Tampereen yliopisto

Joonas Oksanen

DESIGN STRUCTURE -MATRIISIN KÄYTTÖ TUOTTEEN ELINKAAREN AIKANA

Kone- ja tuotantotekniikan laitos
Kandidaatintyö
Huhtikuu 2019

TIIVISTELMÄ

Joonas Oksanen: Design Structure -matriisin käyttö tuotteen elinkaaren aikana
Kandidaatintyö
Tampereen yliopisto
Tekniikan kandidaatti
Huhtikuu 2019

Tässä kandidaatintyössä selvitettiin design structure -matriisin teoriaa ja käyttämistä tuotekehitysprosessin aikana referoimalla Arie Karnielin ja Yoram Reichin (2011) kirjasta “Managing the Dynamics of New Product Development Processes-A New Product Lifecycle Management Paradigm” design structure -matriisin käyttöä sisältävät alueet. DSM on riippuvuusmatriisi, jonka avulla voidaan kuvata prosessin eri vaiheiden välisiä riippuvuuksia. Kun riippuvuudet tiedetään, matriisin avulla on mahdollista tarkastella työjärjestyksen muutoksen vaikutuksia. Uudelleenjärjestämisen avulla voidaan etsiä prosessille optimaalinen työjärjestys.

Työn tarkoituksena oli oppia, miten matriisia käytetään ja miten sitä voidaan hyödyntää tuotteen elinkaaren aikana. Työn tuloksena opittiin matriisin käytön perusteet ja siihen liittyvä teoria. Tuotteen elinkaaren aikaisesta käytöstä tärkeimmäksi todettiin uuden tuotteen valmistus- ja kehitysprosessin suunnitteleminen, mutta kirjan keskittyessä tuotteen kehitysprosessiin muista elinkaaren aikaisista käyttökohteista ei saatu selkeää kuvaa.

Avainsanat: design structure -matriisi, tuotekehitysprosessi, elinkaaren hallinta

ABSTRACT

Joonas Oksanen: The use of Design Structure Matrix over the product's lifecycle
Bachelor's thesis
Tampere University
Bachelor of Science in Technology
April 2019

In this bachelor's thesis's study theory and usage of Design Structure Matrix in a product development process and during products lifecycle by summarizing Arie Karniel and Yoram Reich's (2011) book "Managing the Dynamics of New Product Development Processes-A New Product Lifecycle Management Paradigm". DSM is a tool to describe dependencies within process' activities. When dependencies are known its possible to review effect of chance in order of the activities within process. Reordering the matrix is a tool to find optimal order for process' activities.

Aim of this work was to summarize parts of the book that included information about the matrix's and to learn how to use them and how to exploit them within products lifecycle. As result of the thesis the usage and theory of Desing Structure Matrix was learned. About the use during products lifecycle it was found that the most important use for matrix was during planning of the products development and production process. However, as the focus of the book was products development process no clear picture was obtained how to take advantage of the matrix during other parts of products lifecycle.

Keywords: Design Structure Matrix, product development process, product lifecycle management

ALKUSANAT

Tämä työ on osa Tampereen yliopiston tekniikan kandidaatin tutkintoa. Työn avulla haluttiin tutustua tarkemmin tuotteen elinkaaren hallinnan työkaluihin, ja lopulta aiheeksi valikoitui design structure -matriisit ja niiden käyttäminen tuotteen elinkaaren aikana.

Tässä yhteydessä halutaan kiittää työn ohjaajaa Antti Pulkkista avusta työn tekemisessä.

Tampereella, 10.4.2019

Joonas Oksanen

SISÄLLYSLUETTELO

LYHENTEET JA MERKINNÄT

C-prosessi	current process
CDR	critical design review
DB	design block
DPM	design process matriisi
DSM	design structure matriisi
LDI-levy	laser direct imaging levy
PDR	preliminary design review
PLM	product lifecycle management, elinkaaren hallinta
RT-prosessi	real time process

P, p	todennäköisyys

1. JOHDANTO

Tuotteen elinkaaren ajalta saatava tieto on tämän päivän teollisuudessa merkittävä informaation lähde. Kun tuotteen elinkaaren aikana havaitaan ongelmia, voidaan saadun informaation perusteella tehdä tarvittavia muutoksia tuotteen valmistusprosessiin. Muutosten avulla pyritään varmistamaan, että vastaavia ongelmia ei esiintyisi enää uusien tuotteiden elinkaaren aikana. Tiedon oikeanlainen käyttäminen voi kuitenkin olla haastavaa, ja tiedon keräämisen ja käytön helpottamiseksi onkin luotu paljon erilaisia työkaluja. Niiden avulla varmistetaan tiedon oikeellisuus sekä nopea hyödyntäminen.

Yksi tuotteen kehitys- ja tuotantoprosessien suunnittelun ja kehittämisen työkaluista on design structure matriisi (DSM). Matriisin avulla voidaan tarkastella prosessin vaiheiden välisiä riippuvuuksia ja selvittää, miten työvaiheiden järjestyksen muuttaminen vaikuttaa prosessin kulkuun. Työssä referoidaan Arie Karnielin ja Yoram Reichin vuonna 2011 julkaistua kirjaa “Managing the Dynamics of New Product Development Processes-A New Product Lifecycle Management Paradigm”. Tarkoituksena on selvittää ja referoida, mitä Karniel ja Reich kertovat design structure matriiseista sekä niiden käyttämisestä ja hyödyntämisestä koko tuotteen elinkaaren aikana.

Työn rakenne ja järjestys seuraa referoitavan kirjan rakennetta. Alkuun kerrotaan lyhyesti, mitä kirjassa sanotaan tuotteen elinkaaresta ja sen hallinnasta, minkä jälkeen siirrytään design structure matriisin määrittelyyn ja teoriaan eri käyttötavoista ja vaadittavista tiedoista. Tämän jälkeen siirrytään DSM-pohjaisen prosessin käsittelemiseen, ja miten DSM muutetaan käyttökelpoiseksi prosessiksi. Ennen yhteenvetoa esitetään vielä kirjan esimerkkitutkimuksesta ne osat, joissa design structure matriiseja on todellisessa tilanteessa käytetty. Lopuksi kootaan yhteen kirjan tärkein sisältö DSM:in liittyen ja tehdään johtopäätökset.

2. DESIGN STRUCTURE -MATRIISI

2.1 Tuotteen elinkaari ja elinkaaren hallinta

Tuotteen elinkaari käsittää yleisesti kaikki tuotteen kehitys- ja käyttöprosessien vaiheet aina ensimmäisestä tuoteideasta tuotteen käytöstä poistoon asti. Kirjoittajat toteavat, että product lifecycle management (PLM) eli tuotteen elinkaaren hallinta sisältää kokoelman erilaisia työkaluja, metodeja ja toimintatapoja, joiden avulla yritysten ja organisaatioiden on helpompi selvitä nykyajan suunnitteluhaasteista. Haasteet koskettavat erityisesti uusien tuotteiden kehitysprosesseja. (Stark 2005, Sääksvuori & Immonen 2008, Karniel & Reich 2011 mukaan) Uutta tuotetta suunniteltaessa ei ole vielä olemassa olevaa tietoa kyseisen tuotteen elinkaaren myöhemmiltä vaiheilta. Elinkaaren hallinnan työkalut vaikuttavat koko tuotteen elinkaaren ajan. Karniel ja Reich (2011) huomauttavat, että tuotteen käytöstä poiston sijaan suositeltavampi tuotteen elinkaaren pääte on kierrätys.

Karniel ja Reich (2011) painottavat, että elinkaaren hallinnan työkalujen arvo nojaa kykyyn toimittaa tarkkaa tietoa nopeasti, jotta tiedon jakaminen ja käyttäminen yrityksen ja muiden mahdollisten prosessissa mukana olevien organisaatioiden kesken on mahdollista. He toteavat nopeasti saatavilla olevan tiedon mahdollistavan nopean reagoinnin ilmi tulleisiin ongelmiin, joita tuotteen elinkaaren aikana on havaittu. Nopea reagointi mahdollistaa myös innovaatioiden synnyn. Jos tuotteen jonkin elinkaaren vaiheen aikana havaitaan ongelma, se voidaan ottaa huomioon uusien tuotteiden valmistusprosessissa. Voidaan siis kehittää esimerkiksi uusi valmistusmenetelmä tai rakenteellinen ratkaisu, joka estää samojen ongelmien syntymisen uudemman sukupolven tuotteissa.

Elinkaaren hallinta kehittyi kirjoittajien mukaan tuotetiedon hallinnan järjestelmistä, jotka keskittyivät tuotetiedon eri muotojen hallintaan. Nykyään mikä tahansa tiedosto, joka sisältää tietoa, voidaan säilöä, ja sitä voidaan muokata sekä hallita. Kirjoittajat kuitenkin huomauttavat, että vastaava kyky oli aikanaan merkittävä läpimurto. Vähitellen todettiin, että kyseisten järjestelmien arvo perustui koko tuotteen elinkaaren hallitsemiseen (Sääksvuori & Immonen 2008, Karniel & Reich 2011 mukaan). Lisäksi arvoon vaikuttaa elinkaaren hallinnan eri työkalujen yhteentoimivuus (Subrahmanian et al. 2005, Karniel & Reich 2011 mukaan) eli esimerkiksi, kuinka hyvin eri työkaluista saatua tietoa voidaan käyttää jossain toisessa työkalussa.

Elinkaaren hallintaan liittyy jatkuvasti kehittyvistä järjestelmistä huolimatta aina sama ongelma. Karniel ja Reich (2011) kyseenalaistavat, että käyttävätkö elinkaaren tai prosessin hallinnan järjestelmät uusinta tuotetietoa. Tämän peruskysymyksen taustaksi he toteavat, että tiedon olemassa oleminen ei aina tarkoita sen kunnollista käyttämistä. Esimerkiksi jos tuotteessa havaitaan ongelma, mutta ei tiedetä, miten se ratkaistaan tai korjataan, niin tietoa ongelmasta ei välttämättä pystytä heti käyttämään. Saumaton muuttuvan tuotetiedon käyttäminen prosessin suunnittelussa ja toteutuksessa on kuitenkin tärkeää, kun pyritään selviämään varsinkin uuden tuotteen kehitysprosessissa vastaan tulevista haasteista (Karniel & Reich 2011).

2.2 Määritelmä ja luokittelut

Karniel ja Reich (2011) määrittelevät design structure -matriisin neliömatriisiksi, jonka avulla kuvataan eri elementtien välisiä riippuvuuksia. Matriisi kertoo jokaisen vaiheen sisään menevät syötteet ja ulos tulevan tuotteen. Toimintojen syötteet nähdään matriisin riveiltä ja tuotteet sarakkeilta. DSM antaa keinot, joilla tunnistaa prosessissa tapahtuvat monisuuntaiset informaatiovirrat ja toimintavaiheet, jotka tapahtuvat sarjassa, rinnan tai toistuvasti. Toimintojen välisten suhteiden vuoksi seuraavan vaiheen tuloksella voi olla suora vaikutus sitä edeltäneeseen vaiheeseen, jolloin edellinen toiminto on suoritettava uudestaan. (Karniel & Reich 2011) Esimerkki DSM:sta on nähtävillä kuvasta 2.1. Kuvan matriisissa toiminnot on esitetty riveillä ja sarakkeilla ja kahden toiminnon välinen suhde merkitty numerolla 1. Matriisin diagonaali on tummennettu, koska toiminta ei voi olla riippuvainen itsestään. Poikkeuksena diagonaalin käytössä on toiminnon itseiteraatio, johon palataan työn luvuissa 2.7 ja 4. Diagonaalin alapuolelle jäävät merkinnät esittävät toisiaan seuraavia aktiviteetteja, ja diagonaalin yläpuolella olevat merkinnät kertovat palautteesta,

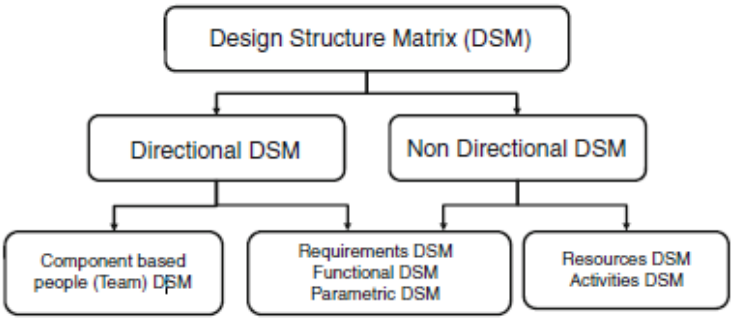
jonka seurauksena vaihe on iteroitava eli toistettava uudestaan sen vaiheen jälkeen, mille riippuvuussuhde on merkitty.

	a	b	c	d	e	f	g	h	i	j	k	l	m
a		1		1									
b							1						
c					1	1							
d						1		1					
e		1									1		
f							1						
g													
h	1												
i			1					1					
j		1	1										
k			1				1		1				
l				1							1		
m	1								1	1			

Kuva 2.1: Design Structure Matriisi (Karniel & Reich 2011, s. 38)

Karniel ja Reichin (2011) mukaan DSM:t on mahdollista luokitella matriisissa käytetyn merkintätavan tai matriisin käyttöluokan mukaan. Perinteinen tapa kuitenkin luokittelee DSM:t neljään pääkategoriaan: komponenttipohjainen DSM, organisaatio- tai ryhmäpohjainen DSM, toimintaperusteinen DSM ja parametriperusteinen DSM. Perinteisessä luokittelussa komponentti- ja organisaatiopohjaiset DSM:eja pidetään staattisina ja olemassa olevia elementtejä kuvaavina, kun taas toiminta- ja parametriperusteisten DSM:ien ajatellaan olevan aikaan perustuvia, koska niissä olevien toimintojen järjestys viittaa tuotteen tai tiedon virtaukseen. (Browning 2001,Karniel & Reich, 2011 mukaan)

Muitakin luokittelumalleja kerrotaan kirjassa. Yhdessä esimerkissä DSM:t on luokiteltu matriisissa olevien toimintojen suhteiden mukaisesti sen mukaan, ovatko niiden väliset suhteet suoria vai epäsuoria. Tämä luokittelutapa erittelee useampia DSM-tyyppejä ja perinteiseen luokitteluun verrattuna tarjoaa enemmän tuotteisiin tai projekteihin liittyviä näkökohtia. Tällaisia DSM:eja ovat funktionaaliset, resurssi- sekä vaatimus-DSM:t. (Karniel & Reich 2011) Edellä esitelty luokittelutapa ja sen sisältämät kategoriat on esitetty kuvassa 2.2.



Kuva 2.2: Esimerkki Design Structure Matriisin luokittelusta (Karniel & Reich, 2011, s.41)

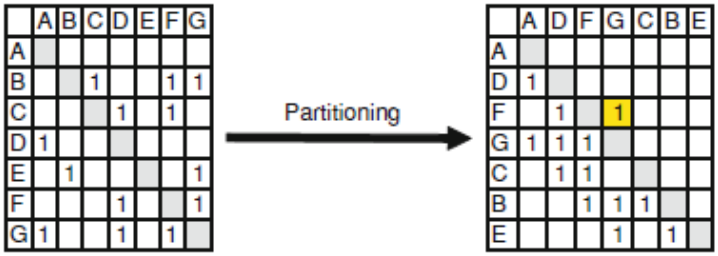
Kuvasta nähdään, miten DSM:t on jaettu suoriin ja epäsuoriin. Esitetyn luokittelun etuna on, että joidenkin tuotetietojen väliset suhteet voidaan esittää sekä suorina että epäsuorina. Kuvasta nähdään vaatimus, funktionaalinen ja parametrinen DSM sisältävät tällaista tietoa. Tämän tyyppistä luokittelua voidaan käyttää apuna, kun määritetään minkä tyyppinen algoritmi sopisi DSM:ään pohjautuvaan suunnitteluun. (Karniel & Reich, 2011)

Karniel ja Reich (2011) kertovat, että DSM-kirjallisuudessa on esitetty kaksi tapaa, miten DSM:a käytetään todellisuudessa. Nämä käyttökohteet ovat olemassa olevan prosessin mallintaminen DSM:n avulla sekä DSM:n käyttäminen prosessissa esiintyvien toimintojen välisten vuorovaikutusten mallintamiseen. Olemassa olevan prosessin mallintamisessa parannetaan prosessia suunnittelualgoritmien avulla. (Karniel & Reich, 2011) Eri algoritmeja on esitelty tämän työn luvussa 3.2. Toimenpiteiden jälkeen uudelleenjärjestelty DSM muutetaan takaisin prosessiksi. Kirjoittajat kertovat, että olemassa olevaa prosessia mallinnettaessa on tyypillisesti löydettävissä reitti kaikkien prosessin toimintojen läpi, mutta reittiä ei ole järjestelmällisesti varmistettu. Sen sijaan toisessa Karnielin ja Reichin esittelemässä käyttötavassa eli vuorovaikutusten mallintami-

sessa vastaavaa reittiä ei heidän mukaansa ole. Vuorovaikutuksia mallinnettaessa DSM:a manipuloidaan, minkä jälkeen DSM muutetaan jälleen takaisin prosessiksi. DSM:n käyttämistä varten tarvitaan siis olemassa oleva prosessi tai vähintään prosessisuunnitelma, jotta prosessia voidaan parantaa matriisin avulla.

2.3 Suunnittelualgoritmit

Suunnittelualgoritmit liittyvät DSM:n uudelleenjärjestämiseen. Kun prosessin sisältämät toiminnot ja niiden väliset riippuvuudet ovat selvillä, voidaan suunnittelualgoritmien avulla selvittää, mikä olisi optimaalisin toimintojen suoritusjärjestys. DSM:iin pohjautuvia suunnittelualgoritmeja on kirjoittajien mukaan kolmea eri päätyyppiä: partitioning, clustering ja sequencing. Partitioning eli jakaminen toteutetaan siten, että matriisin rivit ja sarakkeet järjestetään uudelleen ja tavoitteena on saada uusi järjestys, jossa ei ole palautemerkkejä. Teoreettisessa ideaalitilanteessa palautemerkintöjä ei tulisi lainkaan eli prosessin vaiheitta ei tarvitsisi iteroida. (Karniel & Reich, 2011) Tätä algoritmia käyttäen tehty matriisin uudelleenjärjestäminen on esitetty kuvassa 2.3. Kuvasta huomataan, että riveillä ja sarakkeilla esiintyvien toimintojen (A, B, C jne.) järjestys on muuttunut jakamisen seurauksena ja yhtä merkintää lukuun ottamatta kaikki merkinnät ovat diagonaalin alapuolella. Vain toiminnon F ja G välinen riippuvuus jää diagonaalin yläpuolelle, mikä tarkoittaa sitä, että toiminto F on riippuvainen sen jälkeen tulevasta toiminnosta G. Tämän seurauksena toiminto F on toistettava uudestaan toiminnon G jälkeen. Jakaminen on yleisesti käytetty tapa uudelleenjärjestää DSM sekä saavuttaa prosessi, jossa on mahdollisimman vähän toistoja (Karniel & Reich, 2011).



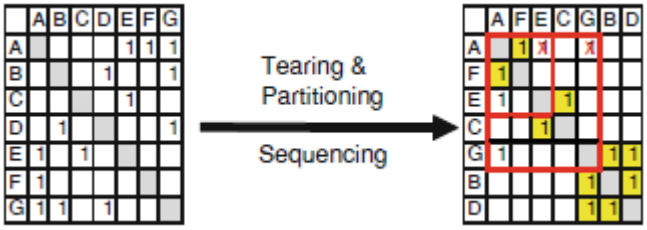
Kuva 2.3: Partitioning.-algoritmin mukainen DSM:n järjestäminen (Karniel & Reich, 2011, s.42)

Clustering eli klusterointialgoritmissa tunnistetaan toisiinsa merkittävästi yhteydessä olevat komponentit, kun kyseessä on komponentti- tai ryhmäpohjainen DSM (Browning 2001, Karniel & Reich 2011 mukaan). Yksinkertainen klusterointi-esimerkki on esitetty kuvassa 2.4, mistä huomataan, että toisiinsa riippuvaiset toiminnot on järjestetty matriisin riveille ja sarakkeille lähelle toisiaan, jolloin iteroinnit voidaan suorittaa mahdollisimman nopeasti.

Kuva 2.4: Klusterointi-algoritmin mukainen DSM:n järjestäminen (Karniel & Reich 2011, s. 43)

Karnielin ja Reichin (2011) mukaan kolmatta algoritmia sequencingia eli sekvensointia voi määritellä algoritmeiksi, joita käytetään sekä palautemerkintöjen minimoimiseen että toisiinsa kytkettyjen toimintojen järjestämiseksi samaan sykliin, jolloin palautelinkit olisivat lähellä diagonaalia. Kuvan 2.5 matriisissa toimintojen väliset suhteet ovat molempiin suuntiin riippuvaisia toisistaan, jolloin esimerkiksi partitioning-algoritmi ei muuttaisi matriisin järjestystä. Kuvan esimerkissä matriisista on manuaalisesti poistettu yhteys toiminnosta C toimintoon E sekä yhteys G:stä A:han,

jonka jälkeen partitioning-algoritmin käyttäminen mahdollista. Sekvensoinnin avulla on mahdollista päästä samaan lopputulokseen. (Karniel & Reich 2011)



Kuva 2.5: Sekvensoinnin mukainen DSM:n järjestäminen (Karniel & Reich 2011, s. 43)

Karniel ja Reich esittelevät myös algoritmin, jossa yhdistyvät päätyypeistä klusterointi ja sekvensointi. Tätä optimointialgoritmiksi kutsuttua työkalua voidaan käyttää laskentatehtävien uudelleenjärjestämiseen, kun kyseessä on rajoitettu käänteistekniikkaa käyttävä prosessi. Kirjoittajien mukaan laskentatehtäviltä vaaditaan kyky ratkaista eri tasojen väliset geometriset rajoitukset. Tehtävien uudelleenjärjestelyn tavoitteena on kokonaislaskennan parantaminen. Muiden edelle esitettyjen algoritmien tavoin myös optimointialgoritmin avulla pyritään löytämään prosessin tehtäville optimaalinen järjestys. (Karniel & Reich 2011) Optimointialgoritmi on esitelty tarkemmin luvussa 2.4.

Uudelleenjärjestäviä algoritmeja tarvitaan, koska varsinkaan suunnitteluvaiheessa olevasta uudesta tuotteesta ei tiedetä tarkkoja määritelmiä, sillä niitä ei ole vielä tehty. DSM ei ole siis staattinen, vaan muuttuu tuotetiedon karttuessa. Näin ollen tuotetiedon muuttuessa myös DSM muuttuu, jonka seurauksena myös DSM:iin pohjautuvan prosessin tulisi muuttua. (Karniel & Reich 2011)

2.4 Optimointialgoritmin käyttö

Optimointia tarvitsevaksi ongelmaksi määriteltiin muutettu DSM, joka sisältää sekä suoria että epäsuoria linkkejä toimintojen välillä. Epäsuorissa linkeissä informaatio kulkee toimintojen välillä, mutta toimintojen suoritusjärjestyksellä ei ole merkitystä. (Karniel et al. 2005, Karniel & Reich 2011 mukaan) Muutos viittaa kirjoittajien mukaan siihen, että uudelleenjärjestämisen jälkeen suorista linkeistä voi tulla palautelinkkejä. He lisäävät, että epäsuorat linkit voivat ikään kuin muuttaa suuntaansa ja olla aina eteenpäin meneviä linkkejä. Jos toimintosilmukan sisällä yksi linkki voi vaihtaa suuntaansa, silmukka hajoaa ja silmukan toiminnot voidaan järjestää ilman palautelinkkejä (Karniel & Reich 2011). Optimoinialgoritmi yhdistää aiemmin esiteltyjä algoritmeja, ja algoritmin käytön kerrotaan sisältävän seuraavat vaiheet:

1. Matriisin jakaminen alkujärjestykseen ja toimintasilmukoiden löytäminen.
2. Tarkastetaan jokaisesta toimintasilmukasta sisältääkö se epäsuoria linkkejä. Jos silmukka sisältää epäsuoran linkin, muuta väliaikaista suuntaa silmukan hajottamiseksi ja siirry takaisin vaiheeseen yksi. Jäljelle jäävät silmukat muodostavat optimointivaihetta varten tarvittavat klusterit eli rykelmät.
3. Optimointialgoritmin käyttö suurten silmukkajoukkojen hajottamiseen lisäämällä pieniä joukkoja tai toimintoja epäsuorilla linkeillä sekä matriisin uudelleenjärjestämisellä. Vaiheen tuloksena syntyy optimaalisia klustereita.

Klusteri eli rykelmä osoittaa joukon toimintoja, jotka ovat samankaltaisia ja näin ollen suunniteltu toteutettavaksi rinnakkain (Karniel & Reich 2011). Toimintasilmukka saattaa kertoa, että silmukan toiminnot tulisi toteuttaa yhdessä, koska ne ovat riippuvaisia toisistaan. Tästä syystä vaiheessa kaksi tunnistetut silmukat ovat ehdokkaita klusteroinnille. Kolmannen vaiheen optimointialgoritmista kirjoittajat kertovat, että sillä etsitään parempia klusterointituloksia minimoimalla kustannusfunktion arvo (Karniel et al. 2005, Karniel & Reich 2011 mukaan). Kirjoittajat esitellevät yhden vaihtoehdon kustannusfunktiosta, jota käytetään myös kirjan tutkimusesimerkissä. Tutkimusesimerkkiä käsitellään työn luvussa 4. Kustannusfunktiolle annettu kaava on

$$\left[\sum_{c} N_c \left(F \sum_{ic > jc} A_{ic,jc} + C \sum_{ic < jc} A_{ic,jc} \right)\right] + F \sum_{id > jd} A_{id,jd} D_{id,jd}^{q} + C \sum_{id < jd} A_{id,jd} D_{id,jd}^{q} + \sum_{id} A_{id,id} D_{id,jd}^{q}, \quad (1)$$

missä N_c on klusterin koko, *A* elementtien välinen linkki, *D* rangaistusfunktio, *i* ja *j* elementtejä/toimintoja ja kirjaimet *F* ja *C* vakioita. Kaavan 1 kustannusfunktiossa elementtejä *i* ja *j* ilmaistaan kahdella tavalla: samassa klusterissa olevien elementtien merkintöinä ovat ic ja jc, ja eri klustereissa olevilla elementeillä id ja jd. *A* kuvaa elementtien välistä linkkiä, ja sille osoitetaan ensisijainen arvo. *A*:n alaindeksit kertovat, onko kyseessä samassa vai eri klusterissa olevat elementit. Funktio *D* toimii linkin *A* rangaistuskertoimena, jos linkki on eri klustereissa olevien elementtien välillä, ja se vastaa elementtien *i* ja *j* välistä etäisyyttä. Funktion *D* arvo saadaan laskettua kaavalla

$$D_{id,jd} = abs(i - j) + 1, \quad (2)$$

missä i ja j ovat samat elementit kuin kustannusfunktiossa. Kaavassa 1 esiintyvä funktion *D* potenssi *q* on pienimmästä vaaditusta klusterin koosta johdettu parametri. Myös kaavan 1 vakioilla ja klusterin koolla kerrotaan elementtien välisiä linkkejä. Vakio *F* on eteenpäin menoa kuvaava vakio (kirjassa Forward constant), jota voidaan käyttää, kun $A_{ij} > 0$ ja $i > j$. Vakio *C* on suljetun silmukan vakio (Closed loop), jota voidaan käyttää, kun $A_{ij} > 0$ ja $i < j$. Vakiot ovat aina suurempia kuin nolla.

Karniel ja Reich (2011) kertovat, että vaihe 1 saattaa johtaa tulokseen mikä sisältää elementtejä, jotka kuuluvat useampaan klusteriin. Jos elementillä *id* on syötteitä useammassa klusterissa, sen itseisarvo $A_{id,id}$ kerrotaan funktiolla D_{id}^{q}, missä D_{id} on kaikkien klusterien kokojen summa, joka lasketaan kaavalla

$$D_{id} = \sum_k N_k, \quad (3)$$

missä *k* kulkee ensimmäisestä klusterista, johon *id* kuuluu, viimeiseen asti, ja N_k on klusterin koko.

Optimointialgoritmilla etsitään parempaa klusterointia uudelleenjärjestämällä matriisia ja luomalla erilaisia klustereita. Klusteroinnin kandidaatti luodaan seuraavilla muutoksilla (Reich & Fenves 1992, Karniel & Reich 2011 mukaan):

1. Siirrä elementti klusterista toiseen.
2. Poista elementti klusterista ja tee siitä uusi klusteri.
3. Yhdistä kaksi klusteria toisiinsa.
4. Jaa klusteri kahteen erilliseen klusteriin.
5. Muuta kahden elementin järjestys klusterin sisällä.
6. Muuta klustereiden järjestystä.

Kirjoittajien mukaan klustereiden määrää tai niiden kokoa ei ole ennalta määritelty, ja alkuperäinen arvaus on jakamisella saatava tulos. Klustereiden ja elementtien valinta on satunnainen.

2.5 Vaadittavat tiedot ja niiden käyttäminen

DSM:n rakentamista ja käyttämistä varten tarvittavien tietojen kerääminen voidaan tehdä kyselyiden ja haastatteluiden avulla (Eppinger et al. 1994; Whitfield et al. 2002; Browning and Eppinger 2002, Karniel & Reich 2011 mukaan). Eri kyselyihin tai haastatteluihin osallistuneilla henkilöillä voi kuitenkin olla toisistaan poikkeavat mielipiteet suunnitteluun liittyvien toimintojen vuorovaikutuksista ja riippuvuuksista. Tiedonhankinta niiden avulla on kuitenkin resurssitehokasta. (Karniel & Reich 2011) Osallistujien eriävien näkökulmien johdosta paremmaksi tiedonkeruutavaksi on todettu ryhmäkeskustelu (Danilovic 1999; Danilovic and Browning 2007; Karniel & Reich,

2011 mukaan) Ryhmäkeskustelun avulla eri näkökulmat tulevat samalla kertaa esille, ja on mahdollista päästä yhteisymmärrykseen käsiteltävänä olevasta asiasta/tiedosta.

Matriisin rakentamista ja päivittämistä varten tarvitaan tietoa käsiteltävän prosessin vaiheiden välisistä riippuvuuksista, prosessin numeerisista arvoista sekä todennäköisyyksistä. Binaaririippuvuuksien tiedot ovat kirjoittajien mukaan helppo määritellä tai olettaa oikeiksi, vaikka ryhmän jäsenillä olisi erimielisyyksiä. Numeeristen arvojen arviointi on vaikeampaa, ja yleisesti niiden kohdalla käytetään riippuvuuksien asteikkoa: korkea, keskiverto, pieni sekä nolla (eli ei riippuvuutta. Todennäköisyyksien arvioinnissa on vaikea onnistua siten, että saataisiin kehitettyä hyödyllisiä malleja. (Karniel & Reich 2011)

Riippuvuustietojen saamiseksi vaadittavat toiminnallisuudet löytyvät useimmista tuotteen elinkaaren hallinnan työkaluista. Kirjoittajat kertovat, että PLM-työkaluilla on kyky hallita toimintojen välisiä suhteita ja niiden avulla on mahdollista lisätä manuaalisesti uusia suhde arvoja, joiden avulla DSM:n luominen ja päivittäminen on mahdollista. Automaattinen tiedonhankinta on mahdollista esimerkiksi CAD-ohjelmalla, jonka avulla luodusta mallista on mahdollista ottaa tiedot tuotteen geometrisista riippuvuuksista, mutta tämä voi johtaa liian suuren matriisin syntymiseen. Liian suureksi kasvanutta matriisia on mahdoton käyttää ja ymmärtää. (Karniel & Reich 2011) Kirjoittajien mielestä manuaalinen lähestymistapa on houkuttelevampi, koska sen avulla on mahdollista osoittaa suunnitteluprosessin kaikki suhdetyypit.

Karniel ja Reich (2011) esittävät ehdotuksen prosessista, jonka mukaisesti DSM tiedot voidaan kerätä. He alustavat ehdotustaan toteamalla, että DSM-prosessin ongelmana on muuttuvien tietojen käytön kohdistaminen koko suunnitteluprosessiin. Käyttökelpoisen tuotetiedon karttuessa yksityiskohtaisuuden taso muuttuu ja mukautuu, minkä johdosta vaadittua yksityiskohtaisuutta ei voida määrittää etukäteen. Tietojen keräämiseksi koko suunnitteluprosessin ajalta tarvitaan työkalu, joka voi olla esimerkiksi jokin PLM:n työkaluista, jollaa muita tuotteen riippuvuuksia hallittaisiin (Karniel & Reich 2011). Tiedonkeruuprosessi sisältää seuraavat vaiheet:

1. Tuotekohtaisten suunnittelutoimintojen tunnistaminen
2. Riippuvuuksien tunnistaminen ja niiden arvon osoittaminen
3. Vaikutus arvojen summaaminen jokaiselle solulle, jotta numeerinen DSM voidaan luoda
4. Skaalattujen todennäköisyyksien osoittaminen, jotta todennäköisyyksiin pohjautuva DSM voidaan luoda

Yksinkertaistetussa mallissa jokaiselle tuotteen komponentille on osoitettu oma toiminto (Karniel & Reich 2011). Kirjoittajat toteavat, että yksinkertaistamisen avulla siirtoprosessin automatisointi helpottuu, mutta tuotteen rakenteen ja osoitettujen toimintojen välillä voi silti esiintyä monimutkaisia riippuvuussuhteita. Riippuvuudet toimintojen välillä määritetään niiden vaiheiden välisiksi riippuvuuksiksi, joilla on vaikutusta prosessi suunnitteluvaiheessa tapahtuvaan toimintojen järjestämiseen. Näin ollen suunnitteluparametrien vaikutukset ja niiden suunnat tulee määrittää. Kirjassa annetaan lyhyt ja yksinkertainen esimerkki, joka sisältää kaikki tiedonkeruuprosessin vaiheet. Kuvassa 2.6 on esitetty tuotteesta saatu tieto (Karniel & Reich 2011). Rivit ja sarakkeet (W, X, Y. Z) kuvaavat tässä esimerkissä tuotteen komponentteja, mutta ne voisivat myös olla toimintoja. Kuvassa suluissa olevat komponenttien nimet liittyvät kirjassa olevaan tutkimusesimerkkiin, joka esitetään tarkemmin luvussa 4. Soluille on merkattu parametrit sekä niiden vaikutusarvot.

	W	X	Y	Z
W (frame)		Weight 3	Weight 1 Size 1 Location 1	
X (drum)	Grip 3 Pressure 3			
Y (laser)				Processing Time 3
Z (control)		Acceleration 3 Resolution 1		

Kuva 2.6: Tuotetiedot (Karniel & Reich 2011, s. 53)

Tuotetietojen ja parametrien vaikutus arvojen perusteella esimerkissä on luotu taulukko, missä on nähtävillä toimintojen välisten riippuvuuksien suunnat, vaikuttavat parametrit ja vaikutuksien arvot. Riippuvuudet ovat nähtävillä taulukosta 2.1 (Karniel & Reich 2011). Nähdään, että taulukon

tiedot ovat samat, mitä kuvassa 2.6, mutta ne on esitetty selkeämmin, minkä ansiosta seuraavan vaihe eli arvojen summaaminen ja DSM:n luominen on yksinkertaisempaa.

Taulukko 2.1: Toimintojen väliset riippuvuudet ja niiden vaikutusarvot (Karniel & Reich 2011, s. 53)

From	To	Parameter description	Influence	Value
X	W	Weight	Med	3
X	Z	Acceleration	Med	3
X	Z	Rotation resolution	Low	1
Y	W	Weight	Low	1
Y	W	Size	Low	1
Y	W	Location	Low	1
W	X	Gripping force	Med	3
W	X	Beam pressure	Med	3
Z	Y	Processing time	Med	3

Kahden ensimmäisen vaiheen suorittamisen jälkeen saatu tieto on kerätty ja muutettu käyttökelpoiseksi. Taulukosta löytyvien tietojen avulla voidaan rakentaa numeerinen DSM. Kirjan teksti seuraa esimerkissä kirjoittajien esittämää prosessia, ja summaa kahden komponentin väliset vaikutusarvot yhteen. Esimerkiksi komponenttien Y ja W välillä vaikuttaa kolme parametria, kun vaikutussuunta on Y:stä W:hen. Kaikkien riippuvuuksien vaikutus on pieni eli 1, jolloin summaamalla arvot yhteen saadaan vaikutuksien summaksi 3. Kaikki loput taulukosta 2.1 löytyvät vaikutukset on summattu yhteen samaa tapaa käyttäen, ja saatujen summien avulla on luotu numeerinen DSM, joka on nähtävillä kuvassa 2.7a (Karniel & Reich 2011).

	W	X	Y	Z
W		3	3	
X	6			
Y				3
Z		4		

(a)

	W	X	Y	Z
W		0.25	0.25	
X	0.5			
Y				0.25
Z		0.33		

(b)

Kuva 2.7: (a) Numeerinen DSM ja (b) todennäköisyyksien mukaan tehty DSM (Karniel & Reich 2011, s. 54)

Skaalattuja todennäköisyyksien arvoja tarvitaan prosessin simulointia varten. Kirjoittajien tekemässä arvojen määrittämisessä käytetään lineaarista skaalausta, missä todennäköisyyden arvo p = 0,5 on valittu riippuvuuksien maksimiarvoksi. Valinnan on mielivaltainen, ja valittu arvo on ainoa, mikä on vahvistettu jossakin todellisessa tapauksessa. (Karniel & Reich 2011) Tämän esimerkin oikeiden todennäköisyyksien validoimiseksi vaadittaisiin lisätutkimusta. Karnielin ja Reichin (2011) mukaan lineaarinen muuttaminen on perusteltua, koska DSM:n solujen arvojen väliset suhteet pysyvät muuttumattomina. Tämän ansiosta matriisin mahdolliseen uudelleenjärjestelyyn voidaan käyttää joko numeerista tai todennäköisyksien pohjalta tehtyä DSM:a.

Neljännen vaiheen mukainen esimerkissä tehty numeeristen arvojen skaalaaminen todennäköisyys arvoiksi DSM:iin on esitetty kuvassa 2.7b. Komponenttien W ja X välillä on numeerisen DSM:n suurin riippuvuus, jolloin sille on annettu todennäköisyyden maksimiarvo eli 0,5. Tämän jälkeen muut esiintyvät vaikutusarvot 4 ja 3 on muutettu vastaamaan maksimiarvoja siten, että todennäköisyyksien suhde maksimiarvoon 0,5 on sama kuin numeeristen arvojen suhde maksimiarvoon 6. (Karniel & Reich 2011)

2.6 Simulointi DSM:n avulla

Karniel ja Reich (2011) toteavat, että DSM:iin pohjautuvia simulaatioita on käytetty moniin eri tarkoituksiin, joita ovat muun muassa matriisin uudelleenjärjestäminen sekä prosessiin muuttujien laskeminen. Tyypillisesti DSM:iin perustuvat suunnittelualgoritmit eivät käytä matriisin diagonaalia, mutta jos simuloitavan prosessin jokaisella vaiheella on vain yksi simulaatioparametri, niin diagonaalisia soluja voidaan käyttää parametrien esittämiseen.

DSM:iin pohjautuva mallinnus helpottaa tunnistamaan iteratiivisia silmukoita ja ohjaa prosessin suunnittelua, mutta siitä huolimatta vain DSM-rakennetta käyttäviä suunnittelualgoritmeja kritisoidaan, koska ne ovat epäpäteviä optimoimaan prosessia (Browning & Eppinger 2002, Abdelsalam & Bao 2006, Karniel & Reich 2011 mukaan). Kirjoittajien mukaan DSM arvojen käytöstä simulaatiotarkoituksiin ei ole yleistä sopimusta, minkä seurauksena monia eri tapoja DSM:n muuttamiseksi prosessilogiikaksi on käytössä. Prosessilogiikoista kerrotaan tarkemmin työn luvussa 3.1. Kirjoittajat esittävät useita esimerkkejä erilaisista tulkintatavoista. Eräs Karnielin ja Reichin esittämä tulkintatapa perustuu tarvittavien toistokertojen täsmälliseen määritykseen. Tulkinnan oletuksena on, että kaikkiin silmukan vaiheisiin vaikuttaa iteraatio, jonka seurauksena kaikki vaiheet tulisi tehdä uudelleen. (Abdelsalam & Bao 2006, Karniel & Reich 2011 mukaan) Karnieli ja Reichin (2011) mielestä tähän tulkintaa perustuva prosessi ja sen vaiheet ovat sarjassa toisiinsa nähden. Prosessi etenee DSM rakenteen mukaisesti, ja palautelinkit osoittavat tarvittavat toistomäärät. DSM-metodin ja simuloinnin hyödyt on kirjoittajien mukaan laajasti keskusteltu ja todistettu kirjallisuudessa.

Mallinnuksen tavoitteeksi on määritetty ennustavan mallin luominen parantamaan johdon päätöksentekoa (Smith & Morrow 1999, Karniel & Reich 2011 mukaan). Karniel ja Reich (2011) ovat määrittäneet kriteerit prosessia kuvaavalle mallille, jotta sellainen malli antaisi hyödyllisiä ennustuksia. Mallin validoinnin tasoja on esitetty kirjassa neljä: pätevyys, soveltaminen realistisiin datajoukkoihin, päätöksenteon ohjaaminen kokeellisessa ympäristössä sekä päätöksenteon ohjaaminen todellisissa tilanteissa. Pätevyys osoittaa, että mallin ja sen sisältö vaikuttavat päteviltä tuotteen suunnittelun tunteville tahoille. Mallin soveltamista varten tarvittavat datajoukot on kerätty kirjallisuudesta. Kirjoittajat kertovat, että malleilla osoitetaan mahdollisuus ohjata päätöksentekoa. Päätöksenteolla kokeellisessa ympäristössä pyritään esittämään päätöksenteon parantuminen. Kokeellisessa ympäristössä esiintyvät ongelmat ovat Karniel ja Reichin mukaan usein erittäin yksinkertaisia, joten päätöksenteon demonstroimista voi olla vaikea todentaa. Validoinnin ja simuloinnin lopullisena päämääränä on päätöksenteon ohjaaminen todellisissa tilanteissa.

2.7 Itseiterointi

Karniel ja Reich (2011) toteavat, että DSM-kirjallisuudessa ei käsitellä itseiterointia, mutta sen määrittäminen on kuitenkin tärkeää simulointitarkoituksiin. Heidän mukaansa suunnittelutoiminnon itseiteroinnin voi aina jakaa suunnitteluun, tarkistukseen ja päätöksentekoon. Erottelu voi olla perusteltua, jos eri resurssit suorittavat toisistaan eroavia toimintoja, mutta vain yhtä resurssia käyttävälle prosessille erottelu on tarpeeton.

Kirjoittajat kertovat, että itseiteraatiot edustavat käytännöllisiä prosesseja. Esimerkiksi tällaisesta tilanteesta annetaan jokin suunniteltu osa, joka ei täytä sille annettuja vaatimuksia ja on näin ollen tehtävä uudestaan. Itseiteroinnin käyttäminen simulaatiossa parantaa simulaation vaihtoehtoja, mutta samaan aikaan simulaation toteutus monimutkaistuu. Simulaatiota vaikeuttaa iteraatiotapahtumien osoittaminen, joita tarvitaan simulaation onnistumiseksi. Itseiteraatio merkitään DSM:iin poikkeuksellisesti diagonaalin soluille, joita muuten ei käytetä arvojen merkitsemiseen.

3. DSM-POHJAINEN PROSESSI

3.1 Logiikka ja rajoitteet

Prosessilogiikkaa esitettäessä käytetään eri symboleja ilmaisemaan JA/TAI logiikkaa (Clarkson & Hamilton 2000, Browning 2001, Karniel & Reich 2011 mukaan). Muita JA/TAI logiikan lisäksi kirjassa esiintyviä logiikoita ovat Split ja Join eli erkaantuminen ja yhdistyminen. Prosessissa tämä voidaan tulkita tarkoittavan toiminnan jälkeen tapahtuvaa erkaantumista, missä yhden toiminnon jälkeen on mahdollista suorittaa kaksi tai useampia toimintoja, sekä vastaavasti useamman toiminnon jälkeen tapahtuvaa yhdistymistä, missä toiminto on mahdollista suorittaa vasta, kun kaikki edeltävän vaiheen toiminnot on suoritettu. Lisäksi Karniel ja Reich (2011) puhuvat Lopetus- ja Aloitus-logiikoista, missä ennen ja jälkeen todellisten prosessin vaiheiden on lisätty loogiset toiminnot, jotka ilmaisevat milloin prosessi alkaa ja loppuu.

DSM-malleilla on vaikea erotella syöte- ja tuotelogiikoita toisistaan, ja monissa malleissa niiden logiikat eivät eroa toisistaan mitenkään. Tyypillisesti logiikat ovat kuitenkin erilaiset, jolloin DSM-malleja käyttämällä päädytään herkästi määrittelyongelmiin. Ongelma on mahdollista välttää esittämällä syöte- ja tuotelogiikat erillisissä matriiseissa. (Karniel & Reich 2011) Kirjoittajat toteavat, että nykyiset DSM algoritmit eivät täytä kaikkia prosessilogiikan vaatimuksia. Tällainen logiikkavaatimus on välttää sellaisten toimintojen uudelleenjärjestämistä, joidenka järjestyksellä on erityinen merkitys (Abdelsalam & Bao 2006, Karniel & Reich 2011 mukaan). Eli esimerkiksi testaus ei saisi olla prosessissa ennen suunnittelua, mutta DSM algoritmien avulla olisi mahdollista järjestää matriisi siten, että vaiheiden järjestys olisi väärä.

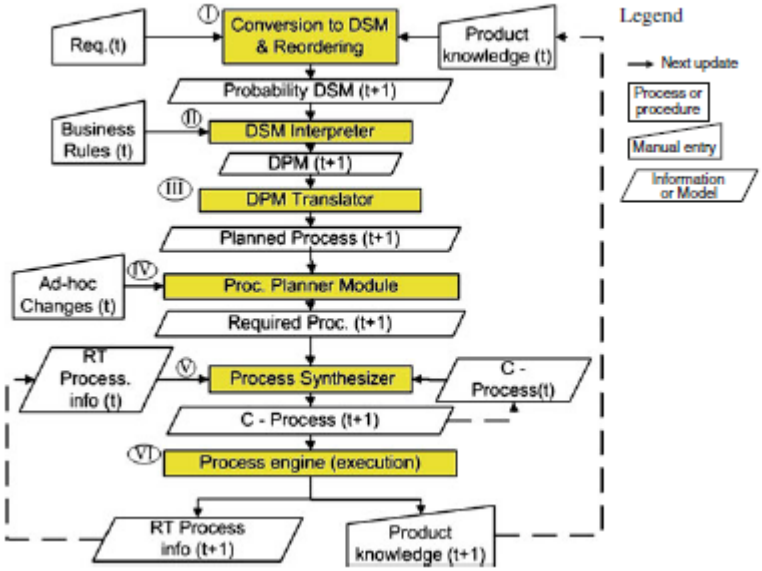
Kirjoittajat toteavat myös, että logiikan määrittämisessä prosessille havaitaan useita ongelmia. Ongelmia tulee heidän mukaansa vastaan muun muassa prosessilogiikan määrittämisessä, logiikan esittämisessä ja mallintamisessa sekä logiikan tarkistamisessa prosessin oikeellisuuden varmistamisen yhteydessä. Karniel ja Reichin (2011) mielestä logiikan tarkistamisesta ei piitata DSM-pohjaiseen simulaation keskittyvässä kirjallisuudessa, vaikka tarkistamisen tärkeys olisikin tiedossa. Iteraatioita sisältävät prosessit sisältävät luonnostaan loogisia ongelmia, minkä seurauksena prosessilogiikan tarkkaa määrittämistä tarvitaan.

3.2 DSM:n muuntaminen prosessiksi

Karniel ja Reich (2011) ehdottavat prosessin hallitsemiseksi suljetun silmukan ohjausprosessia, jonka avulla ohjataan dynaamisesti muuttuvaa suunnitteluprosessia yhdessä dynaamisen prosessijärjestelmän avulla. Prosessista kerrotaan, että se sisältää kahdenlaisia syötteitä: ulkoiset ja sisäiset. Ulkoiset syötteet eivät ole linjassa prosessin etenemisen kanssa, vaan voivat muuttua jatkuvasti. Sisäiset syötteet ovat riippuvaisia prosessin todellisesta etenemisestä. Kirjoittajat toteavat, että prosessia käytetään myös simulaatioissa. Simulaatiotulokset voivat auttaa prosessiin liittyvässä päätöksenteossa, kuten luvussa 2.6 todettiin.

Esitetyssä prosessin kehyksessä on kolme lohkoa: integroitu prosessigeneraattori, prosessimoottori ja tuotepohjainen prosessijärjestelmän generaattori. Prosessigeneraattorin tehtävänä on tuottaa dynaaminen prosessijärjestelmä, ja se yhdistää ulkoisia syötteitä vaadituista muutoksista sisäisitä syötteistä saatavaan palautteeseen. Tietojen avulla generaattori luo prosessisuunnitelman. Prosessimoottori toteuttaa prosessijärjestelmän mukaisen prosessin ajoajan. Moottorilla oletetaan olevan vastaavat kyvyt kuin nykyisen työnkulun moottoreilla. Tuotepohjainen prosessijärjestelmän generaattori luo tarvittavan prosessijärjestelmän saatujen tuotetietojen perusteella.

Suljettu iteratiivinen metaprosessi esitetään diagrammina, jossa prosessin toimintoja seuraa tulokset ja tuotetietojen syötteet. Karniel ja Reichin (2011) kuvaus on määritetty laskennallisen näkymään kehykseksi, ja se on nähtävillä kuvassa 3.1. Prosessin vaiheet ensimmäisestä kolmanteen suorittaa tuotepohjainen prosessijärjestelmän generaattori. Vaiheet neljä ja viisi suorittaa integroitu prosessigeneraattori, ja viimeisen vaiheen toteuttaa prosessimoottori. (Karniel & Reich 2011)



Kuva 3.1: Suljetun metaprosessin malli (Karniel & Reich 2011, s. 117)

Prosessissa DSM:n käyttäminen on osa prosessin suunnittelujaksoa, joka pitää sisällään vaiheet 1, 2 ja 3.Suunnittelu keskittyy kirjoittajien mukaan pääsääntöisesti vaiheeseen yksi, jossa kerätty tuotetieto muunnetaan DSM:ksi. Kuvasta 3.1 huomataan, että prosessi alussa käytetään tuotetietoa ajanhetkellä *t*. Prosessin aikana kerätty tuotetieto ajanhetkellä *t* + 1 palautuu mallissa takaisin vaiheeseen yksi, mikä tekee prosessista suljetun ja mahdollistaa sen jatkuvan kehittymisen. Ensimmäisen vaiheen aikana tunnistetuille komponenteille osoitetaan suunnittelutoiminnot, ja jokaiselle toiminnolle määritetään parametrit. Tietojen muuttaminen DSM:ksi tapahtuu luvussa 2.5 esitetyllä tavalla. Kuvasta 3.1 nähdään, että tietojen muuttamisen jälkeen vaihe yksi sisältää myös matriisin uudelleenjärjestämisen, jota käytetään prosessisuunnitelman luomiseen. Oletuksena suunnitelman luomisessa on, että prosessi käynnistyy heti (Karniel & Reich 2011).

Kuvasta 3.1 voidaan lukea vaiheen kaksi olevan DSM:n muuntaminen prosessisuunnitelmaksi. Vaiheen aikana järjestetty DSM muutetaan design process matriisiksi (DPM). Muutos toteutetaan lisäämällä matriisiin suunnittelutoimintojen ympärille loogiset toiminnot. Lisäämisen jälkeen osoitetaan deterministiset linkit, joiden todennäköisyys $p = 1$. Osoittaminen on tehtävä vasta DSM:n uudelleenjärjestämisen jälkeen, koska muuten prosessiin voisi syntyä palautelinkki todennäköisyydellä $p = 1$, minkä seurauksena syntyisi loppumaton sykli. (Karniel & Reich 2011) Linkkien avulla yhdistetään loogiset toiminnot suunnittelutoimintoihin (Karniel & Reich 2007, Karniel & Reich 2011 mukaan).

Kahden ensimmäisen vaiheen jälkeen metaprosessissa ei enää esiinny DSM:a ennen kuin suljettu prosessi palaa päivitetyn tiedon kanssa takaisin ensimmäiseen vaiheeseen. Meta-prosessi jatkuu kuvan 3.1 mukaan siten, että prosessin suunnittelu jakson viimeisenä vaiheena on DPM:n muuntaminen vastaavaksi C-prosessiksi. C-prosessi eli current process määritellään Karniel ja Reichin (2011) toimesta suunnitelluksi prosessisuunnitelmaksi, jota tulisi seurata ajon aikana. Tämä voidaan lukea myös kuvasta 3.1, jossa vaiheiden kolme ja neljä välisessä kohdassa on suunniteltu prosessi ajanhetkellä *t* +1.

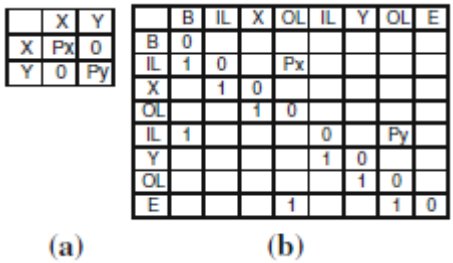
Prosessin suunnittelujakson jälkeen prosessin neljännessä vaiheessa prosessijärjestelmä päivitetään perustuen tuoreimpaan tuotetietoon, ja prosessin kaaviokuvasta nähdään, että päivittämisen seurauksena syntyy required process scheme eli vaadittava prosessisuunnitelma. Viidennen vaiheen toteuttaa prosessin syntetisaattori, joka käyttää hyväksi C-prosessia, vaadittua prosessia sekä RT-prosessin eli ajoaika prosessin todellista tilaa, ja laskee niiden avulla C-prosessin ajanhetkelle *t* + 1. Meta-prosessin viimeisessä vaiheessa prosessimoottori toteuttaa suunnitellun prosessin.

3.3 Muuntamisen toteutus

Meta-prosessin toisessa vaiheessa DSM muutetaan DPM:ksi, kuten luvussa 3.2 kerrottiin. Esimerkki muutoksesta on esitetty kuvassa 3.2. Karnielin ja Reichin (2011) mukaan DSM suunnittelun tulosten muuttamiseksi tarvitaan rakenteellisia määritelmiä, minkä avulla voidaan varmistaa tiettyjen prosessin ominaisuuksien toteutuminen. Muuntamiselle on määritelty seuraavat vaatimukset (Karniel & Reich 2007b, Karniel & Reich 2011 mukaan):

1. Prosessilla pitäisi olla määritetyt aloitus- ja lopetustoiminnot.
2. Mistä tahansa prosessin vaiheesta pitäisi olla mahdollista saavuttaa prosessin loppu.
3. Kun prosessi loppuu, pitäisi
    a. kaikkien toimintojen ja niiden iteraatioiden olla suoritettuna.
    b. jokaisen toiminnon olla suoritettuna vähintään kerran.

Karniel ja Reich (2011) suosittelevat algoritmia, jonka avulla DSM-suunnitelma muutetaan DPM:ksi. Algoritmissa lisätään aluksi loogiset aloitus- ja lopetustoiminnot ennen ja jälkeen prosessiin suunniteltuja toimintoja. Tämän jälkeen aloitustoiminto yhdistetään kaikkiin rinnakkaisiin toimintoihin, joilla ei ole syötelinkkiä. Lopuksi lopetustoimintoon yhdistetään kaikki toiminnot, joilla ei ole tuotelinkkiä. (Karniel & Reich 2007a, Karniel & Reich 2011 mukaan)



Kuva 3.2: DSM:n muuntaminen DPM:ksi (Karniel & Reich, 2011, s.125)

Kuvassa 3.2 kaksi toimintoa X ja Y sisältävä yksinkertainen DSM on muutettu DPM:ksi. Kuvasta nähdään, miten prosessiin on algoritmin mukaisesti lisätty aloitus- ja lopetustoiminnot kuvaamaan prosessin alkamista ja loppumista. Lisäksi molempien toimintojen ympärille on lisätty syöte- ja tuotelogiikat (*IL* ja *OL*). Toiminnon X syötelogiikka on yhdistetty aloitustoimintoon, ja toiminnon Y tuotelogiikka lopetustoimintoon.

3.4 Toimintojen väliset suhteet

3.4.1 Itsenäinen toiminto

Karniel ja Reich (2011) kertovat, että itsenäisellä suunnittelutoiminnolla on tietty kestoaika. Itsenäinen toiminto voi myös toistua, ja itseiteraation todennäköisyys kuvataan toimintoa esittävän DSM:n diagonaalilla. Sen sijaan toiminnon kestoaikaa ei kirjoittajien mukaan kuvata DSM:ssa. Jotta yksittäinen suunnittelutoiminto voidaan muuntaa DPM:ksi, on toiminnon ympärille lisättävä loogisia toimintoja. Näitä olivat aloitus-, lopetus-, syöte- ja tuotelogiikat. Seuraavat säännöt liittyvät loogisten toimintojen täytäntöönpanoon (Karniel & Reich 2011):

S1 Loogisen toiminnon kestoaika on nolla, ja määritelmänsä mukaisesti se ei toistu itsestään eli sillä ei ole itseiteraatiota.
S2 Suunnittelutoiminnolla on yksi syötteenä annettu linkki ja yksi tuotteena lähtevä linkki.

S3 Loogisilla toiminnoilla voi olla useita syöte- ja tuotelinkkejä, mutta molempia on oltava vähintään yksi. Kolmas sääntö sisältää kaksi poikkeusta: Aloitus logiikalla ei ole syötelinkkejä eikä lopetus logiikalla tuotelinkkejä.

3.4.2 Rinnakkain olevat toisistaan riippumattomat toiminnot

Karniel ja Reich (2011) sanovat, että kaksi toimintoa voivat olla keskenään rinnakkaisia, mutta täysin toisistaan riippumattomia. Prosessissa rinnakkain olevat toiminnot tapahtuvat samansuuntaisesti, mutta eivät vaikuta toisiinsa millään tavalla. Rinnakkain riippumattomien toimintojen prosessissa on paljon samankaltaisuuksia itsenäisen toiminnon kanssa. Toiminnot voivat kirjoittajien mukaan iteroitua toistuvasti. Lisäksi he toteavat, että tilanteet, joissa rinnakkain riippumattomat toiminnot esiintyvät vaativat selkeät loogiset määritelmät annetulle aloitus- ja lopetuslogiikalle. Kirjassa esitetään logiikoiden täytäntöönpanosäännöt, jotka luovat toimintavan, jonka avulla voidaan hallita rinnakkaisia käynnistyksiä ja lopetuksia useamman polun simuloinnissa (Karniel & Reich 2009, Karniel & Reich 2011 mukaan). Esitetyt täytäntöönpanosäännöt olivat seuraavat:

S4 Aloitus-toiminnon logiikka muotoa Split-And.
S5 Lopetus-toiminnon logiikka muotoa Join-And.
S6 Jos toiminnolla ei ole edeltävää lähdettä, se olisi yhdistettävä aloitustoimintoon.
S7 Jos toiminnolla ei ole kohdetta, se olisi yhdistettävä lopetustoimintoon.

Molemmat rinnakkaisista toiminnoista hyväksyvät linkit aloitustoiminnosta Split-AND logiikan mukaisesti, ja toiminnot alkavat rinnan. Molemmat toiminnot ovat myös linkitettyinä lopetus toimintoon Join-AND logiikan mukaisesti. Lopetustoiminnon suorittaminen sallitaan vasta, kun molemmat toiminnot ovat suorittaneet kaikki iteraationsa. (Karniel & Reich 2011) Syötelinkeillä ja iteraatiot mahdollistavilla palautelinkeillä voi olla erilliset logiikkaoperaattorit. Seuraavat täytäntöönpanosäännöt kirjoittajat ovat määritelleet syöte- ja tuotelogiikoille:

S8 Suunnittelutoimintoihin menevillä linkeillä on AND-logiikka ensimmäisessä iteraatiossa.
S9 Linkillä lopetustoimintoon on XOR-logiikka (Exclusive-OR) muiden linkkien kanssa.
S10 Palautelinkeillä on OR-logiikka.

Karniel ja Reich (2011) kertovat, että yksinkertaisissa tapauksissa syötelogiikka määrittelee hyväksyttävät signaalit etenevistä linkeistä. Esimerkkitapauksessa toiminto alkaa vasta, kun kaikki sitä edeltävät toiminnot on suoritettu, tai vaihtoehtoisesti vasta kun signaali mistä tahansa palautelinkistä on saatavilla. Tuotelogiikan mukainen menettely puolestaan voi lähettää signaaleja kaikille palautelinkeille tai kaikille syötelinkeille, mutta ei molempiin samanaikaisesti.

3.4.3 Sarjassa olevat toiminnot

Suunnitteluprosessissa sarjassa olevien toimintojen järjestys voi olla seurausta standardoinnista (Sered & Reich 2006, Karniel & Reich 2011 mukaan). Karniel ja Reich (2011) antavat esimerkiksi tilanteen, jossa standardiosan A suunnittelu voi vaikuttaa osan B suunnitteluun, mutta osalla B ei ole vastaavaa vaikutusta osaan A. Sarjassa oleville toiminnoille X ja Y pätee kirjoittajien mukaan eri loogiset rajoitukset toimintojen ensimmäisellä toteutuskerralla verrattuna myöhempiin iteraatioihin. Ensimmäisen toteutuksen jälkeen toiminto X ei voi lähettää signaalia loogiseen lopetustoimintoon, vaan signaali on lähetettävä seuraavana sarjassa olevaan toimintoon Y. Muuten toiminto Y jäisi kokonaan suorittamatta. Karniel ja Reich (2011) ovat määritelleet sarjassa olevien toimintojen X ja Y väliselle linkille todennäköisyydeksi *Pxy*, joka muutetaan prosessissa linkiksi toiminnosta X lähtevän tuotelinkin ja toimintoon Y menevän syötelinkin välille. Kun todennäköisyys *Pxy* on voimassa, ei prosessin tulkinta iteraatioiden kohdalla ole yksiselitteinen. Kirjoittajat toteavat, että ongelmana on, että jos toiminto X iteroituu, voiko toiminto Y alkaa rinnakkain

edeltävän toiminnon kanssa, vai pitääkö sen odottaa kaikkien X:n iteraatioiden valmistumista. Täytäntöönpanoa varten annetaan seuraavat säännöt:

S11 Signaalin lähetyksessä sarjassa olevan vaiheen valmistumisesta voidaan seurata seuraavia Business Rule -vaihtoehtoja.
- a. Signaalin lähetys vasta, kun toiminto on suorittanut kaikki iteraationsa
- b. Signaalin lähetys seuraavaan vaiheeseen, kun toiminto on suoritettu ensimmäisen kerran. Signaali lopetustoimintoon voidaan kuitenkin lähettää vasta, kun prosessin kaikkien vaiheiden kaikki iteraatiot on suoritettu.

Ensimmäisessä tapauksessa *a* on kyseessä sarjassa oleva toiminto, missä iteraatiot tapahtuvat itseiteraation avulla. Koska toiminto Y voi alkaa vasta X:n suoritettua kaikki iteraationsa, ja iteraatiot tapahtuvat sarjassa, myös koko on prosessi sarjassa. Tilanteessa *b* on kyseessä sarjassa olevat toiminnot, jotka voidaan toteuttaa rinnakkain. (Karniel & Reich 2011) Toiminnolla X voi siis olla iteraatioita, jotka voidaan toteuttaa samaan aikaan rinnakkain toiminnon Y kanssa. Jotta sarjassa olevat toiminnot voidaan toteuttaa samanaikaisesti, tarvitaan lisäyksiä täytäntöönpanon sääntöihin. Karniel ja Reich (2011) antavat seuraavat lisäykset:

S12 Signaali lopetustoimintoon: toisen tai sitä myöhemmän toteutuksen jälkeen toiminto voi lähettää signaalin lopetustoimintoon, kun samaan aikaan seuraavat sarjassa olevat toiminnot ovat alkaneet (tai valmistuneet).
- a. Toisen tai myöhemmän toteutuksen jälkeen toimintoa seuraa sen jälkeen sarjassa olevat toiminnot.
- b. Toisen tai myöhemmän toteutuksen jälkeen seuraavat sarjassa olevat toiminnot voivat toteutua, tai lopetustoiminto toteutuu, mutta ei molemmat.

S13 Saman toiminnon iteraatiot eivät voi tapahtua rinnakkain.
- c. Kun toimintoa suoritetaan, syötelinkin signaalit ohjataan kyseiseen toimintoon.
- d. Kun toimintoa suoritetaan, syötelinkin signaalit ohjataan toiminnon seuraavaan iteraatioon, eikä sitä voida aloittaa ennen edellisen iteraation valmistumista.

Rinnakkain tapahtuvat iteraatiot estetään säännöllä 13, jotta vältyttäisiin toimintojen nopealta lisääntymiseltä simulaatiota tehdessä. Yleisesti oletetaan yksittäisen resurssin tekevän lisää iteraatioita samasta toiminnosta. Jos kaksi resurssia toteuttavat saman komponentin suunnittelua, niin nämä resurssit tulisi määritellä selvästi. (Sadiq & Orlowska 1999, Karniel & Reich 2011 mukaan)

3.4.4 Kytketyt toiminnot

Tosiinsa kytketyillä toiminnoilla voi olla vaatimuksia jaksottaiselle toteutukselle, esimerkiksi testauksen toteuttaminen vasta suunnittelutoimintojen suorittamisen jälkeen. Kytketyt toiminnot voivatkin alkaa sekä sarjassa, että rinnakkain. (Karniel & Reich 2011) Molemmille tapauksille on omat sääntönsä, jotka on esitetty seuraavasti:

S14 Kytketyn toiminnon toteutuksen aloitus:
- a. Sarjassa olevan toiminnon voi aloittaa, kun sitä DSM:n mukaan edeltävä toiminto on valmistunut vähintään kerran.
- b. Rinnakkaisten toimintojen tapauksessa toiminto voi alkaa rinnan muiden samassa toimintasilmukassa olevien toimintojen kanssa.

Tapauksen *a* toteutus ja logiikka on vastaava kuin edellisessä luvussa 3.4.3 esitetyllä sarjassa olevan toiminnon säännön *11a* mukaisella tapauksella. Jälkimmäinen tapaus on puolestaan samankaltainen säännön *11b* kanssa, mutta kirjoittajat kertovat syötelogiikan olevan erilainen. Ero toteutetaan käytännössä asettamalla aloitustoiminnosta linkki, jonka todennäköisyys $p = 1$. Tämän seurauksen säännön 12 mukaiset vaihtoehdot korvataan seuraavilla kirjassa kerrotuilla vaihtoehdoilla:

S15 Signaalin lähettäminen toiseen tai myöhempään kytketyn toiminnon toteutuskertaan tehdään käyttäen yhtä seuraavista vaihtoehdoista:
- a. Kytketyn toiminnon tulisi valmistuttuaan linkittyä seuraavaan toimintoon

b. Kytketty toiminto voi valmistuttuaan linkittyä lopetustoimintoon.

Kytkettyjen toimintojen lisätoteutus voidaan tehdä sekä sarjan tai rinnakkain olevalle toteutukselle. Siinä palautelinkkejä sovelletaan samaan tapaan kuin ne on matriisissa ilmaistu. Lisätoteutus on suoraviivaisempi tätä kytkettyjen toimintojen suorittamiselle, mutta sen käyttämistä varten tarvitaan todisteita. (Karniel & Reich 2011) Toteutus vaatii muutoksia sääntöön 11, jotta voidaan ilmaista toiminnon aikaista ja myöhäistä aloittamista. Aikaisella aloituksella tarkoitetaan toiminnon aloittamista, kun edellinen toiminto on suorittanut vähintään yhden iteraatioistaan. Myöhäisessä aloituksessa odotetaan, että edellinen vaihe on suorittanut kaikki iteraationsa. Syöte- ja tuotelogiikoiden säännöt lisätoteutukselle on annettu kirjassa:

S16 Syöte- tuotelogiikoiden vaihtoehdot myöhäiselle (a) ja aikaiselle (b) toteutukselle voi seurata seuraavia sääntöjä:

a. Tuotelogiikka: Signaalin lähetys eteenpäin vasta, kun kaikki iteraatiot suoritettu.
   Syötelogiikka: Toiminnon aloittaminen vasta, kun kaikki edeltävän vaiheen iteraatiot on suoritettu.
b. Tuotelogiikka: Signaalin lähetys eteenpäin, kun ensimmäinen toteutus on valmistunut, mutta signaali lopetustoimintoon vasta kaikkien iteraatioiden valmistumisen jälkeen.
   Syötelogiikka: Toteutuksen aloittaminen, kun kaikki edeltävät toiminnot ovat suorittaneet vähintään yhden iteraation.

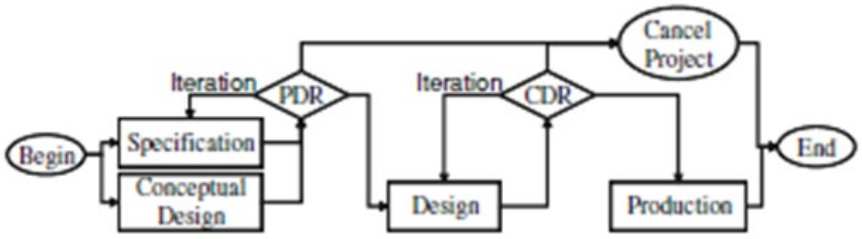
Säännön 16a toteuttaminen on kirjoittajien mielestä monimutkaista päätösalgoritmien avulla, mutta helppoa WF-verkon peruslogiikkaa käyttämällä. Sen sijaan sääntö 16b on heidän mielestä helppo toteuttaa päätösalgoritmin avulla, mutta vaikeaa WF-verkon avulla. WF-verkkoja ja päätösalgoritmia ei käsitellä tässä työssä tarkemmin.

4. TOTEUTTAMINEN KÄYTÄNNÖSSÄ

4.1 Tuote ja tuotekehitysprosessi

Karniel ja Reich (2011) demonstroivat dynaamista prosessin suunnittelua laajan tutkimusesimerkin avulla. Tutkittavana prosessina on laser direct imaging (LDI) -levyn suunnittelu. Esimerkki demonstroi vaiheet, joiden ympärille prosessin eteneminen rakentuu, miten tuotetieto muutetaan vaadituksi prosessisuunnitelmaksi, uuden tuotetiedon yhdistämisessä syntyvän ohimenevän prosessin sekä lopulta simulaatioon pohjautuvan päätöksentekoprosessin. Tässä työssä keskitytään DSM:n käyttöä sisältäviin ja niitä edeltäviin prosessin vaiheisiin. DSM:n hyödyntämisen jälkeen tulevat vaiheet esitetään lyhyesti.

Karniel ja Reich kertovat saaneensa tutkimusesimerkin lähtötiedot LDI-levyn suunnittelua varten johtavalta suunnitteluorganisaatiolta, joka otti aktiivisesti osaa monien eri yritysten esipainin-kuvantamisjärjestelmien suunnitteluun ja kehitykseen. Esitettyjä tietoja ei kerätty kehitysprosessin aikana, mutta niitä käytetään ikään kuin ne olisi kerätty vastaavan prosessin aikana. Prosessin vaiheiden aikana paljastuneet tuotetiedot ovat rajallisia, koska halutaan esimerkin avulla simuloida tiedon kehittymistä uuden tuotteen tuotekehitysprosessin aikana. LDI-levyn tuotekehitys käynnistettiin esimerkissä ennalta määrätyn prosessijärjestelmän mukaisesti. Prosessijärjestelmäksi valittiin high-level prosessi, joka on esitettynä kuvassa 4.1.



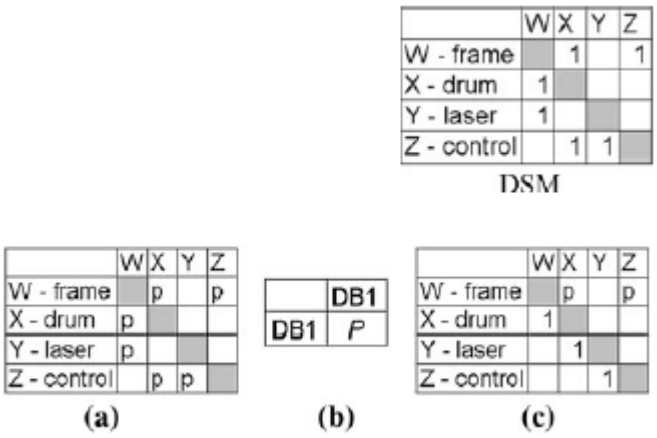
Kuva 4.1: High-level prosessin malli (Karniel & Reich 2011, s. 25)

Kirjoittajien käyttämä high-level prosessi sisältää rinnakkaisia toimintoja, päätöspisteitä sekä iteraatioita. Kuvasta 4.1 nähdään, että määrittely ja käsitteellinen suunnittelu alkavat rinnakkain. Näiden vaiheiden jälkeen tulee prosessin ensimmäinen päätöspiste preliminary design review (PDR), jossa tehdään päätökset siitä, jatkuuko projekti seuraavaan vaiheeseen, toistetaanko päätös pistettä edeltävät vaiheet vai lopetetaanko prosessi. PDR:n jälkeen, jos projektia päätetään jatkaa, siirrytään projektin varsinaiseen suunnitteluun, jota seuraa prosessin seuraava päätöspiste critical design review (CDR), jossa tehdään päätös, että jatketaanko projektia, toistetaanko edellinen vaihe vai lopetetaanko projekti. Jos projektia jatketaan CDR:n jälkeen, siirrytään tuotantoon, jonka jälkeen prosessi on valmis. Karniel ja Reich (2011) kertovat, että high-level prosessin määritelmässä prosessimallia saatetaan pitää suhteellisen muuttumattomana, minkä lisäksi prosessi saattaa koskea useampia tuotteita.

Esimerkkiprosessin varsinainen täytäntöönpano alkaa yleisen prosessin määrittelyllä, missä prosessissa on looginen alkutoiminto, yleinen toiminto sekä lopetustoiminto. Yleinen toiminto korvataan ajan myötä aliprosesseilla, joilla on omat alku- ja lopputoiminnot. RT-prosessi seurasi C-prosessia. Saavuttaessaan yleisen toiminnon ennalta määritetty toiminnon osoittaminen suoritetaan, minkä seurauksena high-level prosessi valitaan joko manuaalisesti tai simulaation avulla. Karniel ja Reich (2011) huomauttavat, että esimerkissä PDR ja CDR toteutetaan suoraviivaisesti lisäämällä loogiset toiminnot ja palautelinkit.

4.2 Käsitteellinen suunnittelu

Uutta tuotetietoa syntyy esimerkissä, kun prosessi saavuttaa käsitteellisen suunnittelun vaiheen. Tuotetiedon kerääminen vaiheen aikana oletetaan suhteellisen yksinkertaiseksi. Demonstraatiota varten kirjoittajat ovat olettaneet tuotteen koostuvan neljästä pääkomponentista: rungosta, rummusta, laser-alijärjestelmästä sekä ohjausjärjestelmästä. Tässä vaiheessa oletettiin myös, että vaiheiden väliset suhteet tunnistetaan binaariarvoiksi, ja suhteellisia vaikutusarvoja ei vielä määritetä. Tuotetietojen avulla suunnitellaan käsitteellisen suunnittelun vaihetta, jossa DSM toimii apukeinona suunnitelman tarkennukselle. Tuotetietojen perusteella esimerkissä rakennetut DSM:t on esitetty kuvassa 4.2.



Kuva 4.2: Käsitteellisen suunnittelun suunnittelua varten luodut DSM:t (Karniel & Reich 2011, s. 194)

Kuvassa 4.2 on ylimpänä tuotetietojen perusteella luotu DSM, ja alemmalla tasolla DSM eri muutosten jälkeen. Kuvan 4.2a DSM on todennäköisyyksiin pohjautuva DSM, jossa numeeriset binaariarvon ”1” merkinnät on korvattu todennäköisyydellä *p*. Koska vaikutusten suhteellisia arvoja ei vielä oltu määritelty, oletetaan vaikutusarvot samanarvoisiksi. Kuvassa 4.2b on esitetty DSM:n optimaalisen järjestämisen seurauksena rakennettu suunnittelulohko eli design block (DB). Suunnittelulohko on prosessirakennelma, joka toteuttaa keskinäiset alun ja lopetuksen rajoitukset rinnakkaisille toiminnoille (Karniel & Reich 2007b,Karniel & Reich 2011 mukaan). Kirjoittajat kertovat, että DB voi sisältää yhden tai useamman toiminnon. DB:n solulle tarvittava todennäköisyys *P* kuvaa yhteenlaskettujen itseiteraatioiden yhdistettyä todennäköisyyttä. Todennäköisyys *P* lasketaan kaavalla

$$P = 1 - (1 - p)^6, \qquad (4)$$

jossa *p* on iteraatioiden todennäköisyyden odotettu maksimi. Kuvassa 4.2c on esitetty prosessin sarjassa tehty toteutus, missä eteenpäin suuntautuvat todennäköisyyslinkit on poistettu ja korvattu ”1” merkinnöillä. Prosessin toimeenpanoa varten kirjoittajat vähensivät toteuttamisen vaihtoehtoja asettamalla rajoitukseksi, että iteraatioita sallitaan jokaiselle toiminnolle maksimissaan kaksi, ja toiminto Z alkaa vasta edeltävien toimintojen kaikkien iteraatioiden suorittamisen jälkeen. Prosessi voidaan rajoitusten jälkeen suorittaa joko sarjassa, rinnan tai kytkettynä.

4.3 Simulaatio

Simulaation parametreinä esimerkissä käytetään toimintojen kestoa D(a_i), missä a_i sisältää kaikki toiminnot (W, X, Y, Z), sekä todennäköisyyttä *p*, joka on määritelty luvussa 4.2. Kytketyille toiminnoille toimintojen kesto on D(*DB*) = max(D(a_1)), ja itseiteraatioiden yhdistetty todennäköisyys P(*DB)* lasketaan kaavan 4 avulla. Simulaation tuloksena saadaan prosessin kokonaisaika *T*

ja vaadittavat resurssit *R*. Yksinkertaisuuden takia Karniel ja Reich (2011) olettivat yhden resurssin yhtä aktiviteettiaikayksikköä kohden, ja lisäksi kytketyille toimintojen kohdalla oletettiin resurssien olevan aktiviteettien keston summa. Prosessissa ei myöskään tapahdu oppimista, eli iteraatioiden seurauksena työvaiheiden kesto ei lyhene. Simulaation tulokset on esitetty taulukossa 4.1.

Taulukko 4.1: Tutkimusesimerkin simulaation tulokset ((Karniel & Reich 2011, s. 201)

Simulation parameters				General statistical results				Performance	
#	P	D(W,X,Y,Z)	Simulation type	Time min (prob. %)	Time max (prob. %)	Time Avg./ std	Resources Avg./ std	Prob. (%) T < 20	Prob. (%) R < 32
1	0.01	3,7,2,1	Serial	15 (0.98)	45 (0.01)	15.26/1.88	15.26/1.88	98.01	99.96
			Parallel	12 (0.98)	36 (0.01)	12.23/1.65	15.26/1.88	98.02	99.96
			Coupled	9 (94.1)	27 (0.34)	9.55/2.29	15.93/3.82	99.65	99.65
2	0.1	3,7,2,1	Serial	15 (80)	45 (0.64)	17.90/6.42	17.90/6.42	80.80	96.11
			Parallel	12 (80)	36 (0.81)	14.64/5.73	17.90/6.42	81.60	96.11
			Coupled	9 (53.1)	27 (21.9)	15.19/7.27	25.32/12.1	78.04	78.04
3	0.2	3,7,2,1	Serial	15 (0.98)	45 (1.44)	21.39/9.12	21.39/9.12	62.40	84.96
			Parallel	12 (0.98)	36 (2.60)	17.96/8.27	21.39/9.12	64.80	84.96
			Coupled	9 (26.2)	27 (54.44)	20.54/7.67	34.23/12.7	45.56	45.56
4	0.1	6,5,1,1	Serial	13 (80)	39 (0.64)	15.77/5.99	15.77/5.99	80.80	96.11
			Parallel	12 (80)	36 (0.81)	14.68/5.77	15.77/5.99	81.60	96.11
			Coupled	6 (53.1)	18 (21.9)	10.12/4.85	21.94/10.51	100.00	78.04

Taulukosta 4.1 huomataan, että siinä on kolme osaa: parametrit, tilastolliset tulokset ja suorituskyky. Nähdään, että simulaatio tehtiin neljä kertaa, joista kolmessa käytettiin eri iteraation todennäköisyyksiä, ja neljännessä käytettiin eri toimintojen kestoaikoja kuin muissa. Tilastot esittävät prosessin ajan minimin ja maksimin sekä niiden todennäköisyydet. Lisäksi tilastoista selviää kokonaisajan ja resurssien keskiarvot ja -hajonnat. Suorituskyvyn osuus kertoo todennäköisyydet projektin suorittamiselle annetussa aikarajassa $T = 20$ sekä annetuilla resursseilla $R = 32$. Tuloksista huomataan, että prosessi on herkkä parametrien muutoksille. Varsinkin prosessin suorituskyky on herkkä parametrien muutoksiin.

Todetaan tulosten pohjalta, että rinnakkaisella prosessilla on lyhyempi suoritusaika kuin sarjassa olevalla ja resurssit ovat samankaltaiset, joten rinnakkaista prosessia voisi harkita paremmaksi tavaksi toteuttaa prosessi. Karniel ja Reich (2011) ovat kuitenkin sitä mieltä, että prosessin ollessa herkkä parametrien muutoksille ei ole olemassa yleisiä sääntöjä oikean toteutustavan valitsemiselle, vaan päätökset tulisi tehdä tapauskohtaisesti.

4.4 Yksityiskohtainen suunnitteluvaihe

Käsitteellisen suunnittelun jälkeen tuotteesta on saatu selville enemmän tietoa. Tuotetiedon avulla voidaan suunnitella yksityiskohtaista suunnitteluvaihetta. Tuotetiedon kehittyminen on esitetty taulukossa 4.2, josta huomataan neljän alkuperäisen pääkomponentin hajoaminen useampaan komponenttiin. Komponentit (A, B, C, D, E, F, G, H) muodostavat uuden tuotetiedon mukaisen pohjan DSM:n rakentamista varten tarvittavalle tiedolle.

Taulukko 4.2: Tuotetiedon kehittyminen (Karniel & Reich 2011, s. 202)

Initial product decomposition	Current decomposition
X (drum)	B (drum body)
	C (drum rotation system)
Y (Laser)	D (Laser writing head)
	E (Laser driving system)
W (Frame)	A (Chassis and Cover)
	F (Load/Unload unit)
	G (Registration & punch unit)
Z (Control system)	H (Control system)

Taulukossa 2.1 esitettiin alkuperäisen komponenttijaon (X, Y, W, Z) mukaiset komponenttien riippuvuudet ja niiden vaikutusarvot. Tuotetiedon karttuessa ja kehittyessä myös tiedot toimintojen välisistä riippuvuuksista muuttuu, jolloin vaikutusarvot on määritettävä uudelleen. Esimerkissä selvitetyt vaikutusarvot ja riippuvuussuhteet on esitetty taulukossa 4.3. Taulukosta nähdään, mikä komponentti vaikuttaa toiseen komponenttiin, mihin parametriin vaikutus kohdistuu sekä vaikutuksen arvo. Vaikutus määritetään luvun 2 mukaisesti asteikolla pieni, keskitaso ja korkea. Arvoille on annettu numeeriset arvot 1, 3 ja 9.

Taulukko 4.3: Päivitetyn tuotetiedon mukaiset komponenttien riippuvuudet ja vaikutusarvot (Karniel & Reich 2011, s. 203)

From	To	Parameter Description	Influence	Value
B	A	Weight	Med	3
B	A	Size	High	9
B	C	Weight	Low	1
B	C	Moment	High	9
B	G	Size—Number/force of clumps	Med	3
B	G	Size—Number of holes	Med	3
C	A	Weight	Low	1
C	A	Size	Med	3
C	A	Max velocity	Low	1
C	B	Inertia moment	Low	1
C	H	Rotation accuracy	Low	1
C	H	Resolution	Low	1
C	H	Acceleration	Med	3
D	C	Power	Low	1
D	C	Exposure time	Low	1
D	E	Weight	Low	1
D	E	Size—surface area (near drum)	Med	3
D	E	Size—distance	Med	3
D	E	Exposure time	Med	3
D	H	Diodes type	Low	1
D	H	Exposure time	Low	1
E	A	Weight	Low	1
E	D	Moving Speed	Low	1
E	D	Size of moving mechanism	Med	3
E	H	Resolution	Med	3
F	A	Weight	Low	1
F	A	Size	Med	3
F	G	Plate load speed	Low	1
F	H	Load/unload rate	Low	1
F	H	Load/unload time	Low	1
G	B	Weight	Low	1
G	B	Punch force	Low	1
G	B	Beam pressure	Low	1
G	C	Gripper force—radial	Med	3
G	C	Gripper force—contact area	Med	3
G	C	Beam pressure	Med	3
G	H	System type	Med	3
H	D	Processing time	Low	1
H	E	Processing time—location	Med	3
H	E	Processing time—velocity	Low	1
H	E	Processing time—acceleration	Low	1

Riippuvuuksien ja vaikutus arvojen selvittämisen jälkeen seuraavana vaiheena on luvun 2 mukaisesti vaikutusarvojen summaaminen DSM matriisiin. Karniel ja Reich helpottavat vaihetta rakentamalla ensin matriisin, jossa kaikki komponenttien väliset suhteet on esitetty DSM:n mukaisessa järjestyksessä, mutta summaamista ei vielä toteutettu. Rakennettu matriisi on nähtävillä kuvassa 4.1, jossa on esitettynä samat tiedot kuin taulukossa 4.3, mutta matriisimuodossa. Kuvan matriisista nähdään samat parametrit ja niiden vaikutusarvot. DSM:n rakenteen mukaisesti diagonaalin on jo jätetty harmaaksi kuvaamaan, että toiminto ei voi vaikuttaa itseensä. Matriisiin on

jo tehty helpottavia summaamisia, sillä esimerkiksi komponentin H ja E väliset vaikutukset liittyvät kaikki käsittelyaikaan, joten parametri on kaikissa samankaltainen. Tällainen summaaminen perustuu yksityiskohtaisempaan vaikutusten tulkintaan, jonka avulla matriisiesitys helpottuu (Karniel & Reich 2011).

		Chassis & Cover	Drum	Drum Rotation system	Laser head	Writing head drive sys	Load /unload unit	Registration & punching unit	Control system
		A	B	C	D	E	F	G	H
Chassis & Cover	A		Weight 3 Size 9	Weight 1 Size 3 Max velocity 1		Weight 1	Weight 1 Size 3		
Drum	B			Inertia Moment 1				Weight 1 Punch force 1 Beam pressure 1	
Drum Rotation system	C		Weight 1 Inertia Moment 9		Power 1 Exposure time 1			Gripper force 6 Beam pressure 3	
Laser head	D					Speed 1 Size 3			Processing time 1
Writing head drive sys	E				Weight 1 Size 6 Exposure time 3				Processing time 5
Load	F								
Registration & punching	G		Size 6 Surface 1 Roughness				Plate load speed 1		
Control system	H			Rotation accuracy 1 Resolution 1 Acceleration 3	Diodes 1 Exposure time 1	Resolution 3	Load/Unload rate 1 Load/Unload Time 1	System type 3	

Kuva 4.1: Riippuvuuksien ja vaikutus arvojen esitys matriisissa (Karniel & Reich 2011, s. 204)

Kuvan 4.1 esitystavan etuna on, että nyt matriisin saman solun sisällä olevat vaikutusarvot voidaan summata yhteen, ja parametrien nimet voidaan poistaa matriisista kokonaan. Näiden toimenpiteiden jälkeen vaikutusarvoihin perustuva numeerinen DSM on valmis. Valmis DSM on esitetty kuvassa 4.2a. Nähdään, että solujen arvot vastaavat kuvan 4.1 matriisiesityksen summia, mutta diagonaalille on lisätty toiminnon F kohdalle vaikutusarvo 5. Diagonaalille lisätty arvo osoittaa, että toiminnon itseiteraation vaikutus voi kertoa epävarmuudesta kyseistä suunnittelutoimintoa kohtaan. Epävarmuus voi johtua esimerkiksi siitä, että käytössä on uusi teknologia tai komponenttia/toimintoa ei ole aiemmin suunniteltu. Diagonaalille osoitettu vaikutus muutetaan myöhemmin itseiteraation todennäköisyydeksi ja se voidaan osoittaa joko nykyiselle tai seuraavalle vaiheelle. (Karniel & Reich 2011)

	A	B	C	D	E	F	G	H
A	0	12	5		1	4		
B		0	1				3	
C		0	0	2			9	
D				0	4			1
E				10	0			5
F						5		
G		7				1	0	
H			5	2	3	2	3	0

(a)

	A	B	C	D	E	F	G	H
A	0.00	0.52	0.22		0.04	0.17		
B		0.00	0.04				0.13	
C		0.17	0.00	0.09			0.39	
D				0.00	0.17			0.04
E				0.43	0.00			0.22
F						0.22		
G		0.30				0.04	0.00	
H			0.22	0.09	0.13	0.09	0.13	0.00

(b)

Kuva 4.2: Vaikutusarvoihin perustuva DSM ja siitä muutettu todennäköisyyksiin perustuva DSM (Karniel & Reich 2011, s. 205)

Kuvaan 4.2b DSM on muutettu todennäköisyyksiin perustuvaksi DSM:ksi. Muutos on tehty luvun 2.4 mukaisesti lineaarisesti, jolloin korkeimman vaikutusarvon omaavalle solulle on osoitettu suurin todennäköisyys. Luvussa 2.4 kerrottiin, että todennäköisyyden todellisen maksimiarvon selvittäminen vaatisi lisää tutkimusta, joten maksimiarvo päätetään mielivaltaiseen arvioon perustuen. Tutkimusesimerkissä todennäköisyydelle on asetettu maksimiarvoksi p_{max} = 0,52. Kuvasta 4,2b nähdään, että kuvan 4.2a A ja B toimintojen välisen riippuvuuden arvoa 12 vastaa todennäköisyyden maksimiarvo 0,52, ja muut arvot on muutettu lineaarisesti vastaamaan samaa suhdetta. Toiminnolle F diagonaalilla osoitettu itseiteraatio on muutettu samaa tapaa noudattaen kuin muutkin DSM:n arvot. Kirjoittajat toteavat, että toiminnolle olisi voitu myös suoraan arvioida, mikä iteraation todennäköisyys on. Heidän mukaansa arvioitavaan todennäköisyyteen vaikuttaa valitut resurssit, joista esimerkiksi annetaan kokematon insinööri, jonka voidaan olettaa tekevän enemmän toistoja kuin kokenut insinööri. Itseiterointi ei vaikuta matriisin uudelleenjärjestämiseen, joka on tutkimusesimerkin seuraava vaihe, mutta itseiterointi vaikuttaa simulointiin. Karnielin ja Reichin (2011) mukaan itseiteraation voidaan tulkita implisiittisenä toteutetun toiminnon keston pidentämisenä. Tulkinta mahdollistaa tapausten paremman erottelun, oppimiskäyrän täytäntöönpanon sekä määritelmän yhteenlasketulle DB:n itseiteraation todennäköisyydelle, joka esiteltiin luvussa 4.2.

Matriisin uudelleenjärjestely toteutettiin esimerkissä luvussa 2.4 käsiteltyä optimointialgoritmia käyttäen. Karniel ja Reich (2011) käyttävät uudelleenjärjestämiseen komponenttien ja suunnittelutoimintojen yksi yhteen vastaavuuksia. Näiden seurauksena DSM rakenne osoitti, että komponentin F suunnittelutoimintoon ei vaikuta minkään muun toiminnon toteutus, minkä takia siitä tehdään ensimmäinen toiminto. Toiminnolla F aloittamisen etuna on myös, että toiminnon mahdolliset itseiteraatiot tapahtuvat heti prosessin alussa, ja näin ollen toiminnosta F johtuvia viivästyksiä ei tule prosessin keskellä. Komponentin A suunnittelutoiminto ei vaikuta muihin toimintoihin, mutta muut toiminnot vaikuttavat siihen. Tämän seurauksena toiminto A suoritetaan viimeisenä vasta, kun muut toiminnot ovat valmistuneet. Jäljelle jäävät toiminnot yhdistetään, ja ne muodostavat toimintasilmukan (BGCDHE). Optimoidun uudelleenjärjestämisen tuloksena saatu DSM on esitetty kuvassa 4.3, josta nähdään toimintojen uusi järjestys ja tummennetut toimintasilmukat. Kuvan DSM näyttää, että silmukoiden DB-toteutukselle on kaksi vaihtoehtoa: vahvistetuilla reunoilla eroteltu yksi isompi DB (BGCDHE) tai kaksi violetilla ja keltaisella pohjalla merkittyä DB:a (BGC ja DHE). Paremman vaihtoehdon selvittämiseksi on käytetty kaavan 1 kustannusfunktiota.

	F	B	G	C	D	H	E	A
F	0.22							
B		0.00	0.13	0.04				
G	0.04	0.30	0.00	0.00				
C		0.17	0.39	0.00	0.09			
D					0.00	0.04	0.17	
H	0.09		0.13	0.22	0.09	0.00	0.13	
E					0.43	0.22	0.00	
A	0.17	0.52		0.22			0.04	0.00

Kuva 4.3: DSM:n optimaalinen uudelleenjärjestely (Karniel & Reich 2011, s. 206)

Kaavan 1 avulla Karniel ja Reich selvittivät, kumpi DB toteutus on optimaalinen. Lähtötiedoiksi laskennalle annettiin seuraavat tiedot: F = 3, C = 64 ja q = 2,32. Toimintojen välisten linkkien arvot saatiin suoraan kuvan 4.3 DSM:n soluista, ja kuvasta nähdään myös mitkä toiminnot kuuluvat samaan klusteriin ja mitkä ovat eri klusterissa. Arvot sijoitettiin kaavaan 1, ja laskennassa saadut tulokset ovat nähtävissä taulukosta 4.4. Laskennan tuloksiksi saatiin yhdelle DB:lle 518,47 ja kahdelle DB:lle 412,22. Kahden DB:n vaihtoehto on näin ollen optimaalisempi vaihtoehto pienemmän kustannuksensa ansiosta.

Taulukko 4.4: Kustannusfunktiolla tehdyn DB laskennan tulokset (Karniel & Reich 2011, s. 206)

One design block (*BGCDHE*)	$6*(F*(0.3 + 0.17 + 0.39 + 0.13 + 0.22 + 0.09 + 0.43 + 0.22) + C*(0.13 + 0.04 + 0.09 + 0.04 + 0.17 + 0.13)) + F*(0.04*3\hat{}q + 0.09*6\hat{}q + 0.17*8\hat{}q + 0.52*7\hat{}q + 0.22*5\hat{}q + 0.04*2\hat{}q)$	518.47
Two design blocks (*BGC*) (*DHE*)	$3*(F*(0.3 + 0.17 + 0.39) + C*(0.13 + 0.04)) + 3*(F*(0.09 + 0.43 + 0.22) + C*(0.04 + 0.17 + 0.13)) + F*(.04*3\hat{}q + 0.09*6\hat{}q + 0.13*4\hat{}q + 0.22*3\hat{}q + 0.17*8\hat{}q + 0.52*7\hat{}q + 0.22*5\hat{}q + 0.04*2\hat{}q) + C*0.09*2\hat{}q$	412.22

Desing blockien täytäntöönpanossa yksittäistä DB:ia voidaan käsitellä yhdistettynä toimintana, jonka itseiteraatiolla on todennäköisyys ja kesto (Karniel & Reich 2007c, Karniel & Reich 2011 mukaan). Itseiteraation todennäköisyys on laskettu esimerkissä kaavan 4 avulla. Tämän jälkeen toiminnot (B,G,C,D,H,E) korvataan matriisissa design blockeilla (BGC ja DHE), ja niille lasketut itseiteraation todennäköisyydet merkitään matriisin diagonaaleille. Muutosten jälkeinen DSM on esitetty kuvassa 4.4. Design blockien todennäköisyydet ovat nähtävissä samanvärisellä pohjalla (violetti ja keltainen), kuin millä ne oli kuvassa 4.3 korostettu erottumaan muusta matriisista. Lisäksi DSM:iin on lisätty loogiset alku- ja lopetustoiminnot. Luvussa 4.2 todettiin, että DB voi koostua yhdestä tai useammasta toiminnosta. Näin ollen toiminnot F ja A voitaisiin myös käsitellä suunnittelulohkoina. Tällöin toiminnolla F on itseiteraatio samaan tapaan kuin muilla lohkoilla, eikä sen hallinta ole tavallisesta poikkeavaa. Toiminnon A ollessa DB, sillä on palautelinkki, joka täytäntöönpannaan osana prosessijärjestelmän mallia, mutta sen todennäköisyys on nolla. (Karniel & Reich 2011)

	Begin	F	BGC	DHE	A	End
Begin	0.00					
F	1	0.22				
BGC		0.04	0.70	0.09		
DHE		0.09	0.32	0.72		
A		0.17	0.63	0.04	0.00	
End					1	0.00

Kuva 4.4: DSM design block -muokkauksen ja niiden todennäköisyyksien jälkeen (Karniel & Reich 2011, s. 208)

Kuvan 4.4 DSM on samalla esimerkissä rakennetun ja muokatun matriisin viimeinen muoto, jota lähdetään muuttamaan DPM:ksi. Muutos toteutettiin samaan tapaa noudattaen kuin luvussa 3.2 esiteltiin. Muuttamisen jälkeen esimerkissä ei enää käytetä tai käsitellä DSM:ia, joten tämän työn kannalta ei ole olennaista käsitellä prosessin jäljellä olevia vaiheita yhtä yksityiskohtaisesti, vaan ne käydään lyhyesti läpi luvussa 4.6.

4.5 Vaihtoehtoinen tapa

Tutkimusesimerkissä DSM luodaan ja muokataan käyttäen design blockeja, mutta DSM pohjainen suunnitelma voitaisiin toteuttaa myös ilman niitä. Karniel ja Reich (2011) esittelevät, miten DSM toteutettaisiin ilman DB:ja. Luotu DSM on nähtävillä kuvassa 4.5. Toteutuksessa toiminnot F, B ja D aloittavat prosessin rinnakkain, koska yhdelläkään niistä ei ole edeltäviä toimintoja. Loput toiminnot saavat linkit eteen-/taaksepäin siten, että järjestys on kuvan 4.5 mukainen (FBDGCHEA).

	F	B	D	G	C	H	E	A
F	0.22							
B		0.00		0.13	0.04			
D			0.00			0.04	0.17	
G	0.04	0.30		0.00	0.00			
C		0.17	0.09	0.39	0.00			
H	0.09		0.09	0.13	0.22	0.00	0.13	
E			0.43			0.22	0.00	
A	0.17	0.52			0.22		0.04	0.00

Kuva 4.5: Vaihtoehtoisella tavalla ilman design blockeja luotu DSM (Karniel & Reich 2011, s. 213)

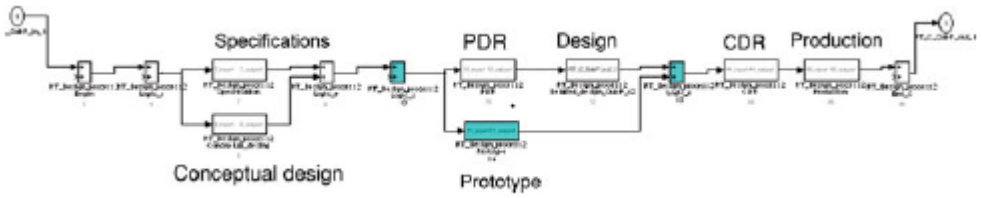
Toteutustapojen erilaisuudesta johtuen eri tavan seurauksena syntyviin prosesseihin syntyy eroja. Ilman design blockeja suunniteltu prosessi saa enemmän sarjassa olevan prosessin piirteitä, minkä seurauksena odotettu prosessin kesto on pidempi. Karniel ja Reich (2011) ajoivat simulaatiot molempien tapojen pohjalta tehdyille prosesseille ja huomasivat, että prosessin minimikeston ollessa huomattavasti pidempi vaihtoehtoisella tavalla, mutta maksimikesto lyhyempi. Lisäksi prosessin kesto vaihteli vähemmän. Resursseja tarvittiin puolestaan vähemmän, koska toteutustavassa hyödynnetään vain tarvittavat resurssit, kun taas DB toteutuksella kaikki toiminnot iteroivat, jolloin resursseja kuluu enemmän.
Vertailun johtopäätöksenä voidaan todeta, että jos prosessin keston tasaisuus on tärkeässä roolissa prosessin tuottaman arvon kannalta, niin tässä esimerkin tapauksessa kannattaisi toteutus tehdä ilman DB:ja. Myös resurssikustannukset olisivat pienemmät. Jos puolestaan prosessista halutaan mahdollisimman nopea, vaikka ongelmatilanteissa prosessi venyisikin erittäin pitkäksi, niin parempi vaihtoehto on tehdä DSM, kuten esitellyssä esimerkkitutkimuksessa se on tehty.

4.6 Tutkimusesimerkin loput vaiheet

DPM:n avulla Karniel ja Reich (2011) rakensivat tutkimusesimerkissään C-prosessin ja RT-prosessin. RT-prosessia simuloitiin 1000 ajon erissä, jotta voitiin tarkistaa simulaatioparametrien täytäntöönpano. Simulaation avulla selvitettiin prosessin kesto ja tarvittavat resurssit eri parametrien arvoilla, ja toteutuksessa oletettiin toimintojen kestojen olevan yhtä suuret. Simulaation tuloksista huomattiin, että prosessin minimikestolla oli suuri vaikutus prosessin kokonaisaikaan ja keskimääräiset resurssitarpeet nousivat, mutta suuren hajonnan takia tuloksissa ei ollut tilastollista eroa.

Simulaatioon lisättiin seuraavaksi oppimismalli, jonka vaikutuksesta toiminnon suorittamiseen kuluva aika lyhenee. Toiminnon suoritusaika voi lyhentyä kertyneen kokemuksen seurauksena tai jos vain osa toiminnosta joudutaan tekemään uudestaan. Oppimismallin lisäämisen seurauksena simuloinnin tulokset muuttuvat siten, että kuluneen ajan maksimi ja keskiarvo pienenevät, sekä prosessin keston hajonta pienenee. Iteraatioilla on pienempi vaikutus prosessin kestoon, koska niistä johtuva ajan lisäys on pienempi.

Simulaatioiden jälkeen high-level prosessin käsitteellinen ja yksityiskohtainen suunnitteluvaihe ovat suoritettu. Vaiheiden jälkeen prosessissa siirryttiin prosessin ensimmäiseen arviointipisteeseen. PDR:n ja sitä seuranneen suunnittelun rinnalle lisättiin prototyyppitoiminto, jonka tarkoituksena on tarkastaa uuden laser diodi -teknologian täytäntöönpano. Karniel ja Reich (2011) suunnittelivat, että prototyyppitoiminto alkaa samanaikaisesti PDR:n kanssa, mikä mahdollistettiin tekemällä high-level prosessin tapauskohtainen muutos. Prototyyppitoiminnon lisääminen high-level prosessiin on esitetty kuvassa 4.6. Lisäksi kuvasta nähdään siniharmaalla merkittynä prototyyppitoiminnon kanssa samanaikaisesti lisätyt loogiset toiminnot Split-And sekä Join-And, jotka mahdollistavat toimintojen suorittamisen rinnakkain.



Kuva 4.6: High-level prosessi, johon lisätty prototyyppitoiminto (Karniel & Reich 2011, s. 215)

Kuvasta 4.6 voidaan todeta, että CDR arviointipistettä ei voida suorittaa ennen kuin suunnitteluja prototyyppitoiminto ovat suoritettu. Tämän jälkeen on mahdollista siirtyä tuotantovaiheeseen.

Yhteenvetona esimerkistä Karniel ja Reich (2011) toteavat, että se demonstroi tuotekehitysprosessin dynaamisia muutoksia. Prosessi kehitettiin vähitellen yksinkertaisesta yhden toiminnon omaavasta prosessista esimerkkitapaukseen parhaiten sopivaksi high-level prosessiksi. Kun ensimmäiset tuotetiedot saatiin, muutettiin saatu tieto prosessisuunnitelmaksi ja sen malliksi. Tuotetiedon pohjalta luotiin ensimmäinen DSM. Päivitetyn tuotetiedon avulla rakennettiin todennäköisyyksiin pohjautuva DSM, joka järjestettiin uudelleen käyttäen optimointialgoritmia. Lisäksi kirjoittajat esittivät DB:ien käyttämisen. Tämän jälkeen useita toimintojen välisiin suhteisiin liittyviä sääntöjä käytettiin, jotta DSM-rakenne saatiin muutettua prosessisuunnitelmaksi. Simulaation avulla ohjattiin päätöksentekoa eri täytäntöönpanon tapojen välillä. Lopuksi esimerkissä kerrottiin, miten dynaamiset muutokset tuotteen rakenteessa aiheuttavat dynaamisia muutoksia koko prosessiin ja todettiin, että simulaation avulla voidaan osoittaa prosessin hallinnan ongelmia.

4.7 DSM:n käytössä havaitut ongelmat

Tutkimusesimerkin pohjalta Karniel ja Reich (2011) kertovat DSM:iin liittyvistä ongelmista, jotka vaativat lisää kehittämistä. DSM:iin tarvittavat tiedot oletetaan olevan saatavilla, mutta todellisia lukuja, joiden pohjalta laskennat tehdään ei ole kirjoittajien mielestä käsitelty riittävästi DSM-kirjallisuudessa. Kirjoittajat kertovatkin havaitsemiaan kehityskohteita, jotka esitellään seuraavaksi.

DSM:n uudelleenjärjestäminen kaavan 1 kustannusfunktiolla pakottaa yksinkertaisen toimintasilmukan palautearvot pienemmiksi tai yhtä suuriksi kuin eteenpäin menevien linkkien. Tämä väite osoittaa, että linkin tarkalla arvolla ei ole merkitystä toiminnot uudelleenjärjestävälle algoritmille, ja vain arvojen väliset suhteet vaikuttavat. Karniel ja Reich (2011) toteavat, että ylhäältä alaspäin lähestymisellä vain kolme arvoa voisi riittää uudelleenjärjestämiseen. Yksityiskohtaisempi lähestymistapa ja suurempi arvojen vaihtelu voi olla hyödyllistä, kun harkitaan DSM:n jakamista DB:hin.

Todennäköisyyksien laskeminen yhdistetyille toiminnoille kaavan 4 avulla perustuu todennäköisyyden esiintymisen laskemiseen itsenäisille tapahtumille. Lähestymistapa on Karniel ja Reichin (2011) mielestä houkutteleva, koska se mahdollistaa DB:ien todennäköisyysarvioiden luomisen ja säilyttää kommutatiivisuuden ja assosiatiivisuuden. Lähestymistapa on kuitenkin validoitava mittauksilla. Toinen vaihtoehto, jota voitaisiin käyttää, on kirjoittajien mukaan pelkkien eteenpäin menevien linkkien käyttö yhdistettävinä linkkeinä, ja palautteen ja itseiteraation todennäköisyyksien laskeminen näille joukoille.

Tutkimuksessa oletetaan, että todennäköisyyksien arvojen tarkkuus ei välttämättä ole tärkeää, kun käytetään ehdotettua runkoa päätöksenteon apuna ja suhteelliset todennäköisyyksien arvot ovat riittävän hyvät. Oletus vaatisi kirjoittajien mielestä lisää testausta ja validointia.

Lisäksi algoritmi alisilmukoiden uudelleenjärjestämiseen voisi olla hyödyllinen optimaalisen algoritmin hakutilan pienentämiseksi. Näiden algoritmien integroimista tulisi Karniel ja Reichin mielestä tutkia lisää. Hakutilan rajoittaminen on heidän mukaan houkuttelevaa laskennallisesta näkökulmasta. Toisaalta alisilmukan uudelleenjärjestämisessä tyypillisesti pyritään mahdollisimman vähäisiin iteraatioihin, mikä ei välttämättä ole optimoinnin tavoite.

5. YHTEENVETO

Työssä referoitiin Arie Karnielin ja Yoram Reichin vuonna 2011 julkaistua kirjaa “Managing the Dynamics of New Product Development Processes-A New Product Lifecycle Management Paradigm”. Työn tarkoituksen mukaisesti kirjasta referoitiin design structure matriiseja käsittelevät osat. Opittiin ymmärtämään, mikä DSM on ja mitä hyötyjä sen käytöstä prosessin suunnittelussa voidaan saavuttaa. DSM esiteltiin työkaluna, jonka avulla on mahdollista tarkastella prosessin eri toimintojen välisten riippuvuuksien suhteita, ja käyttää saatua tuotetietoa prosessin toimintojen optimaalisen työjärjestyksen löytämiseen. Optimaalisen työjärjestyksen löytämiseksi on olemassa suunnittelualgoritmeja, jotka toimivat ohjeina matriisin uudelleenjärjestämiselle. DSM:n rakenteen pohjalta voidaan luoda prosessisuunnitelma, jota puolestaan voidaan simuloida ennen päätöstä kyseisen suunnitelman toteuttamisesta.

Yhtenä työn tavoitteista oli selvittää, miten Karnielin ja Reichin mielestä DSM:a voidaan käyttää koko tuotteen elinkaaren aikana. Tähän kysymykseen kirja ei antanut selkeää vastausta, sillä kirjoittajat keskittyivät DSM:n käyttämiseen uuden tuotteen suunnittelussa. Kirjassa kerrotaan DSM:n käytöstä pääsääntöisesti vain tutkimusesimerkkinä tehdyssä LDI-levyn suunnitteluprosessissa, eikä työkalun soveltamisesta muihin käyttötarkoituksiin saa kunnollista kuvaa. Näin ollen kirjassa korostui DSM:n merkitys tuotteen suunnitteluvaiheessa eli elinkaaren alkuvaiheissa. Tästä pääteltiin, että DSM käyttö on hyödyllisintä, kun prosessia tuotteen valmistamiseen ei vielä ole olemassa. Toisaalta DSM:n rakentamiseen käytetty tuotetieto päivittyy ajan mittaan, eli sitä mukaa kun tuote liikkuu elinkaarensa aikajanalla eteenpäin, mikä voi johtaa dynaamisiin muutoksiin matriisissa ja sitä kautta koko prosessissa. Koko elinkaaren ajalta saatava tieto siis vaikuttaa DSM:n rakenteeseen, mutta muutos prosessissa ei välttämättä vaikuta tuotteeseen josta tieto on saatu, ellei kyseessä ole tuotteen huoltoa tai kunnossapitoa koskeva prosessi. Muulloin tiedon avulla voidaan kehittää tuoteprosessia siten, että mahdollisilta ongelmilta vältytään tai tavoiteltavat parannukset saadaan toteutettua valmistukseen tulevien tuotteiden kohdalla.

Kokonaisuutena työn tavoitteissa onnistuttiin hyvin, vaikka kirja ei antanutkaan selkeää kuvaa DSM:n koko elinkaaren aikaiseen hyödyntämiseen. Paremman näkemyksen saaminen DSM:n käytöstä koko tuotteen elinkaaren aikana vaatii tutustumista muihin asiaa tarkemmin käsitteleviin lähteisiin, mutta kirjasta saatiin tarvittavat perustiedot DSM:n ymmärtämiseen ja sen hyötyihin prosessisuunnitelman teon apuvälineenä.

LÄHTEET

Abdelsalam, H. & Bao, H. (2006). *A simulation-based optimixation framework for product development cycle time reduction.* IEEE Trans Eng Manag 53(1), pp. 69–85.

Browning, T. R. (2001). *Applying the design structure matrix system to decomposition and integration problems: A review and new directions.* IEEE Trans Eng Manag 48, pp. 292–306.

Browning, T. R. & Eppinger, S. D. (2002). *Modeling impacts of process architecture on cost and schedule risk in product development.* IEEE Trans Eng Manag 49(4), pp. 428–442.

Clarkson, P. & Hamilton, J. (2000). *Signposting A parameter-driven task-based model of the design process.* Res Eng Des 12(1), pp. 18–38.

Eppinger, Whitney, Smith & Gebala. (1994). *A Model-based method for organizing.* Res Eng Design 6(1), pp. 1–13.

Karniel, A. & Reich, Y. (2007). *A coherent interpretation of DSM plan for PDP simulation. In proceedings of the international conference on engineering design.* ICED 07, Paris, August.

Karniel, A. & Reich, Y. (2007). *Managing dynamic new product development processes. In: Proceedings of the 17th annual international symposium of the intrenational council on systems enguneering INCOSE'07.* San Diego, CA, June.

Karniel, A. & Reich, Y. (2007). *Simulating design processes with self-iteration activities based on DSM planning.* IEEE Proceedings international Conference Systems Engineering Model, ICSEM'07, 33–41, Haifa.

Karniel, A. & Reich, Y. (2009). *From DSM based planning to design process simulation: a review of procedd scheme logic verification issues.* IEEE Trans Eng Manag 56(4), pp. 636–649.

Karniel, A.;& Reich, Y. (2011). *Managing the Dynamics of New Product Development Processes- A New Product Lifecycle Management Paradigm.* Springer.

Karniel, A. Belsky, Y. & Reich, Y. (2005). *Decomposing the problem of constrained surface fitting in reverse engineering.* Comput Aided Des 37, pp. 399–417.

Reich, Y. & Fenves, S. (1992). *Inductive learning of synthesis knowledge.* Int J Expert Syst: Res App 5(4), pp. 275-297.

Sadiq, W. & Orlowska, M. (1999). *Appluing graph reduction techniques for identifying structural conflicts in process models.* Lect Notes Comput Sci 1626, pp. 195–209.

Sered, Y. & Reich, Y. (2006). *Standardization and modularization driven by minimizing overall process effort.* Comput Aided Des 38(5), pp. 405–416.

Smith, R. & Morrow, J. (1999). *Product development process modeling.* Des Stud 20(3), pp. 237–261.

Stark, J. (2005). *Product lifecycle management: 21st century paradigm for product realization.* New York: Springer.

Subrahmanian, E. Sudarsan, R. Fenves, S. Foufou, S. & Sriram, R. (2005). *Product lifecycle management support: A challenge in supporting product design and manufacturing in a networked economy.* Int J Product Lifecycle Manag 1(1), pp. 4–25.

Sääksvuori, A. & Immonen, A. (2008). *Product lifecycle management, 3rd edn.* Springer.