineet eivät välttämättä hyväksy lähtötietoinaan toisen välineen tuottamaa tietoa. Tästä aiheutuu tarve tietojen muuntamiseen sopivaan muotoon tai pahimmassa tapauksessa jopa seuraavan työvaiheen lähtötietojen syöttämisen käsin. Ongelmaa voidaan vähentää siirtymällä toimintoalustojen ja ympäristöjen käyttöön, mutta täydellinen ja toimiva välineiden yhteensovittaminen on edelleen toteutumatta. Laajassa välineiden yhteensovittamista käsittelevässä kirjallisuuskatsauksessa (Wicks 2004) esitetään monipuolisesti välineiden integraation laajuutta, tyyppejä, standardeja, teollisuuden kokemuksia, kehyksiä, mallinnustekniikoita sekä prosessien ja välineiden yhdistämistä. Tieteellisestä tutkimuksesta, standardeista ja teollisuuden kokemuksista huolimatta loppupäätelmänä on, että vielä ei ole olemassa ideaalista, kaikkiin tilanteisiin sopivaa ohjelmistotuotantoympäristöä ja tapaa välineiden yhdistämiseen. Uusin välineiden yhteensovittamista koskeva standardi (IEEE 2003) esittelee viitekehyksen CASE-välineiden yhteiskytkennälle. Se määrittelee välineen yhdistämisen käyttäjän, organisaation, laitealustan ja muun välineen kesken. Standardi esittelee myös *semanttisen siirtokielen* (semantic transfer language, STL), jonka avulla voidaan siirtää ohjelmiston käyttäytymiskuvauksia välineeltä toiselle. Standardin soveltamisesta käytäntöön ei kuitenkaan löytynyt viitteitä kirjallisuudesta.

3.3.3 Riippuvuussuhteet, muutoksen hallinta ja jäljitettävyys

Tietovarastoon tallennetuilla tietoelementeillä on monenlaisia keskinäisiä suhteita. Suhteiden taso voi vaihdella pelkästä yhteydestä riippuvuuteen tai sitovaan suhteeseen. Tätä suhteiden ylläpitämistä kutsutaan *yhteyksienhallinnaksi* (link management) (Pressman 2000). Kyky seurata näitä riippuvuuksia on ensiarvoisen tärkeää tietovarastoon tallennettujen tietojen eheydelle ja siitä muodostetuille ohjelmistotoimituksille. Yhteyksienhallinta on myös yksi suurimmista tietovaraston tarjoamista hyödyistä ohjelmistoprosessien parantamisessa. Yhteyksienhallinnan yhtenä tärkeänä ominaisuutena on kyky tunnistaa ja arvioida muutoksen vaikutuksia. Esimerkiksi mahdollisuus etsiä kaikki kohteet, joihin tietyn vaatimuksen muutos voi vaikuttaa, auttaa arvioimaan muutoksen vaikeusastetta, muutostyöhön tarvittavaa aikaa ja kustannuksia. Myös mahdollisten virhetilanteita aiheuttavien sivuvaikutusten havaitseminen ja ehkäiseminen helpottuvat. Yhteyksienhallinta auttaa myös tietojen oikeellisuuden ja ajantasaisuuden varmistamisessa synkronoimalla tietovaraston toimintoja. Esimerkiksi oliopohjaista mallinnusmenetelmää käytettäessä voidaan suunnittelumallin tiettyyn kaavioon tehdyn muutoksen vaikutukset nähdä heti myös muissa kaavioissa, dokumenteissa ja jopa koodissa. Myös ryhmätyöskentelyssä on tärkeää, että tehdyt muutokset näkyvät heti myös muille samanaikaisesti työskenteleville.

Vaatimusten jäljitettävyys on yksi yhteyksienhallintaan liittyvä toiminto (Pressman 2000). Sen avulla voidaan jäljittää kaikki tietyn vaatimuksen tai vaatimusmäärittelyn tuloksena tehdyt suunnitelmat, koodit ja valmiit komponentit. Tätä menettelyä kutsutaan *eteenpäin jäljittämiseksi* (forward tracking). Menetelmää voidaan käyttää myös toisin päin. Kun jäljitetään vaatimusta, jonka perusteella tarkasteltava työn tuotos on tehty, nimitetään sitä *taaksepäin jäljittämiseksi* (backward tracking). Jäljittämistä voidaan hyödyntää sekä ohjelmiston kehitysvaiheessa että myöhemmin ylläpitovaiheessa.

Kaikki edellä kuvatut toimenpiteet auttavat omalta osaltaan parantamaan toteutetun ohjelmiston ylläpidettävyyttä ja jopa vähentämään ylläpidon tarvetta. Automaation avulla dokumentit ja koodi saadaan yhtäläisiksi ja niiden oikeat versiot voidaan hakea nopeasti tietovarastosta. Korjausten ja muutosten vaatimat toimenpiteet ja vaikutukset on helpompi määritellä ja analysoida kattavien tietokantatietojen ansiosta. Jäljitettävyyden avulla ylläpidossa koodiin tehtyjen muutoksien vaikutukset voidaan päivittää helposti myös suunnitelmiin.

3.4 Rajoitteita ja haittavaikutuksia

CASE-teknologian avulla on saatu merkittäviä parannuksia ohjelmistoprosessiin, joskaan ei niin merkittäviä kuin aikaisemmin ennustettiin (Sommerville 2000). Yhtenä syynä tähän on ollut se, että ohjelmistotuotanto vaatii luovaa ajattelua ja se ei ole helposti automatisoitavissa. Toisaalta ohjelmistotuotanto on myös tiimityötä ja laajoissa projekteissa kuluu paljon aikaa henkilöiden väliseen vuorovaikutukseen. Kaikkia näitä toimintoja ei voida tukea välineiden avulla. Teknologian käyttöön liittyy monia muitakin haittavaikutuksia ja se voi aiheuttaa rajoitteita prosessien suorittamiseen tai aiheuttaa kokonaan uusia tehtäviä. Tässä kohdassa tarkastelemme lähemmin välineiden käytön rajoitteita ja haittavaikutuksia.

3.4.1 Kustannukset

Hankintahinta mainitaan kirjallisuudessa lähes poikkeuksetta yhdeksi suurimmaksi haittatekijäksi CASE-välineiden käyttöönottamisessa (esim. McMurtrey et al. 2000, Premkumar ja Potter 1995). Tämä tarkoittaa nimenomaan laajoja CASE-ympäristöjä, jotka käsittävät koko ohjelmistotuotannon elinkaaren ja jotka ovat käytössä organisaation kaikilla tasoilla. Pienissä yrityksissä pääoman ja resurssien puute estää usein investoimasta laajoihin CASE-järjestelmiin. Pienissä yrityksissä myös ajatellaan, että CASE-teknologialla saavutetut hyödyt ovat perusteltuja vain laajoissa järjestelmissä. Lisäksi on muistettava, että järjestelmän ja laitteistojen hankinnan korkeat aloituskustannukset eivät riitä, vaan on varauduttava myös hankinnan jälkeisiin ylläpito- ja koulutuskustannuksiin koko järjestelmän käyttöiän ajalle. Liiketaloudellisesti sijoitus CASE-järjestelmiin on yritykselle hankala myös sikäli, että sijoitukselle voidaan odottaa tuottoja vasta useiden vuosien kuluttua. Lisäksi järjestelmän käyttöönotto riippuu hyvin monesta tekijästä, eikä investointia tehtäessä voida olla täysin varmoja käyttöönoton onnistumisesta suunnitellulla tavalla.

CASE-järjestelmän hankkimiseen liittyy myös monia seurannaisvaikutuksia, jotka vaativat sekä henkilöstöresursseja että pääomaa. Jo ennen järjestelmän hankkimista vaaditaan tarkkaa suunnittelua sekä omien tarpeiden ja käytäntöjen arvioimista sopivan järjestelmän valitsemiseksi. Tätä tarkastellaan lähemmin seuraavassa alakohdassa. CASE-järjestelmiä on pidetty myös huonosti mukautettavina (Flynn et al. 1995). Tämän vuoksi tarvitaan järjestelmän ja yrityksessä käytössä olevien menetelmien, stan-

dardien ja käytäntöjen yhteensovittamista. Nämä käyttöönottoa valmistelevat työt vaativat henkilöresursseja ja asiantuntemusta. Jos yrityksen sisältä ei löydy riittävää asiantuntemusta, on uusiin tehtäviin palkattava uutta, erikoisasiantuntemusta omaavia henkilöstöä. Iivari (1996) toteaa CASE-välineiden olevan suhteellisen monimutkaista teknologiaa, jolloin tiedon ja asiantuntemuksen puute voi olla esteenä käyttöönottamiselle. Asiantuntemusta tarvitaan myös käyttöönoton jälkeen järjestelmän ylläpidossa ja käyttäjien opastuksessa. Prosessien kypsyyden kasvaessa liitetään myös CASE-teknologian käyttöönottoprosessi yhdeksi jatkuvan parannuksen kohteeksi. Riippuen organisaation koosta tarvitaan tehtävän hoitoon henkilön tai työryhmän palkkaamista.

3.4.2 Valinnan vaikeus

Tarkoituksenmukaisen välinekokoelman tai CASE-järjestelmän valinta on vaikea, mutta organisaation tulevaisuuden kannalta erittäin merkittävä. Valittavana olevien välineiden ja järjestelmien lukumäärä on suuri ja niiden todellisten ominaisuuksien arvioiminen on hankalaa. On myös vaikea ymmärtää, kuinka ne ovat yhteydessä toisiinsa toiminnallisesti ja teknisesti (Fuggetta 1993). Valintaa vaikeuttavat lisäksi monet huomioon otettavat organisaation sisäiset tekijät, esimerkiksi organisaation koko, käytettävien prosessien kypsyys, kehitettävien ohjelmistojen koko ja sovellusala. Sopivan CASE-järjestelmän valinta ja käyttöönotto muodostavat oman vaativan prosessin, jonka erityispiirteitä tarkasteltiin edellä kohdassa 3.2.

CASE-järjestelmän taustalla on yleensä jokin kehitysmenetelmä (Flynn et al. 1995), joka myös on huomioitava valintaa tehtäessä. Järjestelmän tukemasta menetelmästä tarvitaan käytännön kokemusta, jotta järjestelmän tarjoamia toimintoja osataan hyödyntää riittävästi. Jos organisaatiossa on jo käytössä vastaava menetelmä, ovat aikaisemmat käytännöt ja standardit mukautettavissa mahdollisimman vähäisillä muutoksilla uuteen järjestelmään sopiviksi. Kokonaan uuden menetelmän käyttöönottaminen voi aiheuttaa niin suuria muutoksia organisaation toimintoihin, että siitä aiheutuu kohtuuttomia kustannuksia varsinaisen järjestelmän aiheuttamien kustannusten lisäksi. Jos organisaation prosessi on määrittelemätön, on ensin valittava, määriteltävä ja käyttöönotettava organisaation tarpeisiin sopiva menetelmä ja vasta sen jälkeen valittava siihen sopiva CASE-järjestelmä (Humphrey 1989).

Valinnan onnistuminen on tärkeää organisaation tulevaisuudelle. Päätös tietyn järjestelmän hankkimisesta sitoo organisaation tulevaa toimintaa monin tavoin. Valinnan tulisi

sen vuoksi täyttää mahdollisimman hyvin nykyiset tarpeet ja myös mahdollisia tulevia tarpeita olisi pystyttävä ennakoimaan. Jos järjestelmän valinta osoittautuu epäonnistuneeksi, on uuteen järjestelmään siirtyminen erittäin vaikeaa (Oakes et al. 1992). Tehty investointi on organisaation liiketoiminnan mittakaavassa niin suuri, että ei ole varaa kevyin perustein tehtyyn kokeiluun. Käyttöönoton onnistuessakin tuotto-odotukset alkavat vasta useamman vuoden käytön jälkeen.

3.4.3 Koulutuksen tarve

CASE-järjestelmän pitkä oppimiskäyrä mainitaan kirjallisuudessa hankintahinnan lisäksi yhdeksi CASE-teknologian haittatekijäksi (esim. Flynn et al. 1995, McMurtey et al. 2000). CASE-välineet ovat usein vaikeita käyttää (Premkumar ja Potter 1995) ja vaativat paljon koulutusta sekä ennen käyttöönottoa henkilöstölle että myöhemmin jatkokoulutusta ja uusien käyttäjien koulutusta. Koulutus on tärkeää kaikilla organisaation tasoilla, joissa järjestelmää käytetään. Myös organisaation johdon on tunnettava järjestelmän tarjoamat mahdollisuudet ja toisaalta on opittava käyttämään järjestelmän organisaatiotason toimintoja, esimerkiksi projektinhallinnan toimintoja raportointeineen. Ennakkoon annettu koulutus lisäksi vähentää ennakkoluuloja uutta järjestelmää kohtaan ja vähentää siten henkilöstön vastustusta uutta järjestelmää kohtaan (Albizuri-Romero 2000).

Järjestelmän tarjoamat omat käytönaikaiset tukitoiminnot ja käyttöohjeet sekä toimittajan tarjoamat tukipalvelut ovat tärkeitä ja ne lyhentävät oppimisaikaa. CASE-välineen pitäisi myös aktiivisesti neuvoa käyttäjää eri tilanteissa riippuen käyttäjän kokemuksesta (Huang 1998). Edellä kuvattuihin ominaisuuksiin olisi hyvä tutustua jo järjestelmän valintavaiheessa ja käyttää niitä valintakriteereinä vertailtaessa eri järjestelmien ominaisuuksia.

3.5 Kehityssuunnat

Viime vuosina ohjelmistotuotannossa on koettu murroskausi, jossa erityisesti internetin käytön yleistyminen on aiheuttanut monia muutoksia perinteisiin ohjelmistoihin verrattuna. Yhä useammilla sovellusalueilla ohjelmistoille on tyypillistä, että ne ovat hajautettuina monille palvelimille, ne käsittelevät erilaisista lähteistä haettua tietoa ja niillä on selainpohjainen käyttöliittymä. Tällaisten ohjelmistojen kehittäminen ja yllä-

pitäminen vaatii laaja-alaista tietoa hajautetuista järjestelmistä, tietokannoista ja verkkoteknologiasta (Brown 1997). Monimutkaisten ja laaja-alaisten ohjelmistojen kehityksessä korostuu entisestään tehokkaiden välineiden käyttö. Muutokset ovat aiheuttaneet uusia vaatimuksia myös välineiden toiminnallisuuteen. Tässä kohdassa tarkastelemme lähemmin ominaisuuksia ja kehitysnäkymiä, joita uusien CASE-välineiden odotetaan tukevan.

3.5.1 Hajautetut ohjelmistot ja internet

Tulevaisuudessa ohjelmistoille on tyypillistä, että ne ovat hajautettuja, niiden on päästävä kaupallisiin tietokantoihin, pystyttävä tarkistamaan käyttöoikeuksia, sallittava selainkäyttöliittymä sekä suoritettava kyselyjä erilaisille koneille internetissä. Verkkoteknologian kehittymisen ansiosta ohjelmistot rakentuvat yhä useammin kokoelmista erilaisia toimintoja, jotka on hajautettu eri paikkakunnilla sijaitseville palvelimille (Brown 1997). Tässä tarvitaan edellä kuvattuja toimintoja tukevia välineitä, sillä perinteisiin ohjelmistoihin verrattuna hajautetut ohjelmistot ovat monimutkaisia ja riskialttiita. On kehitettävä soveltuvia, välineiden tukemia arkkitehtuureja, jotta hajautettujen järjestelmien kehitys yksinkertaistuu. Hajautetun järjestelmän kehitystyön vaikeutta kuvaa se, että ohjelmistosuunnittelijan on tyypillisesti tehtävä samanaikaisesti päätöksiä kolmella toisiinsa vaikuttavalla abstraktiotasolla (Brown 1997). Abstraktein taso on *arkkitehtuuritaso*, jolla määritellään ohjelmiston arkkitehtoninen perusmuoto, ohjelmiston ei-funktionaalisten ominaisuuksien perusluonne. *Palvelujen tasolla* määritellään arkkitehtonisten peruskomponenttien rajapinnat ja niiden keskinäinen toiminta sekä tarkastellaan keskinäisen kommunikoinnin yhteensovittamista ja rakenteita, palvelujen vasteaikoja, virhetoleransseja ja menettelytapoja, jos tietoa ei löydetä. Kolmantena tasona on *mekanismitaso*, jolla valitaan toteutuksessa käytettävät tuotteet, kielet, käyttöjärjestelmät sekä muut järjestelmän toimintaan vaikuttavat tekijät.

Internet on hyväksytty yleiseksi tavaksi hankkia tietoa, näyttää haettu tieto käyttäjälle selaimessa ja vaihtaa tietoa selaimessa esitettävien lomakkeiden avulla. Selaimista on siten tullut käyttöliittymien käyttötuntuman osalta *de facto* -standardi (Brown 1997). Tiedon sijainti ja sen alkuperäinen ympäristö tulevat käyttäjälle näkymättömiksi ja merkityksettömiksi. Tiedon hajauttaminen hyväksytään, mutta sen esitystavan on kuitenkin oltava yhtäläinen. Tietoa ei haluta esitettävän pelkästään tekstinä ja numeroita sisältävinä taulukkoina. Käyttäjät odottavat monipuolista multimediaa tiedon esityk-

seen: grafiikkaa, ääntä, videota ja animaatiota, jota selaimet osaavat käsitellä. Selaimet pystyvät automaattisesti käsittelemään erimuotoista tietoa sekä pakkaamaan ja purkamaan tietoa, jotta se voidaan siirtää verkossa tehokkaasti. Käyttäjät odottavat myös saavansa internetistä paljon aineistoa ilmaiseksi tai halvalla muokattavaksi omiin sovelluksiinsa. Internet-sivujen sisällön tuottamiseen ja käsittelyyn onkin kehitetty lukuisia välineitä. Myös ohjelmistosuunnittelussa käytettävien välineiden on tuettava näitä toimintoja, joista tärkeimpiä ovat *Java*-kielen tuki tietoa välittävien lomakkeiden käsittelyyn sekä *HTML*-merkintäkieli selaimessa esitettävien sivujen tekemiseksi.

Internet-pohjaisten ohjelmistojen yleistyessä on kasvanut merkittävästi myös ohjelmistojen käsittelemän tiedon määrä. Tämän seurauksena myös välineiden on pystyttävä käsittelemään tämä eri lähteistä ja erilaisessa muodossa oleva tieto. *Tiedon yhdistäminen* (information integration) onkin haaste tämänhetkiselle ohjelmistokehitykselle. Raja tietokannan ja sisällön hallintajärjestelmien, välimuistien, tietovarastojen sekä muiden tietohallintajärjestelmien välillä on hämärtymässä ja on tarve alustalle, joka tarjoaa yhdistetyn näkymän kaikkiin näihin palveluihin ja niiden tarjoamaan tietoon (Roth et al. 2002). Perustana tällaiselle alustalle on vankka tiedonhallintajärjestelmä, joka tukee *XML* (extensible markup language) (W3C 2004) tietovarastoja. XML sopii relaatiomallia paremmin puolirakenteisen ja rakenteettoman tiedon esittämiseen. Lisäksi XML:n avulla voidaan esittää standardi metamalli näiden tietojen kuvaamiseksi.

Tiedon yhdistämisen ohella XML tarjoaa mahdollisuuden yhdistää eri välineiden välisen tiedon jakamisen. XML on keskitason standardi, joka mahdollistaa joustavampien välineiden kehittämisen (Harrison et al. 2000). Välineiden on oltava sellaisia, että ne tunnistavat XML:n sekä syötteenään että antavat sen tuloksenaan. Tämä mahdollistaa vuorovaikutuksen myös sellaisten välineiden kanssa, jotka voivat epäsuorasti käsitellä XML:ää. Tämän ansiosta kasvavat organisaatioiden mahdollisuudet ja vaihtoehdot liittää käyttämiinsä välineympäristöihin uusia, paremmin heidän prosesseihinsa sopivia yksittäisiä välineitä. Aikaisemmin välineympäristöjen rajoitteena on ollut se, että esimerkiksi eri valmistajien välineiden liittäminen on ollut vaikeaa tai mahdotonta niiden erilaisen tiedon käsittelyn takia.

3.5.2 Komponenttipohjaiset ohjelmistot ja menetelmien kehitys

Jo 1990-luvun loppupuolelta alkaen lisääntynyt suuntaus olemassa olevien moduulien ja koodin uudelleenkäyttöön tulee kasvamaan edelleen. Tämän seurauksena siirrytään

yhä enemmän *komponenttipohjaiseen ohjelmistokehitykseen* (component-based development) (Brown 1997). Tarve tämän suuntaiseen kehitykseen on aiheutunut taloudellisesta paineesta siirtyä käyttämään entistä enemmän valmiita kaupallisia ohjelmistoja (commercial off-the-shelf, COTS) sekä toisaalta edellä kerrottu internetin käytön lisääntyminen.

Komponenttipohjainen ohjelmistokehitys vaatii lukuisia muutoksia ohjelmistojen arkkitehtuureihin ja ohjelmistojen kehitystapoihin. Ohjelmistot kehitetään yksittäisinä, toisistaan riippumattomina moduuleina, joilla on hyvin määritellyt rajapinnat. Komponenttipohjaisen ohjelmistokehityksen menestymistä auttavat oliopohjaisten kielten yleistyminen ja niiden parempi rakenne komponenttien käsittelyyn. Erityisesti Java-kielen rakenne on sopiva komponenttikirjastojen luomiseen. Kirjastojen avulla pienoissovellusten (applet) ja ohjelmistojen rakentaminen on helppoa. Sovellusaluekohtaiset kirjastot ja kehykset ovat yleistymässä. Ohjelmistokehityksessä käytettyjen välineiden on tuettava komponenttikirjastojen tehokasta selaamista, älykästä hakua sekä käyttöön soveltuvimpien komponenttien analysointia ja valintaa. Lisäksi tarvitaan ominaisuuksia uudelleenkäytettävien komponenttien luettelointiin, komponenttien asentamiseen ohjelman osaksi, olemassa olevien komponenttien päivittämistä sekä ulkoisten komponenttien yhdistämistä internetin kautta etäpalvelimelta tai kolmannelta osapuolelta.

Nykyisiä rakenteisia menetelmiä käytetään harvoin yhtenevällä tavalla (Brown 1997). Menetelmien käyttämät käsitteet ovat liian taipumattomia suunnitelmien muuttamiseen koodauksen aloittamisen jälkeen. Suunnitelmien ylläpito vaatii paljon työtä, eivätkä suunnitelmat kerro järjestelmän ylläpitäjälle tarvittavia tietoja ohjelmiston kehittämiseksi. Nämä ongelmat korostuvat tekniikan muuttuessa ja kehitettäessä komponenttipohjaisena laajoja hajautettuja järjestelmiä. Tämän seurauksena on kehitettävä myös komponenttipohjaisessa ohjelmistokehityksessä käytettäviä menetelmiä. Menetelmien on tuettava uusien osien lisäämistä olemassa olevaan ohjelmistoon sekä toisaalta ohjelmistokehityksen aloittamista alusta ja laajojen ohjelmistojen päivittämistä. Menetelmän on myös mukauduttava erikokoisiin tehtäviin, tuettava komponenttien valintaan ja analysointiin liittyvän päätöksenteon dokumentointia sekä komponenttien rajapintojen määrittelyä ja analysointia. Menetelmien kehittyessä on tärkeää, että myös välineitä kehitetään tukemaan selkeästi menetelmien uusia piirteitä. *Oliosuuntautuneet analysointi- ja suunnittelumenetelmät* (object-oriented analysis and design, OOAD) ovat olleet viime vuosina suosittuja, niiden kasvuvaihe jatkuu ja ne tulevat kehittymään edelleen. Sitol (2002) on esittänyt katsauksen tämänhetkisiin OOAD-menetelmiä tuke-

viin välineisiin ja niiden ominaisuuksiin.

3.5.3 Muita välineiden kehitysnäkymiä

Tyypillistä nykyiselle ja myös tulevalle kehitykselle on joka alalla kaikkialle ulottuva tietokoneistaminen. Haasteina ovat uudet teknologiat, kuten kannettavuus, liikkuvuus, langattomuus ja laitteiden yhä lisääntynyt epäyhtenäisyys (Harrison et al. 2000). Rajoitteina ovat erityisesti virran saanti laitteistoille ja ohjelmistojen käytettävyys. Toisaalta on myös tarve saada samat ohjelmat toimimaan useissa erilaisissa laitteissa ja käyttöjärjestelmissä. Palvelujen siirtämisessä erilaisille laitteille tarvitaan tietosisältöjen mukauttamista, jotta laitteet pystyisivät käsittelemään tiedot oikein. Tällaista erilaisten tietosisältöjen mukauttamisprosessia tiedon esittämiseen ja vuorovaikutukseen erilaisten ympäristöjen välillä kutsutaan *siirtokoodaukseksi* (transcoding) (Hori et al. 2000). Siirtokoodauksen suorittamisessa välineiden tuki on tärkeä. Välineen avulla käyttäjän on pystyttävä helposti määrittelemään siirtokoodauksen suoritustapa. Siirtokoodausta tarvitaan myös kehitysympäristöjen toiminnoissa. Tietyssä kehitysympäristössä kehitetty ohjelmisto on pystyttävä muuntamaan toisenlaisissa laitteissa toimiviksi ilman suurta uudelleentehtävää kehitystyötä.

Yksi haaste tulevalle kehitykselle on internetin käytön mukanaan tuoma *elektronisen kaupankäynnin* (e-commerce) lisääntyminen. Elektroninen kaupankäynti poikkeaa perinteisistä sovellusalueista. Kun internetin kautta välitetään tietoa rahan vaihtoon, luottokortteihin ja henkilötietoihin liittyen, on sovelluksille erittäin kriittistä niiden turvallisuus ja luotettavuus. Lisäksi on otettava huomioon ohjelmiston samanaikaisten käyttäjien vaihteleva, joskus hyvinkin suuri lukumäärä (Harrison et al. 2000). Ohjelmiston tietokantapalveluiden on pystyttävä selviytymään luotettavasti ja nopeasti suuresta käyttäjämäärästä ilman palvelun ruuhkautumista ja hidastumista. Edellä kuvattujen ominaisuuksien toteuttamisessa nykyisten menetelmien ja välineiden tuki on ollut puutteellista, joten yhtenä tavoitteena välineiden kehityksessä on tuen lisääminen edellä esitetyille ominaisuuksille. Välineiltä odotetaan myös enemmän tukea ja laajuutta hallintaan liittyviin toimintoihin, esimerkiksi esitystavan, visualisoinnin ja ohjelmiston monimutkaisuuden hallintaan.

4 Vaatimusmäärittelyn välineet

Vaatimusmäärittely on ensimmäinen toiminto ohjelmistoprosessin suorittamisessa. Se on myös erittäin kriittinen vaihe, koska vaatimusmäärittelyssä tehdyt virheet johtavat väistämättä myöhemmin ongelmiin järjestelmän suunnittelussa ja toteutuksessa (Sommerville 2000). Tässä luvussa tarkastellaan lähemmin vaatimusmäärittelyä osana ohjelmistotuotantoa sekä tutustutaan vaatimusmäärittelyssä käytettyihin välineisiin. Aluksi tutustutaan vaatimusmäärittelyn ominaispiirteisiin ja vaikutuksiin muihin ohjelmistoprosesseihin. Seuraavaksi kohdassa 4.2 tarkastellaan vaatimusmäärittelyssä käytettyjä välineitä ja niiden tukea vaatimusmäärittelyprosessien parantamisessa. Kohdassa 4.3 tutustutaan Joensuun yliopistossa kehitettyyn Sfrm-vaatimusmäärittelyohjelmaan ja selvitetään sen ominaisuuksia vaatimusmäärittelyprosessien parantamisen näkökulmasta.

4.1 Vaatimusmäärittelyn asema ohjelmistotuotannossa

Vaatimusmäärittely on keskeisin vaihe koko ohjelmistotuotantoprosessissa ja se on pohjana kaikkien muiden osaprosessien toiminnoissa. Vaatimusmäärittelyn lopputuloksena saatavassa määrittelydokumentissa kuvataan (Santana Filho ja Kochan 2001):

- kaikki ohjelmiston toiminnalliset ominaisuudet,
- rajoitteet, joiden puitteissa ohjelmiston on toimittava,
- ei-toiminnalliset ominaisuudet, kuten laitteisto-, laatu- ja suorituskykyvaatimukset ja
- riittävästi tietoa ohjelmiston sovellusalasta suunnittelua ja toteutusta varten.

Vaatimusmäärittelyprosessin lopputuloksena saatavaa vaatimusmäärittelydokumentti on perusta kaikelle ohjelmistosuunnittelulle, toteutukselle ja testaukselle. Myös ohjelmiston elinkaaren myöhemmissä vaiheissa ylläpidossa ja ohjelmistoa muutettaessa vaatimusmäärittely on keskeinen toimintoja ohjaava dokumentti. Organisaatiotasolla vaatimusmäärittely on lähtötietona riskianalyysien, kustannusarvioiden ja resurssisuunnitelmien tekemisessä. Näiden kaikkien eri tekijöiden kautta se ohjaa myös ohjelmistotuotteen lopullista laatua (Santana Filho ja Kochan 2001).

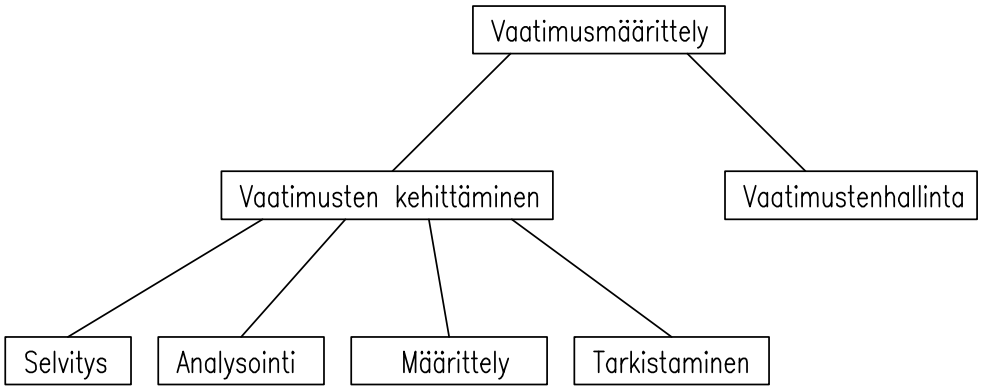
Koska vaatimusmäärittely ohjaa keskeisesti kaikkia muita ohjelmistotuotannon vaiheita, se on erittäin kriittinen koko projektin onnistumisen kannalta. Täsmällinen ja mahdollisimman täydellinen vaatimusmäärittely luo pohjan myös muiden projektin osien onnistumiselle. Vaatimusmäärittelyn puutteet, ristiriidat ja virheet aiheuttavat aina ongelmia, kun ne havaitaan myöhemmissä vaiheissa. Virheiden korjauksessa työmäärä lisääntyy, aikataulut viivästyvät ja riskit kertaantuville uusille virheille kasvavat. Jos virhettä ei havaita lainkaan ohjelmistoprojektin aikana, asiakkaalle toimitettava ohjelmisto joko toimii virheellisesti tai ei kaikilta osiltaan täytä asiakkaan käyttötarpeita. Mitä aikaisemmassa vaiheessa virheet ja puutteet havaitaan, sitä helpompaa ja nopeampaa niiden korjaus on. Davis (1993) on tutkimuksessaan käsitellyt virheen korjauksen suhteellisia kustannuksia ohjelmistoprojektin eri vaiheissa. Sen mukaan suunnitteluvaiheessa havaitun virheen korjaaminen on viisi kertaa kalliimpaa kuin vaatimusmäärittelyssä havaitun virheen korjaaminen. Jos virhe havaitaan vasta hyväksymistestin yhteydessä, on suhteellinen korjauskustannus jo viisikymmenkertainen.

Vaatimusmäärittelyprosessi ei ole pelkästään tekninen prosessi. Se on hyvin paljon ihmisten välistä kanssakäymistä ja vaatii monien eri tieteenalojen soveltamista. Hofmann ja Lehner (2001) kuvaavat, että vaatimusmäärittely on oppimista, ajatusten vaihtamista ja neuvotteluja ja siinä onnistuminen edellyttää teknisten, kognitiivisten, sosiaalisten ja organisaatiotason prosessien yhdistämistä projektin erityistarpeisiin ja luonteeseen sopiviksi.

4.1.1 Vaatimusmäärittelyn ominaispiirteitä

Vaatimusmäärittelyn (requirements engineering) sisältöä on selvitetty kuvassa 5. Vaatimusmäärittely jakaantuu kahteen pääryhmään, varsinaisen vaatimusmäärittelyn laadintaan ja vaatimustenhallintaan (requirements management). Vaatimusmäärittelyn laatiminen toteutetaan ohjelmistotuotantoprosessin alussa ja se on edellytyksenä varsinaiselle ohjelmiston suunnittelulle ja toteutukselle. Vaatimustenhallintaa puolestaan tarvitaan koko ohjelmistotuotantoprosessin aikana sekä ohjelmiston ylläpitovaiheessa mahdollisten virheiden korjaustoimien tai toteutettavien muutosten yhteydessä. Ohjelmiston hyväksymisvaiheessa vaatimusmäärittelyllä on tärkeä rooli. Hyväksymistestissä järjestelmän toimintaa verrataan vaatimusmäärittelyyn ja sen avulla todetaan, että ohjelma on asiakkaan tarpeiden mukainen ja toimii halutulla tavalla.

Vaatimusmäärittelyprosessissa on mukana useita eri osallistujaryhmiä ja ryhmien väli-



Kuva 5: Vaatimusmäärittelyn sisältö (Wiegers 1999).

nen yhteistyö on tärkeää hyvään lopputulokseen pääsemiseksi (Lang ja Duggan 2001). Ohjelmiston toteutukseen osallistuvien eri tahojen lisäksi tärkeinä tekijöinä ovat asiakas ja ohjelmiston tulevat käyttäjät. Tämä korostuu vaatimusten kehittämisen aikana, mutta heidän roolinsa on tärkeä myös vaatimuksista sopimisen ja vaatimusten tarkistuksen yhteydessä. Vaatimusten selvittämisellä pyritään selvittämään ohjelmiston toimiala, järjestelmän tarjoamat palvelut sekä järjestelmän toiminnalliset rajoitteet (Sommerville 2000). Vaatimusten selvittämisessä voidaan käyttää monia erilaisia menetelmiä, esimerkiksi haastatteluja, kyselyjä, dokumentteihin tutustumista, työskentelyä mukana projektissa, tarkkailua, skenaarioita ja prototyyppejä. Käytetyt menetelmät valitaan tapauskohtaisesti riippuen sovellusalasta ja ohjelmiston koosta. Eri menetelmiä käytetään usein rinnakkain, jotta saadaan kerätyksi kattava lähtöaineisto analysoinnin pohjaksi.

Vaatimusten analysointi liittyy läheisesti vaatimusten selvittämiseen. Analysoinnilla pyritään lisäämään kohdealueen tuntemusta, kokoamaan tarvittavat vaatimukset, luokittelemaan ja priorisoimaan vaatimuksia sekä poistamaan ristiriitaisuudet (Sommerville 2000). Tämä on erityisen tärkeä vaatimusmäärittelyn vaihe, jotta vaatimukset saadaan esitettyä täsmällisesti ja projektin jatkon kannalta toteutuskelpoisella tavalla. Tähän pääsemiseksi yksittäisen vaatimuksen on oltava ytimekäs, suunnitelmaan liittyvä, toteuttamiskelpoinen, tarkka, johdonmukainen ja todistettava (Lang ja Duggan 2001). Lisäksi vaatimukset on priorisoitava, järjestettävä selkeäksi rakenteeksi sekä niiden on oltava yksiselitteisiä ja ymmärrettäviä, jäljitettävissä olevia ja ylläpidettäviä. Tulosten

esittäminen täsmällisesti ja johdonmukaisesti pelkästään luonnollisella kielellä on vaikeaa. Sen vuoksi on kehitetty erilaisia mallinnusmenetelmiä ja määrittelykieliä. Menetelmien avulla vaatimuksista ja järjestelmästä erotetaan erilaisia näkökulmia, kuten tieto, toiminnallisuus ja käyttäytyminen (Hofmann ja Lehner 2001). Eri näkökulmat esitetään luomalla järjestelmästä erillisiä malleja. Mallit voidaan esittää esimerkiksi tekstimuotoisena tai graafisesti erilaisina kaavioina.

Yksittäiset vaatimukset voidaan tallentaa joko tietokantaan tai tekstitiedostoon. Riippumatta tallennustavasta vaatimusmäärittelystä tarvitaan raportteja. Tietokannan etuna on, että erilaisten raporttien muodostaminen on helppoa, kun taas tekstimuodossa olevat vaatimukset on laadittava tulostettavan raportin vaatimusten ja esitystavan mukaisesti. Vaatimusmäärittelyn käyttäjiä ovat toisaalta asiakas ja ohjelmiston loppukäyttäjät sekä järjestelmän suunnittelijat, testaajat, ylläpitäjät ja johtajat. Koska eri käyttäjien mielenkiinto ja käsittelytapa vaatimusmäärittelyssä poikkeavat toisistaan, vaatimusmäärittely esitetään usein kahtena erillisenä dokumenttina. Aluksi vaatimukset esitetään vaatimusmäärittelydokumenttina (requirements definition) siten, että sekä asiakas että kehittäjät ymmärtävät sen samalla tavalla ja ovat samaa mieltä siitä, mitä järjestelmän pitäisi tehdä. Sen jälkeen vaatimukset esitetään uudelleen täsmennettyinä ja toteutuksen näkökulmasta tarkasteltuna vaatimusspesifikaationa (requirements specification). Esitystapa voi olla muodollisempi ja matemaattisempi siten, että suunnittelijat voivat käyttää vaatimuksia paremmin hyväkseen siirryttäessä järjestelmän suunnitteluun (Pfleeger 2001).

Vaatimusten tarkistaminen tarkoittaa vaatimusmäärittelyn oikeellisuuden ja täydellisyyden muodollista tarkistamista. Tarkistaminen sisältää vaatimusten kelpoistamisen (validation) ja todentamisen (verification). Kelpoistaminen suoritetaan yhdessä toimittajan ja asiakkaan edustajien kanssa ja siinä varmistetaan, että vaatimusmäärittelyn avulla kuvattu järjestelmä on sellainen kuin asiakas on tarkoittanut ja että se täyttää asiakkaan tarpeet. Todentamisella varmistetaan, että vaatimukset ovat täydellisiä, oikein ja ne ovat keskenään johdonmukaisia ja ristiriidattomia (Hofmann ja Lehner 2001).

Vaatimustenhallinnassa voidaan erottaa ohjelmiston elinkaaren aikana kolme erillistä vaihetta, järjestäytyminen, toteuttaminen ja ylläpitäminen (SQAS 2000). Järjestäytymisvaihe on heti ohjelmistoprojektin alussa ja siinä suunnitellaan projektin vaatimustenhallinnan toteuttaminen, määritellään resurssit, osallistujat ja ympäristö vaatimustenhallinnan toteuttamiselle. Toteutusvaihe käsittää vaatimusten keruun, dokumentoin-

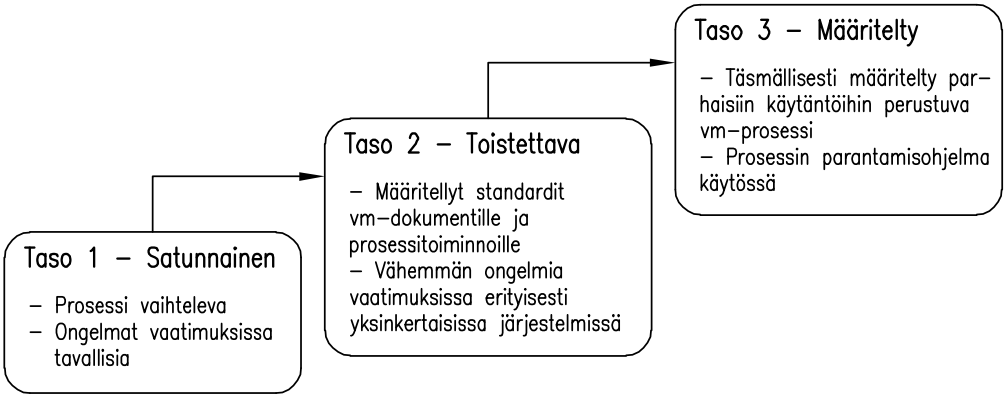
nin, varmentamisen ja muutosten hallinnan. Ylläpitovaihe on pohjimmiltaan edellisiä vaiheita toistava ja kehittävä ohjelmiston vaatimusten muuttumisen seurauksena. Tämän tuloksena saadaan päivitetyt versiot järjestelmämalleihin, määrittelyihin, suunnitelmiin ja ohjelmistoihin sekä parannettu vaatimustenhallintaprosessi.

4.1.2 Vaatimusmäärittelyprosessien parantaminen

Ohjelmistoprosessien kypsyyden arviointiin ja parantamiseen yleisesti käytetyt viitekehykset eivät riittävästi huomioi vaatimusmäärittelyn erityispiirteitä (Sawyer et al. 1998). Vaatimusmäärittely on niissä tyypillisesti käsitelty yhtenä toimintona yleisessä ohjelmistokehitysprosessissa. Kun vaatimusmäärittelyn eri toimintoja ei ole esitetty riittävällä tarkkuudella, ei myöskään prosessien parantamissuunnitelmien kehittäminen ole mahdollista. Kuten edellisessä kohdassa todettiin, on vaatimusmäärittely kuitenkin keskeinen ja tärkeä vaihe koko ohjelmistoprojektin kannalta. Tämän vuoksi prosessien parantaminen vaatimusten keräämisessä, dokumentoinnissa ja hallinnassa on kriittinen koko ohjelmistotuotantoprosessin parantamisen ja menestyksekkään liiketoiminnan kannalta.

Yleisten viitekehysten täydentämiseksi vaatimusmäärittelyn erityispiirteillä Sawyer et al. (1998) esittelevät kolmitasoisen vaatimusmäärittelyprosessin kypsyysmallin (kuva 6) ja siihen liittyvän vaatimusmäärittelyn hyvien käytäntöjen oppaan (requirements engineering good practice guide) (REAIMS 1997, Sommerville ja Sawyer 1997). Näiden avulla organisaatiot voivat arvioida nykyistä vaatimusmäärittelyprosessiaan sekä suunnitella ja asteittain toteuttaa parannuksia. Kypsyysmallin ensimmäinen ja toinen taso, satunnainen ja toistettava, ovat jokseenkin CMM-mallin ensimmäisen ja toisen tason mukaisia. Sen sijaan kolmas taso, määritelty, sisältää CMM-mallin ylimmät tasot (määritelty, hallittu ja optimoiva). Satunnaisella tasolla organisaatiolla ei ole määriteltyä vaatimusmäärittelyprosessia eikä menettelytapoja sen toteuttamiseksi. Laadukkaiden määrittelydokumenttien toteuttaminen määräajassa ja kustannusraameissa on usein vaikeaa ja organisaatio on riippuvainen yksittäisten henkilöiden taidoista ja kokemuksesta. Toistettavalla tasolla organisaatiolla on määriteltynä standardit vaatimusten kuvaamiseksi ja määrittelydokumenttien laatimiseksi sekä käytössä menettelytavat vaatimustenhallintaa varten. Käytössä voi olla myös muutamia edistyneitä välineitä prosessien suorituksen tukemiseksi. Määrittelydokumentit pystytään yleensä laatimaan aikataulun mukaisesti ja ne ovat sovitun laatutason mukaisia. Määritellyllä tasolla or-

ganisaatiolla on määriteltynä ja käytössä hyviin käytäntöihin ja tekniikkaan perustuva vaatimusmäärittelyn prosessimalli. Käytössä on myös aktiivinen prosessin parantamisohjelma ja käytettyjä menetelmiä ja tekniikoita pystytään arvioimaan riittävän täsmällisesti.



Kuva 6: Vaatimusmäärittelyprosessin kypsyystasot (Sawyer et al. 1998).

Hyvien käytäntöjen sisältö vaihtelee pienistä yksittäistä toimintoa kuvaavista laajempiin, eri tekniikoiden, menetelmien tai CASE-välineiden käyttöä sekä prosessin muuttamista kuvaaviin käytäntöihin. Jokaisesta käytännöstä kuvataan sen hyödyt sekä arvio käyttöönottokustannuksista. Sopivien käytäntöjen valinnan helpottamiseksi ne on ohjeistettu ja järjestelty vaatimusmäärittelyn prosessitoimintojen mukaisiksi ryhmiksi käsittäen vaatimusmäärittelydokumentin, vaatimusten keräämisen, analysoinnin, neuvottelut, järjestelmämallinnuksen, tarkistuksen ja vaatimustenhallinnan. Ohjeissa käytännöt on edellisten lisäksi luokiteltu perus-, keski- ja edistyneen tason ryhmiksi. Perustason käytännöt ovat yksinkertaisia standardointiin, hallintaan ja käytettävyyteen keskittyviä. Ne luovat pohjan toistettavalle vaatimusmäärittelyprosessille ja ovat yleensä suhteellisen halpoja toteuttaa ja käyttää. Keskitason käytännöt ovat vaativampia ja niiden avulla saadaan määritelty vaatimusmäärittelyprosessi. Ne keskittyvät systemaattisten ja rakenteisten menetelmien käyttöönottoon. Keskitason käytäntöjen käyttöönotto vaatii perustasoa enemmän aikaa ja niiden toteutus on kalliimpaa. Edistyneen tason käytännöt tukevat jatkuvaa vaatimusmäärittelyprosessia. Ne voivat vaatia erikoisasiantuntemusta ja muutoksia organisaatiotasolla. Käyttöönottokustannukset voivat vaihdella. Edistynyttä tekniikkaa vaativat käytännöt voivat olla kalliita, kun taas toiset voivat olla suhteellisen halpoja, mutta ne vaativat organisaatiotason muutoksia.

Ensimmäinen toimenpide vaatimusmäärittelyprosessin parantamisessa ja kypsyyden arvioimisessa on tämänhetkisen prosessin arvioiminen. Sen avulla saadaan selville, kuinka hyvin prosessi on määritelty ja mitkä ovat sen heikkoja alueita. Tämän jälkeen voidaan aloittaa prosessin parantamisen suunnittelu. Suunnittelun aikana voidaan esittää neljä peruskysymystä, joihin olisi löydettävä vastaus (Sawyer et al. 1998):

- Mitkä ovat nykyisen prosessin ongelmat?
- Mitkä ovat parannusten tavoitteet?
- Kuinka prosessien parannustoimenpiteet on otettava käyttöön, jotta asetetut tavoitteet voidaan saavuttaa?
- Kuinka parannustoimia tulee valvoa ja hallita?

Kun prosessin parantamistavoitteet on määritelty, valitaan tavoitteiden saavuttamiseksi tarvittavat käytännöt. Parannustoimien käyttöönottojärjestys riippuu prosessista ja siihen tarvittavista parannuksista. Jos lähtökohtana on määrittelemätön prosessi, esittävät Sawyer et al. (1998) toteutuksen perusjärjestykseksi:

- Määritellään dokumentille standardirakenne.
- Tehdään dokumentti helposti muutettavaksi.
- Jokaiselle vaatimukselle annetaan yksilöllinen tunniste.
- Määritellään toimintaperiaatteet vaatimustenhallinnalle.
- Määritellään standardimalli vaatimuksen kuvaukselle.
- Käytetään yksinkertaista, yhdenmukaista ja ytimekästä kieltä.
- Järjestetään muodollisten vaatimusten katselmukset.
- Määritellään tarkistuslistat tarkistuksia varten.
- Käytetään tarkistuslistoja vaatimusten analysoinnissa.
- Ennakoidaan mahdollisia ristiriitoja ja suunnitellaan niille ratkaisumallit.

Edellä esitetyt käytännöt keskittyvät dokumentointiin ja vaatimustenhallintaan. Koska ne ovat myös suhteellisen edullisia toteuttaa, päästään näiden käytäntöjen avulla pienillä kustannuksilla toistettavan vaatimusmäärittelyprosessin alkuun.

Kuten ohjelmistoprosessin parantaminen yleensä, myös vaatimusmäärittelyprosessin parantaminen ja käyttöönottaminen on vaativa tehtävä. Kauppinen et al. (2004) ovat esittäneet laajan kirjallisuuteen perustuvan katsauksen avaintekijöistä, joilla on merkittävä vaikutus vaatimusmäärittelyprosessien käyttöönoton onnistumiseksi koko organisaatiossa. Yksi tärkeimmistä tekijöistä on prosessiin osallistuvien henkilöiden mukanaolo. Tällöin prosessi voidaan kehittää sellaiseksi, että siitä on eniten hyötyä siihen osallistuville. Myös kehitetyn prosessin hyväksyminen henkilöstön keskuudessa on helpompaa. Tästä käy selkeästi esille myös se, että prosessien parantamisessa on korostettava tiimityöskentelyn tärkeyttä. Uusista prosesseista tulisi myös olla hyötyä kaikille osallistujille. Prosessien muuttaminen vaatii myös organisaation kulttuurin muuttamista. Kehityshenkilöstön on ymmärrettävä asiakkaan ja loppukäyttäjien vaatimusten tärkeys sekä sitouduttava määrittelemään ja hallitsemaan vaatimuksia systemaattisesti. Jatkuvaan vaatimusmäärittelyprosessiin pyrittäessä on tärkeää asiantuntijan tuki, mittaustieto ja korjaavat toimet. Jotta prosessista saadaan kehittyvä, toteutettujen parannusten on oltava mitattavia ja analysoitavissa. Toteutus pitäisi aloittaa muutamilla pienillä ja edullisilla, mutta suhteellisesti enemmän hyötyä tuovilla parannuksilla ennen kalliin ja uuden tekniikan käyttöönottoa. Uusia muutoksia on myös hyödyllistä testata kokeiluprojekteissa, jotta voidaan havaita muutosten edut ja haitat sekä mahdollisesti korjata niitä ennen laajempaa käyttöä koko organisaatiossa. Muutosten yhteydessä ei saa unohtaa koulutuksen ja harjoittelun merkitystä. Prosessien, menetelmien ja välineiden tulisi myös olla yksinkertaisia ja selkeitä, jolloin niiden oppiminen ja käyttäminen on helpompaa.

Systemaattisen vaatimusmäärittelyn tekemisen ja tässä tarvittavien menetelmien käyttöönoton helpottamiseksi Nikula (2004) on kehittänyt *BaRE-menetelmän* (Basic Requirements Engineering). Tavoitteena on menetelmän helppo ja nopea käyttöönotto pienillä resursseilla sekä helppokäyttöisyys ja menetelmän joustavuus erilaisiin käyttötarpeisiin sovitettaessa. Helppokäyttöisyyteen pyrittäessä menetelmä soveltaa yksinkertaisia, standardoituja tekniikoita ja keskittyy vaatimusmäärittelyn perusasioihin. Menetelmä käsittää käyttövalmiiksi sovitettuina dokumenttipohjat, kuvaa tekniikat ja prosessit vaatimusten kehittämiselle ja vaatimustenhallinnalle, välinetuen vaatimustenhallinnalle sekä koulutuksen. Rajoitteina menetelmän käytölle ovat sovellusalue sekä

organisaation ja kehitettävän ohjelmiston koko. Sovellusalueiksi on rajoitettu pienet hallinnolliset ja liiketoimintajärjestelmät, jolloin dokumenttipohjat on voitu sovittaa valmiiksi näihin sopiviksi. Projektien on oltava kooltaan sellaisia, että vaatimusmäärittelyn voi laatia yksi henkilö ja projektin kokonaiskesto maksimissaan kuuden henkilön toteuttamana on puolesta vuodesta vuoteen. BaRE-menetelmän toimivuutta käsitelleessä tutkimuksessa (Nikula 2004) todettiin, että yritykset pystyivät muutamassa kuukaudessa parantamaan vaatimusmäärittelytoimintaansa merkittävästi vain muutaman työpäivän panostuksella.

4.1.3 Vaatimusmäärittelyn tulevat kehityssuunnat

Vaatimusmäärittelyyn liittyvä tutkimus ja kehitys on ollut voimakasta 1990-luvun alkupuolelta lähtien. Vaikka tuloksia on jo saatukin, on ohjelmistotuotannon kehittyminen niin nopeaa, että haasteita riittää myös tulevaisuuteen. Nuseibeh ja Easterbrook (2000) luovat katsauksen vaatimusmäärittelyyn liittyvän tutkimuksen haasteisiin. Heidän mukaansa yhtenä kehityssuuntana, joka on otettava huomioon myös vaatimusmäärittelyssä, on komponenttipohjaisen ohjelmistokehityksen lisääntyminen. Ohjelmistokomponentteja tullaan käyttämään erilaisissa ohjelmistoissa ja käyttöjärjestelmissä. Tällöin vaatimusmäärittely pitäisi pystyä tekemään yksittäiselle komponentille ja määrittelemään sille oma toimintaympäristönsä. Sen vuoksi on kehitettävä uutta tekniikkaa erilaisten ympäristöjen muodollista mallintamista ja analysointia varten.

Komponenttien käytön lisäksi myös vaatimusten ja mallien uudelleenkäyttö tulee lisääntymään. Monilla sovellusaloilla otetaan tällöin käyttöön viitemallit vaatimusten määrittelyssä. Tämän seurauksena vähenee tarve suunnitella mallit alusta alkaen uudelleen. Sen sijaan voidaan keskittyä tapauskohtaisten erojen mallintamiseen. Tähän liittyy myös Finkelsteinin ja Emmerichin (2000) näkemys siitä, että pitkällä aikavälillä tullaan siirtymään projektin eliniän kestävästä vaatimustenhallinnasta organisaatiotason tietämyksenhallintaan. Tähän kehitykseen tulee vaikuttamaan yleisen tietämyksenhallinnan ja oppivan organisaation kehittyminen ja suosio tulevaisuudessa.

Ohjelmistoarkkitehtuuria on viimevuosina tutkittu aktiivisesti (Finkelstein ja Emmerich 2000). Kuitenkin vaatimusten ja ohjelmistoarkkitehtuurin väliseen vuorovaikutukseen ei ole tähän asti kiinnitetty huomiota. Järjestelmälle sopivan arkkitehtuurin valinnassa ovat vaikuttaneet enemmän ei-toiminnalliset ominaisuudet, kuten suoritusteho ja kuormitus. Tutkimuskohteena tulisi olla esimerkiksi ohjelmistoarkkitehtuurissa

tehtyjen erilaisten valintojen vaikutukset nykyisiin ja tuleviin vaatimuksiin sekä mahdollisiin vaatimusten muunnoksiin koko tuoteryhmässä.

Vaatimusten keruussa on hyödynnettävä enemmän erilaisia lähteitä, kuten multimediaa, jotta vaatimukset voidaan paremmin hyödyntää muodollisessa mallintamisessa ja analysoinnissa. Myös ei-toiminnallisten vaatimusten keräämiseen ja analysointiin käytettyjä malleja on laajennettava. Nuseibeh ja Easterbrook (2000) korostavat myös monitieteellisen koulutuksen merkitystä vaatimusmäärittelyn laadintaan osallistuville henkilöille. Tarvitaan sekä sosiaalisia että teknisiä taitoja. Sosiaalisia taitoja tarvitaan, sillä vaatimusmäärittelyn laadinta edellyttää jatkuvaa kanssakäymistä erilaisten osallistujien kesken, joiden asiantuntemus, intressit ja tavoitteet voivat vaihdella. Teknistä tietämystä puolestaan tarvitaan kommunikointiin järjestelmän suunnittelijoiden, toteuttajien ja testaajien kanssa.

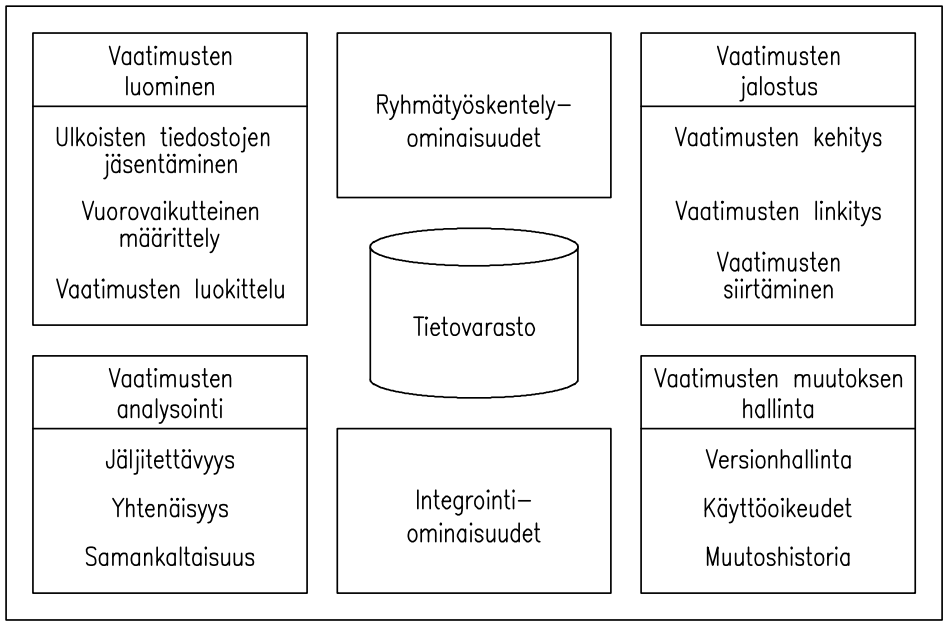
4.2 Välineiden erikoispiirteitä

Vaatimusmäärittelyyn liittyy monia erikoistoimintoja, jotka vaativat erityisesti näitä toimintoja tukevia välineitä. Tässä kohdassa tutustumme tarkemmin näihin erityispiirteisiin ja välineiden ominaisuuksiin. Lopuksi tarkastelemme, kuinka välineiden avulla voidaan tukea vaatimusmäärittelyprosessien suorittamista.

4.2.1 Välineiden ominaisuudet

Vaatimusmäärittelyssä käytettyjä välineitä kutsutaan usein nimellä *vaatimustenhallinnan välineet*. Yphise (2002) on tehnyt vertailevan tutkimuksen nykyisin käytössä olevista vaatimustenhallinnan välineistä ja niiden ominaisuuksista. Kuvassa 7 on esitetty keskeisimmät välineiden tukemat toiminnot. Vaatimustenhallinnan välineille keskeisin elementti on tietovarasto, jonne tallennetaan vaatimusten sisältö sekä muut liittyvät tiedot kuten analysointiin, vaatimusten muotoon ja raporttipohjiin liittyvä aineisto. Tietovarasto voidaan toteuttaa relaatio- tai objektitietokantana, sovelluskohtaisena tietokantana tai jopa tekstitiedostopohjaisena aineistona. Toiminnot kattavat kaikki vaatimusmäärittelyn osaprosessit, tosin eri sovellukset voivat keskittyä enemmän jonkin tietyn osaprosessin tukemiseen, jolloin muiden prosessien tuki voi olla vähäisempää.

Alkuvaiheen vaatimusten selvittämistä tukevia toimintoja ovat suora vuorovaikutteinen



Kuva 7: Vaatimustenhallinnan välineiden toiminnallinen kaavio (Yphise 2002).

vaatimusten määrittely tai ulkoisissa tiedostoissa olevan aineiston jäsentäminen osaksi vaatimusmäärittelyä sekä näihin liittyen vaatimusten luokittelu joko projektin sisäisesti tai hakemiston mukaisesti. Asiakkaalta saadut vaatimukset on jalostettava tarkoiksi toiminnallisiksi vaatimuksiksi ja kehitystehtäviksi. Välineet tarjoavat toiminnot vaatimusten linkittämiseen ja siirtämiseen osaksi projektin osaprosesseja. Vaatimusten analysoinnissa välineiden toiminnot tukevat jäljitettävyyttä, yhtenäisyyttä ja vaatimusten ryhmittelyä sekä vaatimusten tilan seurantaa projektin eri vaiheissa. Välineiden avulla voidaan myös analysoida vaatimusten muutosten vaikutuksia projektiin. Vaatimusten muutoksenhallinnassa välineiden tukemia toimintoja ovat käyttöoikeuksien hallinta, versionhallinta ja muutoshistorian ylläpito.

Erityispiirteenä vaatimustenhallinnan välineille on niiden ryhmätyöskentelyä tukevat toiminnot. Näitä toimintoja tarvitaan, jotta vaatimusmäärittelyn laadintaan osallistuvien, mahdollisesti maantieteellisesti eri paikkoihin hajautettujen henkilöiden työskentely ja yhtäaikainen pääsy keskitettyyn tietovarastoon olisi hallittua ja sujuvaa. Esimerkiksi samanaikainen vaatimusten määrittely, jatkokehitys ja muuttaminen ovat toimintoja, joissa ryhmätyön tuki on erityisen tärkeä. Henkilöiden välistä kommunikointia helpottaa välineiden tuki sähköpostin ja keskusteluryhmien käytölle.

Koska vaatimusmäärittely liittyy kiinteästi muihin ohjelmistotuotannon prosesseihin, on tärkeää, että myös välineet tukevat liittämistä muihin käytettyihin välineisiin, kuten suunnittelun, toteutuksen, testauksen ja projektinhallinnan välineet (Yphise 2002). Vaatimustenhallinnan välineet eivät välttämättä tue kaikkia tarvittavia toimintoja, vaan hyödyntävät tällöin erityisesti näitä toimintoja varten kehitettyjä välineitä. Esimerkiksi projektinhallinnan välineitä voidaan käyttää myös vaatimusmäärittelyprosessien etenemisen seurantaan, versionhallinnan välineitä yksittäisten vaatimusten, dokumenttien ja raporttien versionhallintaan sekä mallinnusvälineitä vaatimusten mallintamiseen. Testausvälineitä voidaan hyödyntää vaatimusten välisten suhteiden, testitapausten ja testitulosten ylläpidossa.

Myös Lang ja Duggan (2001) ovat tutkimuksessaan esittäneet vaatimuksenhallinnan välineiltä vaadittavia ominaisuuksia. Edellä esitettyjen lisäksi välineiden tulisi heidän mielestään ylläpitää yksilöllisiä tunnisteita kaikille vaatimuksille, tarkistaa vaatimusmäärittelyn ja toiminnallisen määrittelyn yhdenmukaisuus ja muodostaa sekä tilapäisiä raportteja että standardimallin mukaisia dokumentteja. Saumaton liittyminen muihin välineisiin ja järjestelmiin tukemalla yhteensopivia protokollia ja standardeja korostuvat myös Langin ja Dugganin esittämissä ominaisuuksissa. INCOSE (2002) esittää laajan katsauksen markkinoilla oleviin vaatimuksenhallinnan välineisiin ja niiden tukemiin ominaisuuksiin. Katsauksessa ominaisuudet on esitetty yksityiskohtaisesti ja ne sisältävät myös järjestelmävaatimukset, käyttöliittymän sekä toimittajan tarjoaman tuen, ylläpidon ja koulutuksen.

4.2.2 Välineiden tuki vaatimusmäärittelyprosesseille

Välineiden tuen merkitys vaatimusmäärittelyprosessien parantamisessa on hyvin samankaltainen kuin niiden merkitys on ohjelmistotuotantoprosessien parantamisessa yleensä. Prosessin kypsyyden kasvaessa myös välineiden tuen merkitys kasvaa. Sawyer et al. (1997) mukaan toistettavalla tasolla organisaatiolla voi olla käytössään muutamia edistyneitä välineitä prosessin suorittamisen tukemisessa, mutta määritellyllä tasolla välineiden tuki kaikissa prosessivaiheissa on välttämätöntä. Toisaalta myös vaatimusmäärittelyssä välineet yksin eivät saa aikaan parannuksia, vaan samalla tarvitaan pitkäjänteistä ja tavoitteellista toimintaa niin organisaatiotasolla kuin yksilöidenkin keskuudessa, jotta välineiden käytöltä odotetut parannukset onnistuisivat. Perusedellytyksenä on, että vaatimusmäärittelyn prosessit ovat määriteltyjä ja käytetyt vä-

lineet soveltuvat määrittelyn mukaiseen toimintaan. Kauppinen et al. (2004) toteavat, että välineiden avulla vaatimusmäärittelykäytännöt saadaan systemaattisiksi. Riskeinä he näkevät sen, että resurssien tarve aliarvioidaan välineiden käyttöönottamisessa ja käytön aikaisessa tuessa tai oletetaan, että välineet yksin ratkaisevat kaikki ongelmat. Hofmann ja Lehner (2001) toteavat, että välineet enemmän häiritsevät kuin tukevat vaatimusmäärittelyn toimintoja, jos prosessi on huonosti määritelty tai koulutus välineen käyttöön on puutteellista.

Perustuen edellä kuvattuihin ohjelmistotuotanto- ja vaatimusmäärittelyprosesseihin sekä niissä käytettyihin välineisiin esitän tämän alakohdan lopuksi mielestäni keskeisimmät välineiden ominaisuudet, joiden avulla voidaan tukea vaatimusmäärittelyprosessien suorittamista. Välineiden yksittäisistä ominaisuuksista keskeisin on niiden tarjoama keskitetty tietovarasto, josta on etua monien osaprosessien suorittamisessa. Mitä laajempaa järjestelmää määritellään, sitä vaikeampaa on jo yksittäisten vaatimusten käsittely ja hallinta ilman välineiden tukea. Kun vaatimukset on tallennettu keskitetysti, se luo edellytykset myös monien muiden toimintojen tukemiselle. Välineiden avulla voidaan tehostaa vaatimusten analysointia. Analysointi voi kohdistua esimerkiksi yksittäisen vaatimuksen ominaisuuksien tai vaatimusten välisen yhtenäisyyden ja ristiriidattomuuden tarkasteluun. Myös vaatimusten välisten riippuvuuksien ja jäljitettävyyden hallintaa on mahdollista tehostaa välineiden avulla.

Ryhmätyöskentelyssä välineiden tuki on tärkeää. Keskitetty tietovarasto tarjoaa yhtenäisen ja ajantasaisen tietolähteen kaikille projektissa samanaikaisesti jopa maantieteellisesti hajautettuina työskenteleville henkilöille. Ryhmätyöskentelyn tehokkaassa toteutuksessa tarvitaan välineiden tukea. Välineiden avulla voidaan hallita käyttöoikeuksia projektin tietovarastoon sekä pitää yllä tallennettujen tietojen muutoshistoriaa ja dokumenttien versionhallintaa.

Vaatimusten tietosisältöä voidaan yhtenäistää keruuvaiheessa määrittelemällä kuvaus vaatimuksesta tarvittavista minimitiedoista. Välineiden avulla voidaan helposti valvoa, että tietojen tallennuksessa noudatetaan tätä standardikuvausta. Samoin voidaan yhtenäistää dokumenttien sisältöä määrittelemällä standardoituja dokumenttimalleja. Välineiden avulla tietovarastosta voidaan tehdä hakuja erilaisilla kriteereillä sekä tulostaa tilapäisiä raportteja.

Edellä esitettyjen toimintojen avulla välineitä voidaan hyödyntää yksittäisten vaatimusmäärittelyn osaprosessien parantamisessa. Ne tukevat myös monilta osin Sawyer

et al. (1998) esittämää vaatimusmäärittelyprosessin parantamisen kymmentä peruskäytäntöä, jotka tulisi toteuttaa lähdettäessä parantamaan määrittelemätöntä prosessia. Vaatimusmäärittelyn välineitä voidaan käyttää myös laajemmin prosessien parantamisessa koko vaatimusmäärittelyprosessin ajalla sekä myös muiden ohjelmistotuotantoprosessien tukemisessa. Välineiden vaatimustenhallinnan toimintojen avulla voidaan parantaa koko vaatimusmäärittelyprosessinhallintaa ja siten omalta osaltaan koko projektin hallintaa. Tässä erityisen tärkeitä ovat välineiden tuki vaatimusten jäljitettävyyden parantamisessa, vaatimusten muutoshistorian ylläpitämisessä sekä kaikkien vaatimusmäärittelyyn liittyvien dokumenttien ja muiden työtulosten versionhallinnassa. Koska vaatimusmäärittely on perustana kaikille muille ohjelmistotuotannon osaalueille, kuten suunnittelulle, toteutukselle ja testaukselle, vaatimusmäärittelyssä käytettyjen välineiden mahdollisimman saumaton integrointi muiden CASE-välineiden kanssa on tärkeää. Tämä edesauttaa näiden muiden prosessien suoritusta, kun CASE-välineillä voidaan hyödyntää lähtötietoina vaatimusmäärittelyssä laadittuja dokumentteja, malleja ja muita tietoja. Erityisen tärkeää tässä on vaatimusmäärittelyn tietovaraston avoimuus ja yhdenmukaisuus CASE-välineiden kanssa. Vaatimustenhallinta ulottuu ajallisesti aina järjestelmän ylläpitoon asti. Järjestelmässä havaittujen korjaus- tai muutostarpeiden yhteydessä voidaan vaatimustenhallinnan välineiden avulla vaikuttaa merkittävästi tarvittavan työpanoksen määrään ja ajalliseen kestoon. Tässä on tärkeässä osassa välineiden muutoksen vaikutusten analysointia tukevat toiminnot. Välineiden avulla voidaan analysoida tarvittavan muutoksen vaikutus kaikkiin ohjelmistoprojektin osatekijöihin kuten määrittelydokumentteihin, suunnitelmiin, koodiin, testitapauksiin ja jopa toimituspakettien eri komponenttien versiotietoihin. Tämä auttaa toimenpiteiden oikeassa kohdistamisessa sekä aikataulu- ja resurssisuunnittelussa. Samalla saadaan järjestelmän vaatimusmäärittely päivitettyä ajan tasalla olevaksi ja muutoksen jälkeistä tilannetta vastaavaksi.

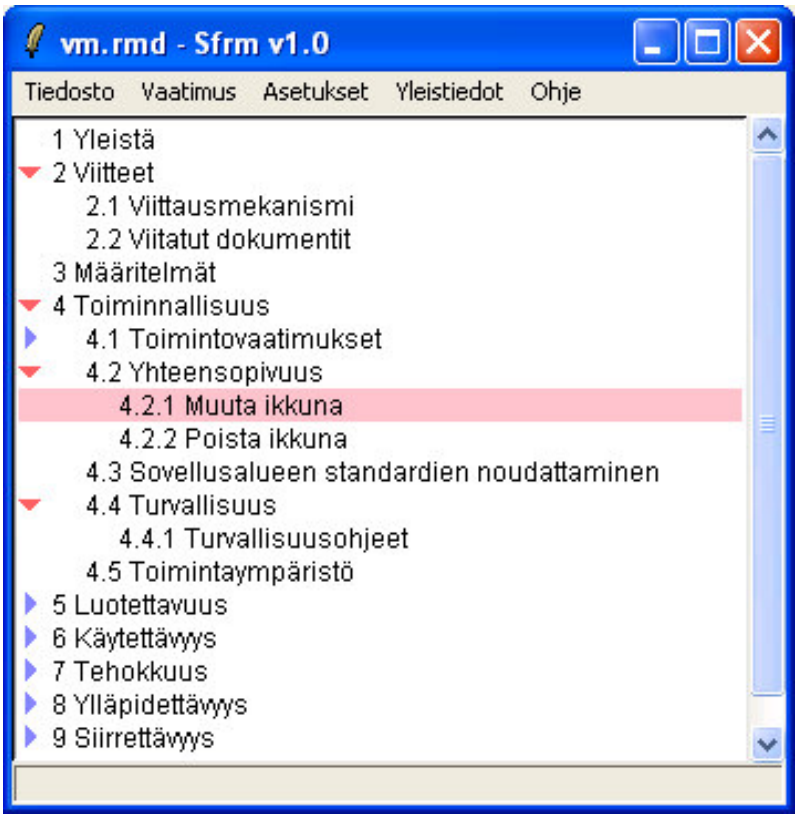
4.3 Sfrm-vaatimusmäärittelyohjelma

Sfrm (software for requirement management) (Kähkönen 2002) on Joensuun yliopistossa professori Sajaniemen ohjauksessa ja hänen määrittelyynsä perustuen erikoistyönä tehty vaatimusmäärittelyn laadintaan ja hallintaan tarkoitettu ilmaiseksi jaettava (shareware) ohjelmisto. Vaatimusmäärittelyn rakenne sekä vaatimuksista tallennettavat tiedot voidaan määritellä vaatimusmäärittelykohtaisesti. Yksittäisten vaatimusten ja vaatimuksesta tallennettavien tietojen lukumäärää ei ole rajoitettu. Vaatimuksiin voi-

daan liittää myös kuvia ja muuta grafiikkaa. Laaditusta vaatimusmäärittelystä voidaan muodostaa erilaisia HTML-tyyppisiä raportteja. Raportti muodostetaan määrittelemällä erilaisia valintakriteerejä, joissa määrätään, mitä vaatimuksia raportissa esitetään sekä mitkä yksittäiset vaatimuksen tiedot otetaan mukaan raporttiin. Kaikki ohjelman tarvitsemat määrittelytiedostot sekä vaatimusten tiedot tallennetaan XML-tyyppisiin tekstitiedostoihin, joten ne on luettavissa ja tarvittaessa muokattavissa millä tahansa tekstieditorilla. Sfrm-ohjelma toimii Unix-, Linux- ja Windows-käyttöjärjestelmissä.

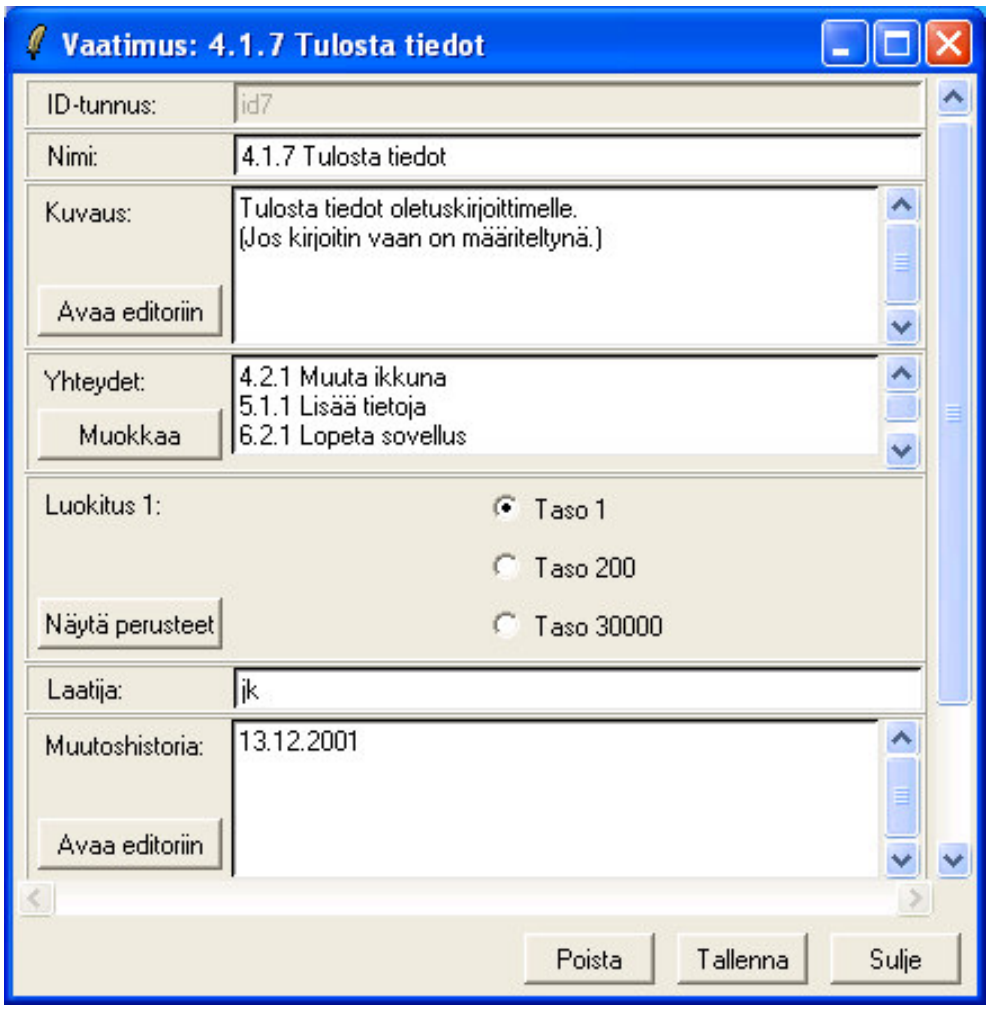
4.3.1 Kuvaus ohjelman toiminnasta ja ominaisuuksista

Vaatimusmäärittelyn avauksen yhteydessä ohjelma lukee asetustiedostosta vaatimusmäärittelyn sisällysluettelon rakenteen ja tulostaa sen hierarkkisena päälomakkeelle (kuva 8). Käyttäjä voi selata sisällysluetteloa sekä avata tai sulkea näkyvillä olevia luettelon tasoja. Vaatimusmäärittelyn avauksen yhteydessä ohjelma lukee asetustiedostosta myös tiedot vaatimuksen rakenteesta ja mahdollisesti jo aikaisemmin tallennettujen vaatimusten tiedot.



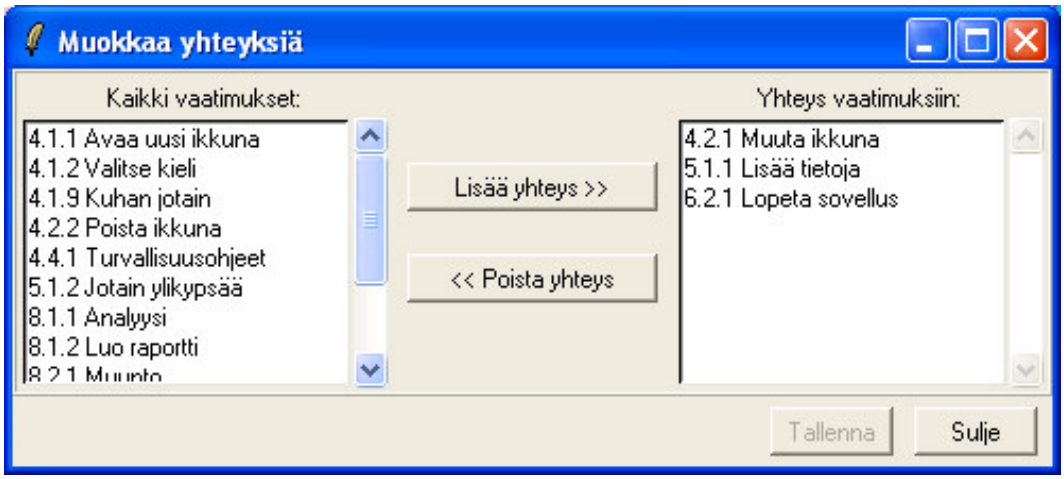
Kuva 8: Sfrm-ohjelman päävalikko.

Uuden vaatimuksen lisääminen aloitetaan valitsemalla päälomakkeen sisällysluettelosta haluttu, hierarkiassa alimmalla tasolla oleva otsikko. Ohjelma avaa kullekin vaatimukselle oman lomakkeen. *Vaatimus*-lomakkeella (kuva 9) esitetään vaatimuksesta asetustiedostossa määritellyt ominaisuudet. Käyttäjä voi lisätä lomakkeella vaatimukselle haluamansa tiedot tai muokata tekstikentän tietoja editorissa. Samanaikaisesti auki olevien vaatimusten lukumäärää ei ole rajoitettu. Ohjelma näyttää aktiivisena olevan vaatimuksen sijainnin pääikkunan hierarkiassa korostettuna. Uuden tallentamattoman vaatimuksen sijainti näytetään korostamalla sen otsikon nimi, jonka alle vaatimus on lisätty. Kun vaatimus tallennetaan, ohjelma lisää vaatimuksen nimen päälomakkeen hierarkiaan ja näyttää nyt sen korostettuna. Lisättävissä olevien vaatimusten lukumäärä on määritelty käyttäjäkohtaisesti asetustiedostossa. Ohjelman suorituksen aikana vaatimusten tiedot tallennetaan ohjelman muuttujiin ja viimeistään ennen istunnon lopettamista tiedot tallennetaan ulkoiseen tiedostoon.



Kuva 9: Vaatimus-lomakkeella voidaan syöttää tai muokata vaatimuksen tietoja.

Vaatimukseen voidaan linkittää toisia vaatimuksia, joihin valitulla vaatimuksella on jokin huomioon otettava toiminnallinen tai muu asiayhteys. Vaatimukseen lisätyt yhteydet näkyvät Vaatimus-lomakkeella, mutta yhteyksiä voidaan muokata erillisellä kuvan 10 mukaisella *Muokkaa yhteyksiä* -lomakkeella.

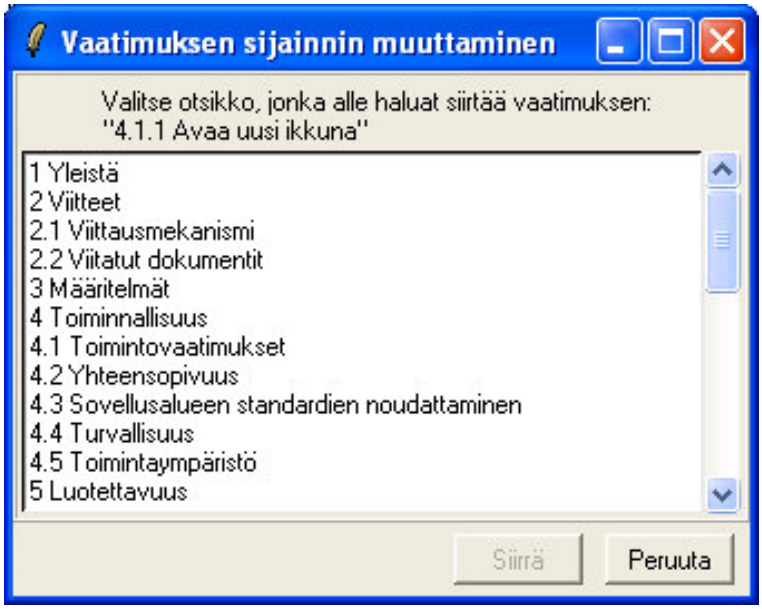


Kuva 10: Muokkaa yhteyksiä -lomakkeella voidaan lisätä tai poistaa valittuun vaatimukseen liittyviä yhteyksiä.

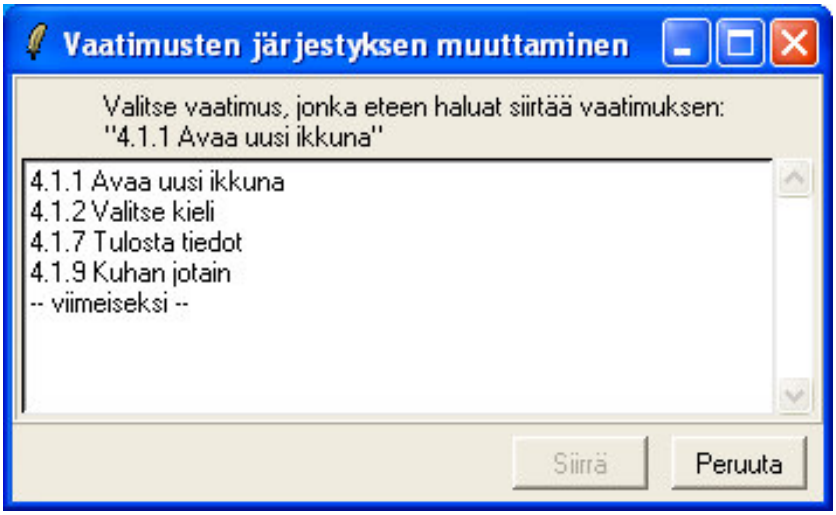
Jo lisättyjen vaatimusten järjestystä hierarkiassa voidaan muokata. Vaatimuksen sijaintia sisällysluettelon hierarkiassa voidaan muokata kuvassa 11 esitettävällä *Vaatimuksen sijainnin muuttaminen* -lomakkeella. Uusi tai siirretty vaatimus lisätään aina saman otsikon alla olevista vaatimuksista viimeiseksi. Näiden vaatimusten keskinäistä järjestystä voidaan muokata kuvassa 12 esitettävällä *Vaatimusten järjestyksen muuttaminen* -lomakkeella.

Vaatimusmäärittelystä voidaan luoda HTML-muotoisia raportteja. Kriteerit halutun raportin muodostamiseksi määritellään *Raportin luominen* -lomakkeella (kuvat 13 ja 14). Raportin esitystavaksi voidaan valita joko pelkkä vaatimusten luettelo tai sisällysluettelon mukainen tulostus otsikkotietoineen. Raporttiin sisällytettävien vaatimusten valintaperusteina voidaan käyttää merkkijonohakua tekstimuotoisille vaatimuksen osatiedolle tai luokitellun osatiedon ominaisuutta. Jos hakuehtoja asetetaan useita, on niiden kaikkien toteuduttava, jotta vaatimus otetaan mukaan raporttiin. Lisäksi määritellään, mitä osatietoja valituista vaatimuksista esitetään raportissa. Muodostettu raportti voidaan tallentaa ja haluttaessa avata selaimeen katseltavaksi.

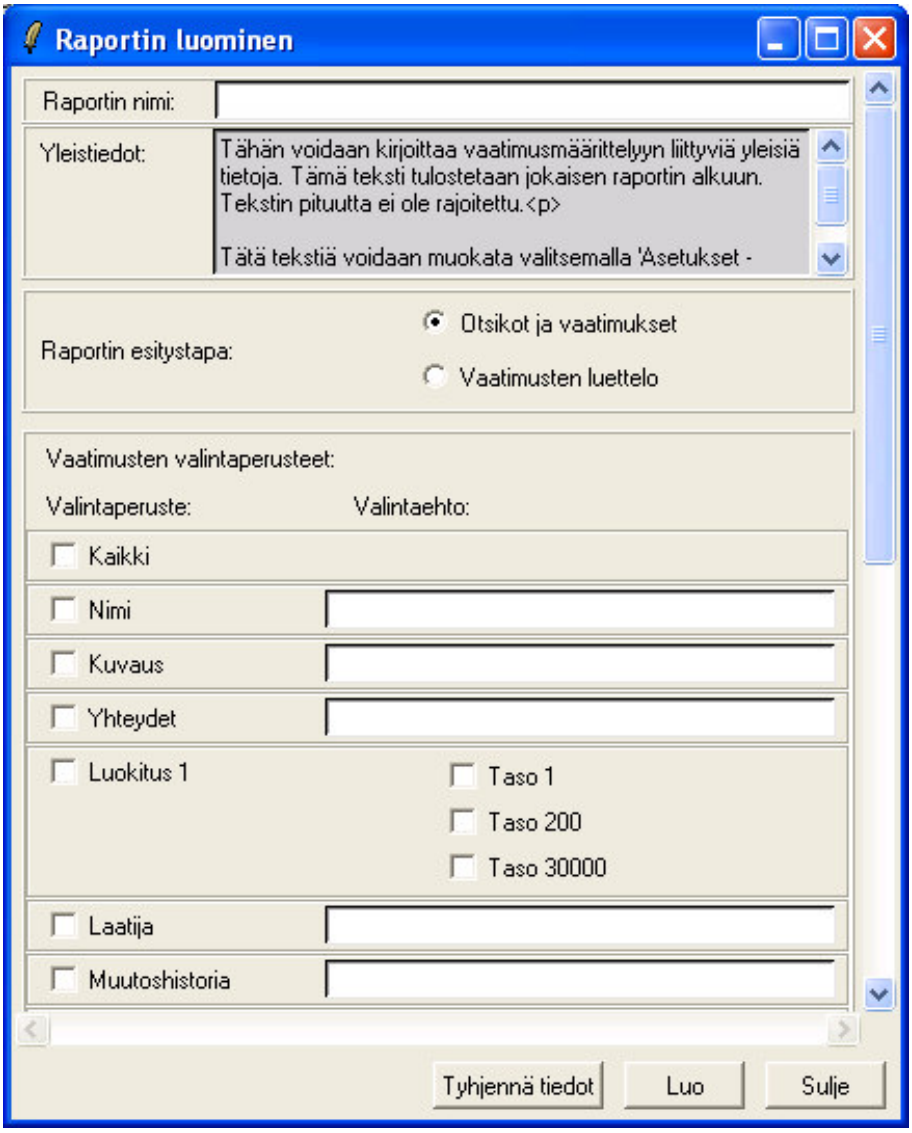
Ohjelma on laadittu *Tcl/Tk*-ohjelmointikielellä (Tool Command Language/Toolkit) (Welch et al. 2003). Ohjelman käyttämiseen tarvitaan neljä XML-tyyppistä tekstitie-



Kuva 11: Vaatimuksen sijainnin muuttaminen -lomakkeella voidaan muuttaa vaatimuksen sijaintia sisällysluettelon hierarkiassa.



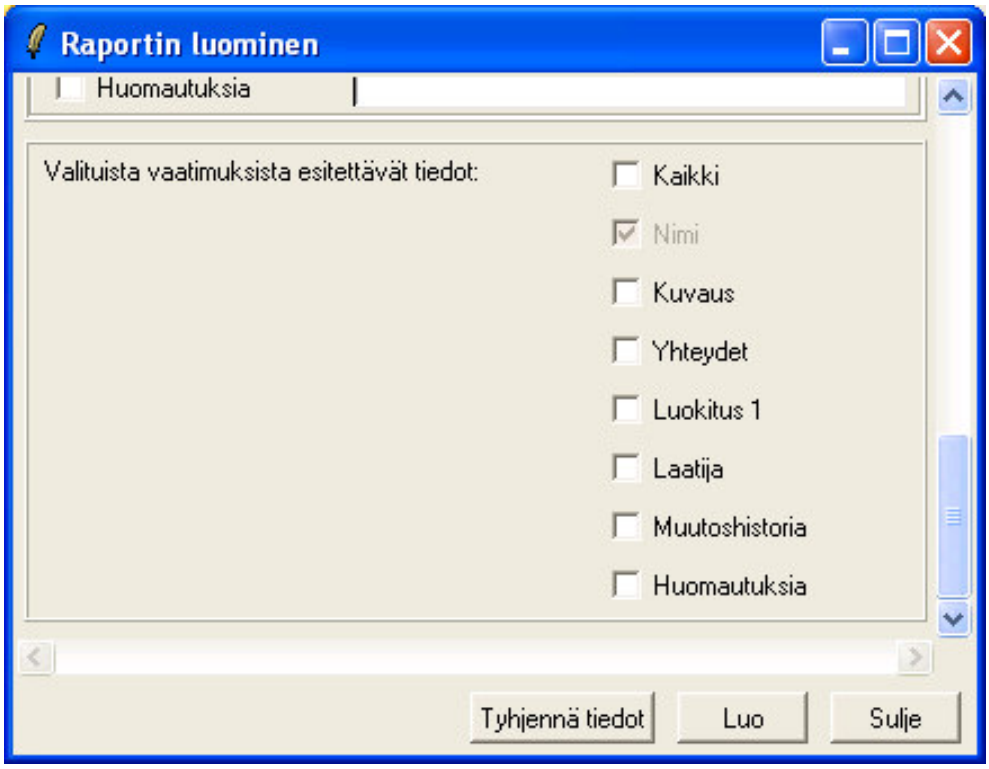
Kuva 12: Vaatimusten järjestyksen muuttaminen -lomakkeella muutetaan saman otsikon alla olevien vaatimusten keskinäistä järjestystä.



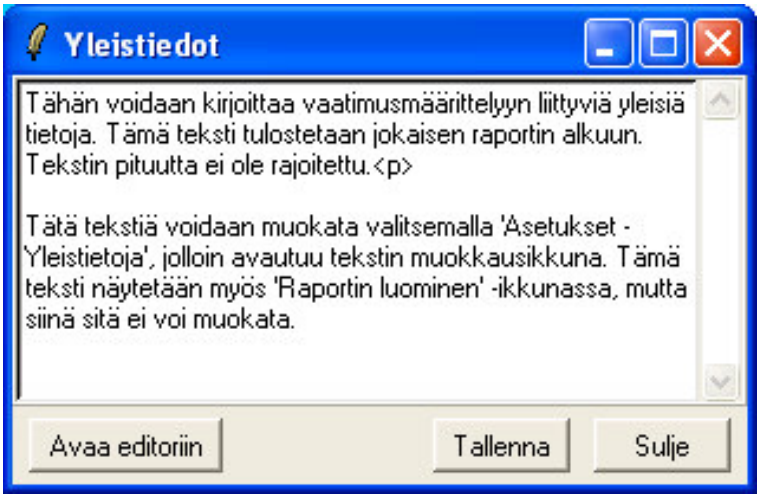
Kuva 13: Raportin luominen -lomake. Raportin nimeäminen, esitystavan ja vaatimuksen valintakriteerien määrittäminen.

dostoa, jotka on määriteltävä ennen ohjelman käyttöä. Näiden tiedostojen avulla määritellään vaatimusmäärittelyn hierarkkinen sisällysluettelo, vaatimuksen tietosisältö, käyttäjäkohtaiset oletusasetukset ja yleistiedot. Yleistiedot voidaan tallentaa tai muokata myös ohjelmassa (kuva 15). Myös laadittujen vaatimusten tiedot tallennetaan vastaavan tyyppiseen tekstitiedostoon. Käyttäjällä ei kuitenkaan ole normaalissa käyttötilanteessa tarvetta muokata tiedostoa ohjelman ulkopuolella.

Vaatimusmäärittelyn hierarkkisen sisällysluettelon määrittelevässä tiedostossa esitetään ohjelman pääikkunassa näytettävän sisällysluettelon runko. Hierarkiatiedosto on rakenteeltaan kuvan 16 esimerkin mukainen. Esimerkissä tiedoston kolme ensimmäis-



Kuva 14: Raportin luominen -lomake. Vaatimuksesta esitettävien tietojen määrittäminen.



Kuva 15: Yleistiedot-lomake.

tä riviä ovat kommenttirivejä, joista ensimmäinen on tiedoston tunnisteena pakollinen esitetyn muotoisena. Sisällysluettelon otsikkorivi sijoitetaan alku- ja lopputunnisteiden `<otsikkoX>` ja `</otsikkoX>` väliselle riville. Tunnisteen nimen viimeisenä merkkinä (X) oleva numero ilmaisee otsikon hierarkiatason luettelossa (vertaa kuva 8). Ohjelmassa vaatimuksia voidaan lisätä vain luettelon hierarkian alimmille tasoille, jotta tässä tiedostossa määritellyn hierarkian eheys säilyy. Sisällysluetteloon sijoitettavien rivien lukumäärää ei ole rajoitettu. Hierarkiatasoja voi olla enintään yhdeksän. Kun esimerkissä sisällysluettelon numerointi aloitetaan numerosta neljä, on vaatimusmäärittelydokumentin luvut 1 - 3 esitetty yleisissä tiedoissa. Tällöin nämä luvut esitetään kaikkien raporttien alussa riippumatta raportin muodostamiskriteereistä.

```
<-- sfrm-hierarchy -->
<-- tämä tiedosto on esimerkki vaatimusmäärittelyn -->
<-- sisällysluettelon hierarkian määrittämisestä -->
<otsikko1>
4 Toiminnallisuus
</otsikko1>
<otsikko2>
4.1 Toimintovaatimukset
</otsikko2>
<otsikko2>
4.2 Yhteensopivuus
</otsikko2>
<otsikko1>
5 Luotettavuus
</otsikko1>
<otsikko2>
5.1 Kypsyys
</otsikko2>
<otsikko2>
5.2 Vikasietoisuus
</otsikko2>
<otsikko1>
6 Käytettävyys
</otsikko1>
...
```

Kuva 16: Esimerkki vaatimusmäärittelyn hierarkian määrittelytiedostosta.

Ohjelmassa vaatimuksen perustietoja ovat vaatimuksen ID-tunnus, nimi, kuvaus ja yhteydet. Ne ovat mukana kaikissa vaatimusmäärittelyissä. Kaikki muut vaatimuksen tiedot ovat vaatimusmäärittelykohtaisia ja ne kuvataan vaatimuksen tietosisällön määrittelytiedostossa. Tietotyyppejä on kolme erilaista: lyhyt yksirivinen teksti, rajoittamaton monirivinen teksti ja luokitus. Tiedosto on rakenteeltaan kuvan 17 esimerkin mukainen. Tiedostossa lyhyen tekstin tunnisteen nimenä on `tekstiL` ja pitkän `tekstiP`. Luokituksen tunnisteen nimenä on `luokitus` (vertaa kuva 9). Luokituksessa alkutunnisteen jälkeiselle riville annetaan luokituksen nimi, seuraavalle riville luokituksen vaihtoehtojen lukumäärä. Lukumäärää seuraaville peräkkäisille riveille kirjoitetaan

```
<-- sfrm-contents -->
<-- tämä tiedosto on esimerkki -->
<-- vaatimuksen sisältämien tietojen määrittelystä -->
<tekstiL>
Laatija
</tekstiL>
<tekstiP>
Muutoshistoria
</tekstiP>
<luokitus>
Vaatimuksen tila
3
Ehdotettu
Hyväksytty
Hylätty
<perusteet>
Vaatimuksen tila -luokituksen valinta tehdään seuraavien
ominaisuuksien perusteella:
Ehdotettu:   Tekstiä 1 ...
Hyväksytty:  Tekstiä 2 ...
Hylätty:     Tekstiä 3 ...
</perusteet>
</luokitus>
...
```

Kuva 17: Esimerkki vaatimuksen tietosisällön määrittelytiedostosta.

luokituksen eri vaihtoehtojen nimet. Luokituksen lopuksi kirjoitetaan vielä luokituksen perusteet. Perusteiden tunnisteen nimenä on `perusteet`.

Käyttäjäkohtaiset oletusasetukset määrittelevässä tiedostossa nimetään käyttäjäkohtaisia asetuksia, kuten oletuseditori, oletusselain sekä käyttäjällä eri vaatimusmäärittelyihin käytettävissä olevat vaatimusnumerot. Tiedosto on rakenteeltaan kuvan 18 esimerkin mukainen. Käyttäjän oletuseditori hakemistopolkuineen sijoitetaan tunnistenimen `editori` väliselle riville. Vastaavasti oletusselaimen tunnistenimenä on `selain`. Sekä oletuseditori että oletusselain voidaan asettaa myös ohjelmassa, jolloin ohjelma tallentaa tiedot tähän oletusasetustiedostoon. Jokaiselle vaatimusmäärittelylle on annettava tässä oletusasetustiedostossa vaatimusten alku- ja loppunumero. Ohjelma antaa jokaiselle vaatimukselle yksilöllisen ID-tunnisteen, alkaen tässä asetetusta alkunumerosta. Jokaiselle uudelle vaatimukselle ohjelma kasvattaa tunnusnumeroa. Kun loppunumeroa vastaava vaatimuskin on jo lisätty, ohjelma antaa virheilmoituksen yritettäessä lisätä seuraavaa vaatimusta. Mikäli vaatimusmäärittelyä laatii useampi henkilö, on kunkin henkilön omassa oletusasetustiedostossa annettava käyttöön eri alku- ja loppunumerot. Numerosarjojen ei tarvitse olla keskenään peräkkäin jatkuvia, mutta ne eivät saa olla päällekkäisiä. Kunkin vaatimusmäärittelyn aloitus- ja lopetustunnisteen nimenä käytetään sen nimeä.

Vaatimusten tallennustiedosto on rakenteeltaan kuvan 19 esimerkin mukainen. Esimer-

```
<-- sfrm-defaults -->
<-- Tämä on käyttäjälle -->
<-- 'jkahko' -->
<-- liittyvä oletusasetuksettiedosto -->
<-- 'C:/Jussi/sfrm/jkahko.rms' -->
<-- Tekijä: jkahko -->
<-- Päiväys: 23.02.2002 16.17 -->
<editori>
/usr/bin/emacs
</editori>
<selain>
/usr/bin/netscape
</selain>
<vm1>
<alkunumero>
1
</alkunumero>
<loppunumero>
99
</loppunumero>
</vm1>
<vm2>
<alkunumero>
251
</alkunumero>
<loppunumero>
499
</loppunumero>
</vm2>
...
```

Kuva 18: Esimerkki käyttäjäkohtaisten oletusasetusten määrittelytiedostosta.

kissä seitsemän ensimmäistä riviä ovat kommenttirivejä. Ensimmäisellä rivillä on vaatimusten tallennustiedoston tunniste ja seitsemännellä tallennushetken päivämäärä ja kellonaika. Lisäksi esimerkissä näkyy yhdestä vaatimuksesta tallennetut tiedot tunnistenimen `vaatimus_id1` sisällä. Aluksi on tallennettu vaatimuksen ID-numero tunnistenimen `id` sisälle. Vaatimuksen sijainti hierarkiassa tallennetaan tunnistenimen `paikka` sisälle ja tunnistenimen `jarjnro` sisälle tallennetaan hierarkiassa samalla tasolla olevien vaatimusten keskinäinen järjestysnumero. Seuraavaksi tallennetaan vaatimusmäärittelyn vakiotiedot nimi, kuvaus ja yhteydet. Vaatimukseen kytketyt yhteydet tallennetaan siten, että kunkin liitetyn vaatimuksen nimet esitetään aaltosuluissa välilyönnillä toisistaan erotettuina. Lopuksi vaatimuksen tietoihin tallennetaan vaatimuksen tietosisällön määrittelytiedostossa asetetut tiedot peräkkäin alkaen tunnistenimestä `tieto_0`. Lyhyet ja pitkät tekstit tallennetaan tiedostoon riveittäin. Luokitustyyppisistä tiedoista tallennetaan ainoastaan tieto siitä, monesko kyseisen luokitusryhmän vaihtoehdoista on valittu. Esimerkissä tunnistenimen `tieto_0` sisälle on tallennettu tieto siitä, että tälle vaatimukselle on valittu luokituksen ensimmäinen vaihtoehto.

Ohjelma vaatii toimiakseen Tcl/Tk-ohjelmointikielen asennettuna työasemaan. Ohjel-

```
<-- sfrm-data -->
<-- Tämä on vaatimusmäärittelyyn -->
<-- 'C:/Jussi/sfrm/vm/vm' -->
<-- liittyvä vaatimusten tallennustiedosto -->
<-- 'C:/Jussi/sfrm/vm/vm.rmd' -->
<-- Tekijä: jkahko -->
<-- Päiväys: 23.02.2002 16.24 -->
<vaatimus_id1>
<id>
id1
</id>
<paikka>
root-4-1
</paikka>
<jarjnro>
1
</jarjnro>
<nimi>
4.1.1 Avaa uusi ikkuna
</nimi>
<kuvaus>
Käyttäjä voi avata uuden ikkunan
napsauttamalla Avaa-painiketta.
</kuvaus>
<yhteydet>
{4.1.7 Tulosta tiedot} {4.2.2 Poista ikkuna}
</yhteydet>
<tieto_0>
rbu0-1
</tieto_0>
<tieto_1>
jk
</tieto_1>
<tieto_2>
29.03.2002 muutettu jk
</tieto_2>
<tieto_3>
Muutosehdotus käsittelyssä
</tieto_3>
</vaatimus_id1>
```

Kuva 19: Esimerkki vaatimusten tallennustiedoston rakenteesta.

ma on laadittu Tcl/Tk:n versiolla 8.3, mutta ohjelmaa voidaan käyttää myös uudemmilla versioilla. Lisäksi tarvitaan selainohjelma raporttien katseluun sekä haluttaessa tekstieditori helpottamaan tekstien muokkaamista. Tekstit voi muokata myös ohjelman tekstikentissä, mutta haluttaessa muotoilla tekstiä ja sijoittaa tekstiin esimerkiksi kuvia, voi tekstin muokkaaminen olla mielekkäämpää käyttäen jotain HTML-koodausta tukevaa tekstieditoria. Käyttäjä voi määritellä vapaasti haluamansa selaimen ja tekstieditorin. Ne voidaan tallentaa ohjelman asetuksiin, jolloin valitut ohjelmat avautuvat oletuksena, kun niitä kutsutaan ohjelmasta. Ohjelman käyttöliittymä voi olla suomen- tai englanninkielinen. Oletuskielenä on suomi, mutta kielen voi vaihtaa ohjelman valikosta milloin tahansa ohjelman käytön aikana.

4.3.2 Soveltuvuus vaatimusmäärittelyohjelmaksi

Tässä alakohdassa tarkastellaan Sfrm-ohjelman soveltuvuutta vaatimusmäärittelyohjelmaksi. Vertailukohtina arviointiin käytetään Sawyer et al. (1998) esittämää vaatimusmäärittelyprosessin parantamisen kymmentä peruskäytäntöä sekä Yphisen (2002) esittämiä vaatimuksenhallinnan välineiden keskeisiä toimintoja. Sfrm-ohjelma tukee selkeästi neljää viidestä ensimmäisestä peruskäytännöstä (taulukko 1). Ensimmäinen ja viides käytäntö, standardirakenteen määrittäminen dokumentille ja vaatimuksen kuvaukselle, tulevat esille jo ohjelman asetustiedostojen määrittelyssä. Sekä vaatimusmäärittelydokumentin rakenne että yksittäisestä vaatimuksesta esitettävät tiedot voidaan määritellä projektikohtaisesti. Kolmannen käytännön, yksilöllisen tunnuksen antamisen jokaiselle vaatimukselle, ohjelma hoitaa automaattisesti. Ohjelman avulla määrittelydokumentin muuttaminen on helppoa: tehdään ensin muutokset vaatimuksiin ohjelmassa ja muodostetaan sen jälkeen raportti uudelleen. Tämä toteuttaa listan toisen käytännön. Loput kymmenestä käytännöstä ovat luonteeltaan sellaisia, että niiden tukeminen välineiden avulla on hankalaa tai jossain tilanteissa välineet voivat jopa hankaloittaa käytäntöjen toteutusta. Sfrm-ohjelma ei kuitenkaan aseta rajoitteita näiden loppujen käytäntöjen toteuttamiselle.

Taulukko 1: Sfrm-ohjelman tukemat käytännöt Sawyer et al. (1998) esittämässä vaatimusmäärittelyprosessin kymmenessä peruskäytännössä.

Prosessien parantamiskäytäntöjen toteutusjärjestys	Sfrm:n tuki
Määritellään dokumentille standardirakenne	Kyllä
Tehdään dokumentti helposti muutettavaksi	Kyllä
Jokaiselle vaatimukselle annetaan yksilöllinen tunniste	Kyllä
Määritellään toimintaperiaatteet vaatimustenhallinnalle	-
Määritellään standardimalli vaatimuksen kuvaukselle	Kyllä
Käytetään yksinkertaista, yhdenmukaista ja ytimekästä kieltä	-
Järjestetään muodollisten vaatimusten katselmukset	-
Määritellään tarkistuslistat tarkistuksia varten	-
Käytetään tarkistuslistoja vaatimusten analysoinnissa	-
Ennakoidaan ristiriitoja ja suunnitellaan niille ratkaisumallit	-

Vertailtaessa Sfrm-ohjelman tukemia toimintoja Yphisen (2002) (vertaa kuva 7) esittämiin vaatimuksenhallinnan välineiden toimintoihin, voidaan eri osa-alueista todeta

seuraavaa:

Tietovarasto: Sfrm-ohjelmassa vaatimusten tiedot sekä kaikki asetustiedot tallennetaan tekstitiedostoihin, eikä tietokantaa ole lainkaan käytössä tietovarastona. Ohjelmalla voidaan tyypillisesti käsitellä useita satoja vaatimuksia, mutta ohjelman suorituskyky heikkenee, jos vaatimusten lukumäärä kasvaa tuhansiin. Koska tietokantaa ei ole käytössä, ovat mahdollisuudet vaatimusten analysointiin rajoitetut. Ohjelma ei sisällä analysointia tukevia toimintoja. Toisaalta raporttien muodostamien erilaisin kriteerein on melko monipuolinen.

Ryhmätyöskentely: Ohjelma tukee ryhmätyöskentelyä osittain, myös hajautettuna, mutta ei kuitenkaan ole reaaliaikainen. Vaatimusmäärittelyä voi laatia useampi henkilö yhtä aikaa, kun heille määritellään käyttöön omat vaatimusten numerosarjat. Käyttäjät näkevät ohjelmassa vain omat vaatimuksensa ja he tallentavat määrittelemänsä vaatimukset omiin tiedostoihinsa, jotka lopuksi yhdistetään yhdeksi kokonaiseksi määrittelyksi. Ohjelmassa ei ole rutiineja näiden yhdistettyjen vaatimusten yhtenäisyyden ja ristiriidattomuuden analysoimiseksi.

Integrointiominaisuudet: Kaikki ohjelman käsittelemät tiedostot ovat XML-tyyppisiä tekstitiedostoja, joten ne ovat helposti luettavissa muilla ohjelmilla. Tiedostoja voi katsella ja editoida millä tahansa tekstieditorilla.

Vaatimusten luominen: Ohjelma tunnistaa vain omat tallentamansa tiedostot. Sisällöltään saman vaatimusmäärittelyn mukaisia tiedostoja voidaan yhdistää ja avata yhdistetty tiedosto ohjelmaan. Suora vaatimusten määrittely ohjelman vaatimuslomakkeella on ohjelman perustoiminto. Vaatimusten luokittelu voidaan määritellä vaatimuksen osatiedoksi.

Vaatimusten analysointi: Sfrm-ohjelma tukee osittain vaatimusten jäljitettävyyttä. Vaatimukselle voidaan antaa yhtenä tietona yhteyksiä muihin vaatimuksiin. Vaatimuslomakkeen Yhteydet-kentästä valitsemalla voidaan avata vaatimus, jolle yhteys on määritetty. Sen sijaan yhteyksien jäljittäminen taaksepäin onnistuu vain kiertoteitse raportin muodostamisen kautta taso kerrallaan. Kun raportin muodostamisehdoksi Yhteydet-kenttään annetaan käsiteltävä vaatimus, saadaan raporttiin mukaan kaikki ne vaatimukset, joista kyseiseen vaatimukseen on liitetty yhteys. Koska vaatimukset ovat HTML-tyyppisiä, voidaan tekstiin sijoittaa hyperlinkkejä muihin asiakirjoihin, esimerkiksi toiminnalliseen määrittelyyn tai suunnitelmiin. Hyperlinkkien avulla voidaan parantaa

jäljitettävyyttä vaatimuksista muihin ohjelmistokehityksen vaiheisiin. Ohjelmassa ei kuitenkaan ole erillistä toimintoa hyperlinkkien lisäämiseen, vaan ne on tehtävä esimerkiksi HTML-koodausta tukevan tekstieditorin avustuksella. Vaatimusten yhtenäisyyden ja samankaltaisuuden analysointiin ohjelmassa ei ole tukea. Myös muut analysointiominaisuudet ja vaatimusten tilan seurannan ominaisuudet puuttuvat.

Vaatimuksen jalostus: Sfrm-ohjelmassa ei ole tukea tämän ryhmän toiminnoille.

Vaatimusten muutoksen hallinta: Versionhallintaan ohjelmassa ei ole tukea. Raportin ja vaatimusten tallennustiedoston alkuun ohjelma tulostaa automaattisesti raportin luontipäivän ja kellonajan, mutta näitä tietoja ei mitenkään hyödynnetä ohjelmallisesti. Käyttöoikeuksien hallinta on ohjelmassa osittainen. Ohjelmalla voidaan avata vain vaatimusmäärittely, joka on nimenomaisesti määritelty käyttäjäkohtaisessa asetustiedostossa. Kukin käyttäjä voi täten avata ja muokata vain hänelle määriteltyä vaatimusmäärittelyä. Yksittäisten vaatimusten tallennustiedostojen ja asetustiedostojen käyttöoikeuksien hallintaan ohjelmassa ei ole toimintoa, joten kuka tahansa voi avata tiedoston tekstieditoriin, katsella ja muokata tietoja. Muutoshistorian hallintaan ohjelmassa ei ole omaa toiminnallisuutta. Muutostiedot voidaan kuitenkin määritellä yhdeksi vaatimuksen osatiedoksi, mutta niiden käyttö on täysin käyttäjän vastuulla.

Ohjelman käyttöönottokustannukset ovat alhaiset. Itse ohjelma on ilmainen ja ainut ulkopuolinen tarvittava ohjelma, Tcl/Tk, on sekin avoimeen lähdekoodin perustuvana saatavana ilmaiseksi. Myöskään ohjelma laiteympäristölle ei ole erityisiä vaatimuksia normaaliin työasemakokoonpanoon verrattuna. Kustannuksia aiheutuu lähinnä työkustannuksina pääkäyttäjän asetustiedostojen määrittämisestä ja varsinaisten käyttäjien tutustumisesta ohjelman käyttöön. Asetustiedostojen määrittäminen voidaan katsoa myös osaksi vaatimusmäärittelyprosessin yleistä määrittelyä, koska siinä määrätään sekä vaatimuksen tietosisältö että laadittavan dokumentin runkorakenne. Kun asetustiedot projektia varten on tehty, ei käyttäjillä ole tarvetta muokata niitä erikseen. Ohjelmaan liittyy yksityiskohtainen HTML-tyyppinen ohje, jonka avulla ohjelmaan tutustumien tai käytönaikaisen avun saaminen on helppoa. Pienenä puutteena ohjelman toiminnassa on häiriötilanteista toipuminen. Istunnon aikana tiedot tallennetaan ohjelman muuttujiin ja tallennetaan tiedostoon vasta erikseen käyttäjän toimesta. Esimerkiksi sähkökatkon sattuessa, tallentamattomat muutokset ja uudet vaatimukset menetetään. Koska ohjelmassa ei ole automaattista varmistustallennusta, tiedostoon tallentaminen vaatii käyttäjän aktiivisuutta.

Johtopäätöksenä edellä kuvatuista ominaisuuksista voidaan todeta, että Sfrm-ohjelma soveltuu erityisesti ensimmäiseksi vaatimusmäärittely- ja vaatimustenhallintaohjelmaksi lähdettäessä parantamaan määrittelemätöntä, kypsyydeltään satunnaisella tasolla olevaa prosessia kohti toistettavaa tasoa. Ohjelma on edullinen ja helppo käyttää ja se tukee monia prosesseja parantavia käytäntöjä. Ohjelman avulla voidaan hallita useita satoja vaatimuksia käsittäviä vaatimusmäärittelyjä. Välineiden tarjoamaa tukea vaatimusmäärittelyprosessien parantamisessa voisi Sfrm-ohjelman lisäksi täydentää muilla ulkopuolisilla välineillä esimerkiksi versionhallinnan osalta. Kun vaatimusmäärittelyprosessia lähdetään kehittämään toistettavalta tasolta kohti määriteltyä tasoa, korostuvat tarpeet vaatimusten analysointiin sekä mittaustiedon keruuseen. Tällöin myös tietovarastoratkaisuna on tietokanta välttämätön. Sfrm-ohjelma ei tue riittävästi näitä kasvavia vaatimustenhallinnan välineeltä vaadittavia ominaisuuksia.

4.3.3 Käyttökokemuksia

Sfrm-ohjelman käyttökokemuksia kuvastaa Nikulan (2004) tutkimus, jossa vertailtiin hänen kehittämänsä BaRE-menetelmän (ks. kohdan 4.1.2 loppuosa) toimivuutta systemaattisen vaatimusmäärittelyn tekemisen aloittamiseksi pienissä organisaatioissa. Sfrm-ohjelma valittiin vaatimustenhallinnan välineeksi yhdessä tutkimuksessa mukana olleista yrityksistä.

Aluksi Sfrm-ohjelmaa käytettiin vaatimusten muutoksenhallintaan. Ohjelmaan tallennettiin uudet vaatimusehdotukset ja vaatimusten muutospyynnöt. Vaatimusmäärittelydokumentti laadittiin aluksi tekstinkäsittelyohjelman avulla. Vaatimusten muutoksenhallinta toteutettiin neljässä kuukaudessa ja saavutettuihin tuloksiin oltiin tyytyväisiä. Hyvistä kokemuksista rohkaistuneena yrityksessä kiinnostuttiin enemmän vaatimusmäärittelyprosessin ja mallidokumentin kehittämisestä. Kehitystä jatkettiin ja kymmenen kuukauden kuluttua tutkimuksen aloittamisesta oli toteutettu koko vaatimusmäärittely-ympäristö. Lopussa myös vaatimusmäärittelydokumentti luotiin Sfrm-ohjelmalla. Tutkimuksen lopussa yritys raportoi vaatimusten dokumentoinnin ja muutoksenhallinnan lisäksi seuraavia parannuksia. Tulevien ohjelmistojulkaisujen suunnittelu on mahdollista perustuen todellisiin ehdotuksiin eikä huhuihin ja uskomuksiin perustuen kuten aiemmin. Samankaltaiset vaatimukset voidaan havaita aikaisin eikä aikaa kulu paljon niiden analysointiin. Vaatimusten standardoitu, sähköinen sijaintipaikka varmistaa vaatimusten ja muutosehdotusten suorittamisen ja hallinnan

systemaattisesti ja yhdenmukaisesti. Toteutetut parannukset ovat ratkaisseet yksilöllisiä ongelmia, tehneet työskentelyn helpommaksi ja tarjonneet yritykselle suhteellista hyötyä aikaisempiin käytäntöihin verrattuna.

Yksitoista kuukautta kestäneen tutkimuksen lopussa suoritetussa SPICE-standardin mukaisessa prosessien kypsyyden arvioinnissa yritys oli saavuttanut hallitun tason (taso 2). Yritys oli selkeästi määritellyt vaatimustenhallintaprosessin ja toimivan tason (tason 1) prosessiominaisuus, prosessin suorittaminen, oli täysin saavutettu. Myös hallitun tason suorittamisen ja työtulosten hallinta oli suurelta osalta saavutettu. Yritys käytti BaRE-menetelmää vaatimusten kokoamiseen ja määrittelyyn. Vaatimustenhallintaan käytettiin sfrm-ohjelmaa. Näiden välineiden avulla yrityksen oli mahdollista saavuttaa vakiintunut ja odotuksenmukainen työskentelytapa ja vaatimusmäärittelydokumenttien ylläpito. Tutkimuksen aikana saavutettiin vaatimusmäärittelyprosessien suorittamisessa yritykselle riittävä kypsyystaso ja jatkossa suunniteltiin parannuksia muilla ohjelmistotuotannon alueilla.

4.3.4 Kehitysehdotuksia

Mietittäessä Sfrm-ohjelmalle kehityskohteita on muistettava sen käyttöalue vaatimusmäärittelyn laadintaa tukevana välineenä prosessien parannustoimien alkuvaiheessa. On keskityttävä ohjelman nykyisten toimintojen kehittämiseen ja vahvistamiseen. Kokonaan uuden toiminnallisuuden liittäminen mukaan ohjelmaan on arveluttavaa, koska se todennäköisesti aiheuttaisi suuria rakenteellisia muutoksia myös nykyiseen ohjelmistoon.

Yhtenä parannuskohteena voisi olla jäljitettävyyden parantaminen. Nyt erityisesti taaksepäin jäljitys on puutteellinen. Jäljitettävyyden parantaminen voidaan toteuttaa helpoimmin lisäämällä raporttityyppi, jossa haetaan yhteyksien perusteella toisiinsa liitetyt vaatimukset ketjussa määrätystä vaatimuksesta eteen tai taaksepäin.

Raportin luontia voidaan parantaa mahdollisuudella tallentaa raportin luomisessa käytetyt kriteerit. Tästä ominaisuudesta olisi hyötyä erityisesti silloin, kun vaatimusmäärittelyä muutetaan ja sen jälkeen halutaan luoda aikaisemman mukainen raportti. Nyt ohjelma tulostaa raportin alkuun sen luomisessa käytetyt kriteerit, mutta myöhemmin luotaessa uudelleen muuttunutta raporttia, on Raportin luominen -lomakkeella määriteltävä kaikki kriteerit uudelleen.

Lisäämällä ohjelmaan automaattinen vaatimusten tiedostoon tallentaminen, voidaan vähentää uhkaa tietojen menettämisestä häiriötilanteissa. Henkilökohtaiseen asetustiedostoon voidaan sijoittaa lisätietona automaattisen tallennuksen aikaväli.

5 Yhteenveto

Ohjelmistoprosessien parantamisella pyritään kehittämään organisaation ohjelmistotuotantoprosessia. Tavoitteena on muuttaa toimintatapoja siten, että ohjelmistotuotteen laatu ja organisaation suorituskyky paranevat. Prosessien suorituksen tehostaminen välineiden avulla on yksi keskeinen tekijä ohjelmistoprosessien parantamisessa. Välineiden soveltuvuuteen sekä niiden käytöstä mahdollisesti saatavaan hyötyyn tietyssä tehtävässä tai projektissa vaikuttavat kuitenkin monet tekijät. Tässä tutkielmassa on perehdytty kirjallisuuden pohjalta CASE-välineiden vaikutukseen ohjelmistoprosessien parantamisessa sekä arvioitu Joensuun yliopistossa kehitetyn Sfrm-ohjelman ominaisuuksia ja soveltuvuutta vaatimusmäärittelyprosessin parantamisessa.

Tutkielman alussa luotiin katsaus erityyppisiin CASE-välineisiin, niiden käyttötarkoituksiin ja erilaisiin luokittelutapoihin. Yksi tapa, jota tutkielmassa käsiteltiin tarkemmin, on ohjelmistoprosessien mukainen luokittelu. Lisäksi CASE-välineet voidaan ryhmitellä niiden ohjelmistoprosesseja tukevien tehtävien laajuuden perusteella. Yksittäinen väline tukee vain yhtä tehtävää ohjelmistoprosessissa. Toimintoalustoissa yhdistetään useita yksittäisiä välineitä yhdessä toimivaksi ohjelmistoksi ja ne tukevat yhtä tai muutamia tehtäviä. Ympäristöt ovat välineiden tai toimintoalustojen kokoelmia ja ne tukevat laajaa osaa tai koko ohjelmistoprosessia.

Tutkielmassa tarkasteltiin välineiden käyttöä ohjelmistoprosessin eri kypsyystasoilla sekä välineiden käyttöönotossa huomioitavia tekijöitä. Lisäksi arvioitiin välineiden käytön etuja ja haittoja ohjelmistoprosessin parantamisen kannalta. Yhtenä keskeisenä havaintona tarkastelussa oli, että välineiden avulla voidaan tukea kypsyydeltään eri tasoilla olevien ohjelmistoprosessien parantamista. Käytettävät välineet ovat kuitenkin erilaisia ohjelmistoprosessien kypsyyden eri tasoilla. Prosessien kypsyyden alimmilla tasoilla voidaan yksittäisten ohjelmistoprosessien suoritusta tehostaa yksittäisen, vain kyseistä toimintaa tukevan välineen avulla. Prosessien kypsyyden kasvaessa lisääntyy prosessien jatkuvan parantamisen tarve ja samalla välineiden merkitys korostuu. Projektin- ja versionhallinnan välineiden tuki korostuu, samoin välineiden keskinäinen tuki toisilleen. Prosessien saamiseksi ennustettavaksi ja optimoiduksi on välineiden tuki välttämätön mittaustiedon keruuseen ja erilaisiin analyyseihin. Myös keskitetty tietovarasto ja välineiden saumaton liittyminen siihen on välttämätöntä. Tarkastelussa havaittiin edelleen, että välineet yksin eivät saa aikaan toivottua parannusta. Välineiden käyttöönottaminen on suunniteltava tarkoin ja se vaatii pitkäjännitteistä työtä ja

pääomia. On tärkeää, että ohjelmistoprosessien menetelmät sovitetaan ensin organisaation tarpeiden mukaisiksi ja vasta sen jälkeen aloitetaan laajan CASE-järjestelmän käyttöönottaminen. Jotta tavoiteltu hyöty saavutetaan, tarvitaan myös muutoksia ja sitoutumista sekä organisaatio- että henkilötasolla. Ylimmän johdon sitoutuminen, tuki ja mukanaolo välineiden käyttöönotossa on välttämätöntä. Tarvitaan organisaatiotason päätöksiä, sovittamista liiketaloudellisiin tavoitteisiin ja pitkän tähtäimen kustannussuunnittelua niin käyttöönottoon kuin käytön aikaiseen ylläpitoon ja koulutukseen. Myös henkilötasolla tarvitaan kaikkien osallistujien sitoutumista ja mukanaoloa suunnittelusta alkaen. Näin voidaan vähentää muutosten vastustamista. Henkilöstön on myös koettava saavansa muutoksista jotain hyötyä omien tehtäviensä suorittamisessa. Tutkielmassa tarkasteltiin myös välineiden käytön etuja ja haittoja ohjelmistoprosessin parantamisessa. Keskeisimpinä etuina todettiin olevan prosessin automatisointi, prosessien välisen tiedon siirrettävyyden tehostaminen, tiedon riippuvuussuhteiden ja jäljitettävyyden hallinta, keskitetty tiedon varastointi ja järjestelmän ylläpidettävyyden parantaminen. Keskeisimpiä haittoja puolestaan todettiin olevan välineiden kalleus, vaativa valinta- ja käyttöönottoprosessi sekä koulutuksen tarve.

Tutkielmassa tutustuttiin lähemmin vaatimusmäärittelyprosessiin, sen erityispiirteisiin sekä vaatimusmäärittelyssä käytettyjen välineiden ominaisuuksiin. Onnistunut vaatimusmäärittely todettiin keskeiseksi koko ohjelmistoprojektin onnistumisen kannalta. Kaikki muut ohjelmistoprosessin vaiheet suunnittelusta ylläpitoon pohjautuvat vaatimusmäärittelyyn. Jokainen määrittelyssä tehty virhe aiheuttaa ongelmia myöhemmissä vaiheissa ja seuraukset ovat sitä suuremmat, mitä myöhemmin virhe havaitaan. Tämän vuoksi ohjelmistoprosessien parantamisessa on erityisen tärkeää keskittyä vaatimusmäärittelyprosessin parantamiseen. Vaatimusmäärittelyssä keskeisiä elementtejä ovat vaatimusten kehittäminen ja vaatimustenhallinta. On tärkeää, että kehitettävästä ohjelmistosta saadaan kootuksi riittävä määrä yksiselitteisiä, keskenään ristiriidattomia vaatimuksia, joiden avulla järjestelmän toiminta pystytään kuvaamaan tarkasti. Vaatimukset on myös pystyttävä esittämään määrittelydokumentteina selkeästi siten, että asiakas ja toteuttajat pystyvät ymmärtämään järjestelmän samalla tavalla ja että järjestelmä pystytään toteuttamaan näiden tietojen perusteella. Koska vaatimusmäärittely on kaikkien muiden osaprosessien perustana, korostuu koko projektin ajan kestävän vaatimustenhallinnan merkitys. Keskeisinä tekijöinä vaatimustenhallinnan onnistumisessa todettiin olevan vaatimusten jäljitettävyys koko prosessiketjun matkalta, toimiva vaatimusten muutoksenhallinta sekä muutoksen vaikutusten analysointi. Vaatimusten ja kaikkien muidenkin työtulosten versionhallinnan toimivuudesta on myös huolehditta-

va. Välineiden tuki on tärkeä kaikkien vaatimusmäärittelyn osaprosessien suorittamisessa. Käyttöönottoon, organisaation ja henkilöstön osallistumiseen liittyvät tekijät on otettava huomioon samalla tavalla kuin CASE-välineillä yleensä.

Tutkielman lopussa tutustuttiin Joensuun yliopistossa kehitettyyn Sfrm-vaatimustenmäärittelyohjelmaan ja arvioitiin sen ominaisuuksia vaatimusmäärittelyprosessien tukemisessa. Tarkastelun yhteenvetona todettiin, että ohjelma on edullinen ja sen käyttöönotto ei vaadi suuria henkilöresursseja eikä käytettävälle laitteistolle ole erityisiä vaatimuksia. Ohjelma soveltuu käytettäväksi ensimmäisenä välineenä, kun kypsyydeltään satunnaisella tasolla olevaa määrittelemätöntä prosessia ryhdytään parantamaan pienin, askel kerrallaan toteutettavin muutoksin. Koska ohjelma ei käytä tietokantaa ja siitä puuttuvat analysointiominaisuudet, ei ohjelma enää sovellu käytettäväksi vaatimusmäärittelyprosessin kypsemmässä vaiheessa siirryttäessä toistettavalta tasolta kohti määriteltyä tasoa.

Ohjelmistotuotanto on ihmiskeskeistä toimintaa. Kaikkia toimintoja ei voida automatisoida ja ihmisen luovaa ajattelua tuskin koskaan voidaan täysin korvata välineillä. Kuitenkin ohjelmistojen jatkuvasti laajentuessa ja tullessa yhä monimutkaisemmiksi on monien toimintojen toteuttaminen ilman välineitä jokseenkin mahdotonta. Välineet ovat keskeisessä asemassa pyrittäessä edelleen parantamaan ohjelmistoprosesseja. Tämä vaatii kuitenkin välineiden jatkuvaa kehittämistä ja sovittamista organisaation toimintoihin unohtamatta organisaatiolta ja henkilöstöltä vaadittavia toimia tässä onnistumiseksi.

Viitteet

Albizuri-Romero, M.B. (2000) A Retrospective View of CASE Tools Adoption. *Software Engineering Notes*, **25**(2), 46-51.

Blackburn, J., Scudder, G., Van Wassenhove, L.N. (2000) Concurrent Software Development. *Communications of the ACM*, **43**(11es), Article No. 4.

Brown, A.W. (1997) CASE in the 21st Century: Challenges Facing Existing CASE Vendors. *Proceedings of the Eighth IEEE International Workshop on Software Technology and Engineering Practice* (toim. Budgen, D., Hoffnagle, G., Trienekens, J.), London, UK, 7/1997, IEEE Computer Society, 268-278.

Christie, A., Levine, L., Morris, E., Zubrow, D., Belton, T., Proctor, L., Cordelle, D., Ferotin, J-E., Solvay, J-P., Segoti, C. (1996) *Software Process Automation: Experiences from the Trenches.* Technical Report CMU/SEI-96-TR-013, Software Engineering Institute, Pittsburgh, PA.

Conradi, R., Fernström, C., Fuggetta, A., Snowdon, R. (1992) Towards a Reference Framework for Process Concepts. *Proceedings of the Second European Workshop on Software Process Technology* (toim. Derniame, J-C.), Trondheim, Norway, 9/1992, Springer-Verlag, Berlin 1992, 3-17.

Curtis, B. (1992) The CASE for Process. *Proceedings of the IFIP WG8.2 Working Conference on the Impact of Computer Supported Technologies in Information Systems Development*, (toim. Kendal, K.E., Lyytinen, K., DeCross, J.I.), Minneapolis, USA, 6/1992, North-Holland, 333-343.

Daneva, M. (1999) A Best Practice Based Approach to CASE-tool Selection. *Proceedings of the Fourth IEEE International Symposium and Forum on Software Engineering Standards*, Curitiba, Brazil, 5/1999, IEEE Computer Society, 100-110.

Dart, S.A., Ellison, R.J., Feiler, P.H., Habermann, A.N. (1987) Software Development Environments. *Computer*, **20**(11), 18–28.

Davis, A.M. (1993) *Software Requirements: Objects, Functions & States*. Prentince Hall, USA.

Dittrich, K.R., Tombros, D., Geppert, A. (2000) Databases in Software Engineering:

A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*. Limerick, Ireland, 2000, ACM Press, New York, 293–302.

El Emam, K., Drouin, J., Melo, W. (1998) *SPICE: The Theory and Practice of Software Process Improvement and Capability Determination*. IEEE Computer Society, Los Alamitos, California.

Fenton, N.E., Neil, M. (2000) Software Metrics: Roadmap. *Proceedings of the Cconference on the Future of Software Engineering*. Limerick, Ireland, 2000, ACM Press, New York, 357–370.

Finkelstein, A., Emmerich, W. (2000) The Future of Requirements Management Tools. *Information Systems in Public Administration and Law*. (toim. Quirchmayr, G., Wagner R., Wimmer, M.) Österreichische Computer Gesellschaft.

Flynn, D., Vagner, J., Dal Vecchio, O. (1995) Is CASE Ttechnology Improving Quality and Productivity in Software Development? *Logistics Information Management*, **8**(2), 8-21.

Forte, G., McCulley, K. (1991) *CASE Outlook: Guide to Products and Services*. CASE Consulting Group, Lake Oswego, Oregon.

Fuggetta, A. (1993) A Classification of CASE Technology. *Computer*, **26**(12), 25–38.

Fuggetta, A. (2000) Software Process: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*. Limerick, Ireland, 2000, ACM Press, New York, 27–34.

Harrison, W., Ossher, H., Tarr, P. (2000) Software Engineering Tools and Environments: A Roadmap. *Proceedings of the conference on The future of Software engineering*. Limerick, Ireland, 2000, ACM Press, New York, 261–277.

Hoffer, J.A., George, J.F., Valacich, J.S. (2002) *Modern Systems Analysis and Design, 3rd ed*. Prentice Hall, USA.

Hofmann, H.F., Lehner, F. (2001) Requirements Engineering as a Success Factor in Projects. *IEEE Software*, **18**(4), 58-66.

Hori, M., Ono, K., Kondoh, G., Singhal, S. (2000) Authoring Tool for Web Content Transcoding. *Markup Languages: Theory & Practice*, **2**(1), 65-80.

Huang, R. (1998) Making Active CASE Tools — Toward the Next Generation CASE. *ACM SIGSOFT Software Engineering Notes*, **23**(1), 47-50.

Humphrey, W.S. (1989) *CASE Planning and the Software Process.* Technical Report CMU/SEI-89-TR-26, Software Engineering Institute, Pittsburgh, PA.

IEEE (1990) *IEEE Std 610.12-1990: IEEE Standard Glossary of Software Engineering Terminology*. Institute of Electrical and Electronic Engineers, USA.

IEEE (1998) *IEEE Std 1012-1998: IEEE Standard for Software Verification and Validation*. Software Engineering Standards Committee of the IEEE Computer Society, USA.

IEEE (2003) *IEEE Std 1175.1-2002: IEEE Guide for CASE Tool Interconnections - Classification and Description*. Software Engineering Standards Committee of the IEEE Computer Society, USA.

Iivari, J. (1996) Why Are CASE Tools Not Used? *Communications of the ACM* **39**(10), 33-47.

INCOSE (2002) *Tool Survey: Requirements Management (RM) Tools.* `http://66.34.135.97/tools/tooltax.html` (13.12.2004).

ISO/IEC TR 15504-2 TR (1998) *Information Technology - Software Process Assesment - Part 2: A Rreference Model for Processes and Process Capability*. International Organisation for Standardisation, Switzerland.

ISO/IEC TR 15504-5 TR (1999) *Information Technology - Software Process Assesment - Part 5: An Assessment Model and Indicator Guidance*. International Organisation for Standardisation, Switzerland.

ISO/IEC TR 15504-9 (1998) *Information Technology - Software Process Assesment - Part 9: Vocabulary*. International Organisation for Standardisation, Switzerland.

Kauppinen, M., Vartiainen, M., Kontio, J., Kujala, S., Sulonen, R. (2004) Implementing Requirements Engineering Process Throughout Organizations: Success Factors and Challenges. *Information and Software Technology*, **46**(14), 937-953.

Kuvaja, P., Similä, J., Krzanik, L., Bicego, A., Koch, G., Saukkonen, S. (1994) *Software Process Assessment and Improvement: The BOOTSTRAP Approach*. Blackwell Publis-

hers, Oxford, UK.

Kähkönen, J. (2002) *Sfrm-ohje, Vaatimusmäärittelyn laadinta- ja ylläpitosovellus*. Erikoistyö, Tietojenkäsittelytieteen laitos, Joensuun yliopisto, Joensuu.
Saatavana myös: `http://cs.joensuu.fi/pages/saja/se/sfrm_ohje/sfrmhelp.html` (13.12.2004).

Lang, M., Duggan, J. (2001) A Tool to Support Collaborative Software Requirements Management. *Requirements Engineering*, **6**(3), 161-172.

Leffingwell, D., Widrig, D. 1999 *Managing Software Requirements: A Unified Approach*. Addison Wesley, New Jersey.

Low, G., Leenanuraksa, V. (1999) Software Qquality and CASE Tools. *Proceedings of Software Technology and Engineering Practice, STEP'99*, Pittsburgh, PA, USA, 8-9/1999, IEEE Computer Society, 142-150.

Mathiassen, L., Sørensen, C. (1996) The Capability Maturity Model and CASE. *Journal of Information Systems* **6**(3),195-208.

McMurtrey, M.E., Teng, J.T.C., Grover, V., Kher, H.V. (2000) Current Utilization of CASE Technology: Lessons from the Field. *Industrial Management & Data Systems*, **100**(1), 22-30.

Mendoza, L.E., Rojas, T., Perez, M.A. (2001) Organizational Indicators for CASE Tools Selection: A Case Study. *Colombian Journal of Computation* **2**(2).
Saatavana myös: `http://www.unab.edu.co/editorialunab/revistas/rcc/pdfs/r22_art3_c.pdf` (13.12.2004).

Nikula, U. 2004 *Introducing Basic Systematic Requirements Engineering Practices in Small Organizations with an Easy to Adopt Method*. Acta Universitatis Lappeenrantaensis 189, Doctoral Dissertation, Lappeenranta University of Technology.

Nuseibeh, B., Easterbrook, S. (2000) Requirements Engineering: A Roadmap. *Proceedings of the Conference on the Future of Software Engineering*. Limerick, Ireland, 2000, ACM Press, New York, 35–46.

Oakes, K.S., Smith, D., Morris, E. (1992) *Guide to CASE Adoption.* Technical Report CMU/SEI-92-TR-15, Software Engineering Institute, Pittsburgh, PA.

Paulk, M.C., Curtis, B., Chrissis, M.B., Weber, C.V. (1993) Capability Maturity Model for Software, Version 1.1. *IEEE Software*, **10**(4), 18-27.

Perry, D.E., Staudenmayer, N.A., Votta, L.G. (1994) People, Organizations, and Process Improvement. *IEEE Software*, **11**, 36-45.

Pfleeger, S.L. (2001) *Software Engineering: Theory and Practice (2nd Edition)*. Prentice Hall, Upper Saddle River, New Jersey, USA.

Premkumar, G., Potter, M. (1995) Adoption of Computer Aided Software Engineering (CASE) Technology: An Innovation Adoption Perspective. *Database*, **26**(2-3), 105-124.

Pressman, R.S. (2000) *Software Engineering - A Practioner's Approach*. McGraw-Hill Inc, Maidenhead, England.

Purvis, R.L., Sambamurthy, V., Zmud, R.W. (2000) The Development of Knowledge Embeddedness in CASE Technologies within Organizations. *IEEE Transactions on Engineering Management*, **47**(2), 245-257.

REAIMS (1997) REAIMS Home Page (Esprit Project 8649). `http://www.comp.lancs.ac.uk/computing/research/cseg/projects/reaims/` (13.12.2004).

Roth, M.A., Wolfson, D.C., Kleewein, J.C., Nelin, C.J. (2002) Information Integration: A New Generation of Information Technology. *IBM Systems Journal*, **41**(4), 563-577.

Santana Filho, O.V., Kochan, K.G. (2001) The Importance of Requirements Engineering for Software Quality. *Proceedings of the World Multiconference on Systemics, Cybernetics and Informatics. Volume I: Information Systems Development*. Orlando, Florida, USA, 7/2001, IIIS, 529-532.

Sawyer, P., Sommerville, I., Viller, S. (1998) Requirements Process Improvement Through the Phased Introduction of Good Practice. *Software Process: Improvement and Practice*, **3**(1), 19-34.

Siltanen, A. (1992) *CASE Adaptation Process and its Success Factors*. Technical Reports TR-2, Department of Computer Science, University of Jyväskylä, Jyväskylä.

Sitol, A.A. (2002) CASE Technology. *Student Conference on Research and Develop-*

ment, Kota Kinabalu, Malaysia, 2002, IEEE Computer Society, 54-57.

Sodhi, J. (1991) *Software Engineering Methods, Management, and Case Tools*. McGraw Hill, Blue Ridge Summit, PA, USA.

Sommerville, I. (2000) *Software Engineering (6th Edition)*, Addison-Wesley Pub Co, New York.

Sommerville, I., Sawyer, P. (1997) *Requirements Engineering: A Good Practice Guide*, John Wiley & Sons, Chichester, England.

SQAS19.01.00 - 2000 (2000) *Guidelines for Requirements Management*. Software Quality Assurance Subcommittee, USA.

Thomas, I., Nejmeh, B.A. (1992) Definition of Tool Integration for Environments. *IEEE Software*, **9**(2), 29-35.

W3C 2004 *Extensible Markup Language (XML)*. The World Wide Web Consortium, `http://www.w3.org/XML/` (13.12.2004).

Welch, B., Jones, K., Hobbs, J. (2003) *Practical Programming in Tcl and Tk, 4th ed*. Prentice Hall PTR, Upper Saddle River, New Jersey, USA.

Wicks M. (2004) *Tool Integration in Software Engineering: The State of the Art in 2004*. Technical Report HW-MACS-TR-0021, Heriot Watt University, Edinburgh, England.

Wiegers, K.E. (1999) *Software Requirements*. Microsoft Press, Washington.

Yphise (2002) *Software Assessment Report: Requirements Management Tools*. Yphise, USA.

Zave, P. 1997 Classification of Research Efforts in Requirements Enigneering. *ACM Computing Surveys*, **29**(4), 315-321.