KEYWORDS product design · axiomatic design · design structure matrix · systems decomposition

The concept of DSM AND ADT INTEGRATION in the PRODUCT DESIGN PROCESS

Mohamed-Larbi Rebaiaia
Department of Mechanical Engineering, University of Laval, Quebec (QC), Canada
mohamed-larbi.rebaiaia@cirrelt.ca

Darli Rodrigues Vieira
Université du Québec à Trois-Rivières, Québec, Canada
darli.vieira@uqtr.ca

ABSTRACT

In manufacturing, product design is often very complex and requires the participation of a large number of services and material resources. Managing these resources can be very costly in terms of time, risk and money. The use of well-known tools (e.g. PERT, CPM and Gantt) alone cannot establish a detailed and optimized planning in the execution of different stages and tasks of a project. To address this problem the Design Structure Matrix (DSM) and the Axiomatic Design Theory (ADT) appear to be an interesting solution. This paper presents a DSM and ADT integrated tool as a part of the decomposition–integration problem in product design development process, where the latter is more concerned with mapping customer's needs from functional requirements to design parameters, while the former is better suited to modelling the interactions and the integration of the design parameters. It also presents some algorithms related to both DSM and ADT used for manipulating projects-based DSM information, coordination and timing requirements, and provides a complementary between them. The discussion in this paper addresses the implementation of concurrent design engineering which is the greatest challenge faced by design managers.

1. Introduction and Definitions

To remain competitive in the current economy manufacturers have been encouraged the creation of departments to promote research, design, and new product development *(NDP)*. Without an efficient policy for new product development, companies may disappear. Continuous innovation in product development *(PDP)* appears to be the success key even though it is complicated, uncertain, dynamic and very risky. The most competitive companies are always looking for new techniques and technologies to market their products that must be better in terms of quality, competitively priced, and have a time of design and manufacturing as short as possible. Kneuper *(2007)* defines the PDP management as a processes control with respect to "three sacred cows": "time, quality and budget". According to these three requirement factors, decreasing the project time or increasing the project quality influence the final project cost. However, the design process proves to be crucial in the factor risk view. One solution to coordinate between these factors is to provide design window flexibilities in order to put back on the process itself and reworking critical parts of the design *(project)* and thus avoiding defects or inconsistencies that may arise during the execution phases. Another admissible solution works for decreasing PDP working time. If possible it avoids that all process tasks are performed independently by adopting the concept of concurrent engineering as a strategy for achieving shorter time to market, reduced development costs, and higher-quality products. Engineers admit that concurrent engineering process do not only consider performance and process design issues, but many other lifecycle issues such as reliability, environmental impact, service, testing, and so on *(1995)*. In practice several companies have reported that by implementing the concept of concurrent engineering as a new method of managing construction equipment projects, they achieved 30% savings in development costs and 60% savings in development time. Defining the PDP process using mathematical frameworks in addition to information exchange processing and computer's facilities could be an advantage for design, communication and coordination.

Several authors define the PDP process as a collection of methods and procedure that companies use to concretize new ideas, market opportunities to design new products, or to improve others and bring them to the market. Traditionally, a PDP process reflects the specific design requirements, objectives and constraints, and follows a series of iterative steps known as product lifecycle stages. A sketch of a series of stages for product cycles that are well-known, stable and under control is shown in **Figure 1** *(2009)*. Nonetheless, PDPs can also vary dynamically so that sophisticated techniques for minimizing the defaults risk and improving sales have to be applied. Krubasik *(1998)* argues that not all product development is similar. In this topic, Zamenopoulos and Alexiou *(2007)* propose a clever spiral form of PDP staged process which requires managers to evaluate risk before starting a new step of the process. A version of the spiral PDP introduced by Barry Boehm *(1986)* is illustrated in **Figure 2**. Note that this version is similar but more detailed than the Xerox copier/printer software presented in Unger and Eppinger *(2009)*. It clearly shows that feedback-based design is still possible for reaching managing objectives.

A PDP planning project involves the completion of hundreds or even thousands parts or entities not simply coordinated or structurally well-linked.

PDP projects are very complex and especially dynamic. Besides, different disciplines and domains can influence the behavior of a product design or a project, therefore, managers and engineers must understand how they interrelate with and influence each other. Danilovic and Brownig *(2004)* give an overview of the environment that interacts with PDP.

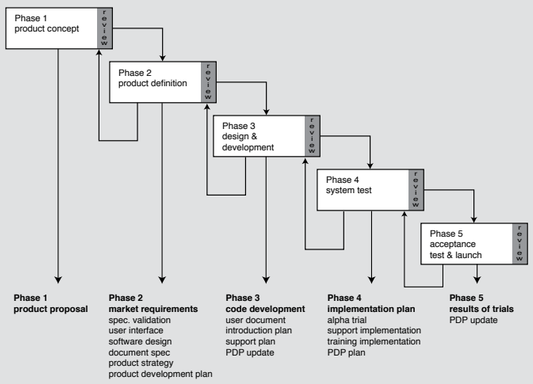


FIGURE 1. *A Software PDP Lifecycle planning*

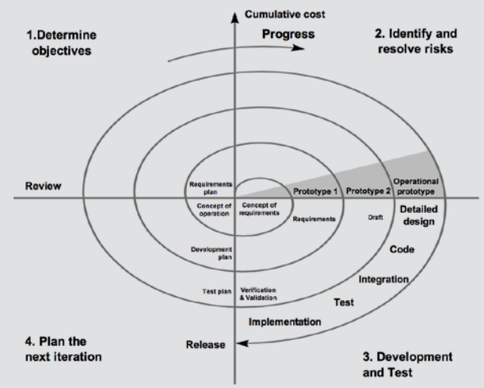


FIGURE 2. *Boehm General Spiral Process (Software Product) [from http://commons.wikimedia.org/wiki]*

Product specifications are a consequence of customer requirements, and many other events that contribute to the success or failure of a project. **Figure 3** illustrates the four domains defined in a PDP process. They are part of the axiomatic design methodology and work such that for each pair of adjacent domains, the left domain represents "what we want to achieve" while the right one represents the design solution of "how we propose to achieve it". The four domains are:

Customer	The benefits customers seek.
Functional	Functional requirements of the design solution
Physical	Design parameters of the design solution
Process	Process variables

Danilovic and Browning *(2004)* precise that this process is like a web such that domains are interrelated and information is flowing back and forth between them.

Many attempts have been applied to use different paradigms to represent the structure of a PDP components in order to reduce as much as possible the relationship between them. One of those solutions is attributed to Steward *(1981)* who introduced the Design Structure Matrix *(DSM)* which is also known as the dependency structure matrix. An alternative representation of the PDP uses the axiomatic design principles, which according to Suh *(1990)*, offers a systematic approach to product design and production planning.

This paper presents two complimentary integrated tools named DSM and ADT as a part of the decomposition–integration problem in product design development process, where the latter is more concerned with mapping customer's needs from functional requirements to design parameters, while the former is better suited for modelling the interactions and the design parameters integration. The main objective of this work concerns first the decomposition process as a part of concurrent engineering techniques. The following section presents a brief summary of concurrency and decomposition. Section 3 details DSM tools and algorithms. ADT methodology is then highlighted in Section 4, while Section 5 shows that the combination of DSM and ADT will generate a more complete tool for designing and organizing the design process of products. Finally, conclusions are drawn and future work outlined.

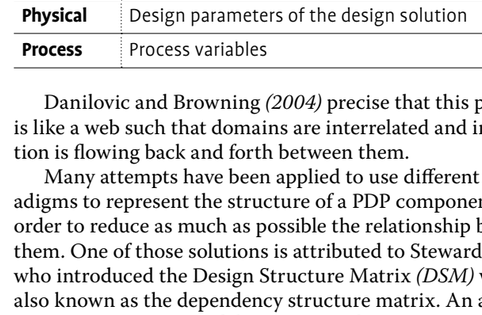


FIGURE 3. *Domains concept in a PDP process*

2. Concurrency and PDP Decomposition

The process of developing new products consumes time, money and human resources, and it is a hard task. The cost of a product development is proportional to the project's resources *(people, machinery, materials)* and duration. When performed sequentially, the effort that is developed in the project often ends up accumulating rework and increasing different types of risks. Of course, this way of working also has impacts on the composition of project costs. In order to minimize the duration of the product development, solutions such as concurrent engineering *(CE)* for products, and concurrent construction *(CC)* in civil engineering and architecture have been widely used in many projects.

Sequential engineering *(SE)* considers that a project is composed by tasks that are executed one after another, and the only constraint that binds them is the precedence information relationship *(exchanges)*. Three types of relationships have been reported by Ulrich and Eppinger *(2004)*: independent, dependant and interdependent. They are also named, "serial", "parallel" and "coupled" *(see* **Figure 6***)*. One obvious thought is that, if PDP process consists of independent tasks, fewer problems may arise by concurrently executing these tasks and the problem becomes manageable. According to Yassine and Braha *(2003)* the CE principles are: iteration, parallelism, decomposition and stability. The advantage of decomposing a PDP design is that it reduces the complexity of the design process *(1993)*.

The main principle of decomposing a product design process is to break it into groups of activities *(subprocess)* while detecting potential activities that could be worked simultaneously *(interleaving parallelism)* or in parallel *(free parallelism)*. Pimmler and Eppinger argue *(1994)* that the device's function is decomposed into multiple sub-functions allowing the team to find solutions for each of the smallest pieces of the design. They propose a three-step method which works as follows: 1 - decomposition of the system into elements; 2 - documentation of the interaction between elements, and clustering the elements into architectural and team chunks. The literature on decomposition process has not reflected the importance of this problem yet, and it is not very extensive except in software engineering systems. Research concerning PDP decomposition started with the work of Alexander *(1964)*, in which it was proposed a PDP process that decomposes designs into minimally coupled groups. Simon *(1981)* suggested that complex designs can be organized using hierarchical structures consisting of nearly decomposed systems. Johnson and Benson *(1984)* developed a two-stage decomposition strategy for design optimization. The strategy is restrictive by considering that sub-processes are independent. Azarm *(1987)* applied decomposition to problem solving using the concept of monotonicity. Rogers *(1989)* incorporated the decomposition principle when conducting research at NASA.

As PDP process may involve coordination of many activities and procedures, problems may arise when describing interactions between the generated sub-parts. Many solutions have been proposed in order to overcome such problem. Warfill and Hill *(1972)* discuss about this issue and suggest that the solution of complex designs lies in understanding and controlling the interactions between the elements. Galbraith *(1973)* proposed a clear distinction between the organization structure and the problem to be solved. Later, Ulrich and Eppinger *(2004)* proposed a solution related to the interactions between decomposed parts of a system so that they can be considered after the architecture is defined.

	A	B	C	D	E	F	G	H
1	X	X	X					
2	X	X	X					
3				X	X			
4				X	X			
5				X	X			
6						X	X	X
7						X	X	X

	A	B	C	D	E	F	G	H
1	X	X	X				X	X
2	X	X	X				X	X
3				X	X	X	X	X
4				X	X	X	X	X
5				X	X	X	X	X
6				X	X	X	X	X
7				X	X	X	X	X

	A	B	C	D	E	F	G	H
1	X	X	X					
2	X	X	X					
3				X	X	X		
4				X	X	X		
5				X	X	X		
6	X	X	X	X	X	X	X	X
7	X	X	X	X	X	X	X	X

FIGURE 4. *Three types of matrices: (1st decomposable, 2nd non-decomposable with overlapping parameters, 3rd non-decomposable with overlapping activities).*

Two well-known tools could be used to represent the information execution of a PDP *(project)*, and they are matrices and graphs. Why matrices and graphs? The answer is simple-- they are natural, visual and could be easily used to represent our thoughts, and are mathematically well-founded. For instance, matrix representing a PDP task and its relationships can be categorized as decomposed in such a way that rows and columns can be grouped in a form that the matrix can be separated into mutually exclusive sub-matrices as shown in **Figure 4** *(first matrix. An entry "X" in this matrix denotes that parameter j (j = A, B, C, …, H)* appears in activity i *(i =1,…, 7)*).

The matrices exhibited in **Figure 4** show three characteristic patterns. The first one presents a simple configuration which clearly highlights three independent subsystems. Moreover, if one draws the graph three sub-graphs can be seen without any interaction with each other *(connected components as defined in graphs theory)*. The others are more difficult to be distinguished as activities and parameters are overlapped. A good solution is precisely to minimize as much as possible the overlapping.

To deal with the overlapping activities, Kusiak *(1993)* proposed four actions which are: 1 - replacing an overlapping activity with an alternative activity that involves different parameters; 2 - decomposition of an activity into sub-activities; 3 - removing an overlapping activity whenever is possible, and 4 - the duration of an overlapping activity can be shortened.

Kusiac et al. *(1993)* proposed two interesting algorithms; they are called Cluster Identification *(CI)* and Branch-and-Bound Algorithms. The CI algorithm proceeds as follows:

Cluster-Identification-Algorithm.

Inputs:

- Incidence matrix as showed in **Figure 5** (first matrix).
- Two Boolean vectors V-rows and V-columns are declared (they will memorize the marking).

Step 0. Set iteration k = 1.

Step 1. Select row I of the incidence matrix and mark and put "1" in the position I of V-rows.

Step 2. For each entry of "X" crossed by the horizontal line ("1" in V-rows) put "1" in V-columns.

Step 3. For each entry of "X" crossed by the vertical line ("1" in V-columns) put "1" in V-columns.

Step 4. Repeat step 2 and step 3 until there are no more crossed-ones of "X" in the matrix.

Step 5. Transform the initial incidence matrix into another one by removing rows and columns marked in V-rows and V-columns.

Step 6. If the last matrix is empty, stop, otherwise increment k and go to step 1.

	A	B	C	D	E	F	G	H
1		X	X		X			
2	X					X		
3				X			X	
4	X					X		
5		X		X				
6			X					
7		X	X		X			X

	B	C	E	H	A	F	D	G
1	X	X	X					
5		X	X	X				
7	X	X	X	X				
2					X	X		
4					X	X		
3							X	X
6							X	

FIGURE 5. *(1st) example of a design matrix, (2nd) resulting matrix after clustering algorithm application.*

3. Design Structure Matrix

The design structure matrix *(DSM)* also known as dependency structure matrix and its manipulating algo-

rithms, was originally developed by Steward *(1981)* to analyse information flow. DSM has been widely used in managing complex projects in various field of study such as building construction, semiconductor, automotive, aerospace, telecom, manufacturing, factory equipment, and so on. The DSM is a useful tool for optimizing the components composition of product development in terms of minimizing interfaces. It has the ability to analyse functions, facilitate the tracking of components in a new design, decompose a system into elements, integrate elements into modules or subsystems and can, for example, perform an optimal sequencing of tasks that need to be performed as part of a product development effort *(Fernandez, 1996)*. Concretely, DSM is just a matrix structure which does not express all the relevant information required for defining logic process, but it is a general method for representing and analyzing system models to better plan complex projects involving interdependencies, facilitating modularity, sequencing to minimize cost and schedule risk in a variety of application areas. DSM representation evolved later to Domain Mapping Matrices *(Danilovic & Browning, 2004)* and Multiple-Domain Matrices *(Gebala & Eppinger, 1991)*. The DSM-based project plan can be converted to a graph-based model or a graphical evaluation and review technique *(graphical PERT)* for representing the static process structure and the dynamic progress logic and calculus. **Figure 8** shows some types of activity relations in DSM and their corresponding representations in a graph represented in **Figure 7**. A DSM can be modeled by an incidence matrix as defined in graph theory. In the DSM, like a project task or a system component, the relationships between activities are represented by marking the cell formed by rows and the corresponding columns. For example, if there is an arrow from Node A to Node B, then a 'X' mark is placed in the intersection cell of Row A and Column B. This means that activity B follows activity A. Diagonal elements have no significance and are normally blacked-out *(see* **Figure 8***)*. A design process is composed of three types of basic behavioral patterns, namely serial *(dependent or decoupled)*, parallel *(independent or uncoupled)*, and iterative *(interdependent or coupled)* ones. **Figure 6** gives a graphical representation of these 3 types of design.

The DSM is most useful when activities are listed in the order of their execution in the project. Changing the order of activities is called partitioning and a partitioned DSM shows which tasks are parallels, series or coupled. The problem of DSM model is its static representation of activities and the lack in exhibiting the timing requirements of tasks as project duration enriches a DSM model such as, for example, to apply simple heuristics directly on the activities graph such as to evaluate the minimal and maximal duration of a project. We can remark that the project information cannot be used directly from the DSM unless we use some algorithms to manipulate them.

According to the mathematical side of DSM interpretation, a DSM associated with a graph is a binary square matrix with m rows and m columns, and the number of marked cells corresponds to the links between nodes which are admitted equal to n. The easiest way to model a directed graph in terms of DSM is to use a binary matrix so that the DSM can be defined as follows:

Definition 1. The DSM is a Boolean matrix $A = [a_{ij}]_{n\times n}$ composed of elements such as each element is defined according to *(1)*.

$$a_{ij} = \begin{cases} 1 \text{ if } (a_j \rightarrow a_i) \\ \text{otherwise} \end{cases} \qquad (1)$$

where a link $(a_j \rightarrow a_i)$ between a_j and a_i denotes that component a_j transfers information to component a_i so that the execution of a_i cannot be proceeded if a_j did not complete its execution. The diagonal cells are blackened. An empty row represents a source node and an empty column represents a terminal node.

Figure 8 presents the DSM correspondent to the graph in **Figure 7**. The product consists of 12 subsystems labeled Modules A, B, C,..., L.

Note that once the design process has been mapped into a DSM, two specific algorithms are applied. The first one concerns the partitioning which consists of re-sequencing the design activities to maximize the coordination between them and to detail the circular path supporting the information exchange. The algorithm identifies the activities in the loop and clusters them as a block on the diagonal of the design matrix *(Gebala & Eppinger, 1991)*[9]. The second algorithm *(tearing)* is to re-sequence within the blocks of coupled activities to find an initial ordering to start the iteration which consists in the removal of dependency between coupled tasks.

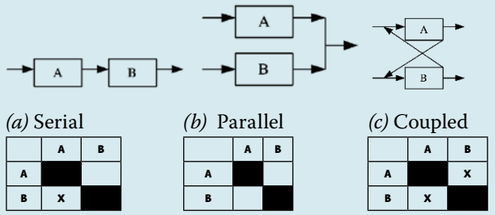


FIGURE 6. *Activity Networks and DSM equivalence*

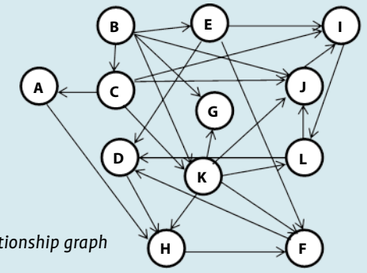


FIGURE 7. *Any relationship graph*

	A	B	C	D	E	F	G	H	I	J	K	L
A			1									
B												
C		1										
D		1			1	1						1
E						1		1			1	
F		1										1
G		1									1	
H	1			1							1	
I			1			1				1		
J		1	1								1	1
K		1	1									
L	1								1	1	1	

FIGURE 8. *DSM corresponding to the graph in the Figure 7.*

3.1 Partitioning a DSM

Partitioning is a special operation that re-orders and re-groups the DSM rows and columns such that the new DSM arrangement does not contain any feedback marks. For complex engineering systems, partitioning algorithms try to move feedback marks as close as possible to the diagonal creating blocks. Collapsing these blocks into single task simplify the form of the matrix and thus those of the project.

There are several heuristics used in DSM partitioning. However, they are all similar, with a difference in how they identify information cycles *(loops or circuits)*. The main objective of such heuristics is attempting to find a sequence of the design activities which allows the DSM matrix to become lower triangular. If the activity tasks can be sequenced so that each one begins only after it receives all the information it requires from its predecessors, then no coupling remains in the design problem.

The following algorithms *(see the three boxes below)* decompose a design matrix if possible.

The grouping of components into a class or a group of components can be understood as the creation of a module composed by the merging of these components. Algorithm 3 consists of identifying loops *(blocks)* of information by powers the adjacency Matrix.

Algorithm 1 (Partitioning)

1. Identify components without input from the rest of the elements in the matrix (source nodes). Those elements can easily be identified by observing an empty column in the DSM. Place those elements to the left of the DSM. If there is more than one source node, put them according to their position in the initial matrix. Once an element is rearranged, it is removed from the DSM (with all its corresponding marks) and step 1 is repeated on the remaining elements.

2. Identify the components that deliver no information to other elements in the matrix (target nodes). Those elements can easily be identified by observing an empty row in the DSM. Place those elements to the right of the DSM. Once an element is rearranged, it is removed from the DSM (with all its corresponding marks) and step 2 is repeated on the remaining elements.

3. If after steps 1 and 2 there are no remaining elements in the DSM, then the matrix is completely partitioned; otherwise, the remaining elements contain information circuits (at least one).

4. Determine the circuits using any method.

5. Collapse the elements involved in a single circuit into one representative element and go to step 1.

Example 1 – Application of Algorithm 1

	A	B	C	D	E	F	G	H
A				1				1
B					1		1	
C					1			
D						1		1
E		1	1				1	
F	1							1
G		1	1					
H	1			1				

FIGURE 9. *Sample DSM.*

	A	H	F	D	E	C	G	B
A		1		1				
H	1			1				
F	1	1						
D		1	1					
E						1	1	1
C					1			
G						1		1
B					1		1	

FIGURE 10. *Partitioned DSM of figure 11 (first step).*

	A-H-F-D	E-C-G-B
A-H-F-D	1	
E-C-G-B		1

FIGURE 11. *Reduced DSM (final step).*

Algorithm 2 (Partitioning)

```
Input: DSM matrix;
S.Modules = null;
    Let n be its dimension
    Identify components without inputs and
    put them in a queue list;
    Start node v (the last one in the queue)
    index = 0 /* nodes number counter */
    Stack = empty /* empty stack of nodes */
    forall v in DSM
     if (v is an unvisited node
       call depth_1st_search(v)
    until (u == v)
end.
procedure depth_1st_search(v)
      index (v) = index
      lowlink(v) = index
      index = index + 1
      push(v) /* Push v on the stack */
      L = Children(v) /* node v successors */
      forall (v, u) in L
       if u not visited yet?
        depth_1st_search(u)
       lowlink(v)= min(lowlink(v), lowlink(v)
       elseif (u in Stack)?
        lowlink(v) = min(lowlink(v), u.index(u))
 if (lowlink(v) == index(v)?
  Copy Stack in  S.Modules
       repeat
        u = Stack.pop
      end
```

3.2 Tearing a DSM

Once the blocks are identified and placed into a block-triangular form, the activities have to be sequenced within the blocks by removing the dependency relationship temporarily. As the last decision of which task must be discarded is up to the manager's judgement, the result of tearing will be highly sensitive from one person to another. To obtain more information and a detailed description of the tearing process, we strongly encourage the reader to read the work of Browning and Eppinger *(2002)*.

Example 2 – Application of Algorithm 2

	A	B	C	D	E	F	G	H	I
A									
B			1						1
C						1		1	
D							1	1	
E	1								1
F	1				1				
G			1						
H		1						1	
I						1			

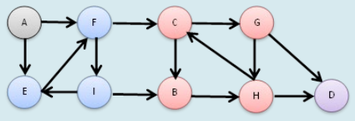


FIGURE 12. *Graph corresponding to DSM of figure 13.*

	E	F	I	C	B	G	H	D
A								
E	1		1					
F	1	1						
I			1					
C			1				1	
B				1	1			
G				1				
H					1	1		
D						1	1	

FIGURE 13. *Partitioned DSM obtained with algorithm 2.*

Algorithm 3.

1. Exclude from the DSM every row and every column because they does not require information from other tasks and provide no information to the other tasks.

2. Multiply the matrix by itself to determine loops.

3. Stop when reaching the higher power.

Example 3 – Application of Algorithm 3

	A	B	C	D	E	F	G
A			1	1			
B							1
C	1	1				1	1
D		1					
E			1			1	
F							
G				1		1	

1. Select tasks free of input and output links (E and F are such tasks), we get:

	A	B	C	D	E	F	G
A			1	1			
B							1
C	1	1				1	1
D		1					
E			1			1	
F							
G				1		1	

2. Remove rows and columns relatives to E and F.

	A	B	C	D	G
A			1	1	
B					1
C	1	1			1
D		1			
G				1	

3. Matrix product: Power 2.

	A	B	C	D	G
A	1	1			1
B				1	
C			1	1	1
D					1
G		1			

4. Matrix product: Power 3.

	A	B	C	D	G
A			1	1	1
B		1			
C	1	1		1	1
D				1	
G					1

5. Matrix product: Power 4.

	A	B	C	D	G
A	1	1		1	1
B					1
C		1	1	1	1
D		1			
G				1	

6. Matrix product: Power 5.

	A	B	C	D	G
A		1	1	1	1
B				1	
C	1	1		1	1
D					1
G		1			

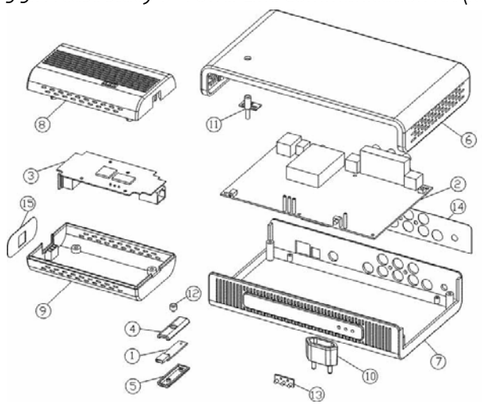
7. Task E and Task F are re-introduced in the final matrix, F as the first column and E as the last row. Task A and Task C have just one entries so they are pushed to the last just before E, and then followed by D with has two entries.

At last, the final matrix is:

	F	B	D	G	C	A	E
F							
B				1			
D		1					
G	1	1					
C	1	1		1		1	
A			1		1		
E	1				1		

3.3 Illustrative examples from the Literature

3.3.1 A case study 1 – Power Line Communication Kit (Reused from [31])



No.	Part Name
1	Key PCBA
2	Main System PCBA
3	Functional PCBA
4	Key Front Cover
5	Key Back Cover
6	Main Top Cover
7	Main Base Cover
8	Functional Top Cover
9	Functional Base Cover
10	Power Plug
11	Power Button
12	Key Button
13	Led Lens
14	Main IO Plate
15	Functional UI Plate

FIGURE 14. *Case study of an existing Power line Communication (PLC) product. (Source: Luh et al., page 15, ref. [31]).*

3.3.2 A case study 2 – (Source: [36])

No.	Part Name
X_1	the operating structure design
X_4	vessel design
X_4	plant layout/general arrangement (GA)
X_4	shipping design
X_5	structure lifting design
X_5	pressure drop analysis
X_8	process engineering
X_8	structural documentation
X_9	size valves
X_{11}	wind load design
X_{11}	seismic design
X_{14}	piping design
X_{14}	process and instrumentation diagram (PandID)
X_{14}	equipment support
X_{17}	pipe flexibility analysis
X_{17}	design documentation
X_{17}	foundation load design
X_{19}	insulation structural design
X_{19}	structural bill of materials (BOM)
X_{20}	assembly design

Components	#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
KeyPCBA	1					1							1			
Functional PCBA	2								1							
Main System PCBA	3						1				1		1			
Key Front cover	4	1		1		1				1		1				
Key back cover	5	1			1											
Main Top cover	6			1				1				1		1		
Main Base cover	7			1			1									
Functional Top cover	8		1	1					1							
Functional base cover	9		1					1								1
Power Plug	10			1			1									
Power button	11			1			1									
Key button	12	1			1											
Led Lens	13			1			1									
Main IO Plate	14			1				1								
Functional UI Plate	15		1												1	

FIGURE 15. *PLC product and its DSM (initial matrix).*

Components	#	3	6	7	11	13	10	14	1	4	5	12	2	8	9	15
Main System PCBA	3			1	1	1										
Main Top cover	6	1		1	1	1										
Main Base cover	7	1	1													
Power button	11	1	1													
Led Lens	13	1	1													
Power Plug	10	1	1													
Main IO Plate	14	1		1												
KeyPCBA	1								1	1						
Key Front cover	4	1					1		1	1	1					
Key back cover	5								1	1						
Key button	12								1	1						
Functional PCBA	2														1	
Functional Top cover	8	1											1		1	
Functional base cover	9												1	1		1
Functional UI Plate	15							1					1			

FIGURE 16. *Matrix result after clustering.*

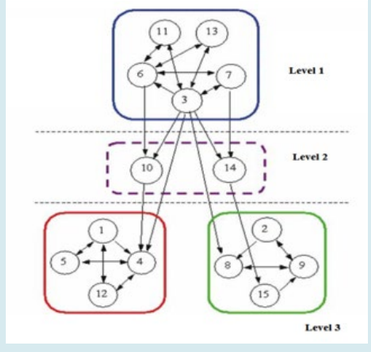
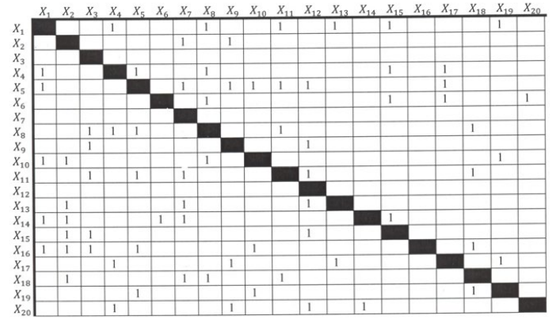


FIGURE 17. *Hierarchical of component interaction.*



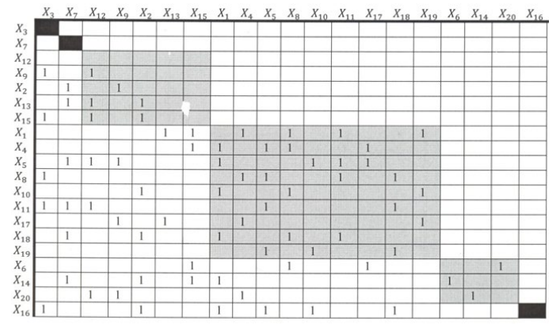


FIGURE 18. *Initial design Table, non-partitioned (first matrix), and Partitioned (second matrix).*

4. Axiomatic Design Theory

Axiomatic design is a systems design methodology developed by Dr. Suh at MIT *(1990, 2005, 1999)* in the Department of Mechanical Engineering in the 90s. This method consists of using matrix methods to systematically analyze the transformation of customer needs into functional requirements, design parameters, and process variables.

Axiomatic design provides a systematic way of satisfying many functional requirements *(FRs)* at the same time without introducing coupling of functions and creating integrated physical systems. ADT provides means of decomposing top-level requirements *(FRs)* and design parameters *(DPs)* until the creation of leaf-level FRs and DPs that can be implemented to construct the system according to the resulting design decision architecture. Let FRs a set of function requirement and DPs its corresponding referred

design parameters be selected by the project team through the development of different solutions so that each DPs is related to a corresponding FR, and related such that a specific DP can be adjusted to satisfy its FR. The design process is composed by four domains as illustrated in **Figure 19** and partially shown in **Figure 3**, namely: 1 - customer domain, functional domain, physical domain and process domain. The role of each domain is detailed in *(Suh, 2005, 1999)*.

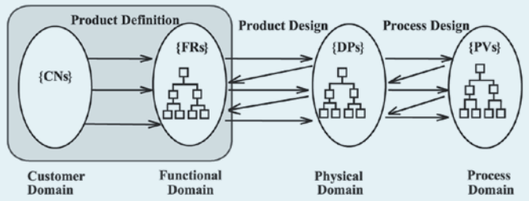


FIGURE 19. *The foundation axioms used in Axiomatic Design Theory are:*

Axiom 1: Maintain the independence of the functional requirements (FRs).

Axiom 2: Minimize the information content of the design such that as defined by equation (2). The formal definition of information content for a particular design parameter is:

$$I = \log\left[\frac{\text{system range}}{\text{common range}}\right] = \log\left(\frac{1}{P_i}\right) \qquad (2)$$

Design range: the tolerance of the admissible variation of DPs.

System range: The capability of the system.

Common range: The overlap between design and system ranges.

P_i : Probability of satisfying (See **Figure 20**).

To compare two different designs based on their information content, it is necessary to compare the sums of information contents of each design. For further details, the reader is invited to read Suh *(2005)*.

Axiom 1 requires that functional requirements be independent of each other, enabling each FR to be satisfied without affecting any other FR.

Axiom 2 provides a quantitative measurement of any design. It is useful in selecting the best design which satisfy Axiom 1.

Furthermore, in ADT there are some theorem and corollary to organize the logic side of the process. They are considered as rules for adopting a sense of interpretation. For example, Corollary 3 says: "Integrate design features into a single physical part if the functional requirements can be independently satisfied in the proposed solution" and Theorem 5, "When a given set of FRs is changed by the addition of a new FR, by substituting one of the FRs with a new one, or by selection of a completely different set of FRs, the design solution given by the original DPs cannot satisfy the new set of FRs any longer. Consequently, a new design solution must be sought".

A typical Design model using the ADT is presented as follows *(Nam, 2001) (At the high level)*:

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \end{Bmatrix} = \begin{bmatrix} A_{11} & A_{12} & 0 \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \end{Bmatrix}$$

FR_1= Satisfy specific customer requirements
FR_2 = Produce economically
FR_3 = Deliver fast
DP_1 = Product variety
DP_2 = Position of the decoupling point
DP_3 = Production flow.

In ADT methodology, the decomposition process starts with the decomposition of the overall functional requirement and its corresponding DP is determined for the same hierarchical level in the physical domain. The iterative process is called zigzagging and at each level of the hierarchy the FRs and DPs can be mapped to each other according to the equation *(3)*. **Figure 21**.

$$\{FR\} = [DM]\{DP\} \qquad (3)$$

The design matrix DM can be expressed as *(4)*.

$$[DM] = \begin{bmatrix} A_{11} & \cdots & A_{1n} \\ \vdots & \ddots & \vdots \\ A_{m1} & \cdots & A_{mn} \end{bmatrix} \qquad (4)$$

where

$$A_{ij} = \frac{\delta FR_i}{\delta DP_i} \quad (i = 1, \ldots, m \text{ and } j = 1, \ldots, n) \qquad (5)$$

A small change in any parameter may cause a deviation in the functional requirement.

$$\Delta FR_i = \frac{\delta FR_i}{\delta DP_i} \Delta DP_i \qquad (6)$$

In linear design A_{ij} are constants. The obtained expression is

$$FR_i = \sum_{i=1}^{n} A_{ij} DP_j \qquad (7)$$

As equation 2 satisfies the Axiom 1 of the ADT, it can be interpreted as "choosing the DPs set which satisfies a given set of FRs". Equation *(3)* shows that each element of the DM matrix is represented as a partial derivative to indicate the strong dependency between FRs and DPs. The derivative values can be "0" or any other value. Using a derivative value of "0" means to say that the functional requirement does not depend on any particular design parameter. In addition, as discussed previously in DSM paragraph, the three types of designs may exist *(uncoupled, decoupled and coupled)* in the sense that uncoupled design occurs when each FR is satisfied by exactly one DP, and the resulting matrix is diagonal. When it is possible to organize DM matrix as a lower triangular, the design is decoupled, and therefore, the FRs can be satisfied. The third case shows that FRs cannot be satisfied independently, exactly as shown in **Figure 6**. Moreover, it can be noticed that decoupled design has less tolerance than uncoupled design, and the increase of the order of design makes the last DP's tolerance smaller. If the number of DPs is greater than the number of FRs then the design is redundant *(Nordlund, 1996)*.

FR_i/DP_i	DENOMINATION
FR_1	Construct the basic information of the product
FR_{11}	Assign an ID number to a new product
FR_{12}	Construct a set of data for a new product
FR_2	Establish the product shape, produce economically
FR_{21}	Check the curvature, panel flatness, and funnel axis profile
FR_{22}	Calculate the three-dimensional shape
FR_{23}	Consider the manufacturability
FR_3	Verify the mechanical characteristics of the product
FR_4	Generate the product drawing
FR_{41}	Represent the shape of the product
FR_{42}	Display accessory of the drawing
FR_{231}	Check the useful screen dimension for the panel
FR_{232}	Evaluate the ejectability
FR_{233}	Examine the deflection angle of a scanning line for the funnel
DP_1	A set of basic data - those supplied by the customer
DP_{11}	Representative code of new product
DP_{12}	A set of specific data for a new product
DP_2	The three-dimensional shape structure (panel/funnel)
DP_{21}	Inside/outside curvature for the product
DP_{22}	Characteristic geometric equation for the product
DP_{23}	A set of data for mold manufacture
DP_{231}	Distance of blending circle center position
DP_{232}	Angle of the side wall
DP_{233}	Inside curvature of the yoke part
DP_3	Loading conditions on the panel and funnel
DP_4	A set of accessory drawing data
DP_{41}	A set of data for product design
DP_{42}	A set of data for accessory

TABLE 1. *Functional and design product for the ADT (Source: http://www.axiomaticdesign.com/technology/ADSChapter5.html*

4.1 Example. Architecture for a Glass-Bulb Design

(Source: http://www.axiomaticdesign.com).

The glass bulb is a unit element of a TV tube, which is sometimes called "Brown tube." The tube consists of a shadow mask, an electron gun, a band, and a glass-bulb that consists of a panel and a funnel. The panel is the front part and the funnel is the rear part of the glass bulb shown in **Figure 22**.

The original manual design process consisted of the following 5 steps.

Step 1. Product requisition which consists in receiving a customer order containing the basic design specifications.

Step 2. Generation of an initial design using a CAD software.

Step 3. Generation of the tube three-dimensional shape using a software program, which uses the output of Steps 1 and 2 as well as part of the information generated in Step 5, and then perform Step 4.

Step 4. Stress analysis.

Step 5. Generation of final design. Some part of this information is used in Steps 3 and 4, requiring iterations between Steps 2 through 5.

The functional requirements of this design are at each level of the process and are presented in the **Table 1**.

The hierarchy and the zigzagging processes are detailed in the **Figure 23**.

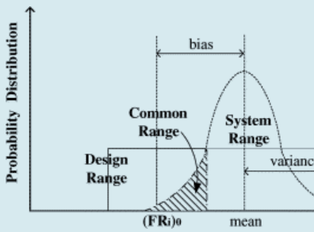


FIGURE 20. *The relationship among design range, the common range, the system range and probability distribution (from [25]).*

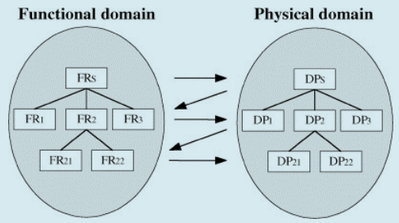


FIGURE 21. *Hierarchical decomposition in ADT process.*

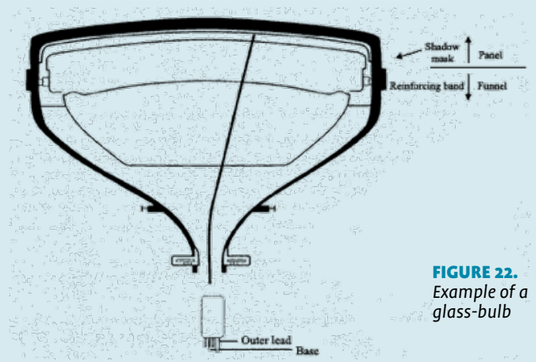


FIGURE 22. *Example of a glass-bulb*

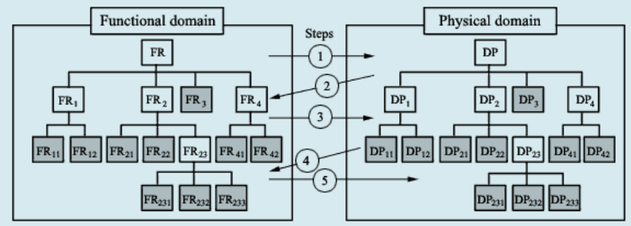


FIGURE 23. *Hierarchical and zigzagging topology.*

The hierarchical relationships between functional and physical domains depicted in **Table 1** are presented in the following matrices illustrated in **Figure 24**.

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & X \\ X & X & 0 & X \\ X & X & X & 0 \\ X & X & 0 & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \end{Bmatrix} \text{(a)}$$

$$\begin{Bmatrix} FR_1 \\ FR_2 \\ FR_3 \\ FR_4 \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 & 0 \\ X & X & 0 & 0 \\ X & X & X & 0 \\ X & X & 0 & X \end{bmatrix} \begin{Bmatrix} DP_1 \\ DP_2 \\ DP_3 \\ DP_4 \end{Bmatrix} \text{(b)}$$

$$\begin{Bmatrix} FR_{11} \\ FR_{22} \end{Bmatrix} = \begin{bmatrix} X & 0 \\ X & X \end{bmatrix} \begin{Bmatrix} DP_{11} \\ DP_{22} \end{Bmatrix} \text{(c)}$$

$$\begin{Bmatrix} FR_{21} \\ FR_{22} \\ FR_{23} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ X & X & 0 \\ X & X & X \end{bmatrix} \begin{Bmatrix} DP_{21} \\ DP_{22} \\ DP_{23} \end{Bmatrix} \text{(d)}$$

$$\begin{Bmatrix} FR_{41} \\ FR_{42} \end{Bmatrix} = \begin{bmatrix} X & 0 \\ 0 & X \end{bmatrix} \begin{Bmatrix} DP_{41} \\ DP_{42} \end{Bmatrix} \text{(e)}$$

$$\begin{Bmatrix} FR_{231} \\ FR_{232} \\ FR_{233} \end{Bmatrix} = \begin{bmatrix} X & 0 & 0 \\ 0 & X & 0 \\ 0 & 0 & x \end{bmatrix} \begin{Bmatrix} DP_{231} \\ DP_{232} \\ DP_{233} \end{Bmatrix} \text{(f)}$$

FIGURE 24. *(High-level coupled design, (b) high-level decoupled design, (c) triangular matrix, it is a decoupled design, (d) decoupled design, (c) diagonal matrix, uncoupled design, (f) uncoupled design.*

5. Integrating ADT and DSM

As ADT is firstly used in the product creation stage, it is incapable of analysing the system's interactions. This mission is treated efficiently by the DSM, however, the DSM cannot address descriptive facilities at the stage of design creation. Therefore, it is logical to believe that their combination will generate a more robust tool for designing and organizing product design process. Many attempts have been made in order to address this process *(Axiomatic design of mechanical systems, 1995)* and *(Maurer, 2007)* the objective was to obtain the design information flow at an early stage of the design, and thus allowing the use of the DSM at the time when the most important decisions about the system and design are made. The procedure for doing this could be as proposed by *(Maurer, 2007)*: *(1)* using existing DSM-based knowledge to accelerate AD; *(2)* conversion of DM to DSM to get the interactions among DPs at the conceptual design stage; *(3)* using derived DSM to evaluate the design result of AD from the DPs interaction view; and *(4)* using derived DSM to conduct project planning at an early design stage.

Dong and Whitney *(2001)* showed that if the AD matrix can be expressed analytically and one design parameter *(DP)* is dominant in satisfying a particular functional requirement *(FR)*, then the triangulated design matrix is equivalent to the DSM of the design parameters. The algorithm proceeds in three major steps:

1. Construct the Design Matrix (DM).

2. In each row of the DM choose the dominant entry.

3. Permute the matrix by exchanging rows and columns so that all dominant entries appear on the main diagonal.

To demonstrate the transition from an ADT to its corresponding DSM, we use a simple example that can be summarized according to the procedure of Dong and Whitney *(2001)*.

We assume the functional requirements at the highest level are available.

Step 1. Construct the design matrix (ADT).

	DP1	DP2	DP3
FR1	X		X
FR2	X	X	
FR3		X	X

Step 2. Choose an output variable from each row. They are indicated by X0.

	DP1	DP2	DP3
FR1	X		X0
FR2	X0	X	
FR3		X0	X

The significance of the output variable choice is:

DP3 = f *(FR1, DP1)*
DP1 = f *(FR2, DP2)*
DP2 = f *(FR3, DP3)*

Step 3. Permute the rows so that the output variables are on the diagonal. Then rename the rows according to the DP of the columns, we get the DSM:

	DP1	DP2	DP3
DP1	X0		X
DP2		X0	
DP3	X		X0

6. Conclusions

Product design process is a very complex task to accomplish as the number of activities and their interaction increases. Decomposition and partitioning may involves preliminary solutions but not sufficient to contribute to facilitate the description of the global design process, sharing the project between team component, and incorporating time, cost and risk requirements. DSM and ADT are two compact representations of the design process architecture information structure, and from which it can be possible to monitor the design process from the beginning to the end. DSM and ADT are able to work independently from one another, however, they are surprisingly more effective when used together.

In this paper a design decomposition model is proposed in which Axiomatic Design Matrices *(ADT)* map the functional requirements to design parameters, while the Design Structure Matrices *(DSM)* provide a structured representation of the system development context. The process of decomposition and integration using matrices as information support is easier when using simple algorithms and heuristics presented in the paper. Furthermore, transition from one model *(DSM)* to another *(ADT)* was clearly identified through the use of several examples. In the near future, this work will include an evaluation and study on advanced algorithms for enhancing and clarifying sequence-coupled tasks based on ADT and DSM tools to lightening up PDP's processes.

authors

Darli Rodrigues Vieira is a professor at the Université du Québec à Trois-Rivières (UTQR), and holds the research chair in Management of aeronautical projects. Previously, he was a professor at the Federal University of Parana (UFPR) and the Instituto Tecnologico de Aeronautica (ITA). He is the author of several works, among which stands out the following recent books: Analysis and design of logistic networks (with Alain Martel), Distribution Centers Project (with Michel Roux), Audit Logistics (with Michel Roux) and Product Design Management (with Abdelaziz Bouras and Denis Debaecker). His professional experience includes over 20 years of activities in the aeronautical, automotive, steel, cosmetics, telecommunications, and ports sectors.

Mohamed-Larbi Rebaiaia is an associate professor of production management and operations management successively at the Department of Mechanical Engineering and the Department of Operation and Decision Systems, both located at Laval University, Canada. From 1983 until 2006 he taught software engineering and operational research at Annaba and Batna Universities, Algeria. M. Rebaiaia received a PhD and Msc in computer sciences and operational research from these two universities and a PhD in industrial engineering at the Department of Mechanical Engineering, Laval University. His primary research interests are in the areas of software engineering, operations and production management, scheduling and timetabling, project management, systems reliability and networks performance and more specifically, techniques for determining the availability of telecommunication systems. He has published more than 60 publications in international journals and conferences' proceedings. Rebaiaia is a member of OR Society in Switzerland.

BIBLIOGRAPHIC references

Alexander, C., Notes on the synthesis of form, Harward University Press, Cambridge, 1964.

Axiomatic design of mechanical systems (1995). Special 50th anniversary combined issue of the journal of mechanical design and the journal of vibration and acoustics, Trans ASME, Vol. 117, pp. 1–10.

Azarm, S., Optimal design using Two-level monotonicity-based decomposition method, ASME design automation conference, Boston, Mass., pp. 41-48. 1987.

Blecker T., Complexity and variety in mass customization systems: analysis and recommendations, Management decision, Emerald, Vol. 44, NO 7, pp. 908-929, 2006.

Boehm B, A Spiral Model of Software Development and Enhancement", ACM SIGSOFT Software Engineering Notes, ACM, 11(4):14-24, August 1986.

Browning, T. R. and Eppinger S. D, Modeling Impacts of Process Architecture on Cost and Schedule Risk in Product Development IEEE Transactions on Engineering Management, vol. 49, no. 4, pp. 428-442, Nov. 2002.

Chen, S-J and Lin, Li, Decomposition if interdependent task group for concurrent engineering, J. Computers & Industrial Engineering, Vol. 44, N0 3, pp. 435-459, 2003.

Cho S-H and Eppinger S.D., Product Development Process Modeling using advanced simulation, ASME conference, Pittsburgh, Pennsylvania September 9-12, 2001.

Danilovic, M.; Browning, T., A Formal Approach for Domain Mapping Matrices (DMM) to Complement Design Structuring Matrices (DSM). In: Proceedings of the 6th International Design Structure Matrix Workshop, Cambridge, UK, 12.-14.09.2004. Cambridge: University of Cambridge 2004.

Dong and Whitney Qi Dong and D. Whitney, Designing a requirement driven product development process, DTM conference, September 9–12, , pp. 11–20, Pittsburgh, PA, 2001.

Fernandez, C., Integrating analysis of product architecture to support effective team co-location, Master thesis, MIT, 1996.

Galbraith, Jay R., Designing Complex Organizations, Addison-Wesley, Reading, MA, 1973.

Gebala D. A. and Eppinger S.D, Methods for analyzing design procedures, DE-Vol. 31, Design Theory and Methodology, ASME 1991.

Jang B-S, Yang Y-S, Song Y-S, Yeun Y-S and Do S-H, Axiomatic design approach for marine design problems, Marines Structure, Vol. 15, Issue 1, pp. 35-56, Elsevier, 2002.

Johnson, R. C. and Benson, R. C. , A Basic Two-Stage Decomposition Strategy for Design Optimization, ASME Journal of Mechanisms, Transmissions, and Automation in Design, 106, 380-386, 1984.

Kneuper, R., CMMI - Verbesserung von Softwareprozessen mit Capability Maturity Model Integration. Heidelberg: dpunkt, 2007.

Krubasik, E.G., Customize your product development. Harvard Business Review, November–December, 1998.

Kusiak, A. and J. Wang, "Decomposition of the Design Process," Journal of Mechanical Design, Vol. 115, December, 1993.

Luh D-B, Ko Y-T and Ma C-H, A structural matrix-based modelling for designing product variety, Journal of Engineering design, Vol. 22, N) 1, pp. 1-29, 2011.

Maurer, M., Structural Awareness in Complex Product Design, Dissertation, Technische Universität München, 2007.

Nam P.S., Axiomatic design, MIT-Pappalardo Series in Mechanical Engineering Oxford University Press, New York, 2001.

Nordlund M. An information framework for engineering design based on axiomatic design, Doctoral Thesis, Stockholm, Sweden, Royal Institute of Technology, 1996.

Pimmler T. U. and Eppinger D., Integration analysis of product decompositions, ASME Design Theory and methodology Conference, Minneapolis, Sep. 1994.

Rogers, J., A knowledge-based tool for multilevel decomposition of complex design problem. Hampton, Virgina: NASA TP2903, Langley Research Center, 1989.

Simon, Herbert A. The Sciences of the Artificial, 2nd edition, MIT Press, Cambridge, MA. 1981.

Smith R. P. and Eppinger S., Deciding between sequential and parallel tasks in engineering design, Working paper N0 3858, Sloan School of Management, MIT, Oct. 1995.

Steward, D.V., The design structure system: a method for managing the design of complex systems, IEEE Transactions on Engineering Management, 28 (3), pp. 71–74, 1981.

Suh, N.P., "A theory of complexity, periodicity and the design axioms", Research in Engineering Design, Vol. 11 No.2, pp.116-31, 1999.

Suh, N.P., Complexity: Theory and Applications, Oxford University Press, New York, NY, 2005.

Suh, N.P., The Principles of Design, Oxford University Press, New York, NY, 1990.

Ulrich K and Eppinger S., Product Design and Development. NewYork: McGraw Hill, 2004.

Unger D. W. and Eppinger, S. D., Comparing product development processes and managing risk, Int. journal Product development, Vol. 8, N0 4, pp. 382-402, 2009.

Warfield, John and J. Douglas Hill, (Edited by Benjamin B. Gordon), "A Unified Systems Engineering Concept," Battelle Monograph, No. 1, June 1972.

Yassine A. and Braha D., Complex concurrent engineering and the design structure matrix method, Concurrent Engineering: research and application, vol. 11, N0 3, Sept. 2003.

Zamenopoulos, T. and Alexiou, K., Towards an anticipatory view of design. Design Studies, 28 (4), 411–436, 2007.