Torsti Töllinen

MALLIPERUSTAISTA ARKKITEHTUURIA TUKEVIEN CASE-VÄLINEIDEN ARVIOINTI- JA VALINTAMENETELMÄ

Tietojärjestelmätieteen
pro gradu -tutkielma
4.12.2002

Jyväskylän yliopisto
Tietojenkäsittelytieteiden laitos
Jyväskylä

TIIVISTELMÄ

Töllinen, Torsti Johannes
Malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmä / Torsti Töllinen
Jyväskylä: Jyväskylän yliopisto, 2002.
100 s.
Tutkielma

Malliperustainen arkkitehtuuri (Model Driven Architecture) on koko ohjelmistotuotantoprosessia tukeva ohjelmistoarkkitehtuuri, joka perustuu järjestelmien mallintamiseen teknologiariippumattomasti. Arkkitehtuurin tavoitteena on automatisoida ohjelmistotuotantoprosessia sekä mahdollistaa järjestelmien kustannustehokas siirtäminen teknologiaratkaisusta toiseen. Ohjelmistotuotantoprosessin automatisointi vaatii arkkitehtuuria tukevia CASE-välineitä. Tässä tutkielmassa määritellään ja kuvataan malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmä.

Tutkielma on tutkimusotteeltaan käsitteellis-konstruktiivinen. Tutkielman käsitteellis-teoreettisessa osuudessa esitellään kirjallisuuskatsaukseen perustuen malliperustainen arkkitehtuuri ja luodaan kattava kuva olemassa olevista CASE-välineiden arviointi- ja valintamenetelmistä. Tutkielman konstruktiivisessa osassa kuvataan ja määritellään tutkielman tuloksena kehitetty CASE-välineiden arviointi- ja valintamenetelmä malliperustaiselle arkkitehtuurille.

Tutkielman keskeisimpänä tuloksena on malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmä. Tämän lisäksi tutkielmassa on määritelty malliperustaiseen arkkitehtuuriin liittyvät keskeisimmät käsitteet sekä kuvattu arkkitehtuurin, ohjelmistotuotantoprosessin vaiheiden ja ohjelmistotuotantomenetelmien suhteet.

AVAINSANAT: CASE-välineiden arviointi- ja valintamenetelmä, vaihejako, arviointikriteeri, Model Driven Architecture, Unified Modeling Language

KIITOKSET

Neljän kuukauden työn jälkeen on tullut aika kiittää tämän tutkielman tekemisen mahdollistaneita ja siinä avustaneita henkilöitä. Pro gradu -tutkielman aiheenvalintani olisi tuskin koskettanut malliperustaista arkkitehtuuria, jollei TietoEnator Oyj:n teknologiajohtaja Ari Hirvonen olisi siitä ollut kiinnostunut. Kiitos myös muille TietoEnator Oyj:n henkilöille työtäni koskeneista asiantuntevista kommenteista.

Tutkielmani ohjaajana on toiminut professori Heikki Saastamoinen, jonka suora ja tiivis palaute on ollut suureksi avuksi tutkielmani eri vaiheissa. Kiitokset ansaitsee myös opiskelukaverini Janne Kulin, jonka kommentit ovat olleet erityisen tarpeellisia tutkielman konstruktiivisen osuuden osalta. Lopuksi haluan kiittää Piia Jaatista monista mielenkiintoisista metodisista keskusteluista ja asiantuntevista kielellisistä kommenteista.

SISÄLLYS

1 JOHDANTO 6

2 MALLIPERUSTAINEN ARKKITEHTUURI 9
2.1 Ohjelmistoarkkitehtuuri 9
2.2 Peruskäsitteistö 11
2.2.1 Malli 11
2.2.2 Näkökulma 11
2.2.3 Sovellusalusta 12
2.2.4 Transformaatio 12
2.3 Ohjelmistotuotannon ongelmat ja malliperustainen arkkitehtuuri 12
2.4 Arkkitehtuurin mallit 14
2.4.1 Sovellusalustariippumaton malli 15
2.4.2 Sovellusalustariippuvainen malli 16
2.5 Mallien transformaatio ja ohjelmakoodin generointi 17
2.6 Standardit malliperustaisessa arkkitehtuurissa 22
2.6.1 Unified Modeling Language 23
2.6.2 Meta-Object Facility 26
2.6.3 Common Warehouse Metamodel 29
2.6.4 XML Metadata Interchange 30
2.7 Malliperustaisen arkkitehtuurin mukainen ohjelmistotuotantoprosessi 31
2.8 Malliperustaisen arkkitehtuurin haasteet ja mahdolliset edut 34
2.8.1 Haasteet 35
2.8.2 Edut 37
2.9 Yhteenveto 37

3 CASE-VÄLINEIDEN ARVIOINTI JA VALINTA 40
3.1 CASE-välineiden arviointiin sekä valintaan liittyvät ongelmat ja riskit 41
3.1.1 Ongelmia ja riskejä 41
3.1.2 Ratkaisukeinoja ongelmien ja riskien välttämiseksi 43
3.2 Menetelmiä CASE-välineiden arviointiin ja valintaan 45
3.3 Cookin, Lamin, Powellin, Vickersin ja Williamsin menetelmä 48
3.3.1 Vaihejako 49
3.3.2 Arviointikriteerit 52
3.4 Le Blancin ja Kornin menetelmä 53
3.4.1 Vaihejako 53
3.4.2 Arviointikriteerit 56
3.5 ISO/IEC 14102 -standardin mukainen menetelmä 57
3.5.1 Vaihejako 57

3.5.2 Arviointikriteerit 63
3.6 Yhteenveto 64

4 CASE-VÄLINEIDEN ARVIOINTI- JA VALINTAMENETELMÄ 66
4.1 Vaatimuksia menetelmälle 67
4.2 Vaihejako 68
4.3 Arviointikriteeristö 80
4.3.1 Esiarviointikriteerit 81
4.3.2 Arviointikriteerit 82
4.4 Yhteenveto 90

5 YHTEENVETO 92

LÄHTEET 96

1 JOHDANTO

Maailma muuttuu nopeasti. Teknologinen kehitys mahdollistaa uusien ja entistä monimutkaisempien järjestelmien kehittämisen. Uusia sovellusalustoja[1] ilmaantuu markkinoille muutaman vuoden välein ja jokaisen niistä väitetään olevan seuraava suuri askel, jonka varaan voidaan järjestelmiä rakentaa pitkälle tulevaisuuteen. Teknologia kehittyy ja vanhat järjestelmät jäävät elämään omaa elämäänsä. Yrityksillä saattaa olla useita vanhoja, niin sanottuja perinnejärjestelmiä, jotka toimivat eri sovellusalustoilla. Vanhat, usein yritysten liiketoiminnan kannalta kriittiset järjestelmät sisältävät paljon liiketoimintaan suoraan liittyvää logiikkaa, jonka siirtäminen vanhasta arkkitehtuuriratkaisusta uuteen on usein työlästä ja vaikeaa. Ongelmaksi muodostuukin järjestelmien siirtäminen vanhasta arkkitehtuurista uuteen mahdollisimman pienillä kustannuksilla, tai vaihtoehtoisesti vanhojen järjestelmien integroiminen uusien järjestelmien kanssa.

Malliperustainen arkkitehtuuri (*Model Driven Architecture*) on koko ohjelmistotuotantoprosessia tukeva ohjelmointikieli-, ohjelmistotoimittaja- ja sovellusalustariippumaton ohjelmistoarkkitehtuuri, joka mahdollistaa järjestelmien siirtämisen kustannustehokkaasti sovellusalustalta toiselle. Arkkitehtuurin tavoitteena on myös ohjelmistotuotantoprosessin automatisointi sitä tukevien CASE-välineiden avulla. *CASE-välineellä* tarkoitetaan tässä yhteydessä sellaista sovellusta, joka automatisoi ohjelmistotuotantoprosessia, järjestelmien ylläpitoa tai projektinhallintaa.

Malliperustaisen arkkitehtuurin laaja-alainen soveltaminen ohjelmistotuotannossa edellyttää sitä tukevia CASE-välineitä, joille arkkitehtuuri asettaa kuitenkin aivan omanlaisensa vaatimukset. Maailma muuttuu ja sen mukana muuttuvat ja kehittyvät myös eri ohjelmistovälineet ja CASE-välineet. Tämän vuoksi organisaatiot ja yritykset joutuvat yhä useammin arviointi- ja valintatilanteen eteen, jossa on valittava väline

---

[1] Sovellusalustalla tarkoitetaan tässä yhteydessä sellaista ohjelmistojen kehitys- ja suoritusympäristöä, jonka määrittelemien palveluiden avulla voidaan kehittää ja suorittaa sovelluksia.

hyvin samanlaisten välineiden joukosta. CASE-välineiden arviointiin ja valintaan on kehitetty useita erilaisia menetelmiä. Menetelmää, jossa otettaisiin huomioon malliperustaisen arkkitehtuurin vaatimukset CASE-välineiden arvioinnille ja valinnalle, ei kuitenkaan ole aikaisemmin kehitetty.

Tämän tutkielman tavoitteena on kehittää menetelmä malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointiin ja valintaan. Menetelmä koostuu arviointi- ja valintaprosessin määrittelevästä vaihejaosta, sekä arviointikriteeristöstä, joka mahdollistaa eri välineiden yhdenmukaisen arvioinnin. Tutkielman tutkimusongelmana on selvittää, miten malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintaprosessin tulisi edetä, ja toisaalta millaisten arviointikriteerien mukaan CASE-välineitä tulisi vertailla. Tutkielmassa ei käsitellä CASE-välineitä, markkinoilla olevia sovellusalustoja, tai muita teknisiä ratkaisuja yksityiskohtaisella tasolla. Tutkielman lukijan oletetaan tietävän perusteet eri sovellusalustoista ja komponenttimalleista, ohjelmistotuotantomenetelmistä ja järjestelmien mallinnuskielistä.

Tutkielmassa on käytetty käsitteellis-konstruktiivista tutkimusotetta. Tutkielma jakautuu kahteen osaan. Tutkielman kirjallisuuskatsauksessa tarkastellaan malliperustaista arkkitehtuuria ja sen asettamia vaatimuksia CASE-välineille. Tämän lisäksi luodaan kattava katsaus olemassa oleviin CASE-välineiden arviointi- ja valintamenetelmiin. Tutkielman konstruktiivisessa osassa kehitetään kirjallisuuskatsaukseen pohjautuen malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmä. Tutkielman kontribuutioina voidaan pitää seuraavia tuloksia: 1) malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmän kehittäminen sekä 2) malliperustaisen arkkitehtuurin jäsentäminen, sen keskeisten käsitteiden määritteleminen sekä arkkitehtuurin, ohjelmistotuotantoprosessin vaiheiden ja ohjelmistotuotantomenetelmien suhteiden määrittäminen.

Kirjallisuuskatsauksessa lähteinä on käytetty standardeja, tieteellisiä artikkeleita sekä julkaistua kirjallisuutta. Tutkielmassa on käytetty kattavasti lähteiden osalta tämän hetken tietämystä malliperustaisesta arkkitehtuurista ja CASE-välineiden arviointi- ja

valintamenetelmistä. Tutkielman tekohetkellä malliperustaisesta arkkitehtuurista ei ole olemassa julkaistua kirjallisuutta.

Tutkielma koostuu viidestä luvusta. Johdannon jälkeen luvussa kaksi esitellään ja kuvataan malliperustainen arkkitehtuuri, sen perustana olevat standardit sekä arkkitehtuurin mukainen ohjelmistotuotantoprosessi. Tutkielman kolmannessa luvussa tuodaan esille CASE-välineiden arviointiin ja valintaan liittyviä ongelmia, niiden ratkaisukeinoja sekä esitellään seitsemän erilaista ohjelmistovälineiden ja CASE-välineiden arviointi- ja valintamenetelmää. Kolmea näistä menetelmistä tarkastellaan myös tarkemmalla tasolla. Luvussa neljä kehitetään edellisiin lukuihin pohjautuen malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmä. Tutkielman päättää yhteenveto.

2 MALLIPERUSTAINEN ARKKITEHTUURI

Object Management Groupin (OMG[2]) kehittämä malliperustainen arkkitehtuuri (MDA) on koko ohjelmistonkehitysprosessia tukeva ohjelmointikieli-, ohjelmistotoimittaja- ja sovellusalustariippumaton ohjelmistoarkkitehtuuri. Malliperustainen arkkitehtuuri on määritys (specification), joka on tämän tutkielman tekohetkellä vielä äänestysvaiheessa OMG:n viralliseksi standardiksi. Tämän luvun tarkoituksena on kertoa arkkitehtuurista ja sen soveltamisesta, sekä toisaalta myös selventää vielä varsin epäyhtenäistä ja kypsymätöntä käsitteistöä, joka malliperustaiseen arkkitehtuuriin liittyy. Luvun tärkeimpänä tehtävänä on selventää niitä vaatimuksia, joita malliperustainen arkkitehtuuri asettaa CASE-välineiden arviointiin ja valintaan.

Luvun aluksi määritellään ohjelmistoarkkitehtuurin käsite ja verrataan kirjallisuudessa esitettyjä määritelmiä malliperustaiseen arkkitehtuuriin. Tämän jälkeen esitellään itse arkkitehtuuri ja sen perustana olevat standardit. Luvun lopuksi kuvataan arkkitehtuurin mukainen ohjelmistotuotantoprosessi, sekä arkkitehtuurin haasteet ja sen mukanaan mahdollisesti tuomat edut.

2.1 Ohjelmistoarkkitehtuuri

Ohjelmistoarkkitehtuurille on olemassa useita määritelmiä, jotka eroavat toisistaan perusperiaatteiltaankin. Garlanin ja Shawn (1994) mukaan ohjelmistoarkkitehtuuri on erilaisia tietojenkäsittelytehtäviä suorittavien komponenttien ja näiden välisten yhteenliittymien kuvausten muodostama joukko.

---

[2] Object Management Group (www.omg.org) on ei-kaupallinen standardointijärjestö, joka kehittää avoimia standardeja hajautetun tietojenkäsittelyn alalla. Tunnetuimpia OMG:n standardeja ovat Unified Modeling Language -mallinnuskieli (UML) ja Common Object Request Broker Architecture -komponenttimalli (CORBA).

Abd-Allah, Boehm, Clark ja Gacek (1995) näkevät ohjelmistoarkkitehtuurin muodostuvan seuraavista osista:

- joukosta ohjelmistoja ja järjestelmäkomponentteja sekä niiden välisistä yhteenliittymistä ja rajoitteista,
- joukosta sidosryhmien järjestelmälle asettamia vaatimuksia sekä
- selvityksestä, miten ohjelmistot, komponentit, liittymät ja rajoitteet määrittelevät ohjelmiston, joka toteuttaa sille asetetut vaatimukset.

Perry ja Wolf (1992, 40-52) katsovat ohjelmistoarkkitehtuurin muodostuvan suunnitteluelementeistä, joilla on tietty muoto. Suunnitteluelementtejä ovat tietojenkäsittelyä suorittavat elementit (komponentit), elementtien väliset liittymät ja tietoelementit.

Perinteisistä ohjelmistoarkkitehtuurin määritelmistä malliperustaista arkkitehtuuria parhaiten kuvaa Perryn ja Wolfin (1992) määritelmä. Sekään ei kuitenkaan ota kantaa arkkitehtuurin tehtävään. DSouza (2001, 22) on määritellyt ohjelmistoarkkitehtuurin olevan joukko periaatteita, sääntöjä tai suunnittelumalleja (design pattern), jotka rajoittavat ohjelmistojen suunnittelijoita ja antavat toisaalta heille ohjeistuksen (guidline) tietynlaisen järjestelmän kehittämiseen. DSouza (2001, 22) määrittelee ohjelmistoarkkitehtuurin osana arkkitehtuurillisen tyylin (architectural style), jonka mukaisia ovat kaikki kehitettävän järjestelmän määritykset ja suunnitelmat, jotka ilmaistaan jollakin mallinnuskielellä riippuen kulloisestakin arkkitehtuurillisesta tyylistä.

Määritelmien pohjalta voidaan todeta malliperustaisen arkkitehtuurin olevan ohjelmistoarkkitehtuuri, joka muodostaa arkkitehtuurillisen tyylin kehittää ja suunnitella järjestelmiä.

2.2 Peruskäsitteistö

Seuraavissa kappaleissa kuvataan malliperustaiseen arkkitehtuuriin liittyvä peruskäsitteistö.

2.2.1 Malli

Malli (model) on formaali esitys järjestelmän toiminnasta, rakenteesta ja käyttäytymisestä (Model Driven Architecture – A Technical Perspective 2001, 3). Malli voidaan Boochin, Jacobsonin ja Rumbaughin (1999a, 419-422) mukaan myös ymmärtää yksinkertaistettuna kuvauksena todellisuudesta, jossa todellisuuden määrittelee mallinnettava järjestelmä. Malli on formaali silloin, kun se perustuu johonkin kieleen, jolla on hyvin määritelty muoto, semantiikka ja säännöstö. Esimerkiksi kaavio, joka muodostuu suorakulmioista, viivoista ja nuolista ei ole malliperustaisen arkkitehtuurin mukainen malli, jos suorakulmion, viivan tai nuolen merkitystä ei ole formaalisti määritelty.

2.2.2 Näkökulma

Mallit ovat abstraktioita todellisuudesta. Jokainen malli kuvaa kohdealuettaan tietystä *näkökulmasta* (viewpoint), jolloin kaikki näkökulman kannalta epärelevantit yksityiskohdat jätetään kuvauksen ulkopuolelle (Information Technology – Open Distributed Processing – Reference Model: Foundations 1996, 2-3). Malli, joka perustuu tiettyyn abstraktiokriteeriin kuvaa järjestelmää tietystä näkökulmasta. Malliperustainen arkkitehtuuri muodostaa arkkitehtuurilliseen tyyliin, joka määrittää käytettävät mallit. Mallien näkökulmien valinta on kuitenkin riippuvainen kulloisestakin mallinnustilanteesta. (Model Driven Architecture (MDA) 2001, 4) Esimerkiksi ODP-referenssimallin osa 3 (Information Technology – Open Distributed Processing – Reference Model: Architecture 1996, 3-4) määrittää viisi järjestelmien mallintamisessa käytettävää näkökulmaa, jotka ovat: yritysnäkökulma (enterprise viewpoint), informaationäkökulma (information viewpoint), tietojenkäsittelynäkökulma

(computational viewpoint), suunnittelunäkökulma (engineering viewpoint) ja teknologianäkökulma (technology viewpoint).

2.2.3 Sovellusalusta

Perinteisesti *sovellusalustan* ymmärretään olevan sellainen ohjelmistojen kehitys- ja suoritusympäristö, jonka määrittelemien palveluiden avulla voidaan kehittää ja suorittaa hajautettuja sovelluksia. Tällaisia sovellusalustoja ovat esimerkiksi Java 2 Enterprise Edition/Enterprise JavaBeans (J2EE/EJB), Microsoft .NET/COM+ ja Common Object Request Broker Architecture (CORBA). Malliperustaisen arkkitehtuurin yhteydessä sovellusalustan määritelmällä viitataan myös sen sisältämään riippuvuuteen tietystä teknologiasta (Model Driven Architecture (MDA) 2001, 5). Malliperustaisen arkkitehtuurin mukaisen ohjelmistokomponentin toiminnallisuus on erotettavissa sovellusalustan teknologisista yksityiskohdista. Malliperustainen arkkitehtuuri on sovellusalustoihin verrattaessa korkeammalla tasolla. Sen voidaankin todeta olevan sovellusalustojen metamalli (Siegel 2001, 9). Järjestelmiä suunnitellaan arkkitehtuurin mukaisesti ja sovellusalustojen tehtäväksi jää toteuttaa suunnitelmat teknologisista lähtökohdista katsoen.

2.2.4 Transformaatio

Transformaatiolla (mapping/transformation) tarkoitetaan uusien mallien johtamista transformaatiosäännöstön avulla yhdestä tai useammasta mallista (Koch, Uhl & Weise 2002, 5). *UML-profiilit* määrittävät mallien transformaatiossa käytettävät säännöt, joiden mukaisesti lähtökohtana olevasta mallista johdetaan yksi tai useampia tulosmalleja.

2.3 Ohjelmistotuotannon ongelmat ja malliperustainen arkkitehtuuri

Ohjelmistotuotanto on kehittynyt merkittävästi viimeisen kymmenen vuoden aikana olioteknologioiden ja Unified Modeling Language -mallinnuskielen (UML) ansiosta

(Brassard & Cardinal 2002, 1-2). Ongelmia on kuitenkin edelleenkin. Brassard ja Cardinal (2002, 1-2) ovat tunnistaneet viisi ohjelmistotuotannon ongelmaa, joihin malliperustainen arkkitehtuuri pyrkii tarjoamaan ratkaisua:

1) sovellusalustojen, käyttöjärjestelmien ja ohjelmointikielten vaihtuvuus muutamien vuosien välein,
2) liiketoiminnan kohdealueen (business domain) muuttuminen,
3) UML-mallien monimutkaisuus ja kehittymättömyys,
4) UML-mallien ylläpitäminen sekä
5) UML-mallien toteuttaminen.

Sovellusalustat, käyttöjärjestelmät ja ohjelmointikielet vaihtuvat muutamien vuosien välein. Tämän hetken sovellusalustaratkaisuja ovat esimerkiksi J2EE/EJB, .NET/COM+ ja CORBA. Uusia sovellusalustoja ja tapoja kehittää järjestelmiä tulee markkinoille muutaman vuoden välein. Jokainen niistä tarjoaa omat etunsa ja tämän vuoksi yritysten on mahdotonta sitoutua yhteen käyttöjärjestelmään, ohjelmointikieleen tai sovellusalustaan.

Liiketoiminnan kohdealueen muuttumisen vuoksi myös liiketoiminnan vaatimukset muuttuvat. Tämän vuoksi järjestelmiä on ylläpidettävä. Uudet liiketoimintamahdollisuudet, asiakkaiden vaatimukset, ongelmat vanhojen laitteistojen saatavuudessa, osaajien puute ja kehitysvälineiden huonous saattavat myös pakottaa siirtämään järjestelmä kokonaan uudelle sovellusalustalle. Järjestelmien siirtäminen uudelle sovellusalustalle on kallista ja aikaa vievää. Samalla tehdään myös suuri määrä turhaa työtä, koska liiketoimintalogiikka on jo kerran ohjelmoitu järjestelmiin.

UML-mallit ovat vielä monimutkaisia ja kehittymättömiä. Analyysimallin (ks. Booch, Jacobson & Rumbaugh 1999b, 173-215) tulisi olla toteutusteknologiariippumaton. Suunnittelumallin sen sijaan (ks. Booch ym. 1999b, 215-267) tulisi sisältää toteutusteknologiaan liittyvä tietämys ja ratkaisut. Toteutusteknologiaan liittyvien ratkaisujen (tapahtumanhallinta, hakemistopalvelut, pysyvyys ja tietoturva) kuvaaminen

on kuitenkin tällä hetkellä UML-kielellä vaikeaa ja tämän vuoksi myös mallien uudelleenkäyttö (reuse) eri toteutusteknologioiden kuvaamisessa on vaikeaa.

Suurin osa kehitettävistä järjestelmistä suunnitellaan ja mallinnetaan käyttäen UML-kieltä. Järjestelmät toteutetaan kuitenkin edelleen suurimmaksi osaksi käsin ohjelmoimalla ja UML-mallit toimivat yleensä vain ohjeistuksena järjestelmien toteuttajille. *UML-mallien toteuttaminen* käsin on virhealtista. Järjestelmien toteutuksen jälkeen *UML-mallit jäävät usein ylläpitämättä* ja niitä ei saateta ohjelmakoodin kanssa yhdenmukaisiksi. Ennen ei ole ollut järjestelmällistä tapaa varmistaa ohjelmakoodin ja sitä kuvaavan UML-mallin yhteneväisyyttä.

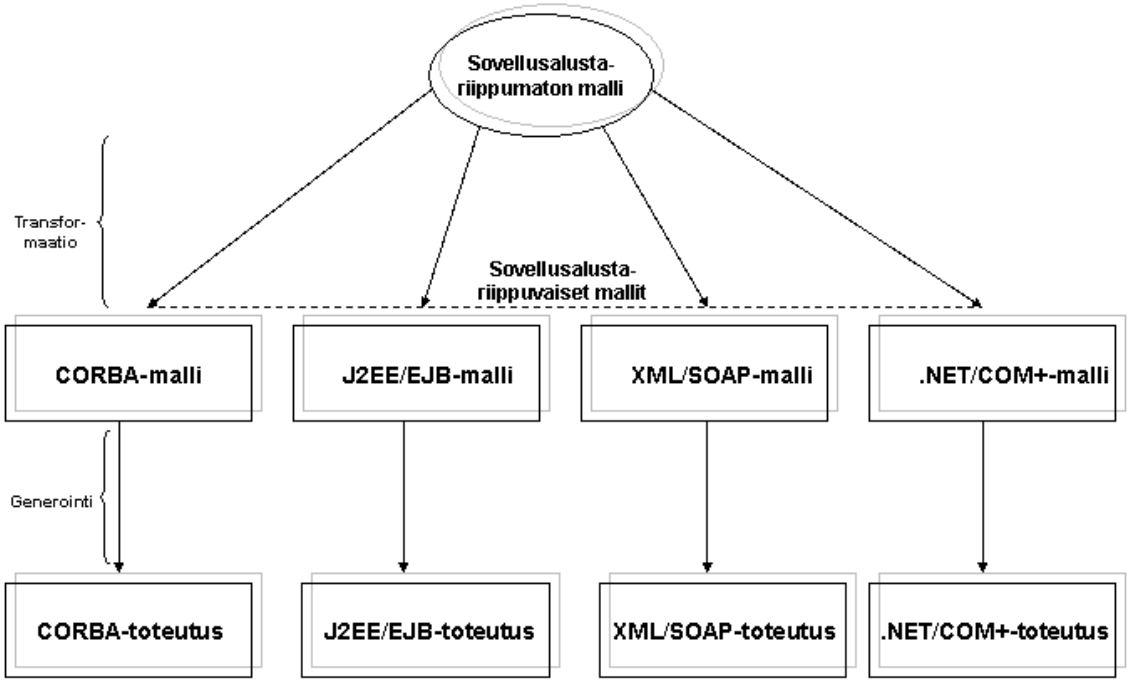
Malliperustainen arkkitehtuuri perustuu järjestelmien liiketoimintalogiikan ja kohdealueen mallintamiseen erillään sen toteuttavasta sovellusalustasta (Soley 2000, 4). Toisena arkkitehtuurin kulmakivenä on ohjelmistotuotannon mahdollisimman suuri automatisointi ohjelmakoodin generoinnilla suoraan malleista (Hazra 2002, 48-49). Arkkitehtuurin mukaisesti järjestelmiä mallinnetaan eri näkökulmista ja abstraktiotasoista (DSouza 2001, 2).

2.4 Arkkitehtuurin mallit

Malliperustainen arkkitehtuuri muodostuu erilaisista malleista, joista kukin kuvaa järjestelmää sen eri osista ja näkökulmista. Arkkitehtuurin mukaiset mallit voidaan jakaa kahteen eri luokkaan: *sovellusalustariippumattomiin* (platform-independent models / PIM) ja *sovellusalustariippuvaisiin* malleihin (platform-specific models / PSM). (Model Driven Architecture (MDA) 2001, 6)

Malliperustaisen arkkitehtuurin mukaisesti kuvatuista sovellusalustasta riippumattomista malleista transformoidaan (mallien transformaatio) sovellusalustasta riippuvaiset mallit halutulle sovellusalustalle. Sovellusalustariippuvaisista kuvauksista generoidaan ohjelmakoodi sovellusalustariippuvaiseksi toteutukseksi. Kuviossa 1 on

esitetty sovellusalustariippumaton- ja sovellusalustariippuvainen malli ja niiden suhde toisiinsa nähden.



KUVIO 1 Sovellusalustariippuvainen ja sovellusalustariippumaton malli.

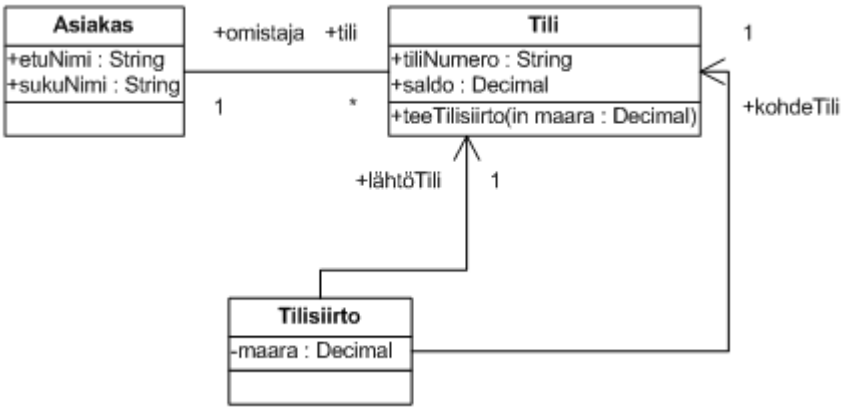
Järjestelmä mallinnetaan ja suunnitellaan UML-kielellä. Sekä sovellusalustariippumaton että sovellusalustariippuvainen malli ovat siten UML-notaation mukaisia. Vasta kunkin sovellusalustariippuvaisen mallin toteutus on lähdekielinen (source code).

2.4.1 Sovellusalustariippumaton malli

Sovellusalustariippumaton malli on formaali määritys järjestelmän toiminnasta ja rakenteesta UML-kielellä ilmaistuna (Model Driven Architecture (MDA) 2001, 6). Sovellusalustariippumaton malli kuvaa järjestelmän ilman sidoksia sen mahdollisesti toteuttavaan teknologiaan (Balcer & Mellor 2002, 11-12). Sovellusalustariippumattoman mallin muodostamiseen voivat osallistua sekä liiketoiminta-alan asiantuntijat että järjestelmien kehittämisen asiantuntijat. Sovellusalustariippumattomuudesta seuraa Siegelin (2001, 5) mukaan kaksi etua. Liiketoiminta-alan asiantuntijat pääsevät alusta lähtien osallistumaan järjestelmän

kehittämiseen ja mahdolliset loogiset liiketoimintaan liittyvät virheet saadaan poistettua mahdollisimman aikaisessa vaiheessa järjestelmän kehittämistä. Lisäksi teknologisen riippumattomuutensa vuoksi sovellusalustariippumaton malli on käyttökelpoinen vuosia ja vaatii ainoastaan muutoksia silloin, kun kohdealue muuttuu.

Kuviossa 2 on esimerkki sovellusalustariippumattoman mallin luokkakaaviosta ilmaistuna UML-kielellä. Luokkakaavio on toteutusteknologiariippumaton kuvaus asiakkaasta, jolla voi olla useita tilejä. Asiakkaan tileihin voi kohdistua tilisiirtoja tai hän voi itse tehdä tilisiirron jollekin toiselle tilille. Tilisiirto ei voi yhtä aikaa käsittää kuin yhden lähtötilin ja yhden kohdetilin.



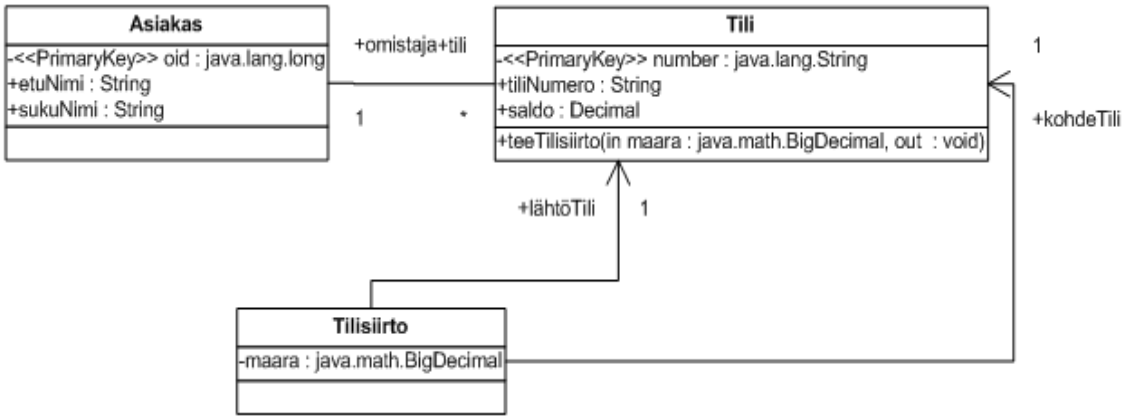
KUVIO 2 Esimerkki sovellusalustariippumattomasta mallista (Koch, Uhl & Weise 2002, 25).

2.4.2 Sovellusalustariippuvainen malli

Sovellusalustariippuvainen malli on formaali määritys järjestelmän toiminnasta ja rakenteesta UML-kielellä ilmaistuna teknologiset lähtökohdat mukaan lukien. Sovellusalustariippuvainen malli kuvaa järjestelmän toteutuksen jonkin toteutusteknologian lähtökohdista käsin. Sovellusalustariippuvainen malli muodostetaan sovellusalustariippumattomasta mallista halutun sovellusalustan mukaiseksi. UML-profiilit määrittävät eri sovellusalustoille erilaiset UML-notaatiot, joiden mukaisia sovellusalustariippuvaisten mallien tulee olla. UML-profiilissa määritetään siis

säännöstö, jonka mukaisesti sovellusalustariippumaton UML-malli transformoidaan sovellusalustariippuvaiseksi UML-malliksi. Tällä hetkellä määritettynä on vasta UML-profiili CORBA-sovellusalustalle (UML™ Profile for CORBA™ Specification 2002), mutta kaikille suosituimmille sovellusalustoille on valmistumassa profiilit.

Kuviossa 3 on esimerkki sovellusalustariippuvaisen mallin luokkakaaviosta. Luokkakaavio kuvaa kuvion 2 mukaisen tilanteen. Kuvio 3 on kuitenkin ilmaistu J2EE/EJB-sovellusalustan mukaisesti ja on siten myös sovellusalustariippuvainen.



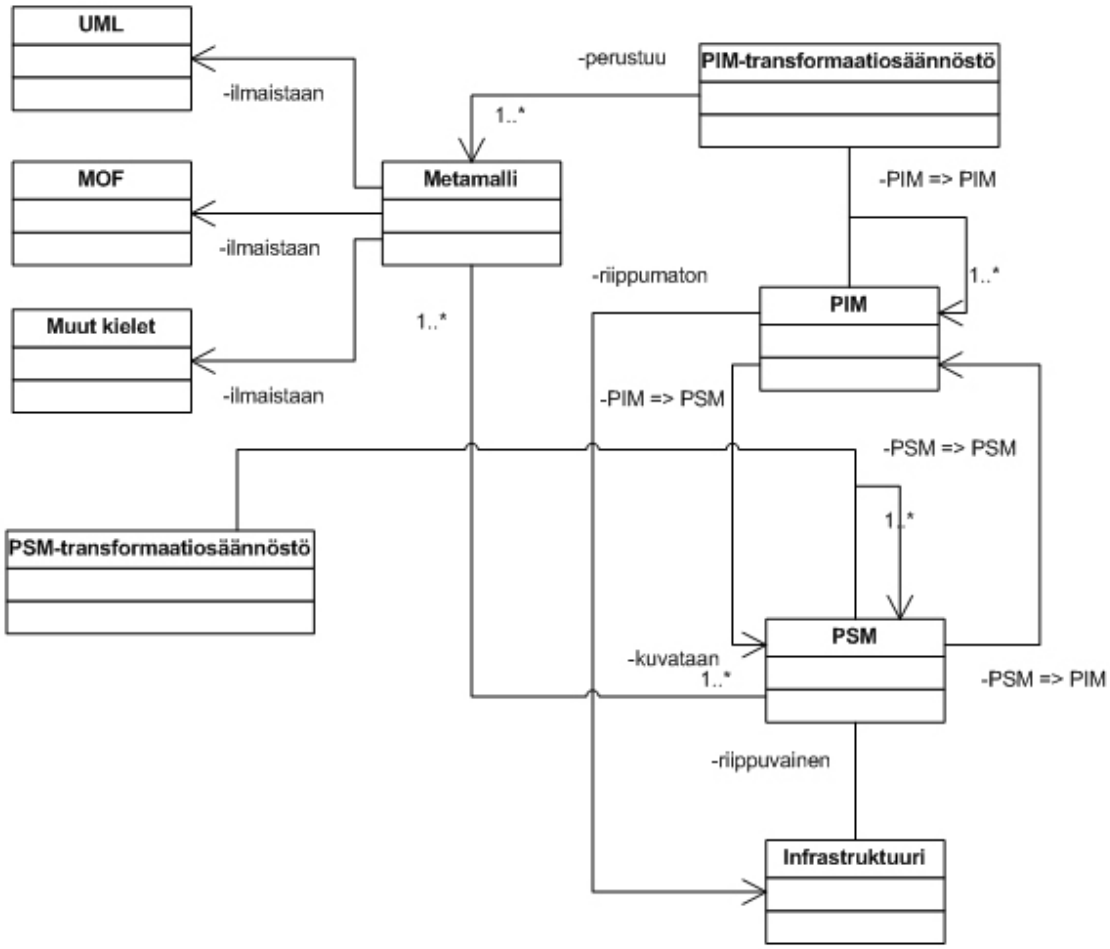
KUVIO 3 Esimerkki sovellusalustariippuvaisesta mallista (Koch ym. 2002, 26).

Kuviossa 3 tarkennetaan J2EE/EJB-sovellusalustan mukaiset komponenttityypit (esimerkiksi entity bean) sekä määritellään mahdolliset attribuutit, jotka toimivat tietokanta-avaimina (primary key) ja attribuuttien tietotyypit (esimerkiksi string).

2.5 Mallien transformaatio ja ohjelmakoodin generointi

Mallien transformaatiolla tarkoitetaan prosessia, jonka avulla voidaan muodostaa uusia malleja lähtökohtana olleesta mallista tietyn voimassa olevan säännöstön mukaisesti (Model Driven Architecture (MDA) 2001, 12). Ohjelmakoodin generoinnilla tarkoitetaan tässä yhteydessä jonkin ohjelmointikielen määrityksen mukaisen lähdekoodin tuottamista automaattisesti olemassa olevasta UML-mallista.

Kuvio 4 esittää metamallia malliperustaisesta arkkitehtuurista ja mallien transformaatiosta (Model Driven Architecture (MDA) 2001, 12).



KUVIO 4 Malliperustaisen arkkitehtuurin metamalli.

Kuviosta 4 nähdään, että mallien transformaatio voidaan suorittaa neljällä tavalla:

1) **Platform Independent Model => Platform Independent Model**. Tätä transformaation muotoa käytetään tarkennettaessa sovellusalustariippumatonta mallia siten, että siihen ei kuitenkaan lisätä sovellusalustariippuvaista tietoa.
2) **Platform Independent Model => Platform Specific Model**. Tätä transformaatiota käytetään, kun halutaan muodostaa sovellusalustariippumattomasta mallista sovellusalustariippuvainen malli sovellusalustan UML-profiilin mukaisesti.

3) **Platform Specific Model => Platform Specific Model**. Tätä transformaation muotoa käytetään tarkennettaessa sovellusalustariippuvaista mallia esimerkiksi lisäämällä siihen tietoa tarvittavista palveluista (service) ja niiden konfiguroinnista.
4) **Platform Specific Model => Platform Independent Model**. Tätä transformaation muotoa käytetään, kun muodostetaan tietyn teknologian mukaisesta toteutuksesta sovellusalustariippumaton malli (reverse engineering).

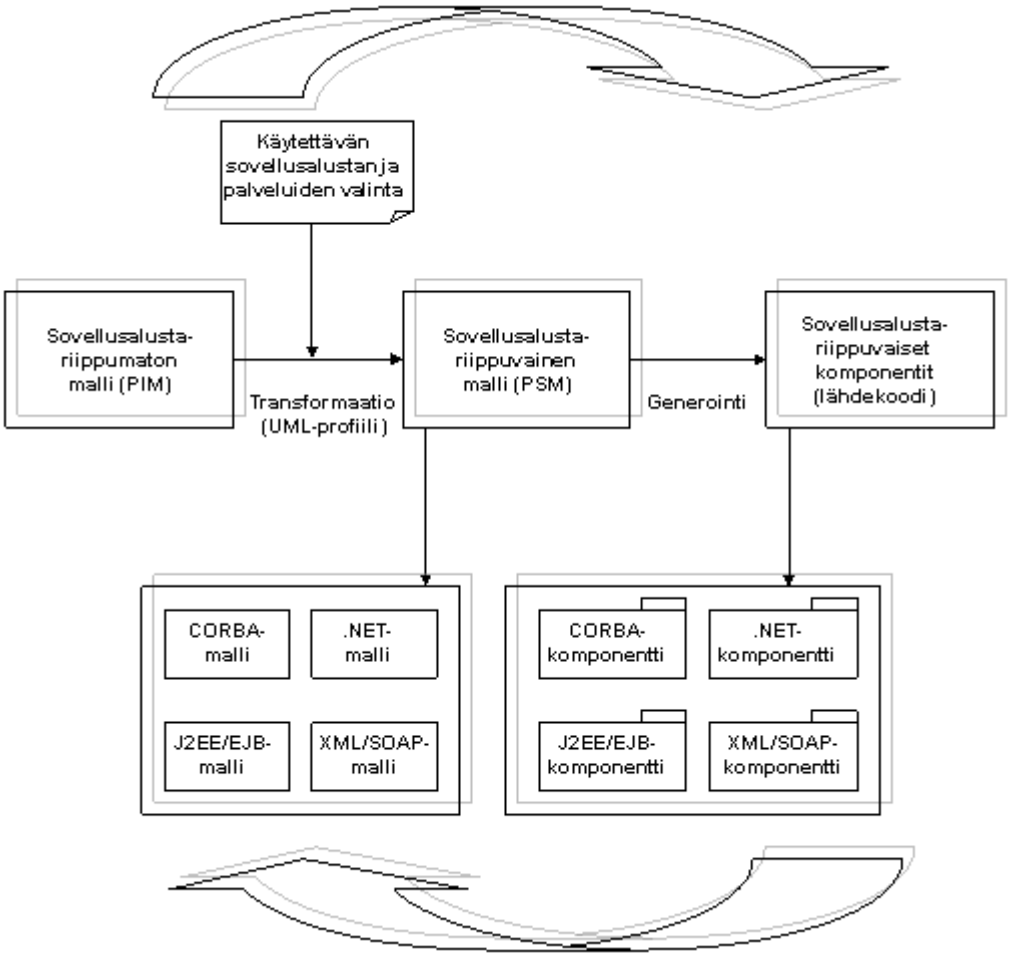
UML-profiilit ja niiden toteutus ovat tärkeässä osassa toteutettaessa mallien välisiä transformaatioita. UML-profiilit määrittävät säännöstön, jonka mukaisesti sovellusalustariippumaton malli muutetaan sovellusalustariippuvaiseksi malliksi. Käytännössä UML-profiilien säännöstöt toteutetaan malliperustaista arkkitehtuuria tukeviin CASE-välineisiin esimerkiksi jonkin skriptikielen avulla tai vastaavasti käyttämällä kokonaan muita sovelluksia transformaation toteuttamiseksi. Tällöin tiedonsiirto välineiden välillä tapahtuu XML Metadata Interchange -kielen (XMI) avulla. On myös mahdollista, että CASE-välineet tukevat esimerkiksi UML-profiileiden tuontia ja vientiä XMI-kielen välityksellä.

Mallien transformaatio ja ohjelmakoodin generointi on iteratiivinen prosessi (Siegel 2001, 7). Sovellusalustariippumattoman mallin ensimmäisen iteraation ollessa valmis siitä muodostetaan sovellusalustariippuvainen malli halutun sovellusalustan UML-profiilin mukaisesti. Kuviossa 5 on esitetty mallien transformaatiosta ja ohjelmakoodin generoinnista prosessikaavio. Prosessikaavio on Hazran (2002, 50) näkemyksestä mukailtu malli, joka osoittaa malliperustaisen arkkitehtuurin mukaisen ohjelmistotuotannon olevan iteratiivista kehittämistä, jonka lopputuloksena on hänen mukaansa eri sovellusalustojen mukaisia ohjelmistokomponentteja[3]. Siegelin (2001, 7)

---

[3] Ohjelmistokomponentti on sellainen järjestelmän fyysinen ja korvattavissa oleva osa, joka toteuttaa jonkin järjestelmään liittyvän toiminnallisuuden ja tarjoaa määritellyn rajapinnan toiminnallisuuden käyttämiseksi (Booch, Rumbaugh & Jacobson 1999a, 345).

mukaan prosessin tuloksena tulisi olla myös kaikki komponenttien varsinaiseen ajoympäristöön liittyvät konfiguraatiotiedot ja tiedostot. Ajoympäristöön liittyvillä konfiguraatiotiedoilla ja tiedostoilla tarkoitetaan tässä esimerkiksi J2EE/EJB-sovellusalustan mukaisia koti- ja etärajapintaluokkia (home/remote interface) sekä komponentin ajonaikaisen toiminnan konfiguraatiotiedostoja (deployment descriptor).



KUVIO 5 Mallien transformaatio- ja ohjelmakoodin generointiprosessi.

Sovellusalustariippumattoman mallin transformaatio sovellusalustariippuvaiseksi malliksi voidaan suorittaa malliperustaisen arkkitehtuurin teknisen raportin mukaan (Model Driven Architecture – A Technical Perspective 2001, 14-15) kolmella eri tavalla:

1) Sovellusalustariippuvainen malli voidaan muodostaa sovelluskehittäjän toimesta suoraan käsin toimimalla halutun sovellusalustan UML-profiilin säännöstön mukaisesti.
2) Sovellusalustariippuvaisen mallin muodostamiseen voidaan osittain käyttää malliperustaista arkkitehtuuria tukevaa CASE-välinettä, joka tietyn algoritmin mukaisesti muodostaa osan mallista. Malli on kuitenkin tarkistettava ja siihen tulee lisätä tarvittavat osat käsin.
3) Sovellusalustariippuvainen malli voidaan muodostaa kokonaan malliperustaista arkkitehtuuria tukevan CASE-välineen avulla, joka tietyn algoritmin mukaisesti muodostaa valmiin mallin halutun UML-profiilin mukaisesti.

Tason 3 mukaiset täydelliset transformaatiot ovat kuitenkin malliperustaisen arkkitehtuurin teknisen raportin (Model Driven Architecture – A Technical Perspective 2001, 14) mukaan tällä hetkellä mahdollisia vain tietyissä rajoitetuissa ympäristöissä. Tällainen tilanne voi olla silloin, kun transformaatiossa ei tarvitse ottaa huomioon mahdollisia vanhoja järjestelmiä (legacy system), transformaation lähtökohtana oleva sovellusalustariippumaton malli on mahdollisimman rikas semanttisesti ja transformaation toteuttavat algoritmit ovat korkealaatuisia.

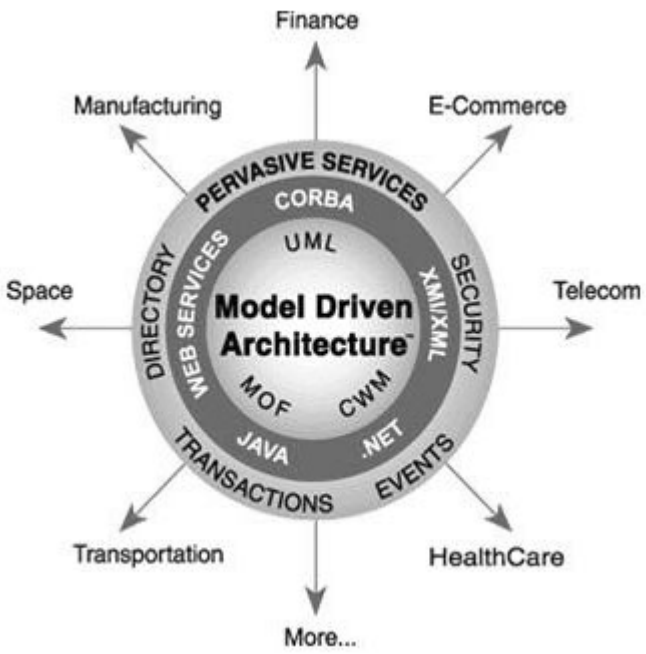
Siegelin (2001, 8) mukaan myös liiketoiminta-alueen ja sillä vaadittavien sovellusten rajoittuneisuus edesauttavat täydellisen transformaation saavuttamista jonkin CASE-välineen avulla. On myös olemassa esimerkkitapaus (Castain 2001), jossa malliperustaista arkkitehtuuria on sovellettu menestyksekkäästi siten, että on kehitetty oma sovellus pankkitoimialalle, jonka avulla on pystytty luomaan tason 3 mukaisia transformaatioita ja myös generoimaan suurin osa ohjelmakoodista.

Ajettavan (excutable) ohjelmakoodin generointi voidaan suorittaa periaatteessa samoilla vaihtoehtoisilla tavoilla kuin sovellusalustariippumattoman mallin transformaatio sovellusalustariippuvaiseksikin (Model Driven Architecture (MDA) 2001, 14-15). Säännöstönä vain toimii halutun sovellusalustan mukainen UML-profiili sekä sovellusalustan ohjelmointikieli ja sen määritys.

Ajettavan ohjelmakoodin generoinnissa on tähän asti ollut vaikeutena toiminnallisten ominaisuuksien generoiminen johtuen UML-kielen puutteista (ks. kappale 2.8.1). Rakenteellisten ominaisuuksien generoiminen on ollut huomattavasti helpompaa niiden yksinkertaisesta luonteesta johtuen (Model Driven Architecture – A Technical Perspective 2001, 14).

2.6 Standardit malliperustaisessa arkkitehtuurissa

Malliperustaisen arkkitehtuurin suhteet Object Management Groupin (OMG) muihin standardeihin, sovellusalustamäärityksiin, perustaviin palveluihin (pervasive services) ja eri toimialoihin on kuvattu kuviossa 5.



KUVIO 5 Malliperustainen arkkitehtuuri (Model Driven Architecture (MDA) 2001, 20).

Kuvion 5 ydin muodostuu kolmesta OMG:n järjestelmien mallintamiseen tarkoitetusta standardista, jotka ovat Unified Modeling Language (UML), Meta-Object Facility (MOF) ja Common Warehouse Metamodel (CWM). Malliperustaisen arkkitehtuurin

ytimeen kuuluvat myös UML-profiilit, jotka kuvaavat UML-notaation kukin tietylle sovellusalustamääritykselle, jotka on kuvattu kuviossa ydintä seuraavassa kehässä.

Perustavat palvelut on kuvattu kuviossa 5 uloimpana kehänä. Nämä palvelut voidaan ymmärtää sellaisina palveluina, jotka jokaisessa sovellusalustamäärityksessä on toteutettu omalla tavallaan. Perustavia palveluita ovat tyypillisesti tapahtumanhallinta, hakemistopalvelut, pysyvyyden turvaaminen sekä tietoturva. Sovellusalustariippumattomassa mallissa perustavat palvelut kuvataan UML-kielellä ottamatta kantaa niiden teknologiseen toteutukseen. Muodostettaessa sovellusalustariippuvainen malli generoidaan tarpeelliset sovellusalustan palveluiden kutsut automaattisesti, mikäli käytettävä CASE-väline sen mahdollistaa. (Model Driven Architecture (MDA) 2001, 20-22)

Kuviossa 5 uloimpana on kuvattu OMG:n toimialamääritykset, joita on tällä hetkellä 31 (Object Management Group 2002). Toimialamäärityksien on tarkoitus tulevaisuudessa kuvata jokaiselle toimialalle omat UML-notaationsa sovellusalustariippumattomien ja sovellusalustariippuvaisten mallien tuottamiseksi. Esimerkiksi rahoituksen toimialamääritys tulee kuvaamaan UML-notaation toimialan sovellusalustariippumattomien mallien muodostamiseen ja oman UML-notaation J2EE/EJB-sovellusalustan sovellusalustariippuvaisten mallien muodostamiseen. Suurin osa tämän hetken toimialamäärityksistä on vain CORBA-sovellusalustalle ja nekin kuvaavat vain kuinka CORBA:n rajapinnat (CORBA IDL) muodostetaan.

Seuraavissa kappaleissa kuvataan tarkemmin OMG:n omat standardit, joiden avulla malliperustaisessa arkkitehtuurissa luodaan, hallitaan, välitetään (interchange) ja julkaistaan malleja.

2.6.1 Unified Modeling Language

Unified Modeling Language on graafinen oliopohjaisten järjestelmien visualisointiin, määrittelyyn, suunnitteluun ja dokumentointiin tarkoitettu mallinnuskieli, joka

hyväksyttiin OMG:n standardiksi vuonna 1997 (OMG Unified Modeling Language Specification 2001). UML-kieltä käytetään malliperustaisessa arkkitehtuurissa sovellusalustariippumattomien ja sovellusalustariippuvaisten mallien muodostamiseen.

Viidessä vuodessa UML on saavuttanut laajan suosion järjestelmien kehittäjien keskuudessa yleiskäyttöisenä mallinnuskielenä (Kobryn 2000, 31-33). UML-kieli koostuu yhdeksästä mallista, joita käyttämällä on mahdollista mallintaa järjestelmien staattinen rakenne ja dynaaminen käyttäytyminen. Järjestelmien staattisen rakenteen mallintamiseen UML-kielessä voidaan käyttää Boochin ym. (1999a, 94) mukaan luokkakaaviota (class diagram), oliokaaviota (object diagram), komponenttikaaviota (component diagram) sekä sijoituskaaviota (deployment diagram). Järjestelmien dynaaminen käyttäytyminen voidaan mallintaa käyttäen käyttötapauskaaviota (use case diagram), sekvenssikaaviota (sequence diagram), yhteistoimintakaaviota (collaboration diagram), tilakaaviota (statechart diagram) ja toimintakaaviota (activity diagram).

UML:n viimeisin virallinen versio on 1.4 ja vuoden 2002 aikana julkaistaan versio 2.0, jonka Kobrynin (2002, 107) mukaan pitäisi korjata edellisissä versioissa ilmenneet ongelmat, joita ovat: laajuus, monimutkaisuus, epätarkka semantiikka, huonot mahdollisuudet järjestelmien toiminnallisuuden kuvaamiseen, laajennettavuuden rajoittuneisuus, huono tuki komponenttipohjaiselle kehittämiselle sekä mallien heikko siirrettävyys eri CASE-välineiden välillä.

Näistä ongelmista malliperustaisen arkkitehtuurin kannalta oleellisimpia ovat erityisesti mallien epätarkka semantiikka, huonot mahdollisuudet järjestelmien toiminnallisuuden kuvaamiseen, laajennettavuuden rajoittuneisuus sekä huono tuki komponenttipohjaiselle kehittämiselle.

Mallien epätarkka semantiikka vaikuttaa mallien transformaatioon sekä ohjelmakoodin generointiin niiden laatua heikentävästi. Järjestelmien toiminnallisuuden kuvaamisessa on ollut UML-kielen edellisten versioiden aikana semanttisia ongelmia. Toiminnallisuutta ja sen rajoituksia on UML-kielessä pystynyt kuvaamaan eri mallien sekä Object Constraint Language -kielen (OCL) avulla. Järjestelmien toiminnallisuuden

mallintaminen niiden avulla ei kuitenkaan ole ollut yhdenmukaista. (Kath 2002, 22) Ongelmaan on Kathin (2002, 22) mukaan olemassa kaksi ratkaisua: UML-profiilit sekä UML Action Semantics -määritys ja formaalimmin määritetty OCL-kieli. UML-profiilit rajoittavat mahdollisuuksia järjestelmän toiminnallisuuden mallintamisen yhteydessä. UML Action Semantics -määritys (mukana versiossa 1.4) mahdollistaa järjestelmien toiminnallisuuden kuvaamisen aikaisempaa paremmin. Määrityksen avulla on mahdollista määritellä formaalisti UML-mallien semantiikka. UML-kielen määrityksen version 2.0 yhteydessä julkaistaan myös OCL-kielen määritys versio 2.0, jonka tarkoituksena on määrittää OCL-kieli formaalisti ja lisätä sen ilmaisuvoimaa, sekä muodostaa metamalli OCL-kielelle (UML 2.0 OCL RFP 2001). Kathin (2002, 23) mukaan nykyinen jo lähes valmis OCL-kielen määritys ei kuitenkaan merkittävästi paranna OCL-kieltä.

Laajennettavuuden rajoittuneisuus ja huono tuki komponenttipohjaiselle kehittämiselle ratkeavat ainakin osittain UML-profiilien ansiosta. Mallien siirrettävyys on mahdollista eri CASE-välineiden välillä XMI-kielen avulla, mikäli CASE-välineet sitä tukevat. UML-kielen laajuus, monimutkaisuus ja sen omaksumisen vaikeus vaikuttavat myös malliperustaiseen arkkitehtuuriin ja sen laaja-alaiseen hyväksymiseen järjestelmien kehittäjien keskuudessa.

UML-profiili on erikoistettu (specialize) UML-metamalli tiettyyn mallintamistarkoitukseen. UML-profiili voi tukea uusia symboleita ja malleja, joiden avulla voidaan laajentaa ja erikoistaa UML:n perus symboli- ja mallijoukkoa. UML-profiili määrittää notaation tietylle sovellusalustalle, jonka mukaisesti malliperustaisessa arkkitehtuurissa tapahtuu mallien transformaatio sovellusalustariippumattomasta mallista sovellusalustariippuvaiseksi malliksi. (Kobryn 2001, 35) Sovelluskehittäjien kannalta oleellista on, että UML-profiileja määritetään mahdollisimman nopeasti suosituimmille sovellusalustoille ja myös, että CASE-välineiden valmistajat toteuttavat UML-profiilit välineissään ja näin tukevat malliperustaista arkkitehtuuria. Tällä hetkellä määritettynä ovat seuraavat UML-profiilit (TAULUKKO 1).

TAULUKKO 1 UML-profiilit.

Profiili	Versio	Kuvaus profiilista
UML Profile for CORBA	1.0	Profiili määrittää UML-notaation CORBA-sovellusalustan mukaisten järjestelmien kehittämiseen (UML™ Profile for CORBA™ Specification 2002).
UML Profile for Enterprise Application Integration (EAI)	–	Dokumentti on hyväksytty ja sitä viimeistellään (OMG). Profiili määrittää metadatastandardin yritystason järjestelmien rajapintojen integrointiin (UML™ Profile and Interchange Models for Enterprise Application Integration (EAI) Specification 2002).
UML Profile for Enterprise Distributed Object Computing (EDOC)	–	Dokumentti on hyväksytty ja sitä viimeistellään (OMG). Profiili määrittää UML-notaation hajautettujen komponenttipohjaisten sovellusten kehittämiseen (UML Profile for Enterprise Distributed Object Computing Specification 2002).
UML Profile for Schedulability, Performance and Time	–	Dokumentti on hyväksytty ja sitä viimeistellään (OMG). Profiili määrittää UML-notaation reaaliaikajärjestelmien ajan, aikataulutuksen ja suorituskyvyn mallintamiseen (UML Profile for Schedulability, Performance and Time 1999).
UML Profile for EJB	1.0	Dokumentista on julkaistu ensimmäinen julkinen versio (Java Community Process). Profiili määrittää UML-notaation EJB-komponenttimallin mukaisten järjestelmien kehittämiseen (UML Profile for EJB 2001).

Malliperustaisen arkkitehtuurin soveltaminen järjestelmäkehityksessä laaja-alaisesti on järkevää vasta, kun UML-profiilit ovat selkeästi määritetyt, ja kun CASE-välineet tukevat profiileja siten, että kaikki arkkitehtuurin määrittämät transformaatiotyypit ovat mahdollisia mukaan lukien mallien muodostaminen suoraan ohjelmakoodista (reverse engineering).

2.6.2 Meta-Object Facility

Meta-Object Facility (MOF) on viitekehys, joka on tarkoitettu teknologiariippumattomien metamallien[4] määrittelemiseen, kehittämiseen ja hallintaan.

[4] Metamalli on malli, joka kuvaa mallien syntaksin ja semantiikan (Koch ym. 2002, 13). Esimerkiksi UML on metamalli, joka kuvaa UML-mallien syntaksin ja semantiikan.

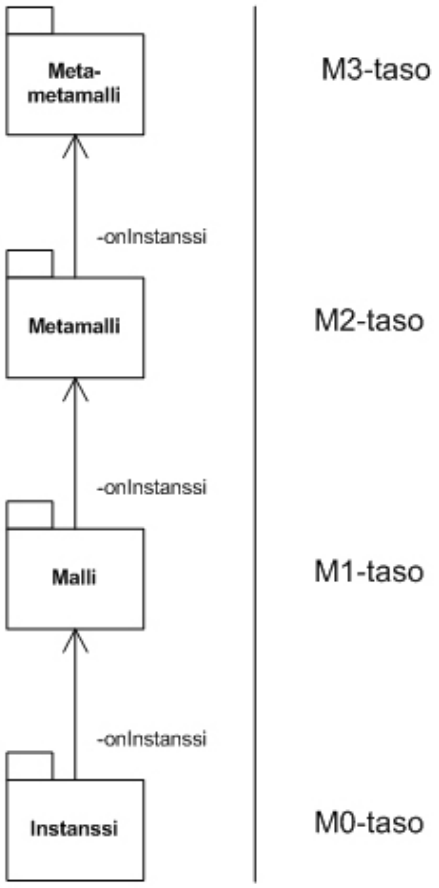
MOF määrittelee myös viitekehyksen metadataa[5] sisältävien tietovarastojen (repository) toteuttamiseen. (Meta Object Facility (MOF) Specification 2002, 15) MOF:n viimeisin virallinen versio on 1.4. Vuoden 2002 puolella OMG julkaisee standardista version 2.0.

Tiedon jakaminen (data interchange) on mahdollista eri järjestelmien välillä, jos järjestelmillä on käytössään toisistaan metadataa (tietoa esimerkiksi tietotyypeistä ja toiminnallisuudesta) ja ne osaavat sitä oikein tulkita ja käyttää. Suurin osa järjestelmistä on kuitenkin toteutettu tällä hetkellä siten, että ne eivät osaa käsitellä ja tulkita toistensa käyttämää metadatamallia. Järjestelmät, jotka käyttävät tiedon määrittelemiseen erilaista syntaksia, semantiikkaa ja rakennetta ovat vaikeita integroida (Java™ Metadata Interface (JMI) Specification 2002, 1-2). MOF on viitekehys, joka määrittelee standardin tavan varastoida metadataa järjestelmistä ja se helpottaa siten myös järjestelmien integrointia.

MOF:n arkkitehtuuri perustuu Crawleyn (2000, 234) mukaan metamallintamisen perinteiselle nelitasoiselle arkkitehtuurille kuvion 6 mukaisesti. Kuviossa M3-tason instanssi määrittelee metamallien rakenteen ja elementit. MOF on tason M3 meta-metamalli. M2-tason instanssit muodostuvat M3-tason määrittelemistä metamalleista. M2-tason instanssi määrittelee metamallien rakenteen ja elementit. UML on esimerkki M2-tason metamallista. M1-tason instanssi määrittelee järjestelmän rakenteen ja elementit M2-tason instanssin määrittelemien käsitteiden mukaisesti. M0-tason instanssi muodostuu järjestelmän ajonaikaisista tietorakenteista tai olioinstansseista. M0-tason instanssit ovat tason M1-mukaisia. MOF:n arkkitehtuurin perusteella voidaan sanoa, että se määrittelee kaikki malliperustaisen arkkitehtuurin mukaiset metamallit ja mallit. MOF-määrityksen mukaisia ovat siten myös UML, CWM (ks. kappale 2.6.3) ja XMI (ks. kappale 2.6.4).

---

[5] Metadata on Changin, Mellorin, Poolen ja Tolbertin (2002, 1) mukaan tietoa tiedosta. Tässä yhteydessä metadata on esimerkiksi tietoa sovelluksien rajapinnoista, toiminnallisuudesta ja tietorakenteista.



KUVIO 6 Meta-Object Facility -viitekehyksen arkkitehtuuri.

MOF määrittelee CORBA-sovelluksille rajapinnan, jonka kautta ne pääsevät ajonaikana käsiksi metadataan. Java™ Metadata Interface Specification -määritys (JMI), joka perustuu MOF-määritykseen, määrittelee puolestaan Java-ohjelmointikielipohjaisille sovelluksille tavan luoda, käsitellä ja hallinnoida metadataa (Java™ Metadata Interface Specification 2002, 1-2).

Malliperustaisen arkkitehtuurin kannalta oleellisin MOF:n käyttökohde on eri ohjelmistotuotannossa käytettävien välineiden integroinnin mahdollistaminen. Hansenin, Heide Dammin, Thomsenin ja Tyrstedin (2000, 94) sekä Keienburgin ja Rauschin (2001, 3945) mukaan on mahdotonta, että koko ohjelmistotuotantoprosessi pystyttäisiin suorittamaan yhdellä ainoalla välineellä. Tämän vuoksi heidän mukaansa on välttämätöntä, että pystytään integroimaan prosessin aikana tarvittavat eri välineet.

Järjestelmien kehittäjät käyttävät siis monia eri välineitä eri tehtäviin ohjelmistontuotantoprosessin aikana. Esimerkiksi kehitettäessä järjestelmää J2EE/EJB-sovellusalustalle saatetaan käyttää UML-kieltä tukevaa CASE-välinettä mallintamiseen, integroitua kehitysympäristöä (integrated development environment) ohjelmakoodin tuottamiseen sekä komponenttienhallintaan ja sijoittamiseen (deploy) liittyviä välineitä sovelluksen hallintaan. JMI-määrityksen avulla voitaisiin muodostaa kolme metamallia, jotka esittävät jokaisen tehtäväkentän kohdealueita (UML-metamalli, Java-metamalli ja J2EE/EJB-metamalli). Tämän jälkeen ohjelmistotuotannossa käytettävät välineet voitaisiin integroida adapterin avulla, joka muuttaisi metamallien mukaisesti välinekohtaiset rajapinnat JMI-rajapintojen mukaisiksi. Palvelut, jotka vaatisivat useiden eri välineiden käyttöä, kuten esimerkiksi ohjelmakoodin ja mallien pitäminen yhdenmukaisina toteutettaisiin käyttäen JMI-määrityksen mukaisia rajapintoja (vrt. KUVIO 7). (Java™ Metadata Interface (JMI) Specification 2002, 7-8)

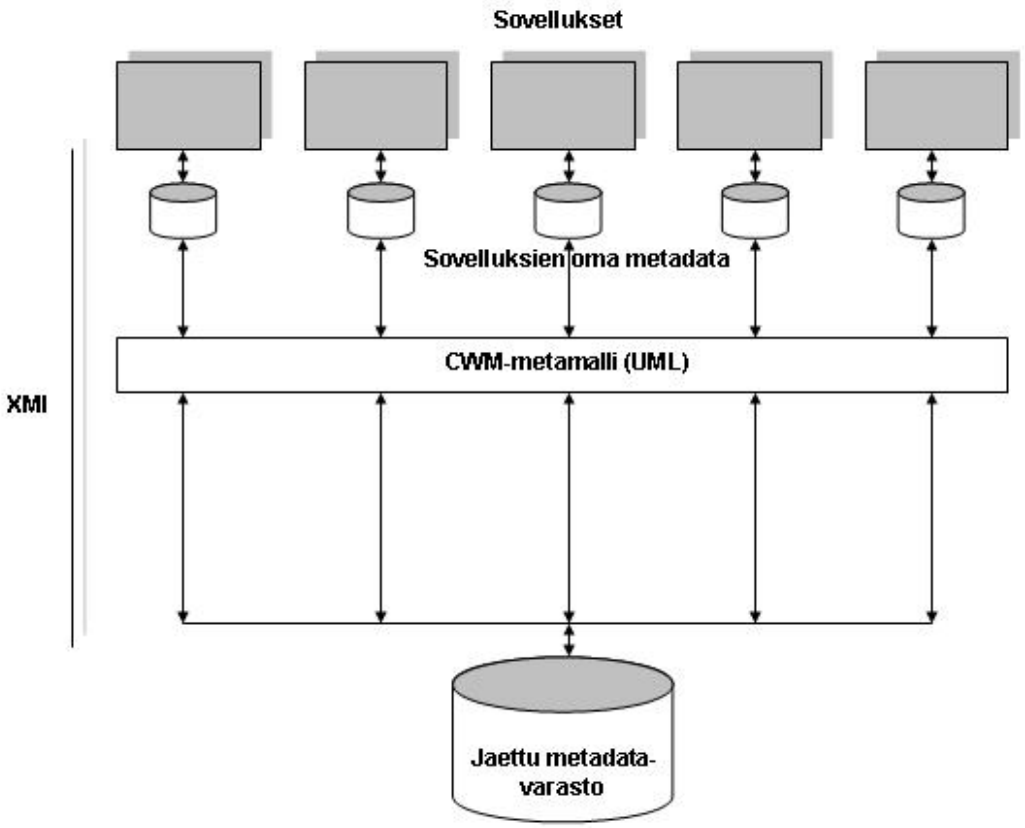
2.6.3 Common Warehouse Metamodel

Common Warehouse Metamodel on tietovarastojen (data warehouse) integrointiin tarkoitettu metadatastandardi, joka määrittelee syntaksin ja semantiikan, joka tarvitaan jaetun metadatan tuomiseen (import) tietovarastoon ja viemiseen (export) tietovarastosta (Common Warehouse Metamodel (CWM) Specification 2001, 20). CWM-standardin virallinen versio on tällä hetkellä 1.0.

CWM:n perustana toimivat UML, MOF ja XMI. UML-kieltä käytetään jaetun metadatan mallien muodostamiseen (CWM-metamalli). XMI-kieltä puolestaan käytetään jaetun metadatan tiedonsiirtoformaattina. MOF määrittelee malliperustaisen arkkitehtuurin metamallit, joten se määrittelee myös CWM:n. MOF ja JMI määrittelevät lisäksi ohjelmallisen rajapinnan, jonka kautta eri sovellukset voivat varastoida ja luoda jaettua metadataa (Chang, Mellor, Poole & Tolbert 2002, 47)

Kuviossa 7 on esimerkki CWM-metadatastandardin käytöstä (Chang ym. 2002, 30-32). Kuvion metadata-arkkitehtuuriesimerkissä erilaiset sovellukset, kuten esimerkiksi CASE-välineet ja integroidut kehitysympäristöt, joiden metadata perustuu omaan

metamalliin luovat, hallinnoivat ja käyttävät jaettua metadataa, joka on varastoitu jaettuun metadatavarastoon. CWM-metamalli toimii sovelluksien ja jaetun metadatavaraston välissä adapterina, jonka tehtävänä on muuntaa sovelluksien käyttämä metadata jaetun metadatavaraston kanssa yhdenmukaiseksi. Sovelluksien metadatamallit mallinnetaan UML-kielellä ja metadatan tiedonsiirtoformaattina käytetään XMI-kieltä.



KUVIO 7 Esimerkki metadata-arkkitehtuurista.

2.6.4 XML Metadata Interchange

XML Metadata Interchange on ohjelmistotuotantoprosessissa käytettävien eri mallinnusvälineiden ja metadatavarastojen väliseen metadatan välittämiseen tarkoitettu standardi (OMG XML Metadata Interchange (XMI) Specification 2002, 9). XMI-standardin viimeisin virallinen versio on 1.2. Vuoden 2002 aikana standardista kuitenkin julkaistaan versio 2.0.

Olioiden esittämiseen ja tiedonsiirtoon sovellusten välillä käytetään paljon eXtensible Markup Language[6]-kieltä (XML). XML ei kuitenkaan ole oliopohjainen kieli eikä siten tarjoa standardia tapaa esittää olioita. Jotta eri olio-ohjelmointikielillä toteutettujen sovellusten ja järjestelmien integrointi olisi mahdollista, on oltava standardi tapa kuvata olioita. UML mahdollistaa olioiden kuvaamisen teknologiariippumattomasti. XMI määritteleekin standardin tavan muuntaa UML-mallit XML-formaattiin ja mahdollistaa myös XML-skeemojen (xml schema, versio 2.0) ja XML-dokumenttimääritysten (document type definition) muodostamisen suoraan malleista. (Brodsky ym. 2002, 3-9)

Malliperustaisessa arkkitehtuurissa XMI mahdollistaa järjestelmien integroinnin ja mallien siirtämisen eri CASE-välineisiin (vrt. KUVIO 7). Malliperustaisen arkkitehtuurin mukaisia järjestelmiä mallinnettaessa, on käytettävien CASE-välineiden tuettava XMI-kieltä.

2.7 Malliperustaisen arkkitehtuurin mukainen ohjelmistotuotantoprosessi

Malliperustaisesta arkkitehtuurin määritys ei kuvaa arkkitehtuurin, ohjelmistotuotantoprosessin vaiheiden, ja ohjelmistotuotannossa yleisesti käytettyjen menetelmien[7] suhdetta lainkaan. Myöskään arkkitehtuurista kirjoitetuissa artikkeleissa ei oteta kantaa sen suhteeseen eri vaihejakomalleihin ja ohjelmistotuotantomenetelmiin. Malliperustainen arkkitehtuuri määrittelee mallien arkkitehtuurin, mutta ei ota kantaa ohjelmistotuotantoprosessin vaiheistukseen. Ohjelmistotuotantoprosessin monimutkaisuuden vuoksi on kuitenkin välttämätöntä jakaa prosessi eri vaiheisiin. Tässä kappaleessa havainnollistetaan malliperustaisen arkkitehtuurin suhdetta ohjelmistotuotannossa yleisesti käytettyihin menetelmiin ja vaihejakomalleihin.

---

[6] eXtensibe Markup Language on World Wide Web Consortium -standardointijärjestön (www.w3c.org) määrittelemä strukturoitu tapa esittää tietoa (Brodsky, Doney & Grose 2002, 18).

[7] Menetelmällä tarkoitetaan tässä yhteydessä järjestelmien kehittämistä ohjaavaa ja sitä vaiheistavaa viitekehystä. Menetelmiä ovat esimerkiksi Unified Software Development Process ja OMT++.

Ohjelmistotuotantoprosessi on Boochin ym. (1999b, 4) mukaan toimenpiteiden joukko, joka tarvitaan käyttäjien vaatimusten (user requirements) muuttamiseksi toimivaksi järjestelmäksi. Ohjelmistotuotantoprosessi on perinteisesti pyritty jakamaan eri vaiheisiin sen monimutkaisuuden vuoksi. Royce (1970, 118-127) esitti ensimmäisenä vesiputousmallin, jonka avulla pyrittiin hallitsemaan suurten ja monimutkaisten järjestelmien kehittämistä jakamalla prosessi eri vaiheisiin. Kolmen vuosikymmenen aikana vaihejakomalleja on esitetty paljon muitakin. Tässä tutkielmassa niitä ei kuitenkaan käsitellä tämän tarkemmin. Yleisesti tunnustetut ohjelmistotuotantoprosessin vaiheet ovat kuitenkin esitutkimus (vaatimusmäärittely), määrittely, suunnittelu, toteutus, integrointi ja testaus (Schach 1990, 46).

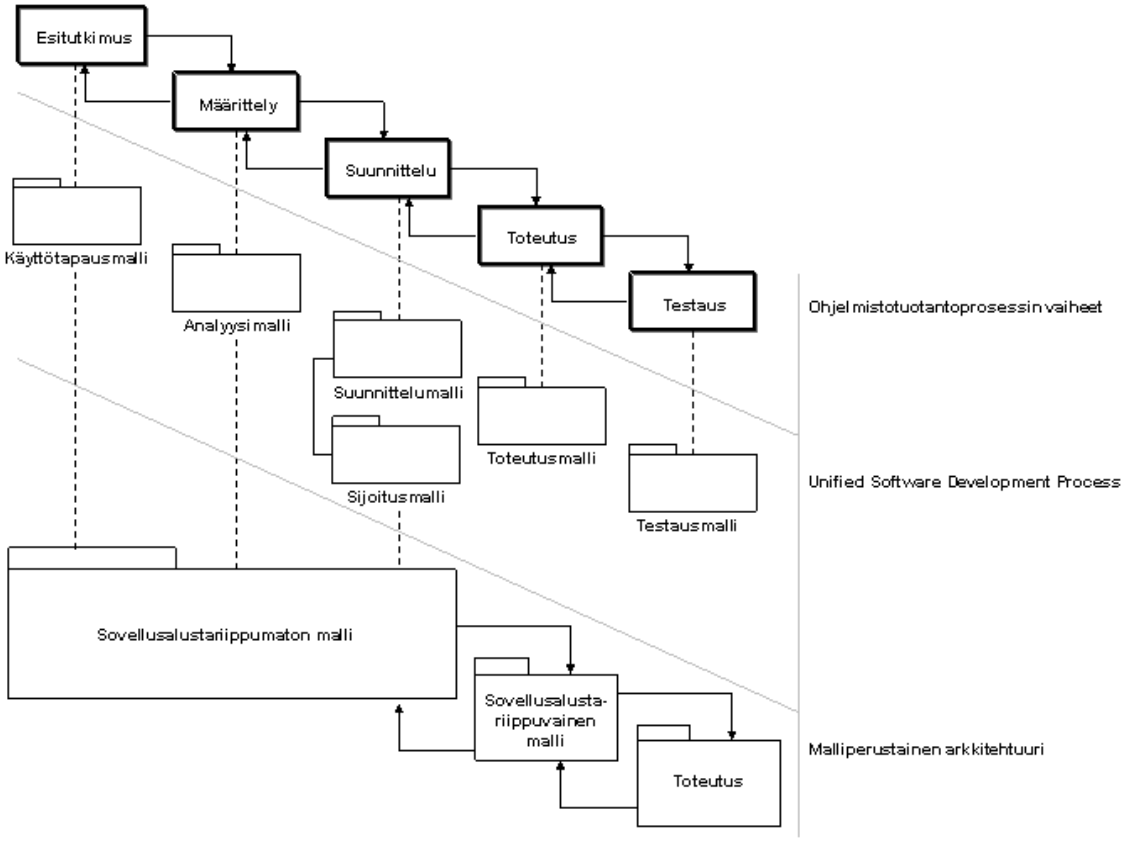
Booch ym. (1999b, 4-5) ovat kehittäneet ohjelmistotuotantoprosessin hallintaan menetelmän, joka perustuu OMG:n UML-kieleen ja on oliokeskeinen sekä iteratiivinen. Unified Software Development Process on komponenttipohjaisten järjestelmien määrittelyyn ja suunnitteluun tarkoitettu menetelmä, joka jakaa ohjelmistotuotantoprosessin eri malleiksi. Malleja ovat käyttötapausmalli, analyysimalli, suunnittelumalli, sijoitusmalli, toteutusmalli ja testausmalli. Vesiputousmallin mukaiseen ohjelmistotuotantoprosessin vaihejakoon mallit sijoittuvat kuvion 8 mukaisesti. On kuitenkin muistettava, että Unified Software Development Process on sekä iteratiivinen että inkrementaalinen prosessi, joten suoraan sitä ei voida verrata esimerkiksi vesiputousmalliin.

Siegelin (2001, 4-9) mukaan malliperustaisen arkkitehtuurin mukainen ohjelmistotuotantoprosessi on yksinkertainen. Ensimmäiseksi kehitetään sovellusalustariippumaton malli, josta mallien transformaation avulla muodostetaan sovellusalustariippumattomia malleja yksi tai useampia. Viimeisenä vaiheena prosessissa on hänen mukaansa sovelluksien automaattinen generointi sovellusalustariippuvaisesta mallista arkkitehtuuria tukevan CASE-välineen avulla. Prosessi kuulostaa yksinkertaiselta, vaikka se todellisuudessa on erittäin monimutkainen. Malliperustainen arkkitehtuuri ei tarjoa minkäänlaista vaiheistusta ohjelmistotuotantoprosessille. Tämän vuoksi ohjelmistotuotantoprosessin vaiheistamiseen tarvitaan perinteisiä menetelmiä. Tällainen menetelmä on esimerkiksi

Unified Software Development Process. Siegelin (2001) määrittämässä prosessissa on täten ongelma: se ei ota mitään kantaa menetelmien ja malliperustaisen arkkitehtuurin suhteeseen.

Malliperustaisen arkkitehtuurin määrityksessäkään (Model Driven Architecture (MDA) 2001) ei viitata lainkaan ohjelmistotuotannossa käytettyjen menetelmien ja sovellusalustariippumattoman sekä sovellusalustariippuvaisen mallin suhteeseen. Arkkitehtuurin määritys ei vielä näiltä osiltaan olekaan valmis. Kehitettäessä järjestelmiä perinteisten ohjelmistotuotantoprosessia tukevien menetelmien mukaisesti (esimerkiksi Unified Software Development Process) määrittelyvaiheessa ei ole otettu kantaa mahdolliseen toteutusteknologiaan, vaan vasta suunnitteluvaiheessa on sitouduttu käyttämään jotakin tiettyä teknologiaa (Booch ym. 1999b, 219). Samalla myös kaikki suunnitteluvaiheen aikana kehitetyt mallit on sitoutettu valitun teknologian mukaisiksi. Malliperustaisen arkkitehtuurin perusajatuksena on mallintaa järjestelmä kokonaan UML-kielellä teknologiariippumattomasti. Tämän vuoksi on perusteltua, että kehitettäessä sovellusalustariippumatonta mallia, jonkin menetelmän mukaisesti, myös suunnitteluvaiheen aikana kehitetyt mallit ovat teknologiariippumattomia. Lisäksi suunnitteluvaiheen teknologiariippumattomuutta puoltaa se tosiasia, että järjestelmästä ja sen toiminnasta on oltava määrittelyvaihetta pidemmälle viedyt suunnitelmat, jotta sovellusalustariippuvaisen mallin transformaatio ja ohjelmakoodin generoiminen olisi ohjelmallisesti mahdollista. Siirryttäessä malliperustaisen arkkitehtuurin mukaiseen ohjelmistotuotantoprosessiin muuttuu siten myös perinteisten ohjelmistotuotannossa käytettyjen menetelmien näkökulma teknologiariippumattomuuteen.

Kuviossa 8 esitetään viitekehys, jonka avulla havainnollistetaan malliperustaisen arkkitehtuurin suhdetta perinteisiin ohjelmistotuotantoprosessin vaiheisiin sekä Boochin ym. (1999b) kehittämään Unified Software Development Process -menetelmään.



KUVIO 8 Malliperustaisen arkkitehtuurin mukainen ohjelmistotuotantoprosessi.

Järjestelmien kehittäminen ilman ohjelmistotuotantoprosessia tukevia CASE-välineitä ei ole järkevää (Booch ym. 1999, 28). Tämän vuoksi oleellinen osa malliperustaista arkkitehtuuria ja etenkin sen käyttökelpoisuutta ovat sitä tukevat CASE-välineet ja niiden toimivuus. Malliperustaista arkkitehtuuria tukevien CASE-välineiden pitää mahdollistaa jonkin ohjelmistotuotannossa yleisesti käytetyn menetelmän käyttäminen, UML-mallien transformaatio profiilien mukaisesti, ohjelmakoodin generointi UML-malleista sekä UML-mallien ja ohjelmakoodin pitäminen yhtenäisinä (round trip engineering). CASE-välineisiin ja niiden toteutukseen liittyvätkin malliperustaisen arkkitehtuurin suurimmat ongelmat ja haasteet.

2.8 Malliperustaisen arkkitehtuurin haasteet ja mahdolliset edut

Seuraavissa kappaleissa käsitellään malliperustaiseen arkkitehtuuriin liittyviä ongelmia ja mahdollisia sen avulla ohjelmistotuotannossa saavutettavia etuja.

2.8.1 Haasteet

Malliperustaisen arkkitehtuurin haasteet ja mahdolliset ongelmat voidaan jakaa kolmeen eri luokkaan:

1) CASE-välineisiin,
2) arkkitehtuuriin ja standardeihin sekä
3) eri teknologioihin liittyviin haasteisiin.

Malliperustainen arkkitehtuuri ja sen hyväksyminen laaja-alaisesti järjestelmien kehittäjien keskuudessa riippuu suurimmaksi osaksi sitä tukevien CASE-välineiden olemassaolosta ja hyvyydestä. Jos sovellusmarkkinoilla on tulevaisuudessa useita arkkitehtuuria ja sen standardeja hyväksikäyttäviä ja toteuttavia välineitä, niin malliperustaisesta arkkitehtuurista voi muodostua seuraava suuri kehitysaskel ohjelmistotuotannossa UML-kielen jälkeen. CASE-välineiden toteuttajat kohtaavat kuitenkin monia haasteita ja ongelmia, joista keskeisimpiä ovat:

- UML-profiilien ja toimialamääritysten toteuttaminen niiden suuren määrän vuoksi.
- Mallien pitäminen yhdenmukaisina riippumatta abstraktiotasosta tai näkökulmasta kehitettäessä järjestelmiä.
- Ohjelmakoodin ja mallien pitäminen yhdenmukaisina (round trip engineering).

Arkkitehtuuriin ja standardeihin liittyvät haasteet ja mahdolliset ongelmat ovat sidoksissa suurimmaksi osaksi UML-kieleen, koska se on malliperustaisen arkkitehtuurin ydinstandardi. Arkkitehtuuriin ja standardeihin liittyviä haasteita ovat:

- Arkkitehtuurin monimutkaisuus, laajuus ja standardien suuri määrä. Arkkitehtuurin omaksuminen vaatii suurimmaksi osaksi käsitteellisten ja vaikeasti hahmotettavien asioiden opettelua.

- UML-kielestä tulee version 2.0 myötä entistä laajempi ja vaikeammin hahmotettava kokonaisuus. UML-kielen notaatioiden määrää lisäävät toimialamääritykset ja UML-profiilit eri toteutusteknologioille.
- UML-kielen semanttinen tarkkuus ei ole ollut edellisissä kielen määrityksissä riittävän korkealla tasolla. Mallien transformaatio ja pitäminen yhdenmukaisina edellyttää semanttisesti tarkkaa ja formaalia kieltä.
- UML-kielen järjestelmien toiminnallisuuden semantiikan mallintamisen puutteiden vuoksi toiminnallisuuteen liittyvän ohjelmakoodin generointi on vaikeaa.
- Malliperustaisen arkkitehtuurin mukaisten mallien tulee olla täsmälleen UML-standardin mukaisia, jotta mallien transformaatio ja ohjelmakoodin generointi olisi mahdollista. Tämä asettaa myös omat vaatimuksensa järjestelmien kehittäjille, joiden on oltava mallintamisen ammattilaisia. UML-mallit eivät enää ole vain apuvälineitä järjestelmien kehittämisessä, vaan ne ovat kiinteä osa järjestelmää.

Teknologisia haasteita malliperustaiselle arkkitehtuurille aiheuttavat muun muassa:

- Sovellusalustojen erilaisuus. Eri komponenttiteknologiat ovat teknologisista lähtökohdistaan hyvin erilaisia. Onko siis lopulta edes mahdollista muodostaa sovellusalustariippumattomasta mallista useita erilaisia sovellusalustariippuvaisia malleja?
- Konfiguraatiotiedostojen ja muiden sovellusten tai komponenttien ajonaikaiseen toimintaan liittyvien tiedostojen muodostaminen on vaikeaa johtuen sovellusmarkkinoilla olevista erilaisista tuotteista. Esimerkiksi EJB-säiliöiden[8] hallinnoimien komponenttien konfiguraatiotiedostot ovat usein palvelinvalmistajakohtaisia.

---

[8] EJB-säiliö on komponenttien ajonaikainen suoritusympäristö, joka huolehtii kaikesta komponentteihin liittyvästä toiminnasta (Monson-Haefel 2001).

2.8.2 Edut

Malliperustaisen arkkitehtuurin mahdollisesti mukanaan tuomat edut ohjelmistotuotantoon ovat riippuvaisia siitä, kuinka sen ongelmat ja haasteet kyetään ratkaisemaan. Soleyn (2000, 7) mukaan malliperustaisen arkkitehtuurin soveltaminen ohjelmistotuotannossa tarjoaa seuraavat edut:

- Järjestelmien kehittäminen eri sovellusalustoille nopeutuu ja siirrettävyys eri sovellusalustojen välillä helpottuu.
- Liiketoimintalogiikka ja teknologia kehittyvät ilman suoranaista sidosta järjestelmään. Ne ovat kumpikin erillisiä järjestelmän osia, joita on helppo kehittää toimintaympäristön muuttuessa.
- Järjestelmäkehityksen kustannukset laskevat ohjelmistokehityksen automatisoinnin vuoksi.
- Järjestelmäkehitykseen liittyvät riskit pienenevät, koska siirtyminen sovellusalustasta toiseen on helpompaa kuin ennen.
- Järjestelmiin sidottu pääoma säilyttää arvonsa paremmin, koska järjestelmään liittyvä logiikka on mallinnettu teknologiariippumattomasti. Teknologiat vaihtuvat muutamien vuosien välein, mutta liiketoimintalogiikka säilyy teknologiasta riippumattomana. Kaikkea ei siis tarvitse aina aloittaa alusta uuden teknologian tullessa markkinoille.

2.9 Yhteenveto

Malliperustainen arkkitehtuuri on koko ohjelmistotuotantoprosessia tukeva ohjelmointikieli-, ohjelmistotoimittaja- ja sovellusalustariippumaton ohjelmistoarkkitehtuuri. Malliperustainen arkkitehtuuri perustuu järjestelmien liiketoimintalogiikan ja kohdealueen mallintamiseen erillään sen toteuttavasta sovellusalustasta. Arkkitehtuurin mukaisesti järjestelmiä kehitetään muodostamalla malleja, jotka kuvaavat järjestelmää eri näkökulmista ja abstraktiotasoista. Malleja on kahdenlaisia: sovellusalustariippumattomia ja sovellusalustariippuvaisia.

Malliperustaisen arkkitehtuurin mukaisessa iteratiivisessa ohjelmistotuotantoprosessissa muodostetaan Unified Modeling Language -mallinnuskielellä (UML) sovellusalustariippumaton malli, josta mallien transformaation avulla muodostetaan sovellusalustariippuvainen malli halutulle teknologiselle ratkaisulle. Viimeiseksi sovellusalustariippuvaisesta mallista generoidaan ohjelmakoodi halutulle sovellusalustalle.

UML-profiilit määrittävät säännöstön, jonka mukaisesti sovellusalustasta riippumaton UML-malli transformoidaan sovellusalustasta riippuvaiseksi UML-malliksi. Mallien transformaatio ja ohjelmakoodin generointi ovat ohjelmallisesti automatisoituja prosesseja, jotka tapahtuvat malliperustaista arkkitehtuuria tukevien CASE-välineiden avulla.

Malliperustaisen arkkitehtuurin määrityksessä ei oteta kantaa arkkitehtuurin, yleisimpien ohjelmistotuotantoprosessin vaihejakomallien ja ohjelmistotuotannossa käytettyjen menetelmien suhteeseen. Tämä on ongelmallista, koska ohjelmistotuotantoprosessi on kokonaisuudessaan varsin monimutkainen, ja vaatii siten jakamista helpommin hallittaviin ja ymmärrettäviin kokonaisuuksiin. Malliperustaisen arkkitehtuurin määrityksen voidaankin todeta olevan näiltä osin vielä keskeneräinen. Tutkielman yhtenä kontribuutiona on kuvattu (ks. kappale 2.7) malliperustaisen arkkitehtuurin suhde perinteiseen vesiputousmalliin ja Unified Software Development Process -menetelmään. Kuvauksesta ilmenee, että ohjelmistotuotantoprosessin vaiheista esitutkimus, määrittely ja suunnittelu ovat osa sovellusalustariippumattoman mallin muodostamista ja siten myös teknologiariippumattomia. Perinteisesti suunnitteluvaiheessa on jo otettu kantaa kehitettävän järjestelmän toteutusteknologiaan, joten malliperustainen arkkitehtuuri tuleekin muuttamaan ohjelmistotuotantoprosessin suhdetta teknologiariippumattomuuteen näiltä osin.

Malliperustainen arkkitehtuuri helpottaa yritystason järjestelmien integrointia (enterprise application integration) ja edesauttaa järjestelmien siirrettävyyttä sovellusalustalta toiselle sekä parantaa järjestelmien kehittämiseen liittyvien investointien tuottoastetta (return on investment). Keskeisin ongelma arkkitehtuurissa

on UML-kielen järjestelmien toiminnallisten ominaisuuksien mallintamiseen liittyvät semanttiset puutteet. UML-kielen määrityksen version 2.0 odotetaan kuitenkin ratkaisevan ainakin osan UML-kieleen liittyvistä ongelmista.

Malliperustaisen arkkitehtuurin laaja-alainen soveltaminen ohjelmistotuotannossa edellyttää sitä tukevia CASE-välineitä. Mikäli malliperustaista arkkitehtuuria aiotaan soveltaa ohjelmistotuotannossa, on ensiarvoisen tärkeää ymmärtää arkkitehtuurin vaatimukset sitä tukeville CASE-välineille. CASE-välineet ovat oleellinen osa malliperustaista arkkitehtuuria, joten myös CASE-välineiden arviointiin ja valintaan on kiinnitettävä erityistä huomiota.

3 CASE-VÄLINEIDEN ARVIOINTI JA VALINTA

Boochin ym. (1999, 28) mukaan järjestelmien kehittäminen ilman ohjelmistotuotantoprosessia tukevia CASE-välineitä ei ole järkevää. CASE-välineiden tarve huomattiin jo 1980-luvun lopussa ja tällä hetkellä CASE-välineitä on sovellusmarkkinoilla useita satoja. Ongelmaksi yrityksille ja organisaatioille muodostuukin CASE-välineiden arviointi ja valinta. CASE-välineiden arviointia ja valintaa on tutkittu paljon. Myös erilaisia menetelmiä on kehitetty CASE-välineiden arviointi- ja valintaprosessin yhdenmukaistamiseksi ja ohjeistukseksi. Tämän luvun tarkoituksena on muodostaa kokonaiskuva kirjallisuudessa esitetyistä CASE-välineiden arviointi- ja valintamenetelmistä, sekä esitellä kolme CASE-välineiden arviointimenetelmää tarkemmin.

Luvun aluksi jäsennetään ohjelmistovälineiden arviointiin ja valintaan liittyviä riskejä ja ongelmia. Ongelmien välttämiseen ja riskien minimointiin esitetään myös kirjallisuudesta löydettyjä ratkaisukeinoja. Seuraavaksi esitellään lyhyesti erilaisia CASE-välineiden ja ohjelmistovälineiden arviointiin sekä valintaan tarkoitettuja menetelmiä. Luvun lopuksi kuvataan tarkemmin kolme CASE-välineiden arviointiin ja valintaan tarkoitettua menetelmää.

CASE-väline (Computer-Aided Software/Systems Engineering) on ohjelmistotuotantoprosessin yksittäisiä osia tai tehtäviä automatisoiva väline (Koskinen 2000, 44). Yleisesti CASE-välineellä tarkoitetaan sellaista sovellusta, joka automatisoi ohjelmistotuotantoprosessia, järjestelmien ylläpitoa tai projektinhallintaa (McClure 1989). *CASE-ympäristö* on CASE-välineiden joukko, jonka avulla pyritään kattamaan koko ohjelmistotuotantoprosessin vaatima välinetuki (Marttiin 1998). CASE-ympäristö muodostuu joukosta CASE-välineitä, jotka kattavat eri osia ohjelmistotuotantoprosessista (Koskinen 2000, 44). CASE-ympäristö voi muodostua esimerkiksi tietovarastosta (repository), ylemmän tason CASE-välineestä (upper CASE tool), alemman tason CASE-välineestä (lower CASE-tool), versionhallintavälineestä, konfiguraationhallintavälineestä ja dokumentointivälineestä.

3.1 CASE-välineiden arviointiin sekä valintaan liittyvät ongelmat ja riskit

CASE-välineiden arviointi- ja valintaprosessiin liittyy monia ongelmia ja riskejä, joihin arvioijat usein törmäävät. Seuraavissa kappaleissa käsitellään CASE-välineiden arviointiin ja valintaan liittyviä ongelmia sekä esitetään ratkaisukeinoja niihin.

3.1.1 Ongelmia ja riskejä

CASE-välineen arviointi ja valinta on monimutkainen prosessi, jonka aikana arvioijan täytyy määritellä arviointikriteerit[9], arvioida (evaluate) eri välineitä kriteerien mukaisesti ja muodostaa arvion perusteella päätös valittavasta CASE-välineestä. Pratherin (1993, 60) mukaan ei ole olemassa parasta CASE-välinettä, vaan ainoastaan hyviä valintoja tiettyihin tilanteisiin. CASE-välineen arviointi- ja valintaprosessin voidaan todeta epäonnistuneen, kun valittu väline ei vastaa sille määriteltyjä vaatimuksia ja tavoitteita (Prather 1993, 60).

Cook, Lam, Powell, Vickers ja Williams (1997, 81) ovat kokemustensa perusteella listanneet seitsemän CASE-välineiden arviointi- ja valintaprosessissa muodostuvaa riskiä ja ongelmaa. Riskejä ja ongelmia ovat heidän mukaansa:

- päätöksenteossa käytettävien kriteereiden monimutkaisuus,
- arviointipäätöksen riskialttius ja epävarmuus,
- päätöksentekoon liittyvät paineet,
- päätöksenteon aikajänteen pituus,
- päätöksentekoprosessi ja vastuut,
- jälkipäätöksenteon likinäköisyys (post-decision myopia) sekä

---

[9] Arviointikriteerit ovat ominaisuuksia, piirteitä tai toiminnallisuuksia joiden avulla CASE-välineitä arvioidaan. CASE-välineessä esimerkiksi on jokin tietty haluttu ominaisuus tai ei.

- kehitysympäristön jatkuva muuttuminen ja kypsymättömyys.

Edellisten lisäksi Pratherin (1993, 60-62) mukaan ongelmia CASE-välineiden arvioinnissa saattavat aiheuttaa myös:

- arviointi- ja valintaprojektin vähätteleminen ja resurssien aliarviointi,
- esivaatimuksien huono tila ja
- suorituskykytestin tekemättä jättäminen.

Päätöksenteossa käytettävät kriteerit ovat monimutkaisia, usein subjektiivisia ja vaikeita saattaa yhteismitallisiksi. CASE-välineiden arviointi monimutkaistuu eksponentiaalisesti välineiden ominaisuuksien lisääntyessä. Arvioijien ongelmana ovat usein myös epämääräiset arviointikriteerit, kuten esimerkiksi laatu, käytettävyys tai ominaisuudet.

Arviointipäätökset ovat riskialttiita ja epävarmoja. Päätöksenteossa tarvitaan tietoa välineen mahdollisesti mukanaan tuomista eduista ja riskeistä. Kriteerien määrittämisen ja vertailun vaikeuden vuoksi kuitenkin epävarmuustekijöiden määrä on suuri. Arviointipäätös voikin yleensä olla vain hyvä arvaus, jolle on perusteluja löydettävissä.

Päätöksentekotilanteeseen liittyvät paineet voivat myös aiheuttaa ongelmia CASE-välineiden arvioinnissa ja valinnassa. CASE-välineen arviointi- ja valintaprosessin voi laukaista sisäinen tai ulkoinen ärsyke. Sisäinen ärsyke voi olla esimerkiksi toimintaympäristön muutos. Ulkoinen ärsyke voi olla esimerkiksi teknologian kehittyminen. Kiireinen aikataulu aiheuttaa päätöksentekotilanteessa paineita ja ongelmia. Päätöksentekoprosessiin liittyy aina myös muutos ja vastuu sen onnistumisesta. Vastuuta voidaan vältellä ja tämän vuoksi jättää arviointi- ja valintaprosessi kokonaan dokumentoimatta.

Jälkipäätöksen likinäköisyydellä tarkoitetaan arviointi- ja valintaprosessin päättymistä hankintapäätökseen. CASE-väline saatetaan hankintapäätöksen jälkeen luovuttaa ilman

koulutusta varsinaisille käyttäjilleen. Arviointi- ja valintaprosessi voi olla onnistunutkin ja teknisesti väline täyttää sille asetetut tavoitteet, mutta välinettä ei osata eikä haluta hyödyntää puutteellisen käyttöönottovaiheen vuoksi.

CASE-välineen toimintaympäristö muuttuu jatkuvasti. Muutoksia aiheuttavat muun muassa teknologiset muutokset, markkinat, välinetoimittajat ja päivitykset. Muutokset aiheuttavat epävarmuutta myös jo CASE-välineen arviointi- ja päätöksentekotilanteessa.

CASE-välineiden arviointi- ja valintaprosessin ei saa vähätellä. Arviointi- ja valintaprosessin tulee olla hyvin määritelty projekti, jolla on käytössään riittävät resurssit. Jokainen CASE-välineen arviointi- ja valintaprosessi on erilainen riippuen organisaatiosta ja käyttöympäristöstä. Ongelmia aiheuttavat lisäksi *määrittelemättömät esivaatimukset*. Organisaatiossa tulee olla jo entuudestaan käytössä ohjelmistotuotantoprosessia tukeva menetelmä ja myös välineen tarkoitus tulee olla määriteltynä. Ennen kuin lopullista valintapäätöstä tehdään, *tulee tehdä suorituskykytesti*, jolla lopullisesti todellisessa käyttöympäristössä todennetaan välineen ominaisuudet ja suorituskyky.

3.1.2 Ratkaisukeinoja ongelmien ja riskien välttämiseksi

Cook ym. (1997) sekä Prather (1993) antavat myös ohjeita, joiden avulla edellä mainitut ongelmat ja riskit voidaan välttää CASE-välineiden arviointi- ja valintaprosessissa.

Cook ym. (1997, 82) ehdottavat seuraavia keinoja, joilla ainakin osa CASE-välineiden arviointiin ja valintaan liittyvistä ongelmista voidaan ratkaista:

- päätöksentekoprosessin dokumentointi ja välinearviointiin sekä valintaan osallistuvien ihmisten sitouttaminen,
- CASE-välineen sovittaminen ympäristöönsä,
- CASE-välineen elinkaaren jatkuva hallinta,

- CASE-välineen kokeileminen todellisessa ympäristössä ennen ostopäätöksen tekemistä sekä
- omien ohjelmistotuotantoon liittyvien prosessien jatkuva kehittäminen.

Arviointi- ja valintaprosessien dokumentointi on tärkeää, koska sen avulla ihmiset sitoutetaan projektiin ja tulevaan valintapäätökseen. Huonoja päätöksiä voidaan vähentää myös lisäämällä arviointipisteitä prosessiin, jolloin päätöksentekoon liittyvät paineet vähenevät. Välineen loppukäyttäjien tulisi myös osallistua arviointi- ja valintaprosessiin. *Sovitettaessa CASE-välinettä käyttöympäristöönsä* tulee myös huolehtia muutoksenhallinnasta, koulutuksesta ja mahdolliset ongelmat pitää käsitellä nopeasti. *CASE-välineen elinkaarta* ja sen käyttökelpoisuutta on jatkuvasti tarkkailtava, jotta tunnistetaan riittävän ajoissa uuden välineen tarve. CASE-välineen elinkaareen voivat vaikuttaa sekä organisaation sisäiset että ulkoiset tekijät. CASE-välineen arviointi- ja valintaprosessin onnistumisen kannalta on ensiarvoisen tärkeää *kokeilla välinettä* ennen hankintapäätöksen tekemistä, koska välineestä voi paljastua selviä puutteita esimerkiksi suorituskykyyn liittyen todellisessa käyttöympäristössä. Joskus tilanne voi olla myös sellainen, että CASE-väline on hyvä, mutta *organisaation omissa ohjelmistotuotantoon liittyvissä prosesseissa on puutteita*. Puutteita voi olla muun muassa ohjelmistotuotannossa käytettävässä menetelmässä, jota välineen pitäisi tukea. Huonoin on tilanne silloin, kun CASE-välinettä ollaan hankkimassa korvaamaan menetelmä. CASE-välineen pitää toteuttaa ja tukea menetelmää, ei määritellä sitä.

Prather (1993, 60-61) painottaa erityisesti ongelmien ratkaisussa seuraavia keinoja:

- arviointikriteereiden tulee olla kaikkien projektiin osallistuvien hyväksymiä ja ennen kaikkea ymmärtämiä sekä
- välinekandidaatit tulee nopeasti rajata muutamaan ja keskittyä niiden arviointiin kunnolla.

CASE-välineiden arviointiin ja valintaan on kehitetty useita erilaisia menetelmiä. Arviointi- ja valintaprosessiin liittyvät ongelmat ja riskit välttääkin parhaiten

käyttämällä omiin tarkoituksiin ja tilanteeseen sopivaa menetelmää. Seuraavassa kappaleessa esitetään useita erilaisia CASE-välineiden ja ohjelmistovälineiden arviointiin ja valintaan kehitettyjä menetelmiä.

3.2 Menetelmiä CASE-välineiden arviointiin ja valintaan

Teknologian kehittyminen mahdollistaa uusien ja toiminnallisilta ominaisuuksiltaan entistä monipuolisempien järjestelmien ja sovellusten kehittämisen. Yritykset ja muut organisaatiot joutuvat tämän vuoksi usein valintatilanteeseen, jossa täytyy vertailla useita markkinoilla olevia samantyyppisiä sovelluksia ja valita niistä omiin tarpeisiin sopivin. Välineiden arviointi- ja valintaprosessi ei kuitenkaan ole yksinkertainen. Välineet saattavat olla ominaisuuksiltaan erittäin monipuolisia ja valintatilanteessa tulee ottaa myös muut tekijät kuin pelkät tekniset ominaisuudet huomioon. Välinearvioinnin ja -valinnan monimutkaisuuden vuoksi on kehitetty erilaisia menetelmiä, jotka avustavat arvioijaa vaiheistamalla prosessin eri vaiheisiin ja määrittämällä arviointikriteeristön, jonka mukaisesti eri välineitä tulee arvioida. Taulukossa 2 kuvatuista menetelmistä tällaisia yleiskäyttöisiä menetelmiä ovat Rowleyn (1993) ja Cookin ym. (1997) kehittämät menetelmät.

Yleiskäyttöisten menetelmien lisäksi on kehitetty erilaisia menetelmiä myös tarkemmin määrätyille sovellusalueille, joista CASE-välineet ovat yksi esimerkki. CASE-välineiden arviointiin ja valintaan on kehitetty useitakin menetelmiä, mutta tässä tutkielmassa keskitytään niistä ainoastaan oleellisimpiin. CASE-välineiden arviointi- ja valintaprosessi on myös standardoitu kansainvälisen standardointijärjestön toimesta (International Organization for Standardization / International Electrotechnical Commission). Taulukossa 2 kuvatuista menetelmistä CASE-välineiden arviointi- ja valintamenetelmiä ovat Kitchenhamin, Lawin ja Lewinin (1997) DESMET-menetelmä, Duk Choin, Griffinin, Mehandjiskan, Pagen ja Phillipsin (1998) menetelmä, Kornin ja LeBlancin (1992) menetelmä, Mosleyn (1992) menetelmä sekä Information Technology - Guideline for Evaluation and Selection of CASE-tools (1995) -menetelmä.

TAULUKKO 2 Menetelmiä ohjelmistovälineiden sekä CASE-välineiden arviointiin ja valintaan.

Menetelmä/Tutkija(t)	Vaihejako	Arviointikriteereiden luokittelu	Kuvaus menetelmästä
Cook, Lam, Powell, Vickers & William (1997)	1) Esiarviointiprosessi 2) Arviointiprosessi 3) Jälkiarviointiprosessi	1) Liiketoimintaan liittyvät asiat 2) Arviointikeskeiset asiat 3) Ulkoisiin sidosryhmiin liittyvät asiat 4) Välinetoimittajan tukeen liittyvät asiat 5) Rahoitukseen liittyvät asiat 6) Välinekohtaiset asiat 7) Käyttöönottoon liittyvät asiat 8) Välineen elinkaareen liittyvät asiat	Ohjelmistovälineiden arviointi- ja valintamenetelmä, joka esittää yksityiskohtaisen arviointikriteeristön ja vaihejaon välinearviointiin sekä valintaan. Menetelmä on helposti siirrettävissä erityyppisten ohjelmistovälineiden valintaan.
DESMET-menetelmä (Kitchenham, Law & Linkman 1997)	1) Arvioitavien välineiden valitseminen 2) Kriteeristöstä päättäminen 3) Kriteeristön priorisointi 4) Tulosten luottamustason päättäminen 5) Arvostelujärjestelmästä päättäminen 6) Arvioinnin suorittaminen 7) Tulosten analysointi 8) Tulosten julkaiseminen	1) Välinetoimittajaan liittyvät kriteerit 2) Välineen kypsyyteen liittyvät kriteerit 3) Taloudelliset kriteerit 4) Kulttuurilliset, sosiaaliset ja tekniset kriteerit 5) Välineen sopivuus sovellusalueelle 6) Arvioitavan välineen luotettavuus	Ohjelmistotuotantoprosessia tukevien menetelmien ja välineiden arviointiin käytettävä menetelmä. Ei ole yksityiskohtaisen tarkka, vaan jättää paljon arvioijan oman harkintakyvyn varaan.
Duk Choi, Griffin, Mehandjiska, Page & Phillips (1998)	Ei vaihejakoa	1) Opittavuus 2) Tehokkuus 3) Joustavuus 4) Kestävyys 5) Palaute	Menetelmä on tarkoitettu oliopohjaisten CASE-välineiden käytettävyyden arviointiin. Menetelmä ei anna ohjeistusta arviointi- ja valintaprosessin läpiviemiseksi, vaan ainoastaan määrittää kriteeriluokituksen välineen käytettävyyden arviointiin.

(jatkuu)

TAULUKKO 2 (jatkuu)

Menetelmä/Tutkija(t)	Vaihejako	Arviointikriteereiden luokittelu	Kuvaus menetelmästä
Information Technology - Guideline for Evaluation and Selection of CASE-tools (1995)	1) Alustusprosessi 2) Rakenteistamisprosessi 3) Arviointiprosessi 4) Valintaprosessi	1) Ohjelmistotuotantoprosessin elinkaareen liittyvät kriteerit 2) CASE-välineen käyttöön liittyvät kriteerit 3) Yleiset laatukriteerit 4) Yleiset kriteerit	ISO/IEC 14102 on CASE-välineiden arviointiin ja valintaan tarkoitettu kansainvälinen standardi, joka määrittelee vaihejaon prosessille sekä myös tarkan kriteeristön.
Korn & LeBlanc (1992)	1) Mahdollisten ehdokkaiden valinta ja listan muodostaminen välineistä 2) Järjestelmäkehityksen vaatimukset parhaiten täyttävän CASE-välineen valinta 3) Käyttäjien vaatimusten ja valitun CASE-välineen ominaisuuksien vastaavuuden varmistaminen ja kuvaus siitä, kuinka vaatimukset täytetään	1) Teknisiin vaatimuksiin liittyvät kriteerit 2) Toiminnallisiin vaatimuksiin liittyvät kriteerit 3) Dokumentaatioon ja koulutukseen liittyvät kriteerit 4) Toimittajaan liittyvät kriteerit	Menetelmä määrittää CASE-välineiden arviointiin ja valintaan vaihejaon sekä luokkajaon arviointikriteeristölle. Arviointikriteeristön kuvaus ei ole tarkka, vaan vain pääkohdat mainitaan.
Mosley (1992)	1) CASE-välineiden luokittelu 2) Lyhyt arviointi välineistä 3) Määrällinen välineiden arviointi 4) Räätälöidyn yhteenvedon tekeminen 5) Konsultaation tekeminen välinearvioinnista	Kriteerit jaetaan kahteen luokkaan: 1) Yleisiin kriteereihin ja 2) Sovelluskohtaisiin kriteereihin	Menetelmä määrittää vaihejaon CASE-välineiden arviointi- ja valintaprosessille ja kriteereiden luokituksen.
Rowley (1993)	1) Kohteiden määrittely 2) Vaihtoehtojen arviointi 3) Määrittely 4) Valinta ja suunnittelu 5) Toteutus 6) Arviointi ja ylläpito	Kriteerit on jaettu kahteen luokkaan: 1) kaikkiin ohjelmistoihin sopiviksi ja 2) tietynlaisiin ohjelmistoihin sopiviksi	Yleiskäyttöinen ohjelmistovälineiden arviointi- ja valintamenetelmä, joka on käytännönläheinen, mutta suppea.

Kaikki taulukossa 2 kuvatuista menetelmistä - lukuun ottamatta Duk Choin ym. (1998) menetelmää - määrittävät sekä vaihejaon että kriteeristön välineiden arviointiin ja valintaan. Menetelmistä löytyy paljon yhteisiä piirteitä sekä vaihejaon että kriteeristön osalta. Yhteisiä vaihejakoon liittyviä piirteitä ovat muun muassa alustavien ehdokasvälineiden valinta, arviointiprosessi sekä itse välineen valinta. Kriteeristön osalta yhdenmukaisuutta löytyy muun muassa teknisten ja toiminnallisten ominaisuuksien määrittämisestä, välinetoimittajan tuesta, toimittajasta, käytettävyydestä sekä kriteereiden jaosta yleisiin ja välinekohtaisiin kriteereihin.

Seuraavissa kolmessa kappaleessa esitellään ja kuvataan tämän tutkielman kannalta oleellisimmat menetelmät tarkemmin. Menetelmät on valittu niiden laajuuden ja kattavuuden mukaan. Jokainen kuvattavista menetelmistä määrittää CASE-välineiden arviointi- ja valintaprosessille selkeän vaihejaon sekä yksityiskohtaiset arviointikriteerit, joiden avulla CASE-välineiden arviointi ja valinta voidaan suorittaa. Arviointikriteerit kuvataan seuraavassa vain oleellisimmilta osiltaan (pääkriteerit ja kriteereiden luokitus), koska ne ovat menetelmissä hyvin samansuuntaisia ja niitä on myös useita kymmeniä.

3.3 Cookin, Lamin, Powellin, Vickersin ja Williamsin menetelmä

Cook ym. (1997) kehittämä viitekehys on ohjelmistovälineiden arviointiin ja valintaan tarkoitettu yleispätevä menetelmä, joka määrittelee arviointi- ja valintaprosessin sekä arviointikriteeristön arvioinnin suorittamiseksi. Menetelmä pohjautuu kahteen osakokonaisuuteen: hyvin määriteltyyn prosessiin ja ominaisuuksien tarkistuslistaan (issue checklist).

Arviointi- ja valintaprosessin tarkoituksena on saavuttaa riittävä ymmärtämys ongelma-alueesta, johon välinettä ollaan valitsemassa, sekä oppia ymmärtämään valinnan kohteena olevia välineitä. Arviointi- ja valintaprosessissa on kiinnitettävä erityistä huomiota prosessin hyvään hallintaan ja sen dokumentointiin.

Ominaisuuksien tarkistuslista jakautuu kahteen osaan: yleisiin arviointiasioihin sekä välinekohtaisiin arviointiasioihin. Riippuen arvioinnin ja valinnan kohteena olevasta välineestä, ominaisuuksien tarkistuslistaan lisätään välinekohtaisia arviointikriteerejä. Välinekohtaiset arviointikriteerit voivat olla esimerkiksi CASE-välineiden tapauksessa, jonkin teknologian tai arkkitehtuurin asettamia vaatimuksia välineelle (esimerkiksi malliperustaisen arkkitehtuurin tai UML-kielen). Ominaisuuksien tarkistuslistan avulla on mahdollista kiinnittää huomiota enemmän ei-teknisiin arviointikriteereihin sekä määrittää arviointikriteerit tarkemmin.

3.3.1 Vaihejako

Arviointi- ja valintaprosessi jakautuu kolmeen päävaiheeseen, jotka ovat:

1) esiarviointiprosessi (pre-evaluation process)
2) arviointiprosessi (evaluation process) ja
3) jälkiarviointiprosessi (post-evaluation process).

Esiarviointiprosessi

Esiarviointiprosessi muodostuu seitsemästä tehtävästä (activity), joiden laajuus ja tarkkuus ovat arviointitilannekohtaisia. Tehtävät esiarviointiprosessissa ovat:

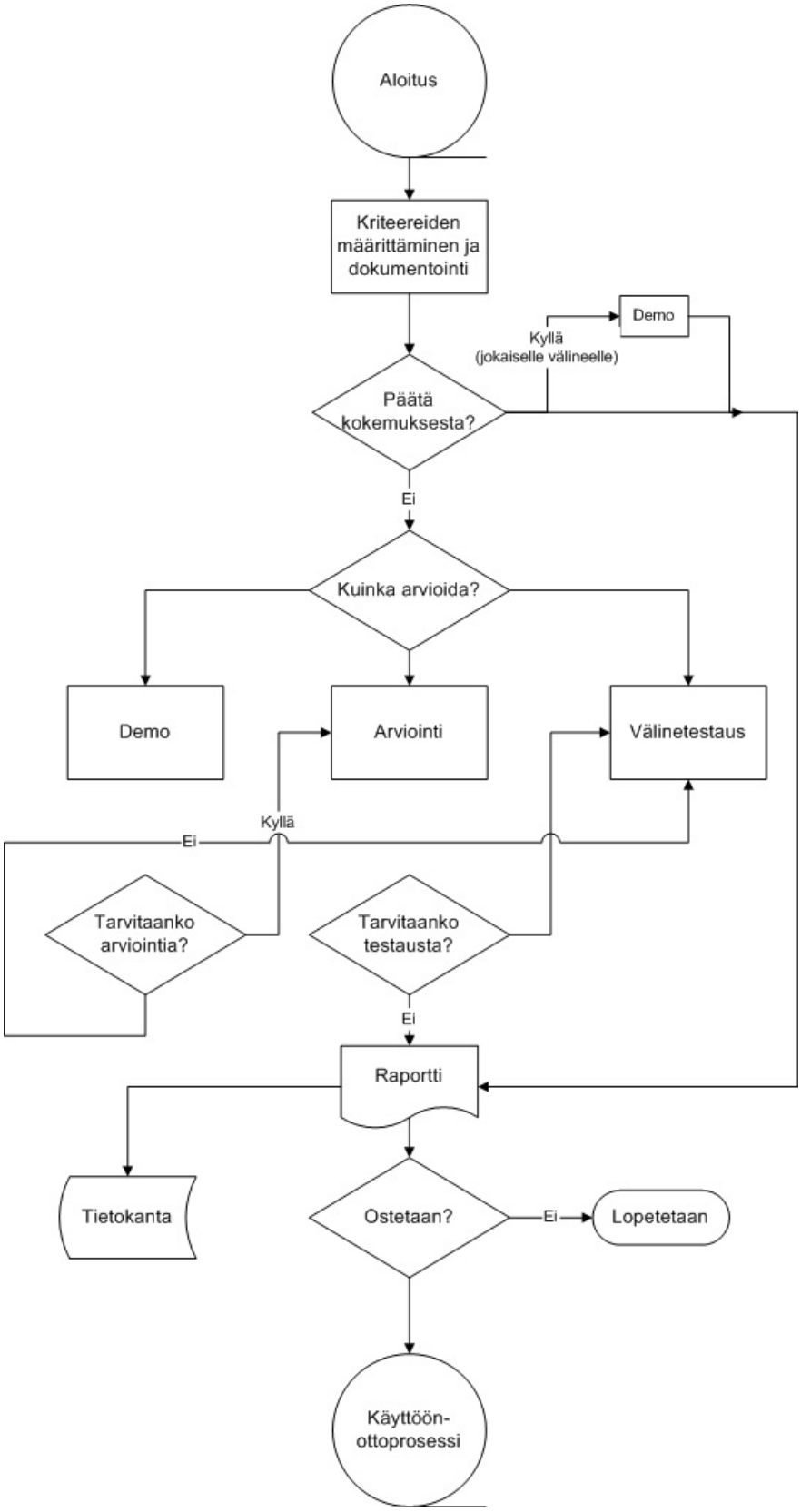
1) arviointi- ja valintaprosessin laukaisevien tekijöiden tunnistaminen ja dokumentointi (esimerkiksi ongelmat vanhoissa välineissä),
2) lausunnon muodostaminen ongelmatekijöistä, jotka välineen tulisi ratkaista,
3) arviointikriteeristön johtaminen edellisistä tehtävistä,
4) arvioitavien välineiden etsiminen ja listaaminen arviointikriteeristön mukaisesti,
5) edellisten tehtävien perusteella päätöksen tekeminen välineen ostamisesta tai kehittämisestä itse,
6) arviointiehdotuksen tekeminen, jossa kuvataan ja perustellaan arviointistrategia sekä

7) päätöksen tekeminen arviointiprosessin keskeyttämisestä tai jatkamisesta.

Arviointiprosessi

Arviointiprosessivaihe perustuu välineiden vaiheittaiseen arviointiin, jossa tehokas ja taloudellinen, välinevalintapäätöksen luonteelle ja riskeille sopiva arviointiprosessi on tärkeässä asemassa. Arviointiprosessi jakautuu tehtäviin kuvion 9 mukaisesti. Ensimmäisenä tehtävänä arviointiprosessissa on kriteereiden määrittäminen ja niiden dokumentointi. Arviointikriteeristö voidaan johtaa esiarviointiprosessin tehtävästä kolme tarkemmalle tasolle. Arviointikriteereiden määrittämisen jälkeen tehdään aikaisempiin kokemuksiin perustuen päätös arvioinnin jatkamisesta tai valintapäätöksen tekemisestä. Mikäli päädytään valintapäätökseen jo tässä vaiheessa, voidaan päätöksen oikeellisuudesta vielä varmistua testaamalla jokaista välinettä, jonka jälkeen muodostetaan lopullinen päätös. Seuraavana tehtävänä on arviointitavasta päättäminen. Arviointitapoja on kolme: demonstraation tekeminen, arviointi kriteeristön mukaisesti sekä välinetestaus todellisessa käyttöympäristössä tai projektissa.

Demonstraatioita käytetään välineiden etsinnän yhteydessä jo esiarviointiprosessissa ja niitä tulisikin arviointiprosessin yhteydessä käyttää vain pienimmissä arviointi- ja valintaprojekteissa. Välinearviointi arviointikriteeristön (ominaisuuksien tarkistuslista) mukaisesti tulisi tehdä aina riippumatta arvioinnin ja valinnan tasosta. Tärkeimmissä välinearviointiprojekteissa tulisi myös aina tehdä välinetestaus (pilot study) todellisessa käyttöympäristössä. Kaikki kolme arviointitapaa on suositeltavaa toteuttaa ja jokaisen arvioinnin lopuksi dokumentoida välinevalinnan syyt tai hylkäyspäätökset. Arviointivaiheiden jälkeen kirjoitetaan raportti arviointiprosessista ja se tallennetaan myöhempää käyttöä varten. Raportin tuloksien perusteella tehdään lopullinen ostopäätös. Kuviossa 9 on kuvattu Cookin ym. (1997, 84) esittämän CASE-välineiden arviointi- ja valintamenetelmän mukainen vaihejako.



KUVIO 9 Cookin ym. (1997, 84) CASE-välineiden arviointi- ja valintaprosessi.

Jälkiarviointiprosessi

Jälkiarviointiprosessi tarkoittaa välineen käyttöönottoa ja arviointi- sekä valintaprosessin kehittämistä edelleen saatujen kokemuksien mukaisesti. Välineen käyttöönotossa tulee huomioida erityisesti koulutuksen tärkeys. Mosleyn (1992, 32) mukaan välineen arviointi ja valinta on vain jäävuoren huippu. Loppuosa jäävuoresta muodostuu sen käyttöönottoprosessista. Cook ym. (1997, 85) toteavat, että menestyksekkääseen käyttöönottoprosessiin liittyvistä tekijöistä tarvittaisiin enemmän empiiristä tutkimusta.

3.3.2 Arviointikriteerit

Ominaisuuksien tarkistuslista on Cookin. ym. (1997) kehittämä viitekehys välinearvioinnissa käytettävien arviointikriteereiden muodostamiseen. Ominaisuuksien tarkistuslistassa jaetaan arviointiin liittyvät asiat yleisiin ja välinekohtaisiin asioihin. Yhteensä eri asialuokkia on kahdeksan:

- Liiketoimintaan liittyvät kriteerit rajoittavat välinevalintaa esimerkiksi rajaamalla hinnan tai hankintapolitiikan.
- Arviointikeskeiset asiat pitää ottaa huomioon suunniteltaessa arviointi- ja valintaprosessia. Arviointikeskeisiä kriteereitä ovat esimerkiksi arviointimenetelmä ja päätöksenteko.
- Ulkoisiin sidosryhmiin liittyen tulee hyödyntää mahdolliset organisaation ulkopuoliset asiantuntijat ja tietämys.
- Välinetoimittajan tukeen liittyvät asiat ovat oleellinen osa arvioitaessa välineen elinkaaren pituutta ja mahdollisesti tarvittavaa teknistä tukea välineelle.
- Rahoitukseen liittyen tulee kaikki välineen hankintaan ja käyttöön sidoksissa olevat kustannukset selvittää (total cost of ownership).
- Välinekohtaiset kriteerit tulee johtaa arviointi- ja valintatilanteeseen johtaneista ongelmista ja syistä sekä esimerkiksi jonkin teknologian tai arkkitehtuurin vaatimuksista (esimerkiksi malliperustainen arkkitehtuuri).

- Käyttöönottoon liittyvät asiat ovat esimerkiksi välinekoulutuksen ja välineen integroinnin suunnittelua.
- Välineen elinkaareen liittyen on tarkkailtava välineen käyttökelpoisuuden heikkenemistä ja varauduttava ajoissa mahdollisiin uusiin ratkaisuihin.

Jokaisessa luokassa kuvataan oleellisimmat arviointikriteerien muodostamisessa huomioitavat asiat, mutta arviointikriteereitä tulee määrittää tilanteen ja tarpeen mukaan myös asiakohtien ulkopuolelta.

3.4 Le Blancin ja Kornin menetelmä

LeBlancin ja Kornin (1992) kehittämä viitekehys on CASE-välineiden arviointiin ja valintaan tarkoitettu menetelmä, joka määrittelee arviointi- ja valintaprosessin sekä arviointikriteeristön arvioinnin suorittamiseksi. CASE-välineen arviointi- ja valintaprosessista tulee menetelmän mukaan muodostaa projekti, jolla on tarkoin määrätyt resurssit ja aikataulu.

LeBlanc ja Korn kiinnittävät menetelmässään erityisesti huomiota CASE-välineiden tarjoamaan ohjelmistotuotantoprosessin menetelmätukeen ja organisaation menetelmän tuntemukseen. Menetelmässä otetaan myös kantaa siihen mahdollisuuteen, että yksikään arvioinnin kohteena olevista CASE-välineistä ei ole organisaation tarpeisiin sopiva, tai useampia CASE-välineitä joudutaan valitsemaan vaatimuksien täyttämiseksi.

3.4.1 Vaihejako

Arviointi- ja valintaprosessi jakautuu kolmeen päävaiheeseen, jotka ovat:

1) ehdokasvälineiden etsintä,
2) CASE-välineen valinta sekä
3) vaatimusten ja CASE-välineen ominaisuuksien vastaavuuden varmistaminen.

Jokaisen vaiheen tuloksena on hyvin määritelty lopputulos.

Ehdokasvälineiden etsintä

Arviointi- ja valintaprosessin ensimmäisen vaiheen avulla on tarkoitus saada vastaus seuraaviin kysymyksiin: onko olemassa yhtä CASE-välinettä, joka täyttää vaatimukset vai tarvitaanko useampia, minkälaisia CASE-välineitä markkinoilla on tarjolla sekä mitkä CASE-välineet tulisi tarkemmin arvioida?

Vaiheen tuotoksena on listaus tärkeimmät vaatimukset täyttävistä CASE-välineistä ja niiden ominaisuuksista. Listauksen tarkoituksena on rajata mahdolliset tarkemmin arvioitavat välineet kahteen tai kolmeen välineeseen resurssien säästämisen vuoksi. Vaihe jakautuu seuraaviin askeleisiin:

1) ehdokasvälineiden tunnistaminen ja
2) arvioitavien välineiden valinta.

Listattaessa mahdollisia ehdokasvälineitä tärkein kriteeri on ohjelmistotuotantomenetelmän tuki. Jos CASE-väline ei tue organisaatiossa käytössä olevaa ohjelmistotuotantomenetelmää, niin sitä ei oteta edes alustavaan arvioon mukaan. Ehdokasvälineiden tunnistaminen ja listaus suoritetaan etsimällä välineitä, jotka täyttävät ehdokaskriteerit (screening criteria). Päähuomio on arviointi- ja valintaprosessin tässä vaiheessa arviointikriteerien täyttämisen tarkastelussa, ei siinä, kuinka hyvin väline kriteerit täyttää.

Valittaessa välineitä tarkempaan arviointiin tarkastellaan vielä ehdokaskriteereiden täyttämisen lisäksi välinetoimittajan luotettavuutta (esimerkiksi markkinoilla oloaika ja työntekijöiden määrä), mahdollisten sovellusalustojen ja laitteistojen tuen puuttumisen tärkeyttä sekä CASE-välineen dokumentaation laatua. Tämän tarkastelun jälkeen valitaan lopulliset tarkemmin arvioitavat välineet, joita tulee olla kahdesta kolmeen.

CASE-välineen valinta

CASE-välineen valinta -vaiheen tarkoituksena on arvioida tarkastelun kohteena olevia välineitä tarkalla tasolla ja valita organisaation vaatimukset parhaiten täyttävä väline.

Vaiheen päätehtävät ovat:

1) arviointikriteereiden määrittäminen ensimmäisen vaiheen ehdokaskriteereiden pohjalta tarkemmalle tasolle,
2) yksityiskohtaisemman tiedon kerääminen CASE-välineistä sekä
3) CASE-välineiden arviointi ja valinta kriteereiden mukaisesti.

Yksityiskohtaiset arviointikriteerit määritetään ehdokaskriteereiden pohjalta. Tärkeimmän painoarvon kriteereistä saavat CASE-välineen toiminnalliset vaatimukset. Tehtävän tarkoituksena on kehittää kattava toiminnallinen näkymä ehdotettuun sovelluskehitysalustaan ja tiivistää CASE-välineeltä vaadittavat ominaisuudet arviointikriteeristöksi. Määritettäessä CASE-välineen toiminnallisia vaatimuksia tulee myös vaatimusten tärkeystaso sekä organisaation käyttämän ohjelmistotuotantomenetelmän vaatimukset dokumentoida.

Arviointikriteereiden määrittämisen jälkeen *kerätään yksityiskohtaista tietoa CASE-välineistä*, jotta niitä voidaan arvioida kriteeristön mukaisesti. Informaatiota CASE-välineistä voidaan saada muun muassa keskusteluista CASE-välinetoimittajan kanssa. CASE-välinetoimittajalle voidaan myös lähettää yksityiskohtainen kysely CASE-välineen ominaisuuksista.

Arvioitaessa ja valittaessa CASE-välinettä on kiinnitettävä erityisesti huomiota siihen, mitä vaadittavia ominaisuuksia CASE-välineissä ei ole. LeBlancin ja Kornin (1992) mukaan minkä tahansa arviointikriteereiden eri tärkeysasteen huomioonottavan menetelmän käyttäminen arvioinnissa on mahdollista, koska ne muutaman vaihtoehdon ollessa kyseessä antavat kuitenkin saman lopputuloksen. Arvioitavia CASE-välineitä

tulee tarpeen mukaan koekäyttää myös varsinaisilla loppukäyttäjillä ja niiden todellisessa käyttöympäristössä.

Vaatimusten ja CASE-välineen ominaisuuksien vastaavuuden varmistaminen

Vaatimusten ja CASE-välineen ominaisuuksien vastaavuuden varmistaminen suoritetaan ottamalla CASE-väline mukaan todelliseen käyttötilanteeseen. Vaiheen tarkoituksena on varmistaa CASE-välineen sopivuus ja tehokkuus sekä tarjota viimeinen mahdollisuus harkita päätöstä. Vaiheen aikana kehitetään CASE-välineen avulla demonstraatiosovelluksia tai prototyyppejä. Mikäli CASE-väline ei täytä kaikkia sille määritettyjä toiminnallisia vaatimuksia, tulee LeBlancin ja Kornin (1992) mukaan harkita CASE-välinetoimittajan konsultointia ja vaadittavien ominaisuuksien kehittämistä tämän toimesta. Vaihtoehtoisesti voidaan harkita toiminnallisten vaatimusten täyttämistä kehittämällä itse vaadittava lisäosa. CASE-välineen kehittäminen itse ei ole kuitenkaan suositeltavaa LeBlancin ja Kornin (1992) mukaan.

3.4.2 Arviointikriteerit

Ehdokaskriteeristö sisältää ainoastaan muutamia ehdottoman tärkeitä arviointikriteerejä. Arviointikriteereiden tulee keskittyä erityisesti ehdokasvälineiden etsintä -vaiheessa CASE-välineeltä vaadittaviin toiminnallisiin ominaisuuksiin, jotka liittyvät organisaatiossa käytössä olevaan ohjelmistotuotantomenetelmään. CASE-välineen valinta -vaiheessa arviointikriteeristöä laajennetaan käsittämään myös muita arvioinnin alueita. CASE-välineiden arviointikriteerit voidaan jakaa neljään eri luokkaan sekä ehdokasvälineiden etsintä -vaiheessa että CASE-välineen valinta -vaiheessa:

- **Tekniset vaatimukset**. CASE-välineen täytyy olla yhteensopiva organisaatiossa käytössä olevien käyttöjärjestelmien ja laitteistojen kanssa. Muita mahdollisia arviointikriteerejä ovat esimerkiksi ohjelmointikielen tuki, muistin tarve tai tiedon siirrettävyys.

- **Toiminnalliset vaatimukset**. Toiminnallisissa vaatimuksissa korostetaan ennen kaikkea tukea organisaation käyttämälle ohjelmistotuotantomenetelmälle. Toiminnallisia vaatimuksia voivat olla esimerkiksi tuki kaikille ohjelmistotuotantomenetelmän mukaisille vaiheille, integroitu tietovarasto tai standardi käyttöliittymä. Toiminnalliset vaatimukset ovat ehdottomia ja välineen tulee täyttää ne.
- **Dokumentaatio ja koulutus**. CASE-välineen dokumentaation pitää olla yksityiskohtainen, kattava ja helppotajuinen. Tärkeitä arviointikriteereitä voivat olla myös välinetoimittajan tarjoamat koulutukset ja ylimääräinen materiaali.
- **Tiedot välinetoimittajasta**. Tärkeitä arviointikriteerejä voivat olla muun muassa CASE-välineen ylläpidollinen tuki, koulutus, asennuspalvelu, välinetoimittajan rahoituksellinen asema sekä välinetoimittajan markkinoillaoloaika.

3.5 ISO/IEC 14102 -standardin mukainen menetelmä

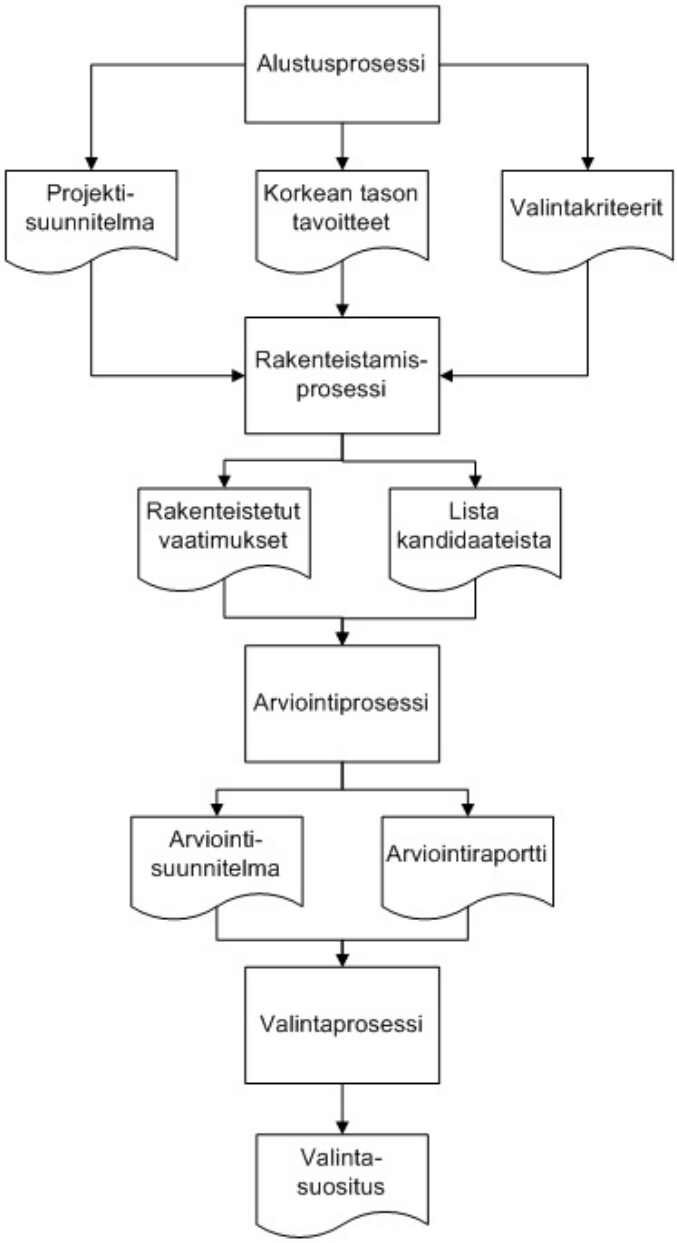
ISO/IEC 14102 on kansainvälinen standardi CASE-välineiden arviointiin ja valintaan (Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995). Standardi määrittelee arviointi- ja valintaprosessin sekä strukturoidun ja erittäin laajan arviointikriteeristön CASE-välineiden arviointiin ja valintaan. Standardin määrittelemät arviointi- ja valintaprosessit ovat yleisluontoisia, joten organisaatioiden on räätälöitävä ne omia tarpeitaan vastaaviksi. Standardin tavoitteena on avustaa organisaatiokohtaisten vaatimusten tunnistamisessa, vaatimusten muuntamisessa CASE-välineiden arviointikriteereiksi sekä määritellä prosessi, jonka avulla on mahdollista valita organisaatiolle sopivin CASE-väline useiden ehdokasvälineiden joukosta.

3.5.1 Vaihejako

CASE-välineiden arviointi- ja valintaprosessi muodostuu neljästä pääprosessista, jotka ovat:

1) alustusprosessi,
2) rakenteistamisprosessi,
3) arviointiprosessi sekä
4) valintaprosessi.

Kuviossa 10 on kuvattu menetelmän mukainen CASE-välineiden arviointi- ja valintaprosessi sekä sen eri vaiheiden päätuotokset.



KUVIO 10 ISO/IEC 14102 -standardin mukainen arviointi- ja valintaprosessi.

Alustusprosessi

Alustusprosessi rakentuu kolmesta tehtävästä, jotka ovat:

1) korkean tason tavoitteiden määritteleminen,
2) valintakriteereiden muodostaminen sekä
3) projektin hallinta ja suunnittelu.

CASE-välineiden arviointi- ja valintaprosessilla täytyy olla organisaation johdon tuki. Tämän vuoksi määritetään CASE-välineiden arvioinnille ja valinnalle yleiset *korkean tason tavoitteet*, joiden tarkoituksena on esitellä prosessia organisaation johdolle ja määritellä projektin tavoitteet ja odotukset.

CASE-välineiden arviointi- ja valintaprosessille määritettyjen yleisten korkean tason tavoitteiden ja odotusten perusteella *muodostetaan valintakriteerit*, joiden tulee olla objektiivisia ja määrällisesti arvioitavissa. Valintakriteereiden tärkeysjärjestys täytyy myös määritellä, jotta arviointikriteereille voidaan määritellä painoarvot.

Korkean tason tavoitteiden ja valintakriteereiden pohjalta *luodaan projektisuunnitelma* sekä arviointi- ja valintaprosessin hallintamekanismi, jonka avulla projektin etenemistä voidaan valvoa.

Rakenteistamisprosessi

Rakenteistamisprosessi muodostuu kolmesta tehtävästä, jotka ovat:

1) vaatimusten määrittäminen,
2) tiedon kerääminen CASE-välineistä sekä
3) ehdokasvälineiden tunnistaminen.

Vaatimukset CASE-välineelle kerätään muun muassa organisaatiosta ja sen toiminnasta, CASE-välineiden käyttäjiltä, vanhoista CASE-välineistä, käytetystä ohjelmistotuotantomenetelmästä ja CASE-välineeltä vaadittavista toiminnallisista ominaisuuksista. Survey-tutkimukset ja haastattelut ovat hyviä tiedonkeruutapoja vaatimusten määrittämisessä. Vaatimukset tulee tunnistaa ja luokitella käytetyn ohjelmistotuotantomenetelmän ja sen vaihejaon, CASE-välineeltä vaadittavien toiminnallisten ominaisuuksien sekä CASE-välineeltä vaadittavien laadullisten ominaisuuksien mukaisesti. Viimeisenä vaiheena ominaisuudet (arviointikriteerit) jaetaan standardin määrittelemällä tavalla aliominaisuuksiksi sekä atomisiksi ominaisuuksiksi ja niille annetaan halutut arvioinnissa käytettävät painoarvot.

Tiedon kerääminen CASE-välineistä on iteratiivinen prosessi, jonka tarkoituksena on kerätä tietoa markkinoilla olevista vaatimukset täyttävistä CASE-välineistä. CASE-välineistä tulee kerätä yleisiä tietoja, kuten välinetoimittajan yleiset tiedot, välineen kustannukset, laitteisto- ja ohjelmistovaatimukset, koulutuksen tarve, välineen toiminnalliset mahdollisuudet, välineen ohjelmistotuotantomenetelmätuki sekä välineen lisensiointitavat.

Varteenotettavien CASE-välineiden tunnistamisen jälkeen valitaan lopulliset välineet arvioitavaksi. Kandidaattivälineiden valinnassa käytetään apuna kriittisten kriteereiden määrittämistä, käyttäjien vaatimuksien vertaamista CASE-välineen toiminnallisiin ominaisuuksiin ja menetelmätukeen, organisaation johdon määrittämiä rajoitteita (esimerkiksi kustannukset, koulutus ja tuki) sekä välinetoimittajan liiketoimintahistorian analysointia. Lopullinen arvioitavien välineiden valinta suoritetaan kriittisten kriteereiden perusteella.

Arviointiprosessi

CASE-välineiden arviointiprosessissa suoritetaan seuraavat tehtävät:

1) arvioinnin valmistelu,

2) CASE-välineiden arviointi ja
3) arvioinnin raportointi.

Arvioinnin valmistelun tarkoituksena on määritellä, kuinka CASE-välineiden arviointi suoritetaan. Valmisteltaessa arviointia määritetään mittayksiköt jokaiselle kriteerille, tunnistetaan ja aikataulutetaan kaikki eri tehtävät, jotka arvioinnin aikana tulee suorittaa sekä joissakin tapauksissa valmistellaan testiprojekti tai -sovellus, jonka avulla CASE-välineitä voidaan arvioida. Arvioinnin valmistelun lopputuloksena on arviointisuunnitelma, jonka mukaisesti CASE-välineiden arviointi tullaan toteuttamaan.

CASE-välineiden arviointi suoritetaan valittujen arviointikriteereiden mukaisesti. Arvioinnissa CASE-välineen ominaisuuksia verrataan arviointikriteereitä vastaan ja jokaiselle ominaisuudelle määritetään mittayksikön mukainen tulos. Mittaustulokset voidaan johtaa testikäyttäjien haastatteluista, välinetoimittajan antamasta informaatiosta, tutkimalla välineen lähdekoodia tai toteuttamalla joitakin testiprojekteja CASE-välineellä. Mittayksiköt voivat olla subjektiivisia tai objektiivisia sekä muodoltaan tekstimuotoisia, numeerisia tai binäärisiä.

Arvioinnin tuloksena on arviointiraportti, joka sisältää arvioinnin kaikista arvioinnin kohteena olleista CASE-välineistä. Jokaisesta CASE-välineestä voidaan myös kirjoittaa arviointiraportti erikseen. Arviointiraportin tulee sisältää ainakin seuraavanlaisia tietoja: CASE-välineen nimi, CASE-välineen versio, välinetoimittaja, käytetty konfiguraatio, kustannukset ja niiden muodostuminen, CASE-välineen tarkoitus ohjelmistotuotannossa, CASE-välineen ohjelmistoympäristö (esimerkiksi menetelmätuki, ohjelmointikielituki, käyttöjärjestelmätuki ja tietokantayhteensopivuus), CASE-välineen toiminnalliset ominaisuudet, tiedot arviointiprosessista sekä lopulliset tulokset kriteerikohtaisesti ja yhteenveto välineestä.

Valintaprosessi

CASE-välineen valinta voidaan suorittaa, kun arviointiraportti on valmis. CASE-välineiden valintaprosessi muodostuu neljästä tehtävästä, jotka ovat:

1) valinnan valmistelu,
2) valinta-algoritmien toteuttaminen,
3) valintapäätöksen suositteleminen sekä
4) valintapäätöksen tarkastelu.

Valmisteltaessa CASE-välineen valintaa muodostetaan valinta-algoritmi, joka määrittää, kuinka kaikki arvioinnin aikana kerätty tieto saatetaan yhdenmukaiseksi arvioinnin kohteena olleiden eri välineiden kesken. Valinnan valmisteluvaiheessa tunnistetaan myös lopulliset valintakriteerit, joiden perusteella CASE-välineen valintapäätös tehdään.

Valinta-algoritmin toteuttamisen lopputuloksena saadaan jokaisesta arvioinnin kohteena olleesta välineestä ja sen ominaisuuksista tekninen yhteenveto. Arvioinnin tuloksia käytetään valinta-algoritmin syötteinä ja tuloksena saadaan informaatiota välineestä.

Valinta-algoritmin toteutuksen jälkeen suositellaan organisaation johdolle arvioinnin mukaan organisaatiolle sopivinta CASE-välinettä tai niiden joukkoa. Lopullinen CASE-välineen valintapäätös perustuu johdon omiin kriteereihin ja arvioinnin tuloksiin. Mikäli päätöksenteko ei ole mahdollista saaduilla tiedoilla, voidaan CASE-välineiden arviointia jatkaa iteratiivisesti, kunnes haluttu tietämys saavutetaan päätöksenteon tueksi. On myös mahdollista, että arvioinnin tuloksena todetaan, että organisaation tarpeisiin sopivaa CASE-välinettä ei ole, jolloin vaihtoehdoiksi jäävät CASE-välineen kehittäminen itse tai olemassa olevien välineiden sovittaminen omiin tarpeisiin sopivaksi.

CASE-välineen arviointi- ja valintaprosessin viimeinen tehtävä on *valintapäätöksen tarkastelu*. Alkuperäisiä CASE-välineelle asetettuja tavoitteita, kriteereitä sekä korkean tason tavoitteita tulee arvioida ja verrata arviointituloksiin sekä suositukseen valittavasta välineestä.

3.5.2 Arviointikriteerit

ISO/IEC 14102 -standardi määrittää laajan käyttäjien tarpeisiin perustuvan luokituksen arviointikriteereille sekä myös tarkan luokitukseen perustuvan arviointikriteeristön. Alustusprosessin aikana määritetyt vaatimukset ja korkean tason tavoitteet luokitellaan rakenteistamisprosessissa standardin määrittelemällä tavalla ominaisuuksiksi, aliominaisuuksiksi sekä alimmalla tasolla atomisiksi ominaisuuksiksi (arviointikriteereiksi). Standardin mukaan kaikkia määriteltyjä atomisia ominaisuuksia ei tarvita jokaisessa arviointi- ja valintaprojektissa, vaan niitä tulee käyttää tarpeen mukaan.

ISO/IEC 14102 standardin mukaan CASE-välineiden ominaisuudet luokitellaan seuraavalla tavalla:

- Toiminnallisuus - ominaisuudet, jotka liittyvät ohjelmistotuotantoprosessin elinkaareen
  - johtamisprosessi
  - ohjelmistotuotantoprosessi
  - ylläpitoprosessi
  - dokumentointiprosessi
  - konfiguraationhallintaprosessi
  - laadunvarmistusprosessi
  - todennettavuusprosessi (verification process)
  - jäljitettävyysprosessi (validation process)
- Toiminnallisuus - ominaisuudet, jotka liittyvät CASE-välineen käyttöön
  - CASE-välineen toimintaympäristö
  - CASE-välineen integroitavuus

  - CASE-välineen sovelluskohtaiset ominaisuudet
- Yleiset laatuominaisuudet
  - toiminnallisuus
  - luotettavuus
  - käytettävyys
  - suorituskyky
  - ylläpidettävyys
  - siirrettävyys
- Yleiset ominaisuudet, jotka eivät liity laatuun
  - hankintaprosessi
  - toteutus
  - tuki
  - välinetoimittajan mahdolliset sertifikaatit

Jokaisella aliominaisuudella on lisäksi joukko atomisia ominaisuuksia. Luokitus ja atomisten ominaisuuksien listaus on kaiken kaikkiaan erittäin kattava ja laaja.

3.6 Yhteenveto

Järjestelmien ja sovellusten kehittämisen automatisoinnin tukena tarvitaan CASE-välineitä (Computer-Aided Software/Systems Engineering). Käytössä olevat teknologiat, kuten ohjelmointikielet, ohjelmistoarkkitehtuurit ja sovellusalustat kuitenkin muuttuvat ja sovelluskehityksen apuvälineiden on muututtava niiden mukana. Yrityksillä ja organisaatioilla on tämän vuoksi jatkuvasti tarve arvioida ja valita erilaisia ohjelmistovälineitä, joihin myös CASE-välineet kuuluvat. Ohjelmistovälineiden ja CASE-välineiden arviointiin sekä valintaan onkin tämän vuoksi kehitetty useita erilaisia menetelmiä, jotka pyrkivät vaiheistamaan ja ohjaamaan arviointi- ja valintaprosessia.

CASE-välineiden arviointi- ja valintaprosessi on monimutkainen, joten siihen liittyy paljon ongelmia ja riskejä. Ongelmia CASE-välineiden arvioinnissa ja valinnassa aiheuttavat muun muassa päätöksenteossa käytettävien kriteereiden monimutkaisuus, arviointipäätöksen riskialttius ja epävarmuus, päätöksentekoon liittyvät paineet,

päätöksentekoprosessi ja siihen liittyvät vastuut, kehitysympäristön jatkuva muuttuminen ja kypsymättömyys sekä arviointi- ja valintaprojektin vähätteleminen ja resurssien aliarviointi. Ratkaisukeinoja ongelmiin ja riskeihin ovat muun muassa päätöksentekoprosessin dokumentointi ja välinearviointiin sekä valintaan osallistuvien ihmisten sitouttaminen, CASE-välineen huolellinen sovittaminen ympäristöönsä, CASE-välineen elinkaaren jatkuva hallinta, CASE-välineen kokeileminen todellisessa ympäristössä ennen ostopäätöksen tekemistä sekä omien ohjelmistotuotantoon liittyvien prosessien jatkuva kehittäminen.

CASE-välineiden arviointiin ja valintaan on kehitetty useita erilaisia menetelmiä. Tässä luvussa esiteltiin yleisellä tasolla seitsemän erilaista ohjelmistovälineiden ja CASE-välineiden arviointi- ja valintamenetelmää. Tarkemmin kuvattiin kolmea tämän tutkielman kannalta oleellisinta ja kattavinta menetelmää (Cook ym. 1997, Korn & LeBlanc 1992 sekä Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995). Kaikki näistä menetelmistä määrittelevät sekä vaihejaon arviointi- ja valintaprosessin läpiviemiseksi että kattavan luokituksen arviointikriteereiden muodostamiseksi. Menetelmillä on myös paljon muitakin yhteisiä piirteitä ja samankaltaisuuksia. Menetelmien vaihejakojen osalta yhdenmukaisia piirteitä ovat muun muassa arviointi- ja valintaprosessin laukaisevien tekijöiden tunnistaminen ja dokumentointi, ehdokasvälineiden tunnistaminen, nopea ehdokasvälineiden rajaaminen sekä arviointikriteeristön täydentäminen tilannekohtaisesti. Arviointikriteereiden osalta yhdenmukaisia piirteitä ovat esimerkiksi kriteereiden jako välinekohtaisiin ja yleisiin kriteereihin, kriteereiden painottuminen erityisesti välineen teknisiin ja toiminnallisiin ominaisuuksiin sekä myös välinetoimittajan huomioiminen osana arviointikriteeristöä.

Luvussa esitellyt ja kuvatut menetelmät eivät kuitenkaan sellaisenaan sovellu malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointiin. Erityisesti arviointikriteeristön osalta malliperustainen arkkitehtuuri asettaa omia vaatimuksiaan. Toisaalta on myös huomioitava, että malliperustainen arkkitehtuuri koskettaa yrityksen tai organisaation koko järjestelmäkehitystä ja järjestelmien integrointia. Tämä asettaa omat vaatimuksensa myös CASE-välineiden arviointi- ja valintaprosessin vaihejaolle.

4 CASE-VÄLINEIDEN ARVIOINTI- JA VALINTAMENETELMÄ

Tässä luvussa esitellään ja kuvaillaan tutkielman edellisten lukujen pohjalta kehitetty CASE-välineiden arviointi- ja valintamenetelmä malliperustaiselle arkkitehtuurille (myöhemmin luvun aikana menetelmä). Menetelmän vaihejako ja arviointikriteeristön viitekehys perustuvat osittain Cookin ym. (1997), Kornin ja LeBlancin (1992) sekä ISO/IEC 14102 -standardin (Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995) esittämiin vaihejakoihin ja arviointikriteeristöihin. Lisäksi lähteinä on käytetty markkinatutkimuslaitos Ovum Ltd:n (Bakker, Budd, George, Ormrod & Stevenson 2000) sekä tutkijoiden tekemiä (Church & Matthews 1995, Greefhorst, Maat & Maijers 1998) CASE-välineiden arviointi- ja valintaraportteja.

Menetelmässä otetaan aiemmin kirjallisuudessa esitettyjä CASE-välineiden arviointi- ja valintamenetelmiä paremmin huomioon CASE-välineiden käytettävyyden arviointi sekä arviointi- ja valintaprosessin projektiluonteisuus. Käytettävyys merkitsee usein käyttäjille samaa kuin koko järjestelmä (Apperley & Duncan 1994, 60). Tämän vuoksi on luontevaa, että käytettävyyden arviointi on tärkeä osa välinearviointia. Lähteinä käytettävyyden arvioinnissa on käytetty Duk Choin ym. (1998) oliopohjaisten CASE-välineiden käytettävyyden arviointiin tarkoitettua menetelmää sekä Nielsenin (1993) käytettävyyden arviointiin tarkoitettuja heuristiikoita. Pratherin (1993, 60) mukaan CASE-välineiden arviointi- ja valintaprosessin tulee olla selkeästi määritelty ja resursoitu projekti. Kirjallisuudessa aiemmin esitetyissä menetelmissä prosessin projektiluonteisuutta ei kuitenkaan ole huomioitu millään tavalla. Kehitetyssä menetelmässä projektin piirteitä on otettu mukaan suoraan arviointi- ja valintaprosessin vaihejakoon.

Luvun aluksi täsmennetään vaatimuksia, joiden mukaisesti menetelmä on kehitetty. Tämän jälkeen esitellään menetelmän mukainen vaihejako CASE-välineiden arviointi- ja valintaprosessille. Luvun lopuksi kuvataan menetelmän arviointikriteeristön viitekehys.

4.1 Vaatimuksia menetelmälle

Tässä kappaleessa esitellään näkökohtia ja vaatimuksia, joiden mukaan menetelmää lähdettiin kehittämään. Kaikkia CASE-välineiden arviointi- ja valintamenetelmälle asetettavia vaatimuksia ei käsitellä, vaan ainoastaan tämän tutkielman kannalta oleellisimpia, joita halutaan erityisesti tässä yhteydessä korostaa.

Käytettävyys jää usein vähälle huomiolle CASE-välineiden arvioinnissa. Tämä näkyy ennen kaikkea katsottaessa menetelmien arviointikriteereitä. Käytettävyyden arviointiin on usein tarkoitettu vain yksi kriteeri, joka sekin saattaa olla ilmaistu suoraan käytettävyytenä. Ongelma tästä muodostuu sen vuoksi, että ihmiset ymmärtävät käsitteen käytettävyys eri tavalla. Monia käytettävyyden osa-alueita jää tällöin arvioimatta. Tutkielman tuloksena kehitetyssä menetelmässä käytettävyyden arviointiin on kiinnitetty enemmän huomiota juuri arviointikriteereiden muodossa. Ne auttavat ainakin huomioimaan suurimman osan käytettävyyteen kuuluvista asioista.

CASE-välineiden arviointi- ja valintaprosessin tulee olla selkeästi määritelty ja resursoitu projekti. Ottamalla menetelmän vaihejakoon suoraan mukaan projektimaisia piirteitä saavutetaan seuraavanlaisia etuja. Arviointi- ja valintaprosessista tulee hyvin dokumentoitu ja resursoitu. Prosessin eri osapuolet ja heidän vastuunsa tulee määriteltyä selkeästi ja prosessin eri vaiheissa on eri sidosryhmille yhteisiä arviointi- ja päätöksentekotilaisuuksia, joissa on mahdollista tehdä prosessiin liittyviä päätöksiä, sekä tarkastella sen tilaa ja jatkamisen järkevyyttä.

Menetelmän vaihejakoon ei sisällytetä käyttöönottovaihetta sen erilaisen luonteen vuoksi. Käyttöönottovaihe on oma projektinsa, jolle tulee määritellä omanlaisensa vaatimukset ja prosessi. Käyttöönottovaiheen ongelmiin pyritään kuitenkin vastaamaan jo arviointi- ja valintaprosessin aikana, siten että välineen loppukäyttäjät eli ohjelmistosuunnittelijat suorittavat CASE-välineiden arvioinnin. Tällä vastataan ennen kaikkea käyttöönottovaiheen aikana usein ilmeneviin käyttäjien hyväksymisongelmiin ja osittain myös vähennetään koulutuksen tarvetta.

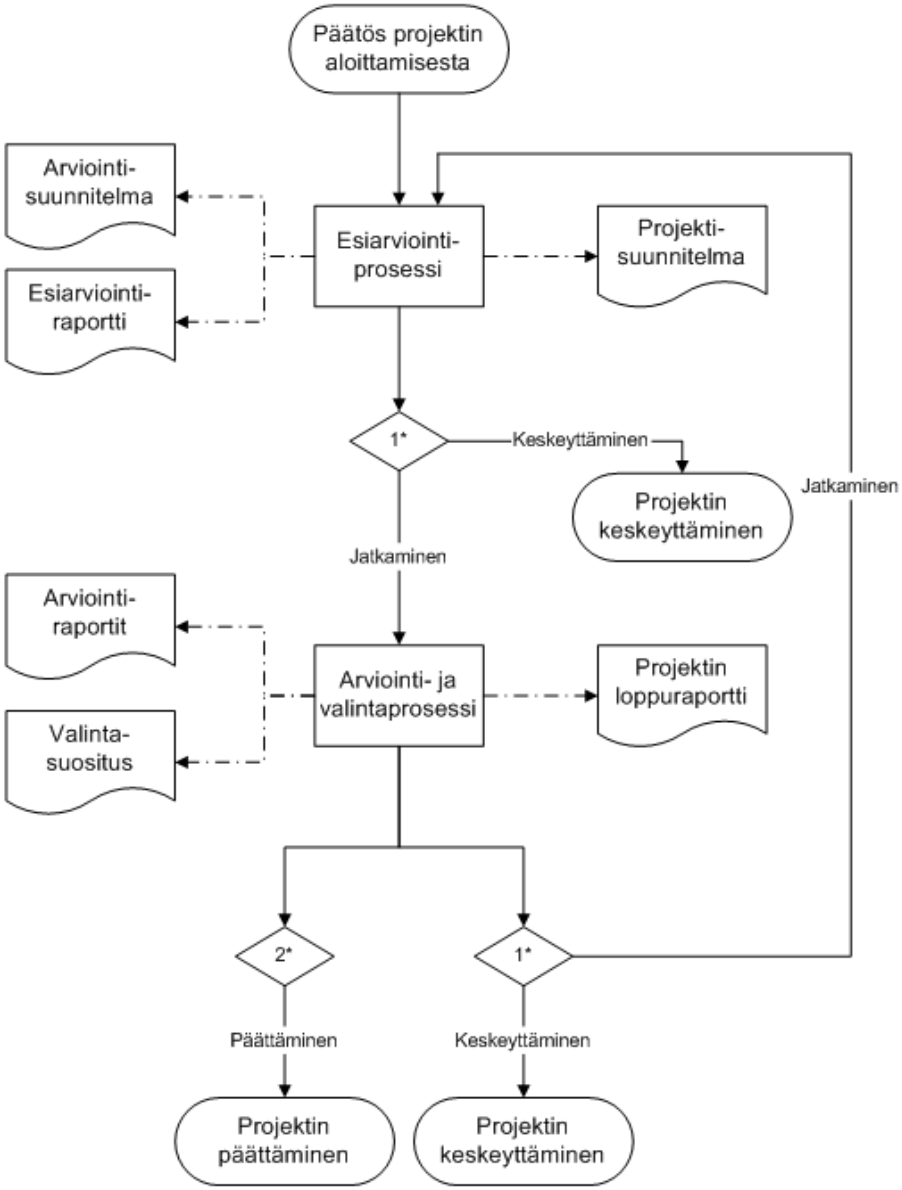
Malliperustainen arkkitehtuuri asettaa omanlaisensa vaatimukset CASE-välineiden arviointi- ja valintamenetelmälle. Malliperustaisen arkkitehtuurin avulla pyritään ratkaisemaan ohjelmistotuotantoprosessin ongelmia laaja-alaisesti. Tällöin prosessin tueksi ei yleensä riitä yksittäinen CASE-väline, vaan integroitu CASE-ympäristö tai joukko hyvin erilaisia CASE-välineitä. CASE-ympäristöt ovat laajoja ja kalliita kokonaisuuksia, joissa erilaisia toiminnallisia ominaisuuksia saattaa olla tuhansia. Tämän vuoksi malliperustaista arkkitehtuuria tukeva CASE-välineiden arviointi- ja valintamenetelmä ei arviointikriteeristön osalta voi olla kuin suuntaa-antava viitekehys, jonka avulla keskeisimmät arkkitehtuurin vaatimukset CASE-ympäristölle ja sen eri CASE-välineille tunnistetaan.

4.2 Vaihejako

Menetelmän vaihejaon tarkoituksena on vaiheistaa CASE-välineiden arviointi- ja valintaprosessi selkeisiin osakokonaisuuksiin, sekä tarjota ohjeistus arviointi- ja valintaprojektin läpiviemiseksi. Menetelmän CASE-välineiden arviointi- ja valintaprosessi jakautuu kahteen pääprosessiin:

1) esiarviointiprosessiin ja
2) arviointi- ja valintaprosessiin.

Kuviossa 11 on esitetty menetelmän mukainen vaihejako sekä vaiheiden lopputulokset. Menetelmän eri vaiheet ja vaiheisiin liittyvät dokumentit kuvataan tarkemmin kuvion jälkeen.



1* Päätös arviointi- ja valintaprosessin jatkamisesta tai keskeyttämisestä.
2* Valintapäätöksen tekeminen valintasuosituksen mukaisesti.

KUVIO 11 Malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmän vaihejako.

Esiarviointiprosessi

Esiarviointiprosessi koostuu kymmenestä tehtävästä, joiden tavoitteena on projektin muodostaminen arviointi- ja valintaprosessista sekä tarkemmin arvioitavien välineiden valitseminen arviointi- ja valintaprosessiin. Esiarvioinnin tehtävät ovat:

1) projektiorganisaation muodostaminen ja projektisuunnitelman tekeminen,
2) arviointi- ja valintaprosessin laukaisevien tekijöiden tunnistaminen (vrt. Cooke ym. 1997),
3) korkean tason tavoitteiden ja rajoitteiden määritteleminen (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995),
4) esiarviointikriteereiden ja arviointikriteereiden täydentäminen sekä varmentaminen,
5) arviointikriteereiden mittayksiköiden ja painoarvojen määrittäminen,
6) ensimmäisen projektin johtoryhmän kokouksen pitäminen,
7) esiarvioitavien ehdokasvälineiden tunnistaminen (vrt. LeBlanc & Korn 1992 ja Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995),
8) ehdokasvälineiden esiarviointi,
9) arviointisuunnitelman täydentäminen sekä
10) toisen projektin johtoryhmän kokouksen pitäminen.

Esiarviointiprosessin ensimmäisenä tehtävänä on *projektiorganisaation muodostaminen ja projektisuunnitelman tekeminen*. Muodostettaessa projektiorganisaatiota on huomioitava, että projektiryhmän jäseniksi tulee riittävästi ohjelmistoalan ammattilaisia, joilla on kokemusta arvioitavien välineiden kaltaisista sovelluksista. Heidän osaltaan on myös varmistuttava resurssien riittävyydestä, koska heidän on myös mahdollisina loppukäyttäjinä käytännössä järkevintä arvioida välineitä. Projektisuunnitelman muodostaminen on projektipäällikön vastuulla. Projektisuunnitelmasta tulisi ilmetä ainakin seuraavat asiat:

- projektin tarkoitus ja tavoitteet,
- projektin tulokset,
- projektin aikataulu,
- projektiorganisaatio ja tehtäväjako,
- projektin valvonta, raportointi ja vastuut,
- projektin aikana tuotettavat dokumentit sekä
- projektin toteutussuunnitelma.

Arviointi- ja valintaprosessin laukaisevia tekijöitä voivat olla esimerkiksi ongelmat aiemmin käytetyissä CASE-välineissä tai kokonaan uuden teknologian käyttöönottaminen organisaation ohjelmistotuotannossa (vrt. Cook ym. 1997). Vaatimukset ja ongelmat, jotka CASE-välineen tulee ratkaista, on määritettävä ja dokumentoitava. CASE-välineitä käyttävät ohjelmistosuunnittelijat ovat ensisijaisesti niitä henkilöitä, jotka vaatimukset ja ongelmat määrittävät. Myös organisaation johto saattaa esittää vaatimuksia, joiden vuoksi arviointi- ja valintaprosessi on käynnistettävä. Projektipäällikön tehtävänä on dokumentoida arviointi- ja valintaprosessin laukaisevat tekijät arviointisuunnitelmaan.

Korkean tason tavoitteet ja rajoitteet ovat organisaation johdon määrittämiä taloudellisia tai strategisia vaatimuksia sekä rajoitteita, joiden puitteissa CASE-väline tulee valita (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995). Korkean tason tavoitteita ja rajoitteita voivat olla esimerkiksi organisaation pitkän aikavälin strateginen linja ja sen vaikutus ohjelmistovälineisiin, kuten laitteisto- ja sovellusalustoihin tai ohjelmointikieliin, sekä projektin käytössä olevat resurssit. Korkean tason tavoitteet ja rajoitteet dokumentoidaan projektipäällikön toimesta arviointisuunnitelmaan. Arviointi- ja valintaprosessin laukaisevista tekijöistä, sekä korkean tason tavoitteista ja rajoitteista, on lopuksi määriteltävä ne, jotka CASE-välineen tulee täyttää. Nämä pakolliset vaatimukset dokumentoidaan esiarviointiraporttiin.

Menetelmän arviointikriteeristö jakautuu kahteen erilliseen osaan: *esiarviointikriteereihin* ja *arviointikriteereihin*. Molemmat näistä kriteeristöistä ovat viitekehyksiä, joissa kuvataan yleisimmät esiarviointi- ja arviointiprosesseissa tarvittavat kriteerit. Esiarviointikriteerit ovat kysymyslauseita, joiden avulla arvioidaan kaikkia sovellusmarkkinoilta löytyviä CASE-välineitä, ja valitaan niistä arviointi- ja valintaprosessiin tarkoituksenmukaisimmat (ks. kappale 4.3.1). Esiarviointikriteereiden viitekehyksessä on listattu tämän tutkielman perusteella olennaisimmat esiarviointikriteerit, joiden mukaisesti esiarviointi tulee suorittaa. Viitekehyksessä ei kuitenkaan välttämättä ole kaikkia tarvittavia esiarviointikriteereitä, tai vastaavasti siinä on joitakin sellaisia, joita ei kaikissa tilanteissa tarvita. Organisaatio- ja

tilannesidonnaisuuden vuoksi esiarviointikriteereiden viitekehystä tuleekin päivittää tarpeen mukaan. Esiarviointikriteereitä voidaan johtaa arviointikriteereiden viitekehyksestä tai organisaation tavoitteista, vaatimuksista ja rajoitteista, jotka on määritetty esiarviointiprosessin edellisten tehtävien aikana. Esiarviointikriteereiden tulee keskittyä välineen teknisiin ominaisuuksiin ja niiden olemassaolon arviointiin, ei jonkin ominaisuuden hyvyyden tarkasteluun. Esiarviointikriteereiden tulee olla myös helposti todennettavissa. Arviointikriteerit ovat arviointi- ja valintaprosessissa käytettäviä kriteereitä, joiden avulla arvioidaan esiarvioinnin läpäisseitä CASE-välineitä. Arviointikriteeristön viitekehykseen on tämän tutkielman tuloksena listattu kaikki olennaisimmat arviointikriteerit, joita malliperustaista arkkitehtuuria tukevien CASE-välineiden arvioinnissa tullaan tarvitsemaan (ks. kappale 4.3.2). Kuten esiarviointikriteereidenkin tapauksessa, myös arviointikriteereiden viitekehystä täytyy tarvittaessa päivittää tilanteesta ja organisaatiosta riippuen. Arviointikriteereitä voidaan johtaa esiarviointiprosessin edellisten tehtävien aikana määritetyistä organisaation tavoitteista, vaatimuksista ja rajoitteista. Myös organisaation käytössä olevat laitteistot, ohjelmistot ja menetelmät vaikuttavat arviointikriteereihin. Esiarviointikriteerit dokumentoidaan esiarviointiraporttiin ja arviointikriteerit arviointisuunnitelmaan valintaprojektissa mukana olevien ohjelmistosuunnittelijoiden sekä projektipäällikön toimesta.

Menetelmän arviointikriteereille täytyy määrittää mittayksiköt ja arviointiasteikot, jotta yksittäisen CASE-välineen pisteytys olisi mahdollista. Mittayksiköllä tarkoitetaan tässä yhteydessä jotakin mitattavissa olevaa suuretta. Tällainen suure on esimerkiksi automaattisesti UML-kaavioista generoitujen ohjelmakoodin rivien määrä. Kaikille arviointikriteereille mittayksikön määritteleminen ei kuitenkaan ole mahdollista. Tällaisia arviointikriteereitä liittyy esimerkiksi välineen käytettävyyden arviointiin. Mikäli arviointikriteerille ei voida määrittää objektiivisesti arvioitavissa olevaa mittayksikköä, niin kriteerin arvioinnista täytyy sopia erikseen, jolloin voidaan määrittää tarkemmin kriteerin arvioinnin tavat ja siinä huomioitavat kohteet. Arviointiasteikolla tarkoitetaan tässä yhteydessä asteikkoa, jossa välineen arviointikriteerin mittayksikön mukainen tulos on muutettu halutun asteikon mukaiseksi, esimerkiksi ohjelmakoodin generoitujen rivien määrää muunnettuna

asteikolle 0-5. Arviointiasteikkona on käytettävä numeerista asteikkoa, jonka numeroskaala on vapaasti valittavissa. Arviointi- ja valintaprojektissa mukana olevien ohjelmistosuunnittelijoiden tehtävänä on päättää jokaisen arviointikriteerin mittayksikön arvojen vastaavuudesta käytettävään arviointiasteikkoon. Arviointiasteikko on sama kaikille eri alikriteeriluokille. Arviointikriteereiden mittayksiköt ja arviointiasteikot dokumentoidaan arviointisuunnitelmaan. Jokaiselle arviointikriteereiden alikriteeriluokalle voidaan myös haluttaessa määrittää painoarvot, jolloin eri alikriteeriluokkien merkitys arvioinnin pisteytyksessä vaihtelee. Alikriteeriluokalla tarkoitetaan tässä yhteydessä kokonaisuutta, jonka tietyt arviointikriteerit muodostavat. Esimerkiksi kaikki käytettävyyteen liittyvät arviointikriteerit muodostavat käytettävyyden alikriteeriluokan. Menetelmän alikriteeriluokkia ovat:

- dokumentaatio ja avustus,
- käytettävyys,
- kehitys- ja toimintaympäristöt,
- kustannukset ja lisensiointi,
- yleiset ominaisuudet,
- sovellusalustariippumaton malli,
- sovellusalustariippuvainen malli,
- toteutus,
- muut ohjelmistotuotantoprosessin vaiheet sekä
- välinetoimittajaprofiili. (ks. TAULUKOT 8, 9 ja 10)

Mikäli arvioinnin eri alikriteeriluokkia halutaan painottaa eri tavoin - esimerkiksi käytettävyyttä halutaan korostaa enemmän kuin kustannuksia ja lisensiointia - niin jokaiselle alikriteeriluokalle voidaan määrittää painokerroin. Tässä vaiheessa tärkeintä on päättää, että halutaanko eri alikriteeriluokkia painottaa eri tavalla, ja jos halutaan, niin kuinka painotus tehdään, ja mitkä ovat eri alikriteeriluokkien painokertoimet. Päätökset ja mahdolliset käytettävät painokertoimet dokumentoidaan arviointi- ja valintaprojektissa mukana olevien ohjelmistosuunnittelijoiden toimesta arviointisuunnitelmaan.

Arviointikriteereiden mittayksiköiden ja painoarvojen määrittämisen jälkeen pidetään *ensimmäinen johtoryhmän kokous*. Projektin johtoryhmän kokouksien tarkoituksena on avustaa arviointi- ja valintaprosessin hallitsemisessa, ja toisaalta asettaa selkeitä pisteitä, jolloin tiettyjä asioita tulee olla tehtynä. Tämän lisäksi johtoryhmän kokoukset tarjoavat mahdollisuuden tarkastella projektia ja sen mielekkyyttä tietyin väliajoin ja ne myös antavat kaikille projektin eri sidosryhmille tietoa projektin tilanteesta. Ensimmäisessä arviointi- ja valintaprojektin johtoryhmän kokouksessa käsitellään projektisuunnitelma sekä alustava arviointisuunnitelma, johon on dokumentoitu arviointi- ja valintaprosessin laukaisseet tekijä, organisaation korkean tason tavoitteet ja rajoitteet, käytettävät esiarviointikriteerit ja arviointikriteerit, sekä arviointikriteereiden mittayksiköt ja mahdolliset käytettävät alikriteeriluokkien painokertoimet.

Tunnistettaessa esiarvioitavia ehdokasvälineitä tutustutaan sovellusmarkkinoilla oleviin CASE-välineisiin (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995). Tietoa olemassa olevista CASE-välineistä saadaan muun muassa organisaation omilta ohjelmistosuunnittelijoilta, ammattilaisten suorittamista CASE-välineiden arviointiraporteista (esim. Bakker ym. 2000) sekä alan ammattilehdistä. Ehdokasvälineistä kerätään tietoa esiarviointikriteerit huomioiden. Esiarviointikriteereihin tarvittavaa tietoa löytyy esimerkiksi usein välineen omasta dokumentaatiosta ja ohjeistuksesta. Luotettavimpia ja puolueettomampia tiedon lähteitä ovat kuitenkin ammattilaisten tekemät CASE-välineiden arviointiraportit, joista löytyy hyvinkin yksityiskohtaista tietoa eri CASE-välineistä ja niiden ominaisuuksista. Tavoitteena on kerätä esiarviointikriteereiden vaatima tietämys kaikista välineistä, jolloin välineiden esiarviointi pystytään suorittamaan.

Tunnistettujen ehdokasvälineiden esiarvioinnissa tarkoituksena on arvioida kaikki sovellusmarkkinoilta löydetyt CASE-välineet esiarviointikriteereiden mukaisesti. Esiarvioinnin tavoitteena on löytää kahdesta kolmeen organisaatiolle soveltuvinta CASE-välinettä, joita arvioidaan tarkemmin arviointi- ja valintaprosessissa. Esiarvioinnin tarkoituksena on siis rajata nopeasti tarkemmin arvioitavat välineet resurssien käytön minimoimiseksi. Esiarviointi on tarkoitettu määrälliseksi ja nopeaksi arvioinnin välineeksi, jonka tarkoituksena on erotella välineet on/ei-tyyppisten

kysymysten avulla. Esiarvioitavassa välineessä joko on tai ei ole kysymyksen osoittamaa ominaisuutta, piirrettä tai toiminnallisuutta. Välineen esiarviointi suoritetaan vertaamalla esiarviointikriteereiden asettamien kysymysten vastauksia arviointia suorittavan organisaation vaatimuksiin, tavoitteisiin ja rajoitteisiin. Jokainen esiarviointikriteeri vastaa yhtä pistettä ja esiarvioinnin lopussa välineet voidaan asettaa pistemäärien mukaiseen paremmuusjärjestykseen, jolloin voidaan valita tarkemmin arviointiprosessin aikana arvioitavat välineet. Tämän vuoksi esiarviointi on nopea suorittaa ja siihen ei kulu paljon resursseja. Esiarvioinnin nopeus ja helppous tulee oleelliseksi kysymykseksi etenkin silloin, kun arvioitavia välineitä on paljon. Esiarvioinnin tulokset dokumentoidaan esiarviointiraporttiin arviointi- ja valintaprojektissa mukana olevien ohjelmistosuunnittelijoiden toimesta. Taulukossa 4 on kuvattu esimerkki esiarviointiraportin sisällöstä ja rakenteesta.

TAULUKKO 4 Esimerkki esiarviointiraportin sisällöstä ja rakenteesta.

Otsikko	Sisältö
Yhteenveto	Yhteenveto on lyhyt referaatti esiarvioinnista ja sen tuloksista. Yhteenvedosta tulisi ilmetä ainakin: – arvioitujen välineiden nimi, – välineiden tekijät, – menetelmät, joita välineet tukevat, – välineiden käyttöjärjestelmät ja käyttöympäristöt, – välineiden vahvuudet, – välineiden heikkoudet, – välineiden hinnat, – lyhyet sanalliset arviot välineistä sekä – arviointiprosessiin siirrettävien välineiden listaus.
Vaatimusten, tavoitteiden ja rajoitteiden listaus	Organisaation CASE-välineelle asettamat pakolliset vaatimukset, tavoitteet ja rajoitteet tulee dokumentoida selkeästi. Vaatimuksia, tavoitteita ja rajoitteita verrataan esiarviointikriteereiden asettamien kysymysten vastauksiin.
Esiarviointikriteerit	Kuvataan täydennetyt esiarviointikriteerit.
Välineiden esiarviointi	Jokaiselle välineelle suoritetaan esiarviointi, jossa selvitetään esiarviointikriteereiden kysymyksiin vastaukset.

Arviointisuunnitelmaan on esiarviointiprosessin aikana dokumentoitu arviointi- ja valintaprosessin laukaisevat tekijät, organisaation korkean tason tavoitteet ja rajoitteet, täydennetty ja varmennettu arviointikriteeristö sekä mahdollisesti käytettävät painokertoimet. Projektipäällikön on *täydennettävä arviointisuunnitelmaa* vielä kuitenkin siten, että se vastaa myös kysymykseen, kuinka CASE-välineiden arviointi tullaan suorittamaan. Arviointisuunnitelmaan on siis lisättävä ainakin resurssien käyttöä

koskeva suunnitelma, jossa otetaan kantaa siihen kuka arvioi mitäkin, milloin, miten ja kuinka paljon arvioijalla on käytettävissään aikaa arviointiin. Taulukossa 5 on esimerkki arviointisuunnitelman sisällöstä ja rakenteesta.

TAULUKKO 5 Esimerkki arviointisuunnitelman sisällöstä ja rakenteesta.

Otsikko	Sisältö
Arviointi- ja valintaprosessin laukaisevat tekijät	Kuvataan arviointi- ja valintaprosessin laukaisevat tekijät ja niiden syyt.
Korkean tason tavoitteet ja rajoitteet	Kuvataan mahdolliset CASE-välineen valintaan vaikuttavat korkean tason tavoitteet ja rajoitteet.
Arviointikriteerit ja painokertoimet	Selvitetään täydennetyt arviointikriteerit, niiden mittayksiköt ja arviointiasteikot sekä mahdollisesti käytettävät painokertoimet.
Suunnitelma arvioinnin suorittamisesta	Kuvataan prosessi, jonka mukaisesti CASE-välineiden arviointi tullaan suorittamaan. Lisäksi määritellään ja kuvataan välinetestaus.
Resurssien käyttö	Määritellään käytössä olevat resurssit ja niiden käyttö arvioinnissa.

Toisessa arviointi- ja valintaprojektin johtoryhmän kokouksessa käsitellään ja hyväksytään tai hylätään esiarviointiprosessin tuotokset, joita ovat: arviointisuunnitelma, esiarviointiraportti ja projektisuunnitelma. Tämän lisäksi päätetään esiarviointiraportin suosituksen perusteella mahdollisesta CASE-välineen ostamisesta tai kehittämisestä itse, sekä vahvistetaan päätös arviointi- ja valintaprosessiin siirtyvistä välineistä. Mikäli esiarviointiraportin mukaan yhtään organisaation vaatimukset ja tarpeet täyttävää CASE-välinettä ei sovellusmarkkinoilla ole, tehdään arviointi- ja valintaprojektin keskeyttämispäätös, jolloin väline joudutaan joko kehittämään itse tai kehittämään jo olemassa olevaa CASE-välinettä. Jos esiarviointiraportin suosituksen mukaan sovellusmarkkinoilta löytyy organisaation vaatimukset ja tarpeet täyttäviä CASE-välineitä, niin tehdään hankintapäätös, jolloin arviointi- ja valintaprojekti siirtyy arviointi- ja valintaprosessiin.

Arviointi- ja valintaprosessi

Tehtävien osalta määrällisesti suurin vaihe oli esiarviointi. Arviointi- ja valintaprosessi on kuitenkin ajalliselta kestoltaan koko arviointi- ja valintaprojektin pisin. Arviointi- ja valintaprosessin tavoitteena on arvioida arviointikriteerien avulla esiarvioinnissa valittuja CASE-välineitä, sekä valita organisaation tarpeet ja vaatimukset parhaiten

täyttävä CASE-väline. Arviointi- ja valintaprosessi koostuu kuudesta tehtävästä, jotka ovat:

1) CASE-välineiden arvioinnin valmistelu (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995),
2) CASE-välineiden arviointi (vrt. Cook ym. 1997),
3) valintasuosituksen tekeminen (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995),
4) (valintasuosituksen todentaminen oikeaksi),
5) projektin loppuraportin tekeminen sekä
6) kolmannen projektin johtoryhmän kokouksen pitäminen.

CASE-välineiden arvioinnin valmistelu koostuu kahdesta osasta: yksityiskohtaisemman tiedon keräämisestä CASE-välineistä ja arviointisuunnitelman päivittämisestä. Tietoa CASE-välineistä saadaan esiarviointiprosessissa esitettyjen lähteiden lisäksi esimerkiksi tekemällä survey-tutkimuksia ja haastatteluja välinetoimittajille (vrt. Bakker ym. 2000). Objektiivisin tiedon lähde on ammattilaisten tekemät CASE-välineiden arvioinnit, joista löytyy yksityiskohtaista teknistä tietoa välineistä. Arviointisuunnitelmaa tulee täydentää esiarviointiprosessin jälkeen vielä välinetestauksen suunnitelmalla. Lisäksi arviointisuunnitelma päivitetään toisesta johtoryhmän kokouksesta saadun palautteen mukaiseksi. Välinetestaus voi olla esimerkiksi jonkin pienen testaustarkoituksiin soveltuvan sovelluksen toteuttaminen alusta lähtien. Välinetestaus tulee kuitenkin suunnitella huolellisesti, jotta sen avulla saataisiin todennettua kaikki oleelliset tekniset vaatimukset ja niiden taso välineessä. Tällaisia teknisiä vaatimuksia voivat olla esimerkiksi UML-kaavioiden eheyssääntöjen tarkistaminen.

Esiarviointiprosessin tuloksena on suositus kahdesta tai kolmesta CASE-välineestä, jotka arviointi- ja valintaprosessin aikana tulee arvioida tarkemmin ja syvällisemmin. Arviointi- ja valintaprosessin päätavoitteena on näiden edellä mainittujen välineiden hyvyyden arviointi arviointisuunnitelman mukaisesti. *CASE-välineiden arviointi - tehtävä* on sekä määrällinen että laadullinen arvioinnin väline, jonka avulla yritetään selvittää onko arvioinnin kohteena olevassa välineessä arviointikriteerin osoittamaa

ominaisuutta, piirrettä tai toiminnallisuutta, sekä kuinka hyvin väline ominaisuuden, piirteen tai toiminnallisuuden toteuttaa. Ominaisuuksien, piirteiden ja toiminnallisuuksien olemassaolo sekä hyvyys todennetaan tarvittavilta osiltaan välinetestauksen avulla. Välinetestaus voidaan toteuttaa siten, että projektiryhmän jäseniksi valitut ohjelmistosuunnittelijat jaetaan ryhmiin ja kukin ryhmä toteuttaa tietyllä CASE-välineellä testisovelluksen ja arvioi välineen teknisen hyvyyden. Hyvyyden arviointi suoritetaan todentamalla välinetestauksen avulla ominaisuuden, piirteen tai toiminnallisuuden arviointikriteerin mittayksikön mukainen tulos, joka muunnetaan arviointiasteikon mukaiseksi pistemääräksi, esimerkiksi asteikolle 0-5. Mikäli arviointikriteerille ei ole ollut mahdollista määritellä objektiivista mittayksikköä, niin arviointiasteikon mukainen tulos on arvioijien yhteisen mielipiteen mukainen. Kaikkien alikriteeriluokan arviointikriteereiden arvioinnin jälkeen alikriteeriluokalle lasketaan pistemäärä, joka on arviointikriteereiden pistemäärien keskiarvo.

Mikäli alikriteeriluokkien mukaisia tuloksia halutaan painottaa osana lopullista välineen pistemäärää, niin seuraavana vaiheena on painotuksien laskeminen alikriteeriluokille. Viimeisenä vaiheena suoritetaan alikriteeriluokkien pistemäärien yhteen laskeminen. Alikriteeriluokkien pistemäärien summa on CASE-välineen arvioinnista saama lopullinen pistemäärä. CASE-välineen arvioinnista saama pistemäärä lasketaan kaavan 1 mukaisesti.

$$\sum_{k=1}^{10}\left(\left(\left(\sum_{n=1}^{m} a_n\right) \div m\right) \times p\right) \qquad (1)$$

m = alikriteeriluokan arviointikriteerien määrä
a_n = arviointikriteerin arviointiasteikon mukainen pistemäärä
p = mahdollinen käytettävä painokerroin

Jokaisen CASE-välineen arvioinnin tulokset dokumentoidaan arviointiraporttiin, joka voi olla esimerkiksi taulukossa 6 esitetyn kaltainen. Arviointiraporttien sisältöön

vaikuttaa oleellisesti käytettävät arviointikriteerit, joten arviointiraporttien sisältöä tulee muokata niiden mukaisesti.

TAULUKKO 6 Esimerkki arviointiraportin sisällöstä ja rakenteesta (vrt. Bakker ym. 2000).

Otsikko	Sisältö
Yhteenveto	Yhteenveto on lyhyt referaatti arvioinnista ja sen tuloksista. Yhteenvedosta tulisi ilmetä ainakin: – välineen nimi, – välineen tekijä, – ongelma, jonka väline pyrkii ratkaisemaan, – menetelmät, joita väline tukee, – käyttöjärjestelmät ja käyttöympäristöt, – viitteet muiden suorittamiin tai aikaisempiin arviointeihin, – vahvuudet, – heikkoudet, – pistemäärä, – hinta sekä – sanallinen arvio välineestä.
Yleiskatsaus	Yleiskatsauksessa esitellään väline ja siitä tulisi ilmetä ainakin seuraavat asiat: – mihin väline on tarkoitettu, – millainen välineen toimintafilosofia on ja kuinka se pyrkii ongelmat ratkaisemaan, – miten väline kattaa eri ohjelmistotuotantoprosessin vaiheet, – yleiskatsaus välineen tukemiin menetelmiin, – välineen eri moduulit ja niiden kuvaus sekä – laitteisto- ja ohjelmistovaatimukset.
Yleiset arviointikriteerit	Kuvataan kaikki yleisten CASE-välineiden arviointikriteereiden mukaiset asiat ja kuinka ne täyttyvät arvioitavassa välineessä. Jokaisen alikriteeriluokan osalta esitetään lisäksi sen pistemäärä.
Malliperustainen arkkitehtuuri	Kuvataan kaikki malliperustaiseen arkkitehtuuriin liittyvien arviointikriteereiden mukaiset asiat ja kuinka ne täyttyvät arvioitavassa välineessä. Jokaisen alikriteeriluokan osalta esitetään lisäksi sen pistemäärä.
Välinetoimittaja	Kuvataan kaikki välinetoimittajaan liittyvien arviointikriteereiden mukaiset asiat ja kuinka ne täyttyvät arvioitavassa välineessä. Jokaisen alikriteeriluokan osalta esitetään lisäksi sen pistemäärä.

CASE-välineiden arvioinnin jälkeen projektipäällikön *tehtävänä on valintasuosituksen tekeminen* (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995). Valintasuositus perustuu arviointisuunnitelmaan ja arviointiraportteihin. Valintasuosituksessa perustellaan ja selvitetään CASE-välineiden arvioinnin tulokset ja niistä tehtävät johtopäätökset. Valintasuosituksessa tulee viitata selkeästi dokumentteihin, joihin se perustuu.

Valintasuositus voidaan tarvittaessa myös todentaa (verify) käyttämällä konsultaatiota tai suorittamalla suositeltavalle CASE-välineelle laajempi käyttötesti. Konsultaatiota tai laajempaa käyttötestiä tulee kuitenkin tarkoin harkita niiden vaatimien resurssien vuoksi. Lisäksi ne pidentävät arviointi- ja valintaprosessia oleellisesti.

Projektipäällikön vastuulla on *projektin loppuraportin tekeminen*. Projektin loppuraportissa on yhteenveto projektista, sen tuloksista ja resurssien käytöstä. Lisäksi loppuraporttiin on hyvä liittää kommentteja CASE-välineen käyttöönottoprosessia varten sekä ajatuksia arviointi- ja valintaprosessin kehittämiseksi.

Kolmannessa arviointi- ja valintaprojektin johtoryhmän kokouksessa tehdään CASE-välineen valintapäätös, joka perustuu valintasuositukseen. Vaihtoehtoisesti voidaan päättää myös arviointi- ja valintaprosessin jatkamisesta tai keskeyttämisestä kokonaan. Arviointi- ja valintaprosessin jatkamispäätöksen voi aiheuttaa esimerkiksi sovellusmarkkinoille tullut uusi CASE-väline, jota pidetään erityisen lupaavana. Johtoryhmän kokouksessa tulee myös tarkastella valintapäätöstä alkuperäisten tavoitteiden ja vaatimusten valossa. Johtoryhmän kokouksen viimeisenä tehtävänä on projektiorganisaation purkaminen ja projektin päättäminen.

4.3 Arviointikriteeristö

Malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmän arviointikriteeristön viitekehys jakautuu kahteen osaan: esiarviointikriteereihin ja varsinaisiin arviointikriteereihin (vrt. LeBlanc & Korn 1992). Esiarviointikriteereitä käytetään esiarviointiprosessissa ja arviointikriteereitä arviointi- ja valintaprosessissa. Molempia viitekehyksiä on täydennettävä arviointitilanteen ja arvioinnin suorittavan organisaation vaatimusten ja tavoitteiden mukaisesti.

4.3.1 Esiarviointikriteerit

Esiarviointikriteerit ovat muodoltaan määrällisiä (quantitative), eivät laadullisia (qualitative). Tästä johtuen myös CASE-välineiden esiarviointi on määrällistä ja esiarviointikriteerit ovat kysymyslauseen muodossa. Esiarvioinnin tarkoituksena on arvioida esiarviointikriteereiden avulla täyttääkö arvioinnin kohteena oleva väline organisaation vaatimukset, tavoitteet ja rajoitteet. Vastauksia esiarviointikriteereiden asettamiin kysymyksiin verrataan esiarviointiprosessissa määritettyihin organisaation tavoitteisiin, vaatimuksiin ja rajoitteisiin. Mikäli väline ei täytä yhtä tai useampaa esiarviointiraportissa kuvattua pakollista vaatimusta, niin sitä ei voida valita arviointiprosessiin. Taulukossa 7 on kuvattu esiarviointikriteereiden viitekehys yksityiskohtaisesti.

TAULUKKO 7 Esiarviointikriteereiden viitekehys.

Kriteeriluokka	Esiarviointikriteeri
Yleiset CASE-välineiden esiarviointikriteerit	– onko välineelle olemassa dokumentaatiota? – millä eri käyttöjärjestelmillä väline toimii? – mitä muita ohjelmistoja väline vaatii toimiakseen ja täyttääkseen tavoitteet? – tukeeko väline organisaation käyttämiä tietokantoja tai yleisimpiä tietokantoja? – tukeeko väline organisaation käyttämiä sovelluspalvelimia tai yleisimpiä sovelluspalvelimia? – tukeeko väline organisaation käyttämiä komponenttimalleja tai yleisimpiä komponenttimalleja? – tukeeko väline organisaation käyttämiä ohjelmointikieliä tai yleisimpiä ohjelmointikieliä? – kuinka paljon väline maksaa yhtä lisenssiä kohden ja mikä on vähimmäislisenssimäärä? – mitkä ovat välineen vuosittaiset käyttökustannukset/tarvittava lisenssimäärä? – päivitetäänkö välinettä säännöllisin väliajoin? – onko väline modulaarinen ja siten laajennettavissa? – onko mahdollista siirtää tietoa välineen eri versioiden välillä? – tukeeko väline käyttöoikeuksien rajoittamista? – onko välineessä työryhmätukea?
Malliperustaiseen arkkitehtuuriin liittyvät esiarviointikriteerit	– tukeeko väline malliperustaista arkkitehtuuria toimintafilosofialtaan? – tukeeko väline malliperustaisen arkkitehtuurin eri osastandardeja? – tukeeko väline UML-kieltä? – tukeeko väline organisaation käytössä olevaa ohjelmistotuotantomenetelmää? – tukeeko väline staattista ja dynaamista mallintamista? – onko välineen avulla mahdollista generoida ohjelmakoodia? – tukeeko väline mallien muodostamista ohjelmakoodista (reverse engineering)?

(jatkuu)

TAULUKKO 7 (jatkuu)

Kriteeriluokka	Esiarviointikriteeri
Välinetoimittajaan liittyvät esiarviointikriteerit	– tarjoaako välinetoimittaja koulutusta välineelle? – mitkä ovat välineen olemassa olevat versiot?

4.3.2 Arviointikriteerit

Arviointikriteerit ovat esiarviointikriteereistä poiketen enemmän laadullisia kuin määrällisiä. Tästä johtuen myös CASE-välineiden arviointi niiden mukaisesti on ajallisesti pitkäkestoisempaa. Arviointikriteerit muodostavat viitekehyksen, jota tulee täydentää organisaatio- ja tilannekohtaisesti. Viitekehyksessä arviointikriteerit on jaettu kolmeen luokkaan:

1) yleisiin CASE-välineiden arviointikriteereihin,
2) malliperustaiseen arkkitehtuuriin liittyviin arviointikriteereihin sekä
3) välinetoimittajaan liittyviin arviointikriteereihin.

Seuraavissa kappaleissa esitellään kunkin luokan alikriteeriluokat ja kuvataan niiden käyttöä arvioinnin osana.

Yleiset CASE-välineiden arviointikriteerit

Yleiset CASE-välineiden arviointikriteerit ovat kriteereitä, jotka liittyvät yleisesti CASE-välineiden arviointiin riippumatta välineen käyttötarkoituksesta. Yleiset CASE-välineiden arviointikriteerit on tässä tutkielmassa luokiteltu viiteen alikriteeriluokkaan, jotka ovat (ks. TAULUKKO 8):

- dokumentaatio ja käytön tuki,
- käytettävyys,
- kustannukset ja lisensiointi,

- kehitys- ja toimintaympäristöt sekä
- yleiset ominaisuudet.

Dokumentaatio ja käytön tuki -alikriteeriluokka jakaantuu kahteen osaan, jotka ovat dokumentaation laatu (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995) ja käytön tuen laatu. Dokumentaatiolla tarkoitetaan välineen mukana tulevia dokumentteja, joita ovat esimerkiksi asennusopas (installation guide), aloitusopas (getting started guide), käyttöohje (user guide), dokumentaatio välineen ohjelmointirajapinnoista (application interface) sekä ohje välineen käytöstä sovelluskehityksen apuvälineenä (tutorial). Käytön tuella tarkoitetaan välineen ohjetoimintoa (help) ja muita mahdollisia sovelluskehitystä tukevia ohjeistuksia sekä välineen reagointia virhetilanteisiin. Mahdollisissa virhetilanteissa välineen tulisi tarjota tilannekohtaista (context sensitive) tukea ohjelmistokehittäjälle. Dokumentaation ja käytön tuen laatua tulee arvioinnissa painottaa erityisesti, mikäli organisaation ohjelmistosuunnittelijoilla ei ole aikaisempaa käyttökokemusta välineestä ja välineelle ei löydy muutakaan julkaistua ohjeistusta.

Käytettävyys -alikriteeriluokka jakaantuu neljään osaan, jotka ovat: käyttöliittymän opittavuus, tehokkuus ja joustavuus sekä virhetilanteiden käsittely. Käytettävyydellä tarkoitetaan CASE-välinettä käyttävien ohjelmistosuunnittelijoiden toimintatapojen, odotusten ja tavoitteiden yhteensopivuutta välineen tarjoamien ominaisuuksien kanssa. Välineen käytettävyys on usein käyttäjille sama asia kuin koko järjestelmä. Tämän vuoksi käytettävyyden arviointi on tärkeä osa koko välinearviointia ja sitä tulisikin painottaa jokaisessa arviointitilanteessa. (vrt. Nielsen 1993 ja Duk Choi ym. 1998).

Kustannukset ja lisensiointi -alikriteeriluokka muodostuu kahdesta kokonaisuudesta: lisensioinnista ja muista kustannuksista. Lisensiointiin liittyvät muun muassa välineen lisenssimääräiset hankintakustannukset ja muu välinetoimittajan lisensiointipolitiikka. Muut kustannukset tarkoittavat välineen ylläpitoon, koulutukseen ja asennukseen liittyviä kustannuksia. Välineen hinta on usein yksi oleellisimmista tekijöistä harkittaessa CASE-välineen ostamista. Erityisen tärkeää on kuitenkin huomata ja selvittää muut välineen hankkimiseen ja käyttämiseen liittyvät kustannukset kuin pelkkä

lisenssiperusteinen hankintahinta. (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995)

Kehitys- ja toimintaympäristöt -alikriteeriluokka on jaettu kolmeen kokonaisuuteen: kehitysympäristöihin, tuettuihin rajapintoihin muihin ohjelmistotuotannon välineisiin sekä välineen toimintaympäristöön. Kehitys- ja toimintaympäristöillä tarkoitetaan välineen tukea eri käyttöjärjestelmille, ohjelmistoille ja ohjelmistotuotantoprosessin menetelmille sekä tekniikoille. (vrt. Bakker ym. 2000 ja Greefhorst ym. 1998) Kehitys- ja toimintaympäristöjen osuutta arvioinnissa tulee painottaa, mikäli arviointia suorittava organisaatio haluaa mahdollisimman monipuolista ja avointa ratkaisua ohjelmistotuotannon apuvälineeksi.

Yleiset ominaisuudet -alikriteeriluokka on jaettu kolmeen kokonaisuuteen: mukautettavuuteen ja ylläpidettävyyteen, toimivuuteen, turvallisuuteen ja luotettavuuteen sekä työryhmätukeen. Alikriteeriluokan arviointikriteerit ovat kriteereitä, joita ei jokaisessa arviointitilanteessa tarvita ja niiden painottaminen arvioinnissa on tilanne- ja organisaatiokohtaista. (vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995 ja Bakker ym. 2000) Taulukossa 8 on kuvattu alikriteeriluokittain yleiset CASE-välineiden arviointikriteerit.

TAULUKKO 8 Yleiset CASE-välineiden arviointikriteerit.

	Alikriteeriluokat	**Arviointikriteeri**
Dokumentaation ja käytön tuki	Dokumentaation laatu[10]	– kokonaisuus (kuvattu koko järjestelmä) – oikeellisuus – yhdenmukaisuus – ymmärrettävyys – helppokäyttöisyys – onko dokumentaatiota välineen tukemasta menetelmästä?

[10] vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995.

(jatkuu)

TAULUKKO 8 (jatkuu)

Alikriteeriluokat		Arviointikriteeri
	Käytön tuen laatu	– kokonaisuus (käytön tuki kattaa koko järjestelmän) – oikeellisuus – tilannekohtaisuus (context sensitive) – yhdenmukaisuus – ymmärrettävyys – helppokäyttöisyys
Käytettävyys[11]	Käyttöliittymän opittavuus	– tunnettuus (kuinka hyvin välineen käyttäjät pystyvät sitä käyttämään ilman koulutusta) – dialogin yksinkertaisuus ja luonnollisuus – käyttöliittymän yhdenmukaisuus – dialogien ennustettavuus – käyttäjien oman kielen käyttö – tilojen ja toimintojen poistumisen selkeys
	Käyttöliittymän tehokkuus	– oikopolkujen käytön mahdollisuus – dialogisiirtymien sujuvuus (vasteajat) – käyttäjälle toiminnoista annettavan palautteen laatu
	Käyttöliittymän joustavuus	– eri hallintavälineiden käytön mahdollisuus dialogien yhteydessä – välineen tuki monen samanaikaisen tehtävän suorittamiselle – välineen käyttöliittymän muokattavuus
	Virhetilanteiden käsittely	– virhetilanteiden ilmoituksien selkeys – poistumistavan hallittavuus – virhetilanteiden vältettävyys – virhetilanteiden peruuttamisen mahdollisuus (askel kerrallaan, viimeisin tilanne vai jokin tietty kokonaisuus) – välineen tarjoama apu virhetilanteisiin liittyen
Kustannukset ja lisensiointi[12]	Lisensiointi	– hankintakustannukset/tarvittava lisenssimäärä – vuosittaiset käyttökustannukset/tarvittava lisenssimäärä – välinetoimittajan lisensiointipolitiikka
	Muut kustannukset	– asennuskustannukset – ylläpitokustannukset – tarvittavat laitteiston/ohjelmiston päivityskustannukset – koulutuskustannukset
Kehitys- ja toiminta-ympäristöt	Kehitysympäristöt[13]	– käyttöjärjestelmät – käyttöliittymät – laitteistovaatimukset – muut vaadittavat ohjelmistot

[11] vrt. Duk Choi ym. 1998 ja Nielsen 1993.

[12] vrt. Bakker ym. 2000 ja Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995.

[13] vrt. Bakker ym. 2000.

(jatkuu)

TAULUKKO 8 (jatkuu)

Alikriteeriluokat		Arviointikriteeri
	Tuetut rajapinnat muihin ohjelmistotuotannon välineisiin[14]	– projektinhallinta – versionhallinta – konfiguraationhallinta – tekstinkäsittely – käyttöliittymän suunnittelu – liiketoiminnan mallintaminen – tuetut tiedostoformaatit
	Toimintaympäristö	– tuetut tietokannat – tuetut sovelluspalvelimet – tuki ohjelmistoarkkitehtuureille – tuki suunnittelumalleille (design pattern) – tuki sovelluskehyksille (framework) – tuetut ohjelmointikielet – tuetut komponenttimallit
Yleiset ominaisuudet	Toimivuus, turvallisuus ja luotettavuus[15]	– varmuuskopiointimahdollisuus – automaattinen tallennusprosessi – tallentaako väline vakavan virhetilanteen ilmettyä työn ja ilmoittaa siitä käyttäjälle? – tuki käyttöoikeuksien rajoittamiselle – välineen vikasietoisuus ja vakavien virhetilanteiden määrä (käyttöaika/vakavat virhetilanteet)
	Työryhmätuki[16]	– yhtäaikaisten käyttäjien määrä – tietojen eheys – työryhmätuen tehokkuus – vasteajat

Malliperustaiseen arkkitehtuuriin liittyvät arviointikriteerit

Malliperustainen arkkitehtuuri asettaa CASE-välineille erityisiä vaatimuksia. Tämän vuoksi on luontevaa, että arviointikriteereiden viitekehyksessä arkkitehtuurin asettamat vaatimukset on eroteltu omaksi kokonaisuudekseen. Malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointilanteessa arkkitehtuuriin liittyviä alikriteeriluokkia on suositeltavaa painottaa muita luokkia enemmän, koska ne ovat välineen toiminnallisuuden ja tarkoituksen kannalta oleellisimpia. Malliperustaiseen

[14] vrt. Bakker ym. 2000 ja Greefhorst ym. 1998.

[15] vrt. Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995.

[16] vrt. Bakker ym. 2000.

arkkitehtuuriin liittyvät arviointikriteerit on tässä tutkielmassa luokiteltu neljään alikriteeriluokkaan, jotka ovat (ks. TAULUKKO 9):

- sovellusalustariippumaton malli,
- sovellusalustariippuvainen malli,
- toteutus ja
- muut ohjelmistotuotantoprosessin vaiheet.

Sovellusalustariippumaton malli -alikriteeriluokkaan kuuluvat arviointikriteerit liittyvät järjestelmien mallintamiseen sovellusalustariippumattomasti sekä CASE-välineen tukeen eri arkkitehtuurin standardeille. Arvioitavan CASE-välineen tulee tukea kaikkia viitekehyksen arviointikriteereiden määrittämiä ominaisuuksia, piirteitä ja toiminnallisuuksia tukeakseen malliperustaista arkkitehtuuria. Ominaisuuksien, piirteiden ja toiminnallisuuksien hyvyyden todentaminen suoritetaan välinetestauksen avulla.

Sovellusalustariippuvainen malli -alikriteeriluokan muodostavat arviointikriteerit liittyvät mallien transformaatioon ja tiedon sekä mallien säilymiseen eheänä ohjelmistotuotantoprosessin aikana. Mallien säilyminen eheänä riippumatta abstraktiotasosta ja näkökulmasta (viewpoint) on erittäin tärkeää arkkitehtuurin perusfilosofian kannalta. Mallien eheyden toteuttaminen CASE-välineissä on erittäin vaikeaa. Mallien eheys ohjelmistotuotantoprosessin aikana voidaan todentaa välinetestauksen avulla.

Toteutukseen liittyvät arviointikriteerit voidaan jakaa kahteen kokonaisuuteen: generointiin ja ohjelmointiin. Alikriteeriluokkien arviointikriteerit liittyvät ohjelmakoodin automaattiseen generointiin ja sen laatuun eri teknologioille sekä ohjelmakoodin kääntäjien (compiler) yleisimpiin ominaisuuksiin. Erityisesti ohjelmakoodin automaattisen generoinnin arvioiminen on tärkeää, koska malliperustaisen arkkitehtuurin yksi perusajatuksista on ohjelmistotuotannon automatisointi.

Muut ohjelmistotuotantoprosessin vaiheet -alikriteeriluokka muodostuu kolmesta osasta: testauksesta, ylläpidosta sekä yhteensopivuudesta ja integroinnista. Näistä osista etenkin testaus saatetaan toteuttaa organisaatioissa hyvinkin erilaisilla tavoilla, menetelmillä ja välineillä. CASE-välineissä tulisi kuitenkin olla tuki muun muassa automaattiselle testitapauksien generoinnille ja ajamiselle. Arvioinnin painotuksen kannalta tärkeimmät osat ovat ylläpito sekä yhteensopivuus ja integrointi. Molemmat näistä alikriteeriluokista liittyvät malliperustaisen arkkitehtuurin perusajatukseen ohjelmistotuotannon automatisoinnista. CASE-välineiden tulisi etenkin tukea UML-kaavioiden muodostamista ohjelmakoodista ja mallien tuontia ja vientiä XMI-kielen avulla. XMI-kielen tuki välineissä takaa mallien siirrettävyyden heterogeenisissakin kehitysympäristöissä.

TAULUKKO 9 Malliperustaiseen arkkitehtuuriin liittyvät arviointikriteerit.

Alikriteeriluokat		Arviointikriteeri
Sovellusalusta-riippumaton malli	Tuki notaatioille, menetelmille ja standardeille	– välineen tuki UML-standardille (versio) – ohjelmistotuotantomenetelmätuki (esitutkimus, määrittely, suunnittelu) – staattisen ja dynaamisen mallintamisen kaavioiden tuki – UML-notaation mukaiset mallien eheyssääntöjen toteutus – tietokannan mallintamisen tuki – OCL-kielen tuki (versio) – tuki MOF-tietovarastoille – mallien yhdenmukaisuus ohjelmistotuotantoprosessin aikana – tietojen reaaliaikainen päivittäminen eri näkymästä toiseen
Sovellusalusta-riippuvainen malli	Mallien transformaatio ja eheys	– tuki eri UML-profiileille – UML-mallien transformaatio profiilien mukaisesti – UML-profiilien muodostamismahdollisuudet – UML-profiilien tuonnin ja viennin mahdollisuudet (import/export) – mallien yhdenmukaisuus ohjelmistotuotantoprosessin aikana
Toteutus	Generointi	– tuki ohjelmointikielille – tuki sovellusalustoille – tuki komponenttimalleille – käyttöliittymän dialogien generointi – tietokantamääritysten generointi – generoinnin laatu (ei mitään, otsikot, otsikot + tilasiirtymät, otsikot + tilasiirtymät + ohjelmakoodi metodeihin) – dokumentaation automaattinen generointi
	Ohjelmointi	– ohjelmakoodin kääntäminen – tuki ohjelmakoodin syntaksille ja sen täydentämiselle – tuki ohjelmakoodin virheenkorjaukselle (debugging)

(jatkuu)

TAULUKKO 9 (jatkuu)

	Alikriteeriluokat	Arviointikriteeri
Muut ohjelmistotuotantoprosessin vaiheet	Testaus	– tuki automaattiselle testitapauksien tunnistamiselle ja generoinnille vaatimusmäärittelyn tuloksista – tuki testitapauksien automaattiselle ajamiselle – tuki ajonaikaisille analyyseille (suorituskyky) – tuki sovelluksen luotettavuuden analysoinnille
	Ylläpito	– tuki tietorakenteiden ja tietoelementtien muuntamiselle UML-kaavioiksi (data reverse engineering) – tuki operaatioiden/prosessien muuntamiselle UML-kaavioiksi (process reverse engineering) – tuki ohjelmakoodin muuntamiselle toiselle ohjelmointikielelle (source code translation) – uml-mallien ja ohjelmakoodin yhtenäisyys (round trip engineering) – tiedon eheys
	Yhteensopivuus ja integrointi	– mahdollisuus käyttää muiden välineiden tietovarastoja – pääsy välineen metadataan ja mahdollisuus sen muokkaamisen – tuki XMI:n käytölle mallien tuontiin (import) ja vientiin (export

Välinetoimittajaan liittyvät arviointikriteerit

Kolmantena arviointikriteereiden luokkana ovat välinetoimittajaan liittyvät arviointikriteerit, joiden avulla muodostetaan välinetoimittajaprofiili. CASE-välineiden sovittaminen organisaation tarpeisiin sopivaksi on usein pitkäkestoinen prosessi, jossa tarvitaan myös välinetoimittajan sitoutumista. CASE-välineen ostopäätös vaatii usein myös pitkän ajan sitoutumista käytettäviin teknologioihin. Näiden syiden vuoksi on tärkeää arvioida myös itse välineen lisäksi välinetoimittajaa.

Välinetoimittajaan liittyvät arviointikriteerit muodostuvat viidestä osasta: toimittajaprofiilista, tuoteprofiilista, teknisen tuen laadusta, koulutuksen laadusta sekä välineen versioinnista ja muista tuotteista. Välinetoimittajaprofiili mahdollistaa välinetoimittajan liiketoiminnan kannattavuuden ja tulevaisuuden, CASE-välineen kypsyyden sekä välinetoimittajan teknisen tuen ja koulutusmahdollisuuksien arvioinnin. CASE-välineen ostopäätöksen kannalta on oleellisen tärkeää, että välinetoimittaja on sitoutunut välineen kehittämiseen edelleen tulevaisuudessa. (vrt. Bakker ym. 2000, Churh & Matthews 1995 ja Information Technology - Guideline for Evaluation and

Selection of CASE-tools 1995) Taulukossa 10 on kuvattu välinetoimittajaan liittyvät arviointikriteerit yksityiskohtaisesti.

TAULUKKO 10 Välinetoimittajaan liittyvät arviointikriteerit.

	Alikriteeriluokat	Arviointikriteeri
Välinetoimittajaprofiili[17]	Toimittajaprofiili	– koko (liikevaihto, työntekijöiden määrä) – markkinoilla oloaika – markkinaosuus – yhteistyökumppaneiden olemassaolo – maantieteellinen sijainti – taloudellinen tilanne – mahdolliset sertifikaatit – tulevaisuuden suunnitelmat
	Tuoteprofiili	– tuotteen ikä – versionumero – myytyjen lisenssien määrä – tuettujen versioiden määrä – jakelukanavien määrä
	Teknisen tuen laatu	– dokumentoitu prosessi tukitilanteisiin – tukipyyntöihin reagoinnin nopeus – tukipyyntöjen dokumentointi – puhelintuki, lähituki tai on-site tuki
	Koulutuksen laatu	– koulutuspaketti välineelle – koulutuksen ajallinen saatavuus – koulutuksen hyödyllisyys
	Välineen versiointi ja muut tuotteet	– olemassa olevat versiot – välinetoimittajan muut tuotteet – välineeseen saatavat mahdolliset kolmannen osapuolen lisäkkeet

4.4 Yhteenveto

Tämän luvun tuloksena on CASE-välineiden arviointi- ja valintamenetelmä malliperustaiselle arkkitehtuurille. Menetelmä määrittelee sekä arviointi- ja valintaprosessia ohjaavan vaihejaon että arviointikriteeristön, jonka avulla CASE-välineiden arviointi on mahdollista. Menetelmän vaihejako ja yleiset osat arviointikriteeristöstä on johdettu osittain aikaisemmista tutkimuksista. Arviointi- ja

[17] vrt. Bakker ym. 2000, Church & Matthews 1995 ja Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995.

valintaprosessin vaihejakoon on kuitenkin aikaisemmista tutkimuksista poiketen pyritty tässä tutkielmassa tuomaan mukaan projektimaisia piirteitä suoraan prosessiin. Lisäksi menetelmän arviointikriteeristössä on otettu aikaisempia CASE-välineiden arviointi- ja valintamenetelmiä paremmin huomioon käytettävyyden arviointi. Menetelmän arviointikriteeristön malliperustaiseen arkkitehtuuriin liittyvät arviointikriteerit on muodostettu tutkielman luvun kaksi pohjalta.

Menetelmän vaihejako jakaantuu esiarviointiprosessiin ja arviointi- ja valintaprosessiin. Esiarviointiprosessin tarkoituksena on muodostaa arviointi- ja valintaprosessille projektiorganisaatio sekä tutustua sovellusmarkkinoilla oleviin CASE-välineisiin, ja esiarviointikriteereiden avulla löytää kahdesta kolmeen organisaation pakolliset vaatimukset täyttävää CASE-välinettä. Arviointiprosessissa arvioidaan esiarviointiprosessissa valittuja välineitä ja valitaan niistä organisaatiolle sopivin.

Menetelmän arviointikriteeristö jakautuu kahteen osakokonaisuuteen: esiarviointiprosessissa käytettäviin esiarviointikriteereihin sekä arviointiprosessissa käytettäviin arviointikriteereihin. Esiarviointikriteerit ovat kysymyslausemuotoisia kriteereitä, joiden avulla voidaan vähäisellä resurssien käytöllä selvittää täyttääkö CASE-väline organisaation tärkeimpiä vaatimuksia. Menetelmän arviointikriteerit on jaettu kolmeen luokkaan: yleisiin CASE-välineiden arviointikriteereihin, malliperustaiseen arkkitehtuuriin liittyviin arviointikriteereihin sekä välinetoimittajaan liittyviin arviointikriteereihin. Menetelmän arviointikriteeristö on viitekehys, jota tulee täydentää organisaatio- ja tilannekohtaiset tekijät huomioon ottaen.

5 YHTEENVETO

Tämän tutkielman tulokset jakautuvat kahteen osaan. Tutkielman keskeisimpänä kontribuutiona on luvussa neljä määritelty ja kuvattu CASE-välineiden arviointi- ja valintamenetelmä malliperustaiselle arkkitehtuurille. Toisena keskeisenä kontribuutiona voidaan pitää malliperustaisen arkkitehtuurin, ohjelmistotuotantoprosessin vaiheiden ja ohjelmistotuotantomenetelmien suhteiden määrittelemistä. Näiden lisäksi tutkielman luvussa kaksi on jäsennetty malliperustaista arkkitehtuuria, joka on vielä varsin kypsymätön, sekä määritelty arkkitehtuurin keskeiset käsitteet.

Malliperustainen arkkitehtuuri on koko ohjelmistotuotantoprosessia tukeva ohjelmointikieli-, ohjelmistotoimittaja- ja sovellusalustariippumaton ohjelmistoarkkitehtuuri, joka muodostaa oman arkkitehtuurillisen tyylinsä kehittää ja suunnitella järjestelmiä. Malliperustainen arkkitehtuuri muodostaa siis mallien arkkitehtuurin, jonka tavoitteena on ohjelmistotuotannon automatisointi, sekä niiden kustannusten minimointi, joita seuraa sovellusalustojen, käyttöjärjestelmien ja ohjelmointikielten vaihtuvuudesta muutamien vuosien välein. Malliperustaisen arkkitehtuurin avainteknologia on Unified Modeling Language -mallinnuskieli, jonka avulla järjestelmien liiketoimintalogiikka mallinnetaan. UML-kieltä käytetään malliperustaisessa arkkitehtuurissa kahteen mallintamisen kohteeseen: sovellusalustariippumattoman mallin sekä sovellusalustariippuvaisen mallin muodostamiseen. Sovellusalustariippumaton malli kuvaa kehitettävän järjestelmän toiminnallisuuden ja käyttäytymisen riippumattomasti eri toteutusteknologioista. Sovellusalustariippuvainen malli kuvaa järjestelmän toiminnallisuuden ja käyttäytymisen jonkin toteutusteknologian mukaisesti. Sovellusalustariippuvainen malli muodostetaan sovellusalustariippumattomasta mallista mallien transformaation avulla. Mallien transformaation mahdollistavat malliperustaista arkkitehtuuria tukevat CASE-välineet sekä UML-profiilit, jotka määrittävät transformaatiossa käytettävän säännöstön.

Malliperustaisen arkkitehtuurin määrityksen mukainen ohjelmistotuotantoprosessi on yksinkertainen. Aluksi muodostetaan sovellusalustariippumaton malli, josta muodostetaan mallien transformaation avulla yksi tai useampia

sovellusalustariippuvaisia malleja. Tämän jälkeen generoidaan ohjelmakoodi halutuille sovellusalustoille. Ohjelmistotuotantoprosessi on kuitenkin luonteeltaan erittäin monimutkainen kokonaisuus, jolloin syntyy tarve jakaa prosessi selkeisiin vaiheisiin. Tällaisia vaihejakomalleja onkin esitetty jo 1970-luvulta alkaen, jolloin esiteltiin ohjelmistotuotannossa käsitteeksi muodostunut vesiputousmalli. Malliperustainen arkkitehtuuri ei vaiheista eikä ohjeista ohjelmistotuotantoprosessia millään tavalla. Arkkitehtuurin määrityksessäkään ei oteta kantaa arkkitehtuurin, yleisimpien ohjelmistotuotantoprosessin vaihejakomallien ja ohjelmistotuotannossa käytettyjen menetelmien suhteeseen. Malliperustaisen arkkitehtuurin määritys onkin tämän perusteella vielä osittain keskeneräinen. Tutkielman toisena keskeisenä kontribuutiona on kuvattu malliperustaisen arkkitehtuurin suhde perinteiseen vesiputousmalliin ja Unified Software Development Process -menetelmään (ks. kappale 2.7). Malliperustaisen arkkitehtuurin mukaisessa ohjelmistotuotantoprosessissa esitutkimus (vaatimusmäärittely), määrittely ja suunnittelu kuuluvat sovellusalustariippumattoman mallin muodostamisprosessiin. Näiltä osin malliperustainen arkkitehtuuri tuleekin muuttamaan perinteistä käsitystä, jossa jo suunnitteluvaiheessa on otettu kantaa toteutusteknologiaan.

Malliperustaisen arkkitehtuurin keskeisimmät ongelmat liittyvät CASE-välineisiin ja UML-profiileihin. Arkkitehtuurin suurimpina haasteina ja ongelmina ovat UML-kielen semanttinen epätarkkuus ja formalismin puute sekä mallien ja ohjelmakoodin pitäminen yhdenmukaisina riippumatta abstraktiotasosta ja näkökulmasta järjestelmää kehitettäessä. Malliperustaisen arkkitehtuurin laaja-alaisen soveltamisen kannalta erityisen tärkeään osaan nousevat arkkitehtuurin toimintafilosofiaa ja eri osastandardeja tukevat CASE-välineet. Mikäli CASE-välinevalmistajat eivät ota malliperustaisen arkkitehtuurin paradigmaa ja eri teknologioita mukaan välineisiinsä, niin arkkitehtuuri voi jäädä ainoastaan määritykseksi. Tutkielman tekohetkellä näyttää kuitenkin siltä, että suuret ohjelmistoalan toimijat, kuten IBM, Borland, Rationale Software ja Hewlett-Packard, olisivat sitoutuneet malliperustaiseen arkkitehtuuriin jo varsin voimakkaasti.

Malliperustainen arkkitehtuuri siis tarvitsee sitä tukevia CASE-välineitä, jotta ohjelmistotuotannon automatisoiminen olisi mahdollista. Kaikkia

ohjelmistotuotantoprosessin vaatimia tehtäviä ei kuitenkaan voida suorittaa yhdellä välineellä. Etenkin malliperustaisen arkkitehtuurin tapauksessa tarvitaan niin sanottuja CASE-ympäristöjä arkkitehtuurin laajuuden vuoksi. CASE-välineitä on sovellusmarkkinoilla useita satoja. Ongelmana onkin valita malliperustaisen arkkitehtuurin vaatiman CASE-ympäristön kannalta organisaatiolle sopivimmat välineet.

CASE-välineiden arviointia ja valintaa on tutkittu paljon. Tämän tutkielman luvussa kolme on muodostettu kattava kirjallisuuskatsaus olemassa oleviin ohjelmisto- ja CASE-välineiden arviointi- ja valintamenetelmiin. Arviointi- ja valintamenetelmä muodostuu vaihejaosta ja arviointikriteeristöstä. Vaihejaon tarkoituksena on vaiheistaa ja ohjeistaa arviointi- ja valintaprosessia. Arviointikriteeristö mahdollistaa eri välineiden yhdenmukaisen arvioinnin, jonka lopputuloksena voidaan arviointia suorittavalle organisaatiolle valita sopivin CASE-väline.

CASE-välineiden arviointi- ja valintamenetelmää, joka ottaisi huomioon malliperustaisen arkkitehtuurin asettamat vaatimukset CASE-välineille, ei ole aikaisemmin määritelty. Tämän tutkielman keskeisimpänä tuloksena on CASE-välineiden arviointi- ja valintamenetelmä malliperustaiselle arkkitehtuurille. Menetelmä määrittelee sekä vaihejaon että arviointikriteeristön. Osia menetelmän vaihejaosta ja arviointikriteeristöstä on johdettu aikaisemmista CASE-välineiden arviointia ja valintaa koskevista tutkimuksista. Menetelmässä on kuitenkin otettu aiempia CASE-välineiden arviointi- ja valintamenetelmiä paremmin huomioon vaihejaon osalta arviointi- ja valintaprosessin projektimaisuus sekä arviointikriteeristön osalta käytettävyyden arviointi. Aikaisemmista tutkimuksista poiketen arviointikriteeristö on jaettu esiarviointikriteereihin ja arviointikriteereihin. Lisäksi on kehitetty arviointikriteerit, joiden avulla voidaan arvioida nimenomaan malliperustaista arkkitehtuuria tukevia CASE-välineitä.

Malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointi- ja valintamenetelmän vaihejako muodostuu esiarviointiprosessista ja arviointi- ja valintaprosessista. Esiarviointiprosessin tärkeimpänä tavoitteena on suorittaa

ehdokasvälineiden valinta. Arviointi- ja valintaprosessissa arvioidaan ehdokasvälineitä ja valitaan niistä organisaatiolle soveltuvin väline. Menetelmän arviointikriteeristö on tilanne- ja organisaatiokohtaisesti täydennettävä viitekehys, joka jakautuu esiarviointikriteereihin ja arviointikriteereihin. Esiarviointikriteerit ovat kysymyslausemuotoisia kriteereitä, joiden tarkoituksena on mahdollistaa ehdokasvälineiden valinta sovellusmarkkinoilla olevien välineiden joukosta. Arviointikriteerit muodostuvat kolmesta kriteereiden luokasta: yleisistä CASE-välineiden arviointikriteereistä, malliperustaiseen arkkitehtuuriin liittyvistä arviointikriteereistä sekä välinetoimittajaan liittyvistä arviointikriteereistä.

Tutkielman tuloksena kehitetty CASE-välineiden arviointi- ja valintamenetelmä on tarkoitettu ensisijaisesti malliperustaista arkkitehtuuria tukevien CASE-välineiden arviointiin ja valintaan. Menetelmää voidaan kuitenkin soveltuvin osin käyttää muunkinlaisten CASE-välineiden ja ohjelmistovälineiden arviointiin ja valintaan. Erityisesti menetelmän vaihejako on sovellettavissa kaikkien ohjelmistovälineiden arviointiin ja valintaan. Arviointikriteeristön osalta CASE-välineiden arvioinnin kannalta yleiskäyttöisiä osia ovat yleiset CASE-välineiden arviointikriteerit ja välinetoimittajaan liittyvät arviointikriteerit. Muiden ohjelmistovälineiden arviointiin soveltuvia kriteereitä ovat esimerkiksi välinetoimittajaan, käytettävyyteen, kustannuksiin ja lisensiointiin sekä dokumentaatioon ja avustukseen liittyvät arviointikriteerit.

Tässä tutkielmassa kehitetty CASE-välineiden arviointi- ja valintamenetelmä malliperustaiselle arkkitehtuurille on teoreettinen. Menetelmän sovellettavuuden ja arviointikriteeristön kattavuuden osalta olisikin tärkeää tehdä empiiristä tutkimusta, jossa menetelmää sovellettaisiin käytännössä CASE-välineiden arviointi- ja valintatilanteessa.

LÄHTEET

Abd-Allah, A., Boehm, B., Clark, B. & Gacek, C. 1995. On the Definition of Software Architecture. USC Center for Software Engineering, USC-CSE-95-TR-500.

Apperley, M. & Duncan, A. 1994. Human-Computer Interface Design in the Software Lifecycle. Teoksessa S. L. Huong, J. Hughes & M. Purvis (toim.) Proceedings of the 5th Annual Conference on Software Engineering Education, Dunedin, November 21-24. Los Alamitos: IEEE Computer Society, 60-64.

Bakker, H., Budd, M., George, G., Ormrod, T.A. & Stevenson, M. 2000. Ovum Evaluates: CASE Products. Ovum Ltd, 2000.

Balcer, M. & Mellor, S. J. 2002. Executable UML: A Foundation for Model-Driven Architecture. New York: Addison Wesley.

Brodsky, S. A., Doney, G. C. & Grose, T. J. 2002. Mastering XMI: Java Programming with XMI, XML and UML. New York: John Wiley & Sons.

Booch, G., Jacobson, I. & Rumbaugh, J. 1999a. The Unified Modeling Language User Guide. New York: Addison Wesley.

Booch, G., Jacobson, I. & Rumbaugh, J. 1999b. The Unified Software Development Process. New York: Addison Wesley.

Brassard, M. & Cardinal, M. 2002. Addressing Problems with Model Driven Architecture [online]. Darien: INT Media Group [viitattu 1.8.2002]. Saatavilla www-muodossa <http://softwaredev.earthweb.com/sdtech/article/0,,12065_1121801,00.html>

Castain, E. H. 2001. MDA – Wells Fargo. Needham: Object Management Group. Saatavilla myös pdf-muodossa < http://www.omg.org/mda_files/MDA%20briefing%20Castain.pdf>[viitattu 2.8.2002]

Chang, D., Mellor, D., Poole, J. & Tolbert, D. 2002. Common Warehouse Metamodel. New York: John Wiley & Sons.

Church, T. & Matthews, P. 1995. An Evaluation of Object-Oriented CASE Tools: The Newbridge Experience. Teoksessa H. Muller (toim.) Proceedings of the Seventh International Workshop on Computer-Aided Software Engineering, Totonto, July 10-14. Los Alamitos: IEEE Computer Society, 4-9.

Common Warehouse Metamodel (CWM) Specification 2001, Object Management Group.

Cooke, B., Lam, W., Powell, A., Vickers, A & Williams, E. 1997. Evaluating Tools to Support Component Based Software Engineering. Teoksessa E. Nahouraii (toim.) Proceedings of the Fifth International Symposium on Assessment of Software Tools and Technologies, Pittsburgh, June 3-5. Los Alamitos: IEEE Computer Society, 80-89.

Crawley, S. 2000. Generating Software from Models. Teoksessa P. Croll & J. Gray (toim.) Proceedings of the International Conference on Software Methods and Tools, Wollongong, November 6-10. Los Alamitos: IEEE Computer Society, 233 -239.

DSouza, D. 2001. Model-Driven Architecture and Integration – Opportunities and Challenges. Needham: Object Management Group. Saatavilla myös pdf-muodossa <ftp://ftp.omg.org/pub/docs/ab/01-03-02.pdf>[viitattu 10.7.2002].

Duk Choi, M., Griffin, D., Mehandjiska, D., Page, D. & Phillips, C. 1998. Teoksessa M. K. Purvis, S. J. S. Cranefield & S. G. MacDonell (toim.) Proceedings of the 1998 International Conference on Software Engineering: Education & Practice, Dunedin, January 26-29. Los Alamitos: IEEE Computer Society, 134-141.

Garlan, D. & Shaw, M. 1994. An Introduction to Software Architecture. CMU Software Engineering Institute, CMU/SEI-94-TR-21.

Greefhorst, D., Matthijs, M. & Maijers, R. 1998. Evaluating OO-CASE tools: OO Research Meets Practise. Teoksessa S. Demeyer & J. Bosch (toim.) Proceedings of the Object-Oriented Technology: ECOOP'98 Workshop Reader, Brussels, July 20-24. Berlin: Springer-Verlag, 486-488.

Hansen, K. M., Heide Damm, C., Thomsen, M. & Tyrsted, M. 2000. Tool Integration: Experiences and Issues in Using XMI and Component Technology. Teoksessa J. M. Jézéquel (toim.) Proceedings of the 33rd International Conference on Technology of Object-Oriented Languages and Systems, Mont Saint-Michel, June 5-8. Los Alamitos: IEEE Computer Society, 94-107.

Hazra, T. K. 2002. MDA Brings Standards-Based Modeling to EAI Teams. Application Development Trends 9(5), 48-52.

Information Technology - Open Distributed Processing – Reference Model: Architecture 1996. ISO/IEC 10746-3, International Organization for Standardization / International Electrotechnical Commission.

Information Technology - Open Distributed Processing – Reference Model: Foundations 1996. ISO/IEC 10746-2, International Organization for Standardization / International Electrotechnical Commission.

Information Technology - Guideline for Evaluation and Selection of CASE-tools 1995. ISO/IEC 14102, International Organization for Standardization / International Electrotechnical Commission.

Java™ Metadata Interface(JMI) Specification 2002. JSR 040, Java Community Process.

Kath, O. 2002. Impacts of Changes in Enterprise Software Construction for Telecommunications: Model Driven Architecture – Assesments of Relevant Technologies. European Institute for Research and Strategic Studies in Telecommunications, Project 1149, EDIN 0288-1149.

Keienburg, F. & Rausch, A. 2001. Using XML/XMI for Tool Supported Evolution of UML Models. Teoksessa R. H. Sprague (toim.) Proceedings of the 34th Annual Hawaii International Conference on System Sciences, Hilton Waikoloa Village Island of Hawaii, January 6-9. Los Alamitos: IEEE Computer Society, 3945-3954.

Kitchenham, B., Law, D. & Linkman S. 1997. DESMET: A Methodology for Evaluating Software Engineering Methods and Tools. Computing & Control Engineering Journal 8(3), 120-126.

Kobryn, C. 1999. UML 2001: A Standardization Odyssey. Communications of the ACM 42(10), 29-37.

Kobryn, C. 2000. Modeling Components and Frameworks with UML. Communications of the ACM 43(10), 31-38.

Kobryn, C. 2002. Will UML 2.0 Be Agile or Ackward? Communications of the ACM 45(1), 107-110.

Koch, T., Uhl, A. & Weise, D. 2002. Model Driven Architecture. Freiburg: Interactive Objects Software GmbH Inc. Saatavilla myös pdf-muodossa < http://cgi.omg.org/docs/ormsc/02-01-04.pdf>[viitattu 16.7.2002]

Korn, W. M. & LeBlanc, L. A. 1992. A Structured Approach to the Evaluation and Selection of CASE Tools. Teoksessa H. Berghel, E. Deaton G. Hedrick & D.

Roach (toim.) Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing: Technological Challenges of the 1990's, Kansas City, March 1-3. New York: ACM Press, 1064-1069.

Koskinen, M. 2000. Process Metamodelling: Conceptual Foundations and Application. University of Jyväskylä, Jyväskylä Studies in Computing, 7.

Marttiin, P. 1998. Customisable Process Modelling Support and Tools for Design Environment. University of Jyväskylä, Jyväskylä Studies in Computer Science, Economics and Statistics, 43.

McClure, C. 1989. CASE is Software Automation. New Jersey: Prentice-Hall.

Meta Object Facility (MOF) Specification 2002. Version 1.4, Object Management Group.

Model Driven Architecture (MDA) 2001. Ormsc/2001-07-01, Object Management Group.

Model Driven Architecture – A Technical Perspective 2001. Ab/2001-02-04, Object Management Group.

Mosley, V. 1992. How to Assess Tools Efficiently and Quantitatively. IEEE Software 9(3), 29-32.

Object Management Group. 2002. Catalog of OMG Domain Specifications [online]. Saatavilla www-muodossa <http://www.omg.org/technology/documents/domain_spec_catalog.htm>[viitattu 2.8.2002]

OMG Unified Modeling Language Specification 2001. Version 1.4, Object Management Group.

OMG XML Metadata Interchange (XMI) Specification 2002. Version 1.2, Object Management Group.

Perry, D. E. & Wolf, A. L. 1992. Foundations for the Study of Software Architecture. ACM SIGSOFT Software Engineering Notes 17(4), 40-52.

Prather, B. 1993. Critical Failure Points of CASE Tool Evaluation and Selection. Teoksessa H. Lee, T. Reid & S. Jazarbek (toim.) Proceeding of the Sixth International Workshop on Computer-Aided Software Engineering, Singapore, July 19-23. Los Alamitos: IEEE Computer Society, 60-63.

Schach, S. R. 1990. Software Engineering. Boston: Aksen Associates Incorporated Publishers.

Siegel, J. 2001. Developing in OMG's Model-Driven Architecture. Needham: Object Management Group. Saatavilla myös pdf-muodossa < ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf>[viitattu 15.5.2002].

Soley, R. 2000. Model Driven Architecture. Needham: Object Management Group. Saatavilla myös pdf-muodossa <ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf>[viitattu 15.5.2002].

Royce, W. W. 1970. Managing the Development of Large Software Systems: Concepts and Techniques. Teoksessa R. H. Thaye (toim.) Proceedings of the IEEE Wescon, Los Angeles, August 25-28. Los Alamitos: IEEE Computer Society, 1-9.

Rowley, J. E. 1993. Selection and Evaluation of Software. ASLIB Proceedings 45(3), 77-81.

UML 2.0 OCL RFP 2001. ad/00-09-03, Object Management Group.

UML™ Profile and Interchange Models for Enterprise Application Integration (EAI) Specification 2002. ptc/02-02-02, Object Management Group.

UML™ Profile for CORBA™ Specification 2002. Version 1.0, Object Management Group.

UML Profile for EJB 2001. Version 1.0, Java Community Process.

UML Profile for Enterprise Distributed Object Computing Specification 2002. ptc/02-02-05, Object Management Group.

UML Profile for Schedulability, Performance and Time 1999. ad/99-03-13, Object Management Group.