A fuzzy method for propagating functional architecture constraints to physical architecture.

Eric Bonjour, Samuel Deniaud, Maryvonne Dulmet, Ghassen Harmel

A Fuzzy Method for Propagating Functional Architecture Constraints to Physical Architecture

Eric Bonjour[1]
ebonjour@ens2m.fr

Samuel Deniaud[2]
samuel.deniaud@utbm.fr

Maryvonne Dulmet[1]
mdulmet@ens2m.fr

Ghassen Harmel[1]
gharmel@ens2m.fr

[1]FEMTO-ST Institute, UMR CNRS 6174 - UFC / ENSMM / UTBM, Dpt AS2M, 24 rue Alain Savary 25000 Besancon, France
[2]UTBM / M3M, 90010 Belfort Cedex - France

Modular product design has received great attention for about 10 years but few works have proposed tools to either jointly design the functional and physical architectures or propagate the impact of evolutions from one domain to another. In this paper we present a new method supporting the product architecture design. In new product development situations or in reengineering projects, system architects could use this method in the early design stages, to pre-determine cohesive modules and integrative elements and to simulate a domain architecture by propagating architecture choices from another domain. To illustrate our approach, we present an industrial case study concerning the design of a new automobile powertrain.

1 Introduction

It has been widely observed that the development of new products has become a critical advantage for organizations acting in competitive environments. Innovative technologies, more demanding customers and regulations are some of the factors that force firms to become innovative.

The product domains are mainly composed of three sub domains [1], which are:

- Customer expectations and life-cycle requirements,
- Functions which are arranged in the functional architecture,
- Sub-systems and components which are arranged in the physical (or design) architecture.

These domains form different views of the product at different levels of abstraction. Requirements correspond to external functions and constraints that the product has to satisfy. These requirements are fulfilled through the realization of the system functions that in turn are realized by the integration of different product components.

In the engineering design field, researchers have developed architecting rules and methods to map functions to physical components [2-6]. Other approaches view the functional model of a system as being described by an abstract functional decomposition that may, but does not need to, have a direct mapping onto physical decomposition of assemblies and subassemblies [7]. Ulrich [3] defines product architectures as "the scheme by which the function of a product is allocated to physical components." A key feature of product architecture is the degree to which it is modular or integrative [8]. In modular architectures, functional models of the product map one-to-one to its physical components. On the other hand, in integrative architectures "several functional elements are each implemented by more than one component, and several components each implement more than one functional element" [3]. In real design situations, designers have to make a trade-off between modular and integral architectures. Hence, many products are hybrid [9]. Their architectures are not fully modular or integral and lie somewhere between the two extremes. The above definitions are mainly based on the functionality of a module. There is another common way of defining a module by only focusing on interactions between elements. Baldwin and Clark [4] define a module as "a unit whose structural elements are powerfully connected among themselves and relatively weakly connected to elements in other units". Browning [8] defines integrative elements as interacting with all of the modules without belonging to any module.

Research, concerning platform-based product development and product family design [10-16], has received huge interest over the last decade since it aims at providing methods to identify common modules and generate product variants with distinctive modules (commonality *vs* variety). According to [10], a non-unique component is defined as a component that is present in at least two products in a product family. It can be either common between the products sharing this component or variant across products. Overviews of current research status on these issues have been developed in [13-16].

Modularity has many advantages but in the engineering design literature, few methods exist to partition a product into modules, even in the special case of single complex products [17-28]. Sharman and Yassine [28] point out that modularity has drawbacks. The inter-module interfaces must allow change to occur within modules without adversely affecting inter-module working. This requires an appropriate definition of interfaces that play the key role of connecting and interacting between components [29, 30]. Dobrescu and Reich [31] described an idea that leads to benefits, where modules need not be shared across all variants but could be shared by only subsets of them. Existing architecting methods use different product architecture representations [28] for instance, oriented graphs [19], diagrams [20] or matrices. The inputs of these methods may be functional

models [20, 32], components interactions [8, 9, 26, 33, 34], a mapping of functions onto physical components [6, 35], or more complex data intended to take into account key factors of the whole life cycle of the system [36, 37].

The general approach when developing complex products is to decompose the product into sub-systems, and if the sub-systems are still too complex, to decompose these into smaller components [18, 38]. In the following development of this article, we consider that product architecture is composed of modules and integrative elements that fulfill system functions. Matrix-based methods are valuable to represent relationships and couplings between product sub-domains or within each domain. We define product architecture by focusing on interactions between elements, similarly to [4, 8]. The architecture identification is achieved by a clustering algorithm, e.g. [39-42].

We are particularly interested in the development of complex products, such as automobile powertrains. In this paper, we propose to provide system architects with a tool that supports the simulation of Product Component (PC) architecture by propagating System Functions (SF) architecture choice. Section 2 presents matrix-based methods for product architecture modeling. Section 3 gives an overview of a matrix-based method that aims at propagating constraints from the functional architecture to the physical one. Section 4 mainly focuses on the fuzzy treatment to obtain a component-based Design Structure Matrix (DSM). Section 5 presents the selected clustering algorithm that allows us to identify the underlying architecture of the DSM. Section 6 exhibits the efficiency of the proposed method by applying it on a powertrain design. Section 7 presents a sensitivity analysis.

2 Matrix-based methods for product architecture modeling

Nowadays, products are becoming more and more complex. In the development of complex products, considerations about product architectures and modularization are crucial. Matrix-based product modeling methods represent the product architecture (product elements and their relationships), shown as a matrix. They are being increasingly used in such works [28, 43], since they can support different research goals: for example, product modularization [21, 25, 26, 36], analysis of technical interactions either within the products or within the project organization [9, 18], design analysis [2], and change propagation analysis [27, 44].

2.1 Different types of matrix-based methods

System architecture modeling relies on different kinds of matrix in order to design a complex system in a systematic and coherent way. Malmqvist [43] distinguished two levels of product analysis that are related to different design goals. Firstly, product-level matrices provide a mapping between a set of properties or other elements and a number of “whole” alternatives. The motivation for such methods is to support decision-making about the entire product life-cycle or about product platforms, of which product-level variants are the “parts”. Developing product platforms requires considerations of common and unique modules within a brand and within a platform, but also requires seeing each product variant as a whole [15, 45]. Secondly, element-level matrices represent the relationships between the elements/parts/components of a single product in a matrix. There are two sub-types of element-level matrices that we briefly review in the next sections: inter-domains matrices and intra-domain matrices.

2.2 Inter-domains matrix or incidence matrix

Inter-domains matrices represent relationships between two domains. These matrices are basically incidence matrices (IM) [46]. They can represent a set of design decisions or relationships between what and how.

Some authors use other names such as Design Matrix (DM) in Axiomatic Design (AD) [2], which captures the relationships between functions and physical design parameters. Indeed, AD pays considerable attention to the relationships between Functional Requirements (FR) and Design Parameters (DP). The former (i.e. FR) correspond to elements in the functional domain and describe design goals. The latter (i.e. DP) correspond to elements in the physical domain and aim at satisfying particular FR. DP are means ("how") to fulfill the FR ("what"). According to AD, the decomposition of a design problem follows a "zigzagging" top-down approach between the hierarchies of the functional and physical domains. The DM indicates how the DP address the FR at each level of the hierarchy. The following equation is used: {FR}=[DM]{DP}. The DM may be uncoupled, decoupled or coupled.

Another example is the Quality Function Deployment (QFD) method [47] that uses several inter-domains matrices to convey the customer requirements throughout the development project. Inspired from QFD, Erixon [36] develops the MFD (Modular Function Deployment) method, using the concept of "module drivers", in order to elaborate a product architecture which takes into account key considerations of the product life-cycle. In [46], an interesting decomposition method has been proposed for complex design problems that are represented in an attribute-component incidence matrix. A formal two-phase process has been described to transform this matrix into a block-angular matrix according to a given set of decomposition criteria. In [48], the authors use binary function-component matrices in order to obtain morphological matrices by using matrix multiplication. In [49], we find a very recent use of inter-domains matrix called Domain Mapping Matrix (DMM). The authors apply clustering directly on DMM. Fixson [35] proposes to create a "Function-component allocation" matrix. In the cells of this matrix, "percentages of a function can be allocated to components that contribute to this function".

We assume in our research work that incidence matrices are of high importance since they help to ensure the cohesion between the product sub-domains and more generally, between project domains (product, process and organization) [50, 51].

2.3 Intra-domain matrix or DSM

Intra-domain matrices represent relationships between elements of the same domain, for example between components. These matrices are usually called DSM: Design (or Dependency) Structure Matrix. DSM [17] is now a popular modeling and analysis tool, especially for purposes of decomposition and integration. DSM can be applied on various levels of abstraction to study interactions between requirements, between functions, between sub-systems or components, between design parameters and between the life-cycle processes [8, 9, 18]. DSM displays the relationships between elements of a system in a compact and visual format. Hence, DSM is used to identify project domain architectures: the architecture of products [18], the architecture of design process [52] or the decomposition of the projects into different teams [9]. DSM is a square matrix with identical elements in rows and columns. Cells along the diagonal have no sense. Reading across a row reveals what other elements the element in that row provides. Scanning down a column reveals what other elements the element in that column depends on. Hölttä and Salonen [53]

present a study that compares different modularization methods: heuristic method, DSM, and Modular Function Deployment (MFD). They conclude that, given the same inputs, all methods partition the products differently but it is important to note that methods developed for single products are poorly suited to modularize product families and MFD is the least repeatable method whereas the DSM-based method the most repeatable.

Dong and Whitney [54, 55] propose a method to obtain a DSM from a DM. This method is useful to forecast system interactions before the detailed design phase and to manage system design iterations. We will present this method further (see Section 6.6, Discussion). Whereas design methods such as Axiomatic Design [2, 56], Quality Function Deployment [47], systems architecting principles [5, 57, 58] or Design rules [4] have described valuable rules, criteria and practices, for defining system architectures or mapping functions onto components, it is worth noticing that few formal methods exist to handle architectural data from both functional and physical views [54, 55, 59]. However, this design step is very important in the identification of appropriate product architectures: "people currently make decisions at the early phase of the design with very limited knowledge about the system […] prediction does not have to be perfect" [55]. When couplings and mapping have not been formally addressed, the integration of the teams' contributions is more difficult and requires numerous design iterations.

3. Propagating functional architecture on physical architecture

In the conceptual design that establishes functional structures or in the preliminary embodiment design that develops the preliminary layouts [58], a crucial design activity corresponds to the mapping between two product sub-domains and then, the choice of the preliminary architectures. The main architecture decisions are represented by the following two inter-domains matrices: the requirements - system functions incidence matrix and the System Functions - Product Components (SF-PC) incidence matrix (IM). These matrices ensure the traceability from the customers' requirements to the component design. In this work, we focus on the simulation of PC architecture starting from the SF architecture. We summarize our propagation method in four steps (see Fig.1). First, we capture the functional architecture of the product. Second, we capture the System Functions *vs* Product Components Incidence Matrix (SF-PC IM). Third, we propagate the SF architecture choices through the SF-PC IM in order to generate the PC DSM. Finally, we identify the SF- and PC- architectures and we compare them.

Insert Figure 1. Propagation method

Step 1: Capture the SF DSM. By interviewing system architects who have to design the product architecture, we identify the list of system functions. Then, we ask them to fill in a SF DSM by identifying and evaluating each coupling between SF. Typologies for DSM interactions have been identified by Pimmler and Eppinger [18]. In this work, the system architect estimates the SF coupling intensity by referring to the set of attributes that the SF affect or share. The same idea is proposed in [46]. Coupling intensity is estimated on the scale 0-10.

Step 2: Capture the SF-PC IM. The system architects have already identified the SF list. They have to elaborate the list of components that are likely to fulfill the functions. So they generate the first level of the product breakdown structure. Then they

have to fill in the SF-PC IM by identifying possible relationships between SF and PC (mapping). Since the product components are also characterized by a set of attributes, they have to estimate the intensity of the relationships by referring to the set of attributes that are shared between a SF and a PC. Similarly to Fixson [35], we define an IM value as the contribution intensity of a component to the fulfillment of a function. This intensity is ranged from 0 to 10. Note that design methods such as QFD or MFD recommend using four values only: 0, 1, 3 or 9. It is rather an exponential scale that gives well-discriminating estimations. We prefer a continuous 0-10 scale because we think that these values could be refined in the detailed design phase by means of behavioral simulations or calculus linking design parameters to functions (design sensitivity analysis). Moreover, we will use a fuzzy processing to discriminate three ranges of contribution intensities.

Step 3: Simulate PC DSM starting from SF DSM and SF-PC IM. Since the documentation of the matrices is rather subjective, we propose a fuzzy treatment. It will implement four construction axioms. This step will be detailed in Section 4.

Step 4: Identify SF- and PC- architectures. By applying a clustering algorithm on SF DSM and PC DSM, we identify the architecture composed of modules and integrative elements. This step will be detailed in Section 5.

4. Simulating PC DSM from SF-PC IM and SF DSM

In this section, we present the propagation method to simulate a PC DSM from a SF DSM and a SF-PC IM. Section 4.1 firstly formulates the axioms which are at the basis of the method for simulating a new DSM. Since intensity values inside DSM and IM are quite imprecise and subjective, we are clearly in a context where the use of fuzzy logic is relevant [61, 62] to manipulate these values. Section 4.2 describes a fuzzy method that allows us to compute potential couplings between each pair of PCs. Section 4.3 presents a simple procedure to aggregate the resulting DSM and filter it in order to ignore meaningless values.

4.1 Axioms

In this section, we explain the axioms that are at the basis of the propagation method, along with the underlying assumptions.

- Axiom 1. Two system functions (SF_i, SF_j) and two components (PC_u, PC_v) can be coupled according to two different ways (Fig. 2).

Insert Figure 2. Coupling a couple of SF and a couple of PC

- Axiom 2. If (SF_i, SF_j) and (PC_u, PC_v) are coupled and if SF_i and SF_j interact, then PC_u and PC_v interact (Fig. 3).

Insert Figure 3. Propagating the (SF_i, SF_j) interaction

- Axiom 3. The intensity of the coupling between PC_u and PC_v is related to both the coupling intensity between (SF_i, SF_j) and (PC_u, PC_v) and the coupling intensity between SF_i and SF_j.
- Axiom 4. We assume that each SF is coupled to itself with an intensity of maximum value, i.e. 10.

Let's explain the most important axiom which is Axiom 2. If SF_i and SF_j interact then they affect a set of shared or coupled attributes. If SF_i (resp. SF_j) is coupled to PC_u (resp. PC_v) then the attributes that they affect are coupled too. Thus, it is possible to identify constraints and then, interactions between attributes affected by PC_u and those affected by PC_v (see Fig.4).

Insert Figure 4. Interaction propagation through attributes

With axiom 3, we assume that the higher the intensities of interactions, the more numerous constraints exist between the sets of coupled attributes and the higher the probability to map the sets of coupled PC attributes. Axiom 4 allows us to handle the case concerning a SF directly coupled to (PC_u , PC_v). Note that the Dong method [55] only deals with these direct relationships.

4.2 A fuzzy method

The propagation method generates one PC DSM for each couple of SF (SF_i, SF_j) by identifying the components coupled to (SF_i, SF_j) and by applying the rules introduced by the axioms and translated into a fuzzy inference system. This system is described hereafter.

Fuzzification and input membership functions

The "fuzzification" stage (see Figure 5 on the left side) corresponds to the transformation of a numerical value through fuzzy variables (input). We choose the structure of the membership functions characterizing the three inputs, by taking into account the architect's reasoning. We define three linguistic variables which are: Low, Medium and High. We use the most common membership function, i.e. a trapezoidal function: Low [0 0 3 5], Medium [3 5 7 9], High [7, 9, 10, 10].

Inferences and fuzzy rules

The fuzzy 'if-then' rules are developed to relate input to output variables [61]. These rules represent the expert's knowledge about the interactions between input variables and their effects on the output. The inference system approximates the way an architect estimates the coupling between two SF. It is based on a set of 13 rules (see Figure 5). This inference system has been implemented with Matlab Toolbox. Our objective has been to generate an inference system that is understandable for architects.

Insert Figure 5. Proposed fuzzy treatment

Defuzzification and output membership functions

A trapezoidal membership function is used for the fuzzy output linguistic variables which are: Weak [0 0 1 3], Average [1, 3, 7, 9] and Strong [7 9 10 10]. The "Defuzzification" stage involves finding a crisp value for the coupling using the output membership function. The aggregated fuzzy output is defuzzified using the "centroid of area" technique. The formula is given in [61]. This is the most widely adopted defuzzification method, which is reminiscent of the calculation of expected values of probability distributions. Table 1 summarizes the main features of the fuzzy inference system. The choice of these linguistic variables aims at limiting the influence of low inputs. Tuning the system has been done to choose the input-output membership

functions and the defuzzification method that are the most appropriate. We tested this fuzzy treatment by applying it to a past engine design. After few changes to the parameters, this fuzzy model has been validated by the system architects since after DSM clustering, the obtained architecture was judged valid. The visual inspection of the clustered DSM clearly revealed distinctive clusters that were meaningful and useful for system architects, who recognized and named them easily.

Table 1. Features of the fuzzy treatment

And method	Min
Or method	Max
Implication	Min
Aggregation	Max
Defuzzification	Centroid

4.3 Aggregate the N PC DSM and filter

In order to obtain the aggregated DSM, we use the average method. Let us consider that we have N SF and M PC. The intensity of the (PC_u, PC_v) interaction in the aggregated PC DSM is computed as follows:

$$DSM(PC_u, PC_v) = \frac{\sum_i \sum_j DSM_{(SF_i, SF_j)}(PC_u, PC_v)}{N^2} \qquad (1)$$

Considering the combination of three zero inputs, the output value is equal to 1.06. This minimum value is directly related to the choice of the membership function of the linguistic variable “Weak” for the output. It is impossible to obtain a zero value by using the centroid defuzzification method. Hence, the PC DSM obtained through the average method is totally dense with no null value. For that reason we propose a procedure to normalize the values on a 0-10 scale and a filtering method in order to limit the final PC DSM density. This density could not be a problem for interpreting the coupling between the components. But the clustering algorithm used for determining the "optimal" architecture is sensitive to the density of couplings. Thus, we need to filter the low values in the aggregated PC DSM. The filter aims at reducing meaningless coupling values by converting them to zero:

$$if\ \ DSM(i, j) \le X\ \ then\ \ DSM(i, j) = 0 \qquad \mathbf{(2)}$$

where X is a parameter that has to be higher than 1.06.

Since the aim of our method is to identify domain architectures and not precisely the valuation of the interactions between elements, the threshold for the clustering can be higher than the minimum value of 1.06. In this case, the filtering will help us reduce DSM density without corrupting the clustering results. The filter used has no impact on the membership of a component to a module. Actually, in the clustering algorithm, the fact of belonging to a module is determined by the highest coupling intensities identified and these intensities are not filtered. The filtering process can be automated and driven by the DSM density. A key requirement is to preserve at least one interaction for each element. In the following development of this paper, the filter is adjusted in order to obtain DSM with a maximum density of 70%.

5. The clustering algorithm

Clustering is a method that, given the mapping of interactions through the use of DSM or graphs, generates modules in a systematic way by optimizing an objective function. In this section, we briefly review interesting clustering algorithms and present the clustering algorithm that we use.

5.1. Previous works

The goal of a clustering algorithm is to group elements together into clusters. Hartigan [42] reviewed the basic approach of clustering and discussed different applications of clustering algorithms. The original goal of clustering was to find similarity between elements and group them together based on a threshold of similarity between elements. Recent algorithms have been developed for optimizing modular design of complex products including simulated annealing [41, 42], genetic algorithms [21, 22, 23] or an algorithm based on the real options theory [24] originally developed by Baldwin and Clark [4].

The clustering algorithm used in our research work is based on an algorithm developed by Idicula [40] and improved by Fernandez [41] and Thebeau [42]. This algorithm is a hierarchical bottom-up (aggregation) clustering algorithm. This algorithm has been often used in modularization work [63] or analyzed to enhance its performance. According to Yu and al. [22, 23], the Thebaud algorithm is likely to be trapped in local optimal solutions. It often fails to accurately predict the formation of "good" clustering arrangements for complex product architectures. Then they propose a new algorithm. They formulate a more complex objective function based on an information theoretic measure of modularity and they use a more robust search strategy, i.e. a genetic algorithm. More recently [64], the previous algorithm has been refined. The authors present an overview of previous dependency rating scheme and propose a new one taking spatial interactions into account. Although researchers criticize the Thebaud algorithm and propose partial comparisons [22, 63], direct benchmarking is unavailable to compare existing clustering methods.

The previous algorithms have been developed for investigating the optimal architecture of a single product without any commonality considerations that arise in family approaches [23]. New approaches enable to better specify modules and interfaces across all the variants in a product family, using a 3D DSM representation [26] or based on the real options literature [65].

5.2. Principles of the Thebaud clustering algorithm

As previously mentioned the Thebaud algorithm is based on an algorithm developed by Idicula [40]. This algorithm assumes an underlying directed graph model for the development effort, and uses a depth-first-search technique to solve the problem. It groups the "project tasks into clusters that are loosely connected with each other, while each cluster consists of densely connected inter-coupled tasks". This clustering algorithm is built around an objective function called the Total Coupling Cost. It attempts to capture the following observations in a mathematical form:

- The cost addressing an interaction is proportional to the importance of this interaction. An important interaction requires more attention, more resources, or more work from designers. Thus, a coupling with high value will have a high coupling cost.
- It is assumed that in a modular architecture, if a given element belongs to a module, it is tightly coupled to other elements in

this module and at the same time, this component is weakly coupled to elements belonging to other modules.

- For single elements, the cost of being a member of a module increases with the number of modules addressing a coupling. This last point reflects the fact that if an element was to be part of different modules, the cost would be proportional to the number of modules where the interaction is addressed. Particularly, an integrative element is an element which should not belong to any module since it has been coupled to many elements belonging to different modules.

For each element in the DSM, the algorithm calculates a coupling cost. Then the sum of the coupling costs for each element gives a total coupling cost. The notations are described in Table 2.

Equations 3 and 4 show the coupling cost for an element. If both elements i and j are in the same cluster k, then the coupling cost is calculated through Equation 3. Otherwise, if no cluster contains both i and j, Equation 4 is used where the entire DSM acts as a cluster containing i and j. Equation 5 is the expression for the total coupling cost function that the algorithm attempts to minimize. The parameter pow_cc plays an important role since it penalizes the size of the clusters in the coupling cost function in Eq. 3 and then it affects their number and their size.

Table 2. Notations for the clustering algorithm

size	the size of the DSM, i.e. the number of elements.
DSM(i,j)	the coupling value (or strength of interaction) between elements i and j. Note that when i=j, DSM(i,i)=0.
cl_size(k)	the number of elements contained in cluster k.
pow_cc	a parameter that controls the type of penalty assigned to the size of the cluster in the coupling cost.

For an interaction between elements i & j that occurs within the cluster k

$$Coupling\ \ Cost\left(e_i, e_j\right) = \left(DSM\left(i, j\right) + DSM\left(j, i\right)\right) \times cl\_size(k)^{pow\_cc} \quad \textbf{(Eq. 3)}$$

For an interaction between elements i & j that occurs outside a cluster

$$Coupling\ \ Cost\left(e_i, e_j\right) = \left(DSM\left(i, j\right) + DSM\left(j, i\right)\right) \times size^{pow\_cc} \quad \textbf{(Eq. 4)}$$

$$Total\ \ Coupling\ \ Cost = \sum_{j=1}^{size} \sum_{i=1, j \neq i}^{size} Coupling\ \ Cost\left(e_i, e_j\right) \quad \textbf{(Eq.5)}$$

In order to maximize the efficiency of the clustering algorithm for exploring the whole solution space, the Thebaud algorithm implements two simulated annealing operations. In two different steps of the algorithm, the decision on what to do next step is not determined through the available data, but instead is run through a random process. These operations allow the algorithm to reach solutions that it would otherwise have left out. The solution space explored by the algorithm is increased, and the likelihood of being "trapped" in a sequence of steps that leads to a sub-optimal solution is minimized.

Finally a comment is required concerning the term "optimal" that is rather an abusive term. First, the problem of clustering is N-P hard. We can not prove that the result obtained is the optimum. Second, defining architecture is a multi-criteria decision making. Other product life-cycle criteria have to be taken into account to refine or validate the obtained architecture.

5.3. Displaying the results

Different ways for displaying the results can be used. First, the clustered DSM can be represented as a graphical matrix with diamonds like in Fig. 9. We choose this representation in order to display the results related to the industrial case we develop in this paper. Direct inspection of the DSM itself is one valid way of interpreting the product architecture [28, 66]. Other representations that could better support interpretation and decision-making may be used, particularly to reveal the inherent hierarchical modularity within the structure, for instance, dendograms [67], molecular diagrams [28], node-link diagrams (or network representations) [33, 66], 3D DSM [26] or colored groupings [21] that represent different strengths of modularity. In the method we propose, we are going to use another hierarchical representation, similar to the dendograms. By increasing the parameter X in the filtering process, we will be able to highlight cohesive modules at different filtering levels.

6. Application on a powertrain architecture

In the framework of a research contract with a French automotive manufacturer, we have analyzed system architects' activities concerning the development of powertrains (engine block and robotized gearbox).

Insert Figure 6. The engine SF DSM

Insert Figure 7. The engine SF-PC Incidence Matrix

We apply our approach to the design of a car engine. System architects have already identified the list of System Functions and components. Figures 6 and 7 show the engine decomposition into 13 SF and 15 components.

6.1. Capturing the SF DSM and the SF-PC IM

In the preliminary stages of the engine development project, system architects have little information about the project domains characteristics. Frequently in this design stage, we have better information about the functional behavior of the product than about its sub-system couplings.

In the engine development project considered in this work, we capture the SF DSM by identifying and evaluating couplings between engine functions (Fig. 6). We use the typology of interactions proposed by Pimmler and Eppinger [18] (spatial, energy, information, materials) to estimate the strength of interactions and we compute a mean value. Note that according to Axiom 4, we add the value 10 on the DSM diagonal.

We also construct the SF-PC Incidence Matrix that exhibits couplings between engine functions and engine components (Fig. 7). In the cells of this matrix, the value corresponds to the weight of a function that can be allocated to components that contribute to this function. The weight is scored on a scale between 0 (no contribution) and 10 (strong contribution).

Without having direct knowledge about component interactions, we will apply the propagation method presented above in order to simulate a PC DSM and then identify the corresponding PC architecture.

6.2. SF architecture identification

By applying the clustering algorithm directly on the SF DSM, we obtain the architecture in Figure 8. This architecture is as follows:

- SF Module 1: Air providing, Fuel providing, Gas Cleaning-up, Combustion and control SFs.
- SF Module 2: Pressure-Torque Conversion, Power Transmission, FLV, Secondary Energy Conversion and Coupling SFs.
- Integrative elements: Functional Volumes, Thermics and Vibration SFs.

Insert Figure 8. SF architecture

The first module corresponds to pre-combustion system functions and groups all the functions leading to the realization of the combustion. The second module corresponds to post-combustion system functions and groups all the functions transforming the energy of combustion into mechanical energy.

The integrative SFs identified are functions that affect all the engine components and so these functions are coupled to all the other engine functions. The architecture identified by the clustering algorithm is empirically identified by the project designers but not formalized because of the high complexity of the engine architecture.

In the following development of this paper, we will analyze the impact of the SF architecture identified on the physical engine architecture.

6.3. PC DSM simulation and clustering

By applying the construction axioms and the fuzzy treatment presented in section 4 on the SF DSM and SF-PC IM, we generate 169 PC DSMs (13x13). After aggregating the 169 PC DSMs using the average method we obtain a dense DSM. In order to increase the clustering algorithm efficiency in identifying the underlying architecture in the PC DSM, the filtering process used a threshold equal to 2.9 (that leads to a density of 70%). By applying the clustering algorithm on the filtered PC DSM, we obtain the physical engine architecture shown in Figure 9. The physical architecture is composed of:

- PC Module 1: This module is composed of 6 components (EGR, Fuel system, Breech, Air intake, Exhaust and Sensors). All these components form the top part of the engine and contribute to create the thermal energy.
- PC Module 2: This second module is also composed of 6 components (Camshaft, Accessory and Synchronous drive, Vacuum and Cooling circuits and Secondary energy generator), most of these components form the bottom part of the engine and contribute to transfer the kinetic energy of the engine.

- There are 3 integrative elements: Casing, Crankshaft and Lubrication. System architects are not familiar with the architecture typology adopted in this work but they all agree on the importance of these components and the multiplicity of their interfaces.

In order to refine our analysis concerning the efficiency of the propagation method, we compared the simulated PC architecture first to the SF architecture and second to a PC architecture directly constructed.

Insert Figure 9. Simulated PC architecture

6.4. Comparison between the SF architecture and the PC architecture

The SF architecture identified in part 6.2 is composed of both modular and integrative functions. The integrative functions are coupled to all the other functions and so to all the components. The effect of these integrative functions on the PC architecture is diffuse. Consequently, it will only be interesting to analyze the impact of the SF modules on the PC architecture. When comparing the modules identified in the two architectures, we notice that there is coherence between the pre-combustion SF module and the physical top engine module. In fact, this engine part essentially contributes to the realization of the combustion.

In the same way there is also coherence between the post-combustion SF module and the bottom engine module. Most of the components in this module contribute to transfer the kinetic energy and so realize post-combustion functions. However, the integrative components mainly contribute to post-combustion functions. The casing is the component where most of post-combustion functions take place and the crankshaft contributes to transferring the kinetic energy.

So, in order to analyze the integrative components, we need to refer to the architects' expertise by submitting the obtained architecture to system architects or by comparing the simulated PC architecture to the architecture of a PC DSM directly constructed by the project designers. In our collaboration with an automotive manufacturer, we were offered the possibility of studying the whole engine development project from its preliminary stages to the detailed design stages. This allowed us to have access to the PC DSM manually constructed.

6.5. Comparison between the simulated and real PC architecture

In this section, we compare the simulated PC architecture obtained in the preliminary stage of the product development to the constructed PC architecture obtained during the detailed design phase. But what makes two DSMs similar? We may answer that they have the same architecture. Figure 10 shows the PC architecture observed after realizing the engine design.

Insert Figure 10. Real PC architecture

When we compare the real PC architecture (Fig. 10) and the simulated one (Fig. 9), we observe that:

- The two architectures are composed of 2 large modules and 3 integrative components (PC 12 is a module with an only element, as shown in Fig.10).
- The modules are almost the same in the two architectures, one corresponding to top engine part and the second to bottom engine part.
- The camshaft (PC 6) is identified in the real PC architecture as belonging to the top engine part. In the simulated PC

architecture, this component belongs to post-combustion module. This difference between the simulated and real PC architectures can be explained by the fact that in its working cycle, the camshaft depends on the kinematic diagram of the bottom engine part. Thus, in the simulated architecture the kinematic dependency appears to be stronger than the physical interfaces with the top engine components.

- In the real architecture, the vacuum circuit (PC 12) has few links with other components and is a module composed of a single component. According to the system architect, the intensities of coupling between this component and the SF may be over-estimated in the Incidence Matrix.

The other differences between the two architectures are linked to the identification of integrative components.

- In the real PC architecture the integrative components are Casing (PC 8), Breech block (PC 3) and Crankshaft (PC 7). These components realize the physical cohesion of the engine externally and internally. They map together all the other engine components. In the simulated PC architecture, integrative components are identified through the importance of their interactions with the other components of the engine. We observe that the simulated architecture identifies the casing and the crankshaft as being integrative but not the breech. This highlights the fact that the global integrative role of the breech is weaker than the two other components. Actually, this component is more integrative internally within the top engine module but it also contributes to the global engine cohesion.
- The last difference between the two lists of integrative components is the identification of lubrication (PC 9) in the simulated architecture as integrative. This may be a consistent choice due to the high number of components interacting with oils, but the multiplicity of interfaces is not sufficient to make a component integrative. The system architect judges that this component role is not important enough to make it integrative.

6.6. Discussion

After comparing the simulated and the real PC architectures, we can conclude that the propagation method proposed in this paper is efficient. To build a PC DSM, the system architects need detailed knowledge of the interactions within the system under development [55]. This knowledge is usually available during the detailed design phase or for an incremental design (re-engineering). The method proposed here is able to simulate the PC DSM and the underlying architecture in the preliminary stages of an engine development project. It may help the system architect to improve the allocations of functions onto components and their traceability through the Incidence Matrix.

Other researchers have presented similar approaches, although different from the proposed method. Tseng and Jiao [59] develop a method to identify modules by analyzing the design matrix (DM) directly. The ROC clustering algorithm used in group technology is applied to the modular analysis of the DM. After a study concerning strengths and weaknesses of DM and DSM, Dong and Whitney [54, 55] conclude that these matrices complement each other. Through a simple matrix transformation procedure, their method starts with a DM and obtains a PC DSM. A strong assumption of this method is that the DM is always

constructed following the rules in AD [2]. Each DP is identified to mainly address an FR. Hence the DM is always square, with higher values for the diagonal marks than for the others. If this assumption is verified, this method provides an efficient way to predict system interactions at multiple levels of the system hierarchy. Then sequences and iterations within the system may be identified by applying a DSM partitioning algorithm.

However, Holmqvist and Persson [68] claim that the independence axiom of AD is an assumption: although independent functions are desirable, it is not always possible to make functional requirements independent. The SF DSM and the SF-PC IM allow the capturing of functional couplings.

Moreover, the DM may be difficult to build in most companies where design knowledge concerning the mapping of functions onto components is not based on the AD assumptions. Therefore, the main differences between the method we propose and the Dong method are, first that we do not assume that the construction of the Incidence Matrix relies on the rules in AD and, second, that we combine the SF DSM and the IM to obtain the PC DSM. Indeed, we take into account potential relationships between two components that may be indirect through two SF (represented both in the SF DSM and the IM) and direct through an only SF (represented in the IM). We assume that the construction of the SF DSM occurs at an earlier stage as the one of the SF-PC IM. This assumption is certainly true in modular or architecture innovations [55, 69]. We try to propagate functional constraints to the physical view through the IM to help the system architects to be aware of the effects of the decisions they make in the early design process. However, the comparison with this method is helpful to identify further improvements we could bring to the proposed method. We built the matrices by interviewing system architects without fixing any rules. This may result in dense matrices where the system may appear to be almost integral. If the method was implemented on a larger system (more than 15 elements) than the engine considered in the case, design rules should be recommended. Otherwise, the obtained DSM may be too dense with close values and the DSM clustering inefficient.

Concerning the Thebaud clustering algorithm, due to the randomness in simulated annealing, several runs of the algorithm are required to provide a good DSM clustering. It is also recommended to change the initial DSM arrangement and to run the algorithm again since, like other algorithms based on simulated annealing, the Thebaud algorithm may be dependent on the initial configuration. Finally, this algorithm contains an interesting consistency analysis of different runs and a likeness measurement index of clusters to help the user to judge the relevance of the suggested architecture.

7. Sensitivity analysis

The proposed method uses many subjective values estimated by systems architects as well as various parameters in the fuzzy inference systems or in the clustering algorithm.

We divide the sensitivity analysis into two studies because the method is clearly split up into two main stages: a matrix transformation through a fuzzy processing and the use of a clustering algorithm.

In other research work concerning modularization algorithms, few sensitivity analyses have been led [63, 70]. Particularly, Dunn and Sussman [63] have studied the sensitivity of the Thebeau algorithm. Their results suggest a relatively strong sensitivity to

the parameter powcc. They recommend the user to set pow_cc to 1 as a good starting point and to increase the value as high as 2 if relatively smaller clusters are desired or else, to decrease the value as low as 0.5. Concerning the sensitivity of the Thebaud algorithm to inaccuracy in the DSM, we have not found related research.

Consequently, we have investigated the sensitivity analysis of the matrix transformation stage with respect to input values. We have run different types of analysis:

A- Input change to simulate an overall inaccuracy in the architects' original estimations. We model inaccuracy by adding a Gaussian offset to the initial value $M_{i,j}$ to obtain the changed input value $M'_{i,j}$. This normal distribution is defined with mean $\mu=0$ and variance SD^2 where SD is the Standard Deviation:

$M'_{i,j} = M_{i,j} + \mathcal{N}(0, SD^2)$

We may consider two cases:

- A-1. a SD1 increase or decrease in SF-PC IM values (SD1 is scored in a range from 0.2 to 1.4 with a step of 0.2).
- A-2. a SD2 increase or decrease in FS DSM values. (SD2 is scored in a range from 0.2 to 1.4 with a step of 0.2).

B- Input change to simulate a Gaussian inaccuracy on a specific SF (a SD3 increase or decrease in the coupling value, with SD3=1: a strong inaccuracy):

- B.1. an integrative function (we choose SF 9).
- B.2. a non-integrative one (we choose SF 2).

C- Idem A.1 and we add the filtering procedure to obtain a density of approximately 70%.

D- Parameter change to simulate inaccuracies in the trapezoidal membership functions. We estimate the sensitivity of the fuzzy processing according to three parameter settings for both input and output trapezoidal membership functions:

Input membership functions:

- Setting 1 = { [0 0 3 5], [3 5 7 9], [7 9 10 10] }
- Setting 2 = { [0 0 3 5], [3 5 6 8], [6 8 10 10] }
- Setting 3 = { [0 0 4 6], [4 6 7 9], [7 9 10 10] }

Output membership functions:

- Setting 4 = { [0 0 1 3], [1 3 7 9], [7 9 10 10] }
- Setting 5 = { [0 0 2 4], [2 4 7 9], [7 9 10 10] }
- Setting 6 = { [0 0 1 3], [1 3 6 8], [6 8 10 10] }

Each run follows the same procedure with 30 different computations of the changed matrix. Then for each type of change, we compute two comparison criteria for judging the effects of input changes on the original PC DSM: the overall output mean and the overall output SD of the difference between the "PC DSM with change" and the "original PC DSM" (without change). In doing so, we assume that firstly, two DSMs are similar if they have the same mark at the same location [55] and secondly, that the output

variation is normally distributed. We have checked the latter hypothesis by plotting the histogram of the output variation: the distribution has been close to the bell curve.

Figure 11 displays the results. They are favorable showing the low sensitivity of the proposed matrix transformation stage as far as the input changes are concerned.

Insert Figure 11. Sensitivity to input inaccuracy

In case A-1, we observe that a change of SD1 on the SF-PC IM values leads to a change of about half SD1 on the original PC DSM. In case A-2, the results show that the output is rather little sensitive and is much less sensitive than in the previous case.

In case B, the change introduced only to an integrative function implies a change of the output value which matches an overall input inaccuracy with a SD of 0.2 (case A-1) and is 70% higher than in the B2 case concerning the change of a non-integrative function (output SD =0.12). We may recommend particular attention to the estimation of the couplings concerning the integrative elements in the SF-PC IM. However, since these elements tend to generate couplings between all components, we may decide to remove them from the IM and to conduct again the propagation analysis.

In case C, we run the filtering procedure after the use of fuzzy processing. We fix the threshold X equal to 2.9. We observe that whereas the mean variation of the PC DSM values is relatively the same, the SD is slightly higher than in case A-1 without any filtering. This observation may be expected since the filtering procedure makes the values around the threshold become zero in one matrix and not in the other one.

Table 3 shows the results concerning the case D. The row (resp. column) corresponds to a change to the parameter setting describing the input (resp. output) functions. We observe that if we change the setting concerning the input functions, the output values are very sensitive (the output SD is estimated on a range between 0.59 and 0.92). But the fuzzy processing has a rather low sensitivity to change in the output functions only (SD=0.24 or 0.41).

Table 3. Sensitivity to parameter change

Input Output	Setting 1	Setting 2	Setting 3
Setting 4	SD =0 (original)	SD = 0.74	SD = 0.73
Setting 5	SD = 0.41	SD = 0.92	SD = 0.80
Setting 6	SD = 0.24	SD = 0.59	SD = 0.80

However, we have to keep in mind that the selected shape comes from discussions with system architects and aims at mimicking their reasoning. We have tested the selected parameters by applying this fuzzy processing to a former engine design.

Finally, we have briefly studied the sensitivity analysis of the global method, that is, the use of the proposed matrix transformation and the Thebaud algorithm. We have run the clustering algorithm ten times to compare the two architectures obtained by

- the original PC DSM and
- the changed one in the case of A-1 with SD=1.

Since we are interested in the arrangement of clusters, a direct gauge of sensitivity is to analyze the number of elements that remain in the same cluster versus the number that do not [63]. We have found that in six runs, the PC DSM clustering reveals the same architecture and in the other four runs, one or two components may be placed in another cluster. We believe that this deviation is acceptable and does not call into question the proposed method. The use of another algorithm such as the ones developed by Yu and al. [22, 23] may be useful since it provides a better likelihood of reaching an optimal DSM clustering, even if it may be at the expense of longer computational time. Further sensitivity analyses are required to study the impact of input inaccuracy on existing clustering algorithms.

8. Conclusions and further work

The propagation method presented in this paper simulates a PC DSM from a given numerical SF DSM and a SF-PC IM, and identifies the underlying PC architecture. If the system architects were not satisfied with the result, they could change the initial matrices and simulate the new PC architecture. This tool could then reduce the number of traditional design iterations. It is based on a new fuzzy inference system that generates a DSM. Then, an existing clustering algorithm groups elements into modules and integrative elements.

The architectures generated by this method are recommendations only but the architects should be aware of the fact that the choice of other modules could increase coordination and teams' efforts to develop the system. The product architectures can deeply influence the design process structure since the product modules should be designed by different design teams independently [9].

In this paper, we intentionally apply the method on two product domains only. More generally, this method may be used similarly to propagate constraints or changes from one project domain to another one, for instance, expectation – SF, product – task, task – team.

The objective of the proposed method is the identification of domains architectures and not the precise valuation of the interactions. That’s why we use fuzzy treatment which is less sensitive to input values variability. Moreover, the filtering step makes the clustering algorithm converge more frequently and rapidly to the optimal architecture without changing the architecture result in itself.

Finally, further work is envisaged. We aim at providing system architects with an integrated tool that allows them to simulate and structure the different project domains in a concurrent manner. In the preliminary design stages ("planning and clarifying the task", "conceptual design" [58]), the system architect who plays the role of project manager too (or strongly collaborates with

him/her) has to design concurrently the preliminary product architecture and the overall project structure, related to the embodiment and detailed design stages. This topic has received much attention recently [49, 51, 71, 72].

In re-engineering situations, the system architect may re-use crucial knowledge structured in past projects and included in different incidence matrices and DSMs. Thus, depending on the modifications introduced by the re-engineering situation, we can assist the system architect by providing him with a tool that allows on the one hand, to test the robustness of the architectures and on the other hand, to propagate modifications and constraints throughout the different domains of the project: product, process and organization. Finally, other industrial applications are in-progress to refine the current propagation method. The question of what makes two DSMs similar may be investigated and a further analysis of unmatched marks should be conducted to study their consequences during the detailed design phase.

REFERENCES

[1] IEEE Std 1220™, 2005, "*Standard for Application and Management of the Systems Engineering Process*", IEEE Computer Society, Revision of IEEE Std 1220-1998.

[2] Suh N. P., 1990, "*Principles of Design"*, Oxford University Press, Cambridge, UK.

[3] Ulrich K.T., 1995, "The Role of Product Architecture in the Manufacturing Firm", *Research Policy*, 24, pp. 419-440.

[4] Baldwin C.Y., and Clark K.B., 2000, "*Design Rules: The Power of Modularity Design*", MIT Press. ISBN 0262024667.

[5] Maier M.W., and Rechtin E., 2002, "*The Art of Systems Architecting*", 2nd edition, CRC Press, New-York.

[6] Liu Y., Chakrabarti A. and Bligh T.P., 1999, "Transforming Functional Solutions into Physical Solutions", *Proceedings of the ASME Design Theory and Methodology Conference, DETC/DTM-8768*, 12-16 September, Las Vegas, NV.

[7] Baxter J.E., Juster N.P., and Pennington A., 1994, "A Functional Data Model for Assemblies Used to Verify Product Design Specifications." *Proceedings IMechE*, Part B, 208, pp. 235–244

[8] Browning T.R., 2001, "Applying the Design Structure Matrix to System Decomposition and Integration Problems", *IEEE Trans. Eng. Mgt*., 48, pp. 292-306.

[9] Sosa M., Eppinger S. and Rowles C., 2003, "Identifying modular and integrative systems and their impact on design team interactions", *ASME Journal of Mechanical Design*, 125, June 2003, pp. 240-252.

[10] Muffatto M. and Roveda M., 2002, "Product Architecture and Platforms: a Conceptual Framework", *Int. J. Technology Management*, 24(1), pp. 1-16.

[11] Farrell S., and Simpson T.W., 2008, "A Method to Improve Platform Leveraging in a Market Segmentation Grid for an Existing Product Line", *ASME Journal of Mechanical Design*, 130(3), 031403 (11 pages).

[12] Khajavirad A., and Michalek J., 2008., "A Decomposed Gradient-Based Approach for Generalized Platform Selection and Variant Design in Product Family Optimization", *ASME Journal of Mechanical Design*, 130(7), 071101 (8 pages).

[13] Simpson T.W., 2004, "Product Platform Design and Customization: Status and Promise", AIEDAM, Special Issue: Platform

Product Development for Mass Customization (January 2004), 18(1), pp.3-20

[14] Zha X.F., and Sriram R.D., 2006, "Platform-based Product Design and Development: A Knowledge-Intensive Support Approach", *Knowledge-Based Systems,* 19, pp. 524-543.

[15] Jiao J., Simpson T., and Siddique Z., 2007, "Product Family Design and Platform-based Product Development: a State-of-the-art Review", *Journal of Intelligent Manufacturing*, special issue on Product family design and platform-based product development, 18(1), pp. 5-29.

[16] Alizon F., Khadke K., Thevenot H.J., Gershenson J.K., Marion T.J., Shooter S.B. and Simpson T.W., 2007, "Frameworks for Product Family Design and Development", *Concurrent Engineering*, 15, pp.187-199.

[17] Steward R. P., and Donald V., 1981, "The Design Structure System: A Method for Managing the Design of Complex Systems", *IEEE Transactions on Engineering Management,* 28(3), pp.71-74.

[18] Pimmler T.U., and Eppinger S.D., 1994, "Integration Analysis of Product Decompositions", *Proc. ASME Design Theory and Methodology Conference (DTM'94),* DE-Vol 68, pp. 343-351.

[19] Kusiak A., and Huang C.C., 1996, "Development of Modular Products", *IEEE Trans. on components, packaging and manufacturing technology – part A*, 19(4), pp. 523-538.

[20] Stone R., Wood K., and Crawford R., 2000, "A Heuristic Method for Identifying Modules for Product Architectures", *Design Studies*, 21, pp.5-31.

[21] Whitfield R.I., Smith J.S. and Duffy A.H.B., 2002, "Identifying Component Modules", *Proceedings of Seventh Int. Conf. on Artificial Intelligence in Design,* Cambridge, United Kingdom, 15-17 July 2002, pp. 571-592.

[22] Yu T., Yassine A. and Goldberg D., 2003, "Genetic Algorithm for Developing Modular Product Architectures", *Proc ASME 15th Int Conf Des Theory Methodol*, Chicago, September 2003.

[23] Yu T., Yassine A. and Goldberg A., 2007, "An Information Theoretic Method for Developing Modular Architectures Using Genetic Algorithms," *Research in Engineering Design*, 18(2), pp. 91-109.

[24] Sharman D. and Yassine A., 2007, "Architectural Valuation using the Design Structure Matrix and Real Options Theory", *Concurrent Engineering*, 15(2), pp. 157-173.

[25] Alizon F., Thevenot H.J. and Shooter S.B., 2006, "Design Structure Matrix Flow for Improving Identification and Specification of Modules", *ASME 2006 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Philadelphia, PA, ASME, Paper No. DETC2006-DTM99524.

[26] Alizon F., Moon S.K., Shooter S.B. and Simpson, T.W., 2007, "Three Dimensional Design Structure Matrix with Cross-Module and Cross-Interface Analysis", *ASME Design Engineering Technical Conferences – Design Automation Conference*, Liou, F., ed., Las Vegas, NV, September 4-7, ASME, Paper No. DETC2007/DAC-34510

[27] Lee H., Seol H., Sung N., Hong Y.-K. and Park Y., 2007, "Analyzing Design Change Impacts in Modular Products with Analytic Network Process", *ASME 2007 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, Las Vegas, NV. Paper No. DETC2007-35880.

[28] Sharman D., and Yassine A., 2004, "Characterizing Complex Product Architectures", *Systems Engineering Journal*, 7(1), pp. 35-60.

[29] Chen K.M, and Liu R.J., 2005, "Interface Strategies in Modular Product Innovation", *Technovation*, 25, pp. 771-782.

[30] ESD Architecture Committee (Crawley E., De Weck O., Eppinger S. et al.) Monograph, 2004, "The Influence of Architecture in Engineering Systems", *Engineering Systems Symposium*.

[31] Dobrescu G., and Reich Y., 2003, "Progressive sharing of modules among product variants", *Computer-Aided Design,* 35, pp. 791-806.

[32] Stone R., McAdams D. and Kayyalethekkel V., 2004, "A product architecture-based conceptual DFA technique", *Design Studies*, May, 25(3), pp. 301-325.

[33] Sosa M., Eppinger S. and Rowles C., 2007, "A Network Approach to Define Modularity of Components in Complex Products", *ASME Journal of Mechanical Design*, 129 (11), pp. 1118-1129.

[34] Blackenfelt M., and Sellgren U., 2000, "Design of robust interfaces in modular products", DETC00, *Proc. of the ASME Design Engineering Technical Conferences*, Paper no. DETC00/DAC-14486, Sept. 10 13, Baltimore.

[35] Fixson S.K., 2005, "Product architecture assessment: a tool to link product, process and supply chain design decisions", *Journal of Operations Management*, 23, pp. 345-369.

[36] Erixon G., 1998, "Modular Function Deployment - A Method for Product Modularisation", *PhD diss.*, Royal Institute of Technology, Stockholm, Sweden.

[37] Gu P., and Sosale S., 1999, "Product modularization for life cycle engineering", *Robotics and Computer Integrated Manufacturing*, 15, pp. 387-401.

[38] Alexander C., 1964, "*Notes on the Synthesis of Form",* Harvard University Press, Cambridge, MA.

[39] Hartigan J., 1975, "*Clustering Algorithms",* Wiley & Sons, New York, NY.

[40] Idicula J., 1995, "Planning for concurrent engineering", Thesis draft, Nanyang Technology University.

[41] Fernandez C., 1998, "Integration analysis of product architecture to support effective team co-location", *Master of Science Thesis*, MIT.

[42] Thebeau R., 2001, "Knowledge management of system interfaces and interactions for product development processes", *Master of Science Thesis*, MIT.

[43] Malmqvist J., 2002, "A classification of matrix based methods for product modeling", *Proceedings of Design 2002*, Cavtat-Dubrovnik, Croatia.

[44] Clarkson P. J., Simons C., and Eckert C., 2001, "Predicting Change Propagation in Complex Design", Proceedings ASME DETC 2001, Paper No DETC2001/DTM-21698, Pittsburgh, PA, USA.

[45] Dahmus J.B., Gonzalez-Zugasti J.P., and Otto K.N., 2001, "Modular Product Architecture", *Design Studies*, 22(5), pp. 409-424.

[46] Chen L., Ding Z. and Li S., 2005, "A Formal Two-Phase Method for Decomposition of Complex Design Problems", *ASME*

Journal of Mechanical Design, 127(2), pp.184-195.

[47] Akao Y., 1990, "*Quality Function Deployment, QFD - Integrating Customer Requirements into Product Design"*, Productivity Press, Portland, ON, USA.

[48] Strawbridge Z., McAdams D., and Stone R., 2002, "A Computational approach to conceptual design", *Proc. DETC 02*, Montreal, Canada, Sept. 29-Oct. 2.

[49] Danilovic M., and Browning T.R., 2007, "Managing complex product development projects with design structures matrices and domain mapping matrices", *Int. Journal of Project Management,* 25, pp. 300-314.

[50] Eppinger S.D., and Salminen V., 2001, "Patterns of product development interactions", *Int. Conf. on Engineering Design, ICED 01, Vol. 1,* Glasgow, 21-23/08, pp. 283-290

[51] Sosa M., 2007, "Aligning Process, Product, and Organizational Architectures in Software Development", *Int. Conf. Engineering Design, ICED'07*, 28-31/08, Paris.

[52] Meier C., Yassine A., and Browning T., 2007, "Design Process Sequencing With Competent Genetic Algorithms", *ASME Journal of Mechanical Design*, 129(6), 566 (20 p.).

[53] Hölttä K., and Salonen M., 2003, "Comparing three different modularity methods", *Proc. of DETC'03, Design Engineering Technical Conferences, ASME 2003*, Chicago, USA, September.

[54] Dong (Hommes) Q. and Whitney D., 2001, "Designing a Requirement Driven Product Development Process", *Proc. of the 13th Int. Conf. on Design Theory and Methodology (DTM 2001)*, Sept. 9-12, Pittsburgh, Pennsylvania, USA.

[55] Dong (Hommes) Q., 2002, "Predicting and Managing System Interactions at Early Phase of the Product Development Process", Ph.D. Diss. (M.E.), Massachusetts Institute of Technology, Cambridge, MA.

[56] Suh N.P., 2001, *Axiomatic Design: Advances and Applications*, Oxford University Press, New-York.

[57] Ulrich K.T., and Eppinger S.D., 2008, *Product Design & Development*, 4th Ed, McGraw-Hill, New-York.

[58] Pahl G., and Beitz W., 1996, "*Engineering Design: a Systematic Approach",* 2nd Ed., Springer-Verlag, London.

[59] Tseng M., and Jiao J., 1997, "A Module Identification Approach to the Electrical Design of Electronic Products by Clustering Analysis of the Design Matrix", *Computers and Industrial Engineering*, 33(1-2), pp. 229-233.

[60] Kurtoglu T., and Tumer I.Y., 2008, "A Graph-Based Fault Identification and Propagation Framework for Functional Design of Complex Systems", *ASME Journal of Mechanical Design*, 130(5), 051401 (8 pages).

[61] Zimmerman H., 1991, "*Fuzzy Set Theory and its Applications"*, 2nd Ed. Kluwer, Boston, MA.

[62] Dubois D., and Prade H., 1980, "Fuzzy Sets and Systems: Theory and Applications", *Academic Press.*

[63] Dunn T.P., and Sussman J.M., 2006, "Design Structure Matrices to Improve Decentralized Urban Transportation Systems", *Transportation Research Record*, pp. 193-200.

[64] Helmer R., Yassine A., and Meier C., 2008, Systematic Module and Interface Definition using Component DSM, *Journal of Engineering Design*, December, DOI: 10.1080/09544820802563226.

[65] Kalligeros K., 2006, "Platforms and Real Options in Large-Scale Engineering Systems", PhD diss., Massachusetts Institute of

Technology, Engineering Systems Division.

[66] Keller R., Eckert C., Clarkson J., "Matrices or Node-link Diagrams: Which Visual Representation Is Better For Visualising Connectivity Models?" *Information Visualization* (2006) 5, 62–76.

[67] Hölttä K., Tang V., and Seering W., 2003, "Modularizing Product Architectures Using Dendrograms", *Proc. of Int. Conf. on Engineering Design,* 19-21 August, Stockholm.

[68] Holmqvist T., Persson M., "Analysis and Improvement of Product Modularization Methods: Their Ability to Deal with Complex Products", *Systems Engineering*, 6(3), pp. 195-209

[69] Henderson R.M. and Clark K.B., 1990, "Architectural Innovation: The Reconfiguration of Existing Product Technologies and the Failure of Established Firms", *Administrative Science Quaterly*, 35, pp. 9-30.

[70] Sered Y., and Reich Y., 2006, "Standardization and Modularization Driven by Minimizing Overall Process Effort", *Computer-Aided Design*, 38, pp.405-416.

[71] Bonjour E., Harmel G., Micaëlli J-P., and Dulmet M., 2008, "Simulating Change Propagation Between Product Architecture and Development Organization", *International Journal of Product Development*, Forthcoming, 2009.

[72] Lindemann U., 2007, "A Vision to Overcome Chaotic "Design for X" Processes in Early Phases", *Int. Conf. of Engineering Design, ICED'07*, 28 - 31 August, Paris.