Architecture-Centric Model-Based Product Development

Tetsuo Tomiyama
Faculty of Mechanical, Maritime and Materials Engineering
Delft University of Technology
Delft, The Netherlands
t.tomiyama@tudelft.nl

***Abstract*—Modern products and systems, including mechatronics systems, are increasingly becoming complex, because of multi-disciplinarity, involvement of multiple stakeholders, and needs to satisfy multiple goals simultaneously. These goals are not only traditional function, cost, and quality, but also time, sustainability, and social responsibility. Their development process needs to be well organized and supported. For this purpose, in the past a variety of methods were developed and successfully applied to the development of complex multi-disciplinary systems, including concurrent engineering, model-based engineering, and systems architecting. This paper is an attempt to integrate these methods and to form a new paradigm named architecture-centric model-based product development, in which systems architecture and function modeling plays a central role because they facilitate multi-disciplinary communication and understanding. Some research efforts of our group related to this topic will be illustrated.**

I. INTRODUCTION

Contemporary products, from consumer electronics, white goods, and passenger cars to professional apparatus and systems such as airplanes, medical equipment, production machines, and even robots, are all complex multi-disciplinary systems. *Complex* means first size; not only physical scale but also an enormous amount of tiny details packed in a limited space. While an Airbus A380 with an impressive cabin space counts two-and-half million unique components, it is also flying CPUs and wire harnesses. Both an Apple iPhone and a Toyota Prius run depending on supposedly more than ten millions lines of source code.

However, size is not the sole factor for their increasing complexity. These modern products are typically multi-disciplinary combinations of mechanical, electrical and electronics, control, software engineering, and even more. This trend of multi-disciplinarity never stops, because certainly it increases the added value of such products and systems and is regarded the source of innovation.

Involving more disciplines will result in more stakeholders, which means increased complexity of the product development process. For instance, considerations on the entire product life cycle also increase due to growing concerns about global sustainability and other cooperate social responsibilities (such as working conditions and worker's welfare). The shift from pure "products" to "product service systems" is also visible.

To tackle the complexity during the product development process, model-based engineering (as opposed to "code-based") seems a promising approach. It is a well-known practice in software engineering for better software development. This model-based design is clearly the trend even in other disciplines, if we look at, e.g., CAD/CAE/CAM for mechanical engineering and dynamic systems modeling tools in systems engineering. Obviously multi-disciplinarity requests integration of such tools, but unfortunately we need to admit that there are gaps among tools used for mechanical design, engineering analysis, systems engineering, and software development. Commercial software package venders have made progress in this direction. For instance, CATIA of Dassault Systems is capable of running systems dynamics simulations based on open Modelica [1], and Siemens Teamcenter offers requirement management capabilities that will better link to systems engineering tools [2]. However, we can still observe insufficiency in the degree of integration and the depth of such tools.

Presently, the most accepted practice for multi-disciplinary product development is concurrent engineering in which

The author gratefully acknowledges the following projects that supported the part of the research work. "Smart Synthesis Tools" and "Automatic Generation of Control Software for Mechatronics Systems" (supported by the Innovation-Oriented Research Programme "Integral Product Creation and Realization (IOP IPCR)" of the Dutch Ministry of Economic Affairs, Agriculture and Innovation), "Darwin" and "Octopus" (carried out under the responsibility of the Embedded Systems Institute in Eindhoven and partially supported by the same ministry under the BSIK program). Philips Healthcare and Océ Technologies were industrial partners of Darwin and Octopus projects, respectively.

mono-disciplinary domain teams collaborate led by a systems architect who should be a truly multi-disciplinary engineer. However, collaboration of mono-disciplinary domain teams requests considerable time and effort for communication and mutual understanding. This is pure overhead. A multi-disciplinary team organization with less emphasis on collaboration and communication is ideal but doesn't seem possible due to the size of the projects. In practice, we do not have yet appropriate tools or methods for better cross-disciplinary communication and mutual understanding.

The concept of "architecture" is often used in the context of "product architecture" in mechanical engineering, which is built upon structural connectivity and related to such concepts as modularity, product family, and product platform [3, 4]. This paper focuses on "systems architecture" in the context of systems engineering, which is wider than "product architecture".

This paper is an attempt to integrate the above-mentioned concepts and approaches related to the product development process of complex multi-disciplinary systems, *viz*., concurrent engineering, model-based engineering, and systems architecture, into a new paradigm named architecture-centric model-based product development. Systems architecture can act as an integration framework that facilitates cross-disciplinary communication and mutual understanding.

The next section analyzes the some fundamental concepts and problems of the development process of complex multi-disciplinary systems. In Section III, we propose the paradigm of architecture-centric model-based product development by combining model-based systems engineering and systems architecting. We argue that to describe systems architecture requires a good ontology and, among others, function plays the central role. Section IV briefly illustrates some tools and methods to support this new paradigm. Section V concludes the paper.

II. Development Process of Complex Multi-Disciplinary Systems

A. Multi-Disciplinarity

Multi-disciplinarity increases the added value of products (e.g., better functionalities, performance, and quality with cheaper cost). On the other hand, multi-disciplinarity increases the complexity of the product development process as well, together with size, multi-disciplinarity, and growing concerns about product life cycle issues and social factors. Primarily this increase of complexity comes from (1) interactions among subsystems and components increase due to multi-disciplinarity, and (2) the number of stakeholders who may not necessarily be able to understand each other very easily also increases.

Since most of (engineering) disciplines are quite advanced, i.e., deeper and narrower, mastering a different discipline is remarkably demanding if not impossible at all. Communication among different disciplines becomes extremely hard, because of confusions and misunderstandings caused by different vocabularies and different interpretations of the same technical terms. One of the major reasons for this is that these engineers are educated and trained in a mono-disciplinary environment.

B. Concurrent Engineering Practices

Concurrent engineering (CE) (or *simultaneous engineering*) [5, 6, 7] is the operational principle of the modern product development process. Before CE, the product development was based on the sequential waterfall model in which cost and quality formed dichotomy, i.e., better product quality almost automatically meant higher cost. Inspired by then very competitive Japanese product development practices in the late 1980s [8], CE was proposed to arrive at much faster (thereby cheaper) product development and higher product quality simultaneously. These two goals are achieved essentially first by frontloading development activities (e.g., production preparation activities) and second by allowing activity overlaps (thus "concurrently") as much as possible.

By frontloading development activities, CE tries to incorporate and make better use of knowledge of later development stages upfront to increase the product quality. This should lead to fewer mistakes and less iteration during the product development, thereby reducing the development time as well. However, this does not necessarily mean that decision-making takes place earlier. On the contrary, CE advocates the least commitment strategy that tries to postpone decision-making as late as possible to explore more options. By allowing activity overlaps, CE tries to remove unnecessary waiting processes and to shorten development time.

CE influenced on the product development organization as well. It does not assume an idealistic product development team composed of multi-functional or multi-domain engineers but rather collaboration of specialized domain engineering teams, which is realistic and preferred in industry.

Initially, the major tool of CE was concurrent (overlapping) execution of sub-processes but gradually CE expanded to cover various computer-based supporting technologies, too. According to various reports, some examples of modern CE technologies can be listed as follows [5-9].

- Various methods and tools, in particular, advanced computer-supported communication tools (e.g., Web-based tools, ontologies, and virtual reality techniques) to improve and enhance communication among stakeholders [10].
- Advanced computer based tools to model, use, share and integrate knowledge and information related to product development, including CAD (Computer-Aided Design), CAM (Computer-Aided Manufacturing), CAE (Computer-Aided Engineering), PDM (Product Data Model), PLM (Product Lifecycle Model), ERP (Enterprise Resource Planning), and ontologies.
- Management and control techniques and tools to improve responsiveness and efficiency of product development processes, such as those based on process management [11, 12], business process modeling, set-based reasoning, and DSM (Design Structure Matrix) [13, 14].

- Methods and tools to take a wide range of life cycle related issues into design consideration, such as Design for X (DfX) [7] and Life Cycle Assessment (LCA).

C. Systems Architecting

The development of complex multi-disciplinary systems can be discussed from two aspects. On one hand, the V model is the most widely used model of the product development process in systems engineering and has three principal sub-processes, *viz.*, decomposition (architecting), implementation, and verification and validation (Fig. 1) [15, 16]. On the other hand, systems engineering deals with "system" as a whole and articulates systems architecture. The process of developing systems architecture is called "architecting". A reference architecture models the architecture of a specific class of products. It works as a guideline for systems architects and helps them to achieve specific goals that the target system needs to realize.

The V model can incorporate the systems architecture model given by Muller in Fig. 1 [17]. In this triangle, the vertical axis signifies the number of elements, hence the level of abstraction. At the bottom level, the number of details of modern products may reach $O(10^6)$ or even $O(10^7)$. The top layer signifies the functional requirements, which can be at most $O(10^3)$ and handled relatively easily even without a sophisticated tool. The bottom layer is the mono-disciplinary detail level for which sophisticated computer-based tools are available. The middle layer is the multi-disciplinary systems level. At this moment, we do not have a perfect tool to handle this level besides systems level description languages such as XML and SysML.

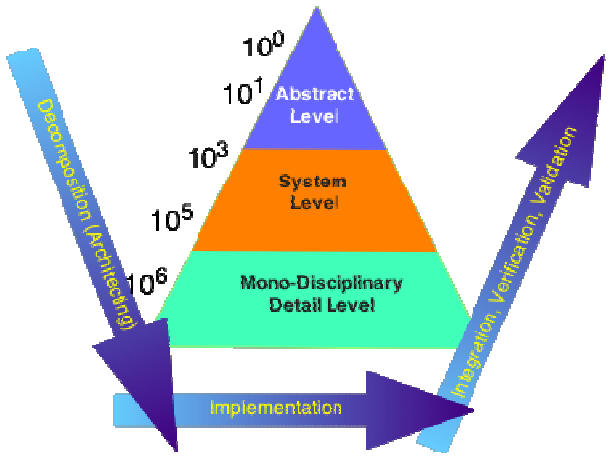

Figure 1. Systems architecture and the V model

In Fig. 1, we go from the top layer to the bottom layer, which is a systems architecting process that includes decomposition and detailing and takes place in the following process:

- Identifying (decomposed) requirements at that level.
- Identifying elements (subsystems and components) needed to achieve those identified requirements
- Describing behaviors of the elements
- Identifying interfaces among the elements
- Describing interactions among the elements.

For each decomposition level, the architect obtains a set of architectural options that should be evaluated against requirements and selects best options. This evaluation is possible only when there is sufficient information to do so. If not, the evaluation should be postponed until all the details are determined by lower level decompositions and implementations. When all sufficient details are obtained through the lowest level implementation, the process moves to the integration, verification, and validation phase.

Besides the above-mentioned generic guidelines, systems engineering does not offer a specific or concrete formal method for systems architecting, especially for systems decomposition. This contrasts with the implementation phase and the integration, verification and validation phase in which much more specifically defined methods are available. This is partially because at the top level, the descriptions are only abstract and generic. An example of formally defined architecting methods is the CAFCR method (Customer objectives, Applications, Functional, Conceptual, and Realization) [17].

D. Problems of the Product Development of Complex Multi-Disciplinary Systems

The product development process of complex multi-disciplinary systems, which is largely influenced by the CE philosophy, is not yet perfect, despite significant effort spent in the development of theories, methods, and tools to support it. Below, we list a variety of problems associated with this [18-20].

1) Communication problem: Dispite the effort to improve communicatons among engineers and stakeholders from various domains, confustions and misunderstanings still happen. Poor communication also results in loss of information.

2) Decomposition problem: Complex multi-disciplinary product development employs the divide & conquer (D&C) principle that divides the whole problem (that cannot be solved as one problem) into subproblems, finds (conquers) smaller subsolutions, and then integrates (or re-assembles) those subsolutions into an overall system level solution. In the first place, we have no established methods for decomposition valid to a variety of complex multi-disciplinary systems. Second, the D&C principle assumes that those divided subproblems are reasonably independent to each other except for interactions at interfaces. However, this is not guaranteed for many cases, because divided subproblems have interactions as the system becomes more complex and packed in a smaller space [21].

3) Integration problem: Design faults found at the integration process are extremely difficult to solve, because they are systems level unpredicted cross-disciplinary problems caused by interferences of subsystems and components that can be individually correct [22]. These will force reworking (process backtracking) of the entire design,

which can prolong the project and be costly. This means the balance between the left side of Fig. 1 and the right side is bad. While the development team often feels that spending too much time in the left side phase is "waste" of time, that might not be so after all.

4) Traceability problem: As the development process goes down to lower subsystems or component level, often high level requirements becomes less relevant. Consequently, they are overseen, forgotten, or even neglected, because there is no traceability management (just like traceability matrix in software engineering [23]) at the middle system level (Fig. 1).

5) Interface problem: Decomposed subproblems should be individually solved, but poor decomposition results in too big subproblems, so-called "coupled problems" (in Suh's Axiomatic Design [24]), or a fully loaded Design Struature Matrix [14]). Interface problems happen at the organizational level as well. Often a product development organization is organized process-wise (marketing department vs. production deparment) or discipline-wise (mechanical engineering vs. electronics department), which makes it for them difficult to collaborate each other, because these divisions are incompatible with the system decomposition.

6) Iteration problem: Despite the advances of computer-based tools (especially of simulation tools), iteration is unavoidable. This problem also leads to the change management issues.

7) Change management problem: The above-mentioned problems lead to design changes. The complexity of the target system and its development process make these changes proliferate; a small change propages to other parts of the system, especially when subsystem decomposition is not well established and subsystems have undefined or unnoticed interactions with each other. This makes change management extremely difficult or even impossible [25, 26]. To prevent this, for example, it is important to improve partitioning of decomposed subsystems as well as to keep good track of design decisions from requirements to all the way through down to details of components (cf. traceabililty problem).

In addition, while traditionally functionalities, quality, cost, and time (time-to-market as well as time-to-deliver) were the primary goals for product development, we also see other rapidly arising issues. These include considerations on sustainability (such as material and energy consumption, emission, toxicity, recyclability, and reusability) and safety. Although safety has been the primary issue for many products, for instance, ISO 26262 functional safety standard on electronics used for vehicles [27] enforces embedding such considerations in the product development process.

E. Neglect of Conceptual Design and Function Modeling

Through observations of various actual product development processes, we have discovered that in industry the conceptual design phase is almost non-existing or neglected, which is a different attitude found in academia. One of the justifications for this neglect is that product development from scratch almost never happens in reality. Most of product development projects are routine design, upgrading/scaling up design, or partial improvement design. Therefore, most of projects do not have to spend time in the conceptual design phase, such as thorough requirement analysis and functional design.

Requirements are documented but very likely these documents will be forgotten until a design review meeting. Functional design in academic sense is rarely conducted. Engineers want to move quickly to later basic and detail later stages, partly due to time pressure and partly because they think they know enough about the functions through past experiences. Consequently, they do not sufficiently explore the problem space and tend to stick to intuitively promising ideas, which often forms the fundamental cause of troubles found in a later stage.

Related to this, the concept of function is also misunderstood and not properly used by engineers during the conceptual design phase. Function modeling is taught at school as part of engineering design but completely neglected in practice. For example, function diagrams advocated by Pahl and Beitz [28], by Stone and Wood [29], or by Umeda and Tomiyama [30] are never created in practice. Engineers do not see any practical applications and advantages of generating such a diagram and they believe that such fundamental concepts as functions are too trivial to spend their precious time.

However, again this should be a wrong perception. In complex multi-disciplinary systems, it is critically essential to establish good communication and mutual understanding among engineers of different disciplines. "Function" describes "what" a system (or a machine) does or performs, whereas "behavior" together with structural information and working principles (physical phenomena) describes "how" this function is achieved. From a viewpoint of non-domain experts, it is important to understand what the machine does rather than details about how it is achieved (behavior, structure, and implementation). In this respect, function is a manifestation of one's understanding about the system and it makes connections among understandings of various stakeholders without going into implementation details, hence facilitating multi-disciplinary communication and mutual understanding.

III. ARCHITECTURE-CENTRIC MODEL-BASED PRODUCT DEVELOPMENT

In the previous section, we investigated the problems associated with the development process of complex multi-disciplinary systems; they are communication, decomposition, integration, traceability, interface, iteration, and change management problems. In addition, we pointed out that the neglect of conceptual design and function modeling seems to be causing various problems of the development process. They should be paid more appropriate attention in complex multi-disciplinary product development. To attack these problems, this paper argues that "architecture-centric model-based product development" would be a promising next generation paradigm.

A. Model-Based Systems Engineering

Model-based engineering (also known as model driven development) is a paradigm proposed in software engineering that has more emphasis on domain application knowledge embedded in models. It is contrasted against "code-based" or "document-based" programming style in which the main focus is on algorithms and data structure. The INCOSE System Engineering Vision 2020 [31] defines model-based systems engineering (MBSE) as "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases".

In more physics-oriented engineering (such as mechatronics), the concept of MBSE can be easily understood in a comparison between drawing-based product development and CAD- or simulation-based product development. Applying MBSE is expected to provide significant benefits over the document-centric approach toward enhanced productivity and quality, reduced risk, and improved communications among the system development team.

We might be able to imagine, for example, a watch combined with health monitoring functions, such as measurement of heart pulse and blood pressure, with communication functions. A CAD model is built and fed into an FEM system to check its strength, dynamic performance, thermal behavior, and so forth. Sensor dynamics is computed using Matlab/Simulink, so the control code is generated from the functional, behavioral and structural descriptions of the system. The thermal behavior should include the CPU's thermal behavior on which the control software is running. When CPU is busy with some intensive computing, the heat dissipation will result in thermal deformation of structure that can interfere with sensors. This means that, for truly multi-disciplinary problems, even multi-physics multi-scale simulation does not suffice.

In drawing-based product development, simulation models are completely separated from the central model (in this case drawings), where as in a modern CAD/CAE environment, a machine model generated with CAD can be (ideally) used for simulations with CAE. If we were able to integrate various types of models used for product development of multi-disciplinary systems, this would become truly model-based engineering. While commercial PLM packages (such as CATIA [1] and NX [2]) try to use a geometric-centric PDM system as the central model, they might find difficulties in, particularly, handling non-geometric information.

An excellent survey on MBSE can be found in [32]. My group has also developed the concept of knowledge intensive engineering which could be called a precursor of MBSE [33, 34]. Some interesting issues about models and modeling methods in MBSE can be found in [35-39].

B. Architecture and Architecting

In this paper, we use ISO/IEC 42010 (which is in fact systems and software oriented standard) in which "architecture" is defined as "fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution" [40]. Therefore, we understand that "architecture-centric model-based product development" is a product development paradigm in which various models linked in the architectural context are used to describe the product. In other words, "system architecture" describes concepts related to various views corresponding to a domain (discipline) specific view including functional, behavioral, and structural decomposition as the fundamental structure of the product. This means "systems architecture" is in fact covering the "product architecture" concept.

In the systems engineering context (see Section II-C), product development begins with translating customer requirements into technical specifications, followed by systems decomposition to hierarchically subsystems and their interfaces. After implementation (or detail design), integration verification of components and then subsystems will be carried out. Finally the total system test validates the system level solution.

During this development process, while details of the target system will be handled separately in respective modeling systems as an independent view (e.g., a functional model, a mechanical CAD model, a control model, an FEM model, etc.), a higher level view represents their relationships with each other. In other words, the architecture specifies how the system is hierarchically decomposed into subsystems and how these subsystems relate to each other. At a certain level and below, a subsystem model will become domain specific. In a domain specific view, components belong to other domains do not necessarily appear. For instance, the software system does not appear at all in physical breakdown of the system or a bill of materials for production, while control software may refer to physical concepts, such as mass, stiffness, etc., that can be computed from those physical concepts.

Therefore, in this architecture-centric model-based product development paradigm, an "architectural model" plays a central role:

- To allow system architecting operations, including decomposition, behavior and interface definition, and integration,
- To store information about the system architecture at different levels in different views,
- To feed appropriate data for verification and validation,
- To store information about reference architectures, and
- To define and manage architecting processes.

C. Ontology of Architectural Modeling

An architectural model needs to be described with a well-defined ontology [33, 41-43]. While there are many research efforts that tried to establish ontologies for particular applications, we emphasize the importance of function modeling here. As discussed in Section II-E, after all, function is a teleological description about a system, although it is subjective. Together with more objective behavioral, structural and phenomenological information, function modeling can play a central role for the architectural model.

In this research, we follow the FBS (Function-Behavior-State) modeling for this purpose [30]. Within FBS, a function is a subjective description in the form of verb-noun pair ("to do something") and is connected to behaviors through function-behavior knowledge. A behavior is a state transition, while a state is a snapshot of the structure including entities, relationships among them, parameters, and their values. A behavior as a state transition is triggered by physical phenomena. Therefore, structural knowledge can be objectively represented at the level below behaviors. Compared with other types of function modeling, especially, so-called transformational function modeling methods [29], the FBS type of function modeling [30, 44] is more flexible, because functions can be purely subjective and may not be limited to those related to transformation (of energy, material, and information) yet still capable of linked to physics. It is also notable that, within FBS, this separation of functions from objective physical notions is intentional, because we have found out from empirical experiments that there is no direct connection between subjective function and objective structural knowledge [45].

These ideas formed the fundamentals of the tools described in the next section.

IV. ARCHITECTING TOOLS

In this section, we briefly highlight our research results related to architecture-centric model-based product development. It is intended to give only references, so the details need to be found in appropriate literature. These tools include the AM (Architecture Modeling) tool, SA-CAD (Systems Architecting Computer-Aided Design), DID (Design Interference Detector), and the combination of workflow modeling and FBS.

A. AM Language and AM Tool [46-49]

The architectural description must allow representing and integrating information that covers a wide spectrum of information regarding architecture. Our group has proposed the Architecture Modeling (AM) language and a tool called the AM tool to create an architectural model [46]. This tool stores AM instances in a digital format for easy access of other software applications.

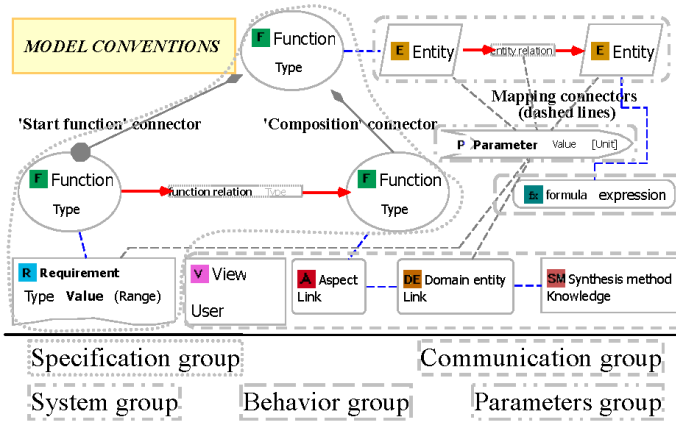

Figure 2. Architectural model in the AM Tool [47]

As shown in Fig. 2, an "aspect" (nodes marked with an "A") makes reference to a common interest of a group of stakeholders, and maps to the set of models (nodes marked with "DE") used to describe and verify properties in the context of such an interest. This aspect contains domain-specific information on the functions, subfunctions, customer requirements, geometry, electrical components, simulation (real-time behavior) and the embedded software. The AM tool connects this set of information to an external analysis tool (for example, implemented in MATLAB). This would help automate performance analysis tasks within the product development [48].

B. SA-CAD [50, 51]

We have developed a prototype CAD system named Systems Architecting CAD (SA-CAD) that offers the following types of support for~~ing~~ system architecting activities [11,8]. ~~The types of support offered by the system are:~~ Fig. 3 depicts the screen hardcopy of the SA-CAD.

- Hierarchical decomposition of function requirements and definition of structural and behavioral descriptions based on the FBS (Function-Behavior-State) modeling [30].
- Generation of architectural options at any level of function hierarchy (i.e., SA-CAD is used to define the top-level decomposition of the system to satisfy the major function requirements as well as the decomposition of a subsystem to satisfy the local function requirements of the subsystem.)
- Visualization of a product model using diverse (functional, behavioral, geometric, and parameter-level) aspect modelers.
- Consistency management of design information defined with the above aspect modelers.

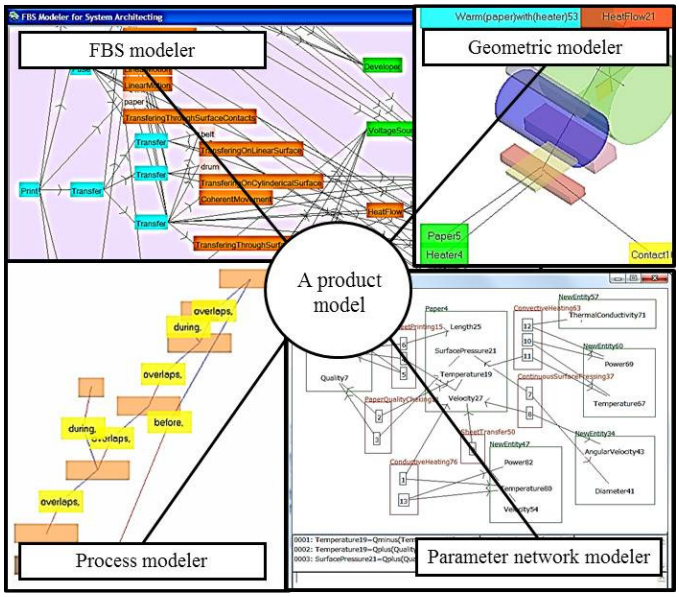

Figure 3. SA-CAD [47]

The systems decomposition process begins with functional requirements. Then, the system architect should find a set of building blocks that can satisfy the given functional requirements. There can be multiple combinations for the same requirements and the best one has to be selected by evaluating the performance. These building blocks are, e.g., machine elements, established components, and mechanisms in mechanical design. In control design, fundamental building blocks are block diagrams that represent sensors, actuators, and controllers. In software design, they can be subroutines and functions.

However, it might be possible that appropriate building blocks are not available, because it is a new technology or design. In this case, the architect needs to perform systems decomposition without building blocks. In this case, as illustrated in Fig. 4, the sSystem architects first identifies a sequence of physical phenomena (called physical process) realizing a desired system behavior (that satisfies a function requirement) and parameters associated to with the phenomena (e.g., parameters in an equation characterizing a physical phenomenon). These parameters constitute a network that can be divided clustered into clusters. While a cluster indicates an embodiment of a component or a subsystem, a set of clusters as a whole defines a module. The sSystem architects compares different clustering possibilities and chooses the best one. These clustering possibilities represent solution options regarding the architecture of the product, so they are called architectural options regarding the organization of clusters. Multiple architectural options are can be derived generated from the network of parameters obtained from a given set of physical phenomena.

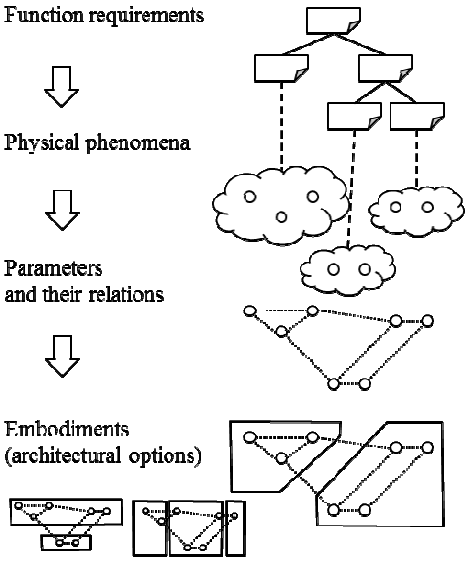

Figure 4. Decomposition Algorithm [47]

C. *Design Interference Detector [22, 52, 58]*

During the system integration and system level verification phases, design failures or design faults made early in the development process can be finally discovered. Often these problems are not component or subsystem level failures but "system level failures", because individual components and subsystems have passed verifications at their level. System level failures are those resulting from the combination of components and subsystems, which we call "design interference" [18]. These problems appear to pop up unexpectedly and can be called "unpredicted problems" [52]. Since these problems often fall in an area between multiple domains, they are hard to solve. Consequently, they can delay the project unacceptably.

An example of the AGV (Automatic Guided Vehicle) system design illustrates such an unpredicted problem [18, 52]. The mechanical engineer who was responsible for the overall system design decided to use induction coils to charge the battery on AGV and determined the locations of charging stations. This charging coil is used also as a traffic signal system. An AGV stops at every charging station and if there is another AGV in the next segment, the AGV is not allowed to proceed to the next segment. The electrical engineer designed the coils both on the vehicle and under the pavement, given such information as the distance between the two coils, maximum charging time, charging frequency, etc. The software engineer programmed the control software that realizes the traffic control logic. However, the electrical engineer was not informed about the distance between charging stations and the mechanical engineer placed two or stations at a too close distance in some places. Because of this, when one AGV stops at a charging station for charging, the coils started to excite the coils of the nearby charging stations. This generated a ghost signal of the next charging station, which prevented this AGV from moving on the next section. This problem of ghost AGVs was fixed after all by modifying the design of the charging coil, but this was a very expensive repair operation of repaving the entire floor of the factory.

This story involved no unusual or strange phenomenon. The physics of every domain was crystal clear at least to experts in that domain. However, this multi-disciplinarity (mechanical, electrical, and software) obstructed flow of critical information. The mechanical engineer didn't tell the distances of the charging stations. In his mind, this was not recognized as important information to be transmitted to the electrical engineer. The electrical engineer was only instructed to design a charging station to satisfy the required performance. Of course, he knew magnetic linkage due to leakage, but he didn't realize he needed to warn the mechanical engineer about such interference. The essence of this problem is, therefore, that cross-disciplinary communication and understanding was hampered by invisible barriers between disciplines, which led to not only miscommunication but also overlook or even neglect of information.

In order to prevent this type of design failures, one idea could be to check a design thoroughly with a computer-based simulation technique, even though only insufficient information can be used. A simulator derives all possible behaviors of the design and helps to check if these behaviors are all expected (or designed). If there is any unexpected behavior, this would potentially cause an "unpredicted problem". For this purpose, qualitative physics (or qualitative reasoning) [53-56] could be useful. Among others, we employ Physical Feature Reasoning System (PFRS) [34] which is a simplified version of Forbus' Qualitative Process Theory

(QPT) [57]. QPT is particularly promising, because it does not assume mono-disciplinary models.

Based on these ideas, DID (Design Interference Detector) [22, 52, 58] was developed on top of KIEF (Knowledge Intensive Engineering Framework) which was also developed by our group [32]. KIEF is capable of simulating qualitatively physical behaviors with QPT as well as PFRS.

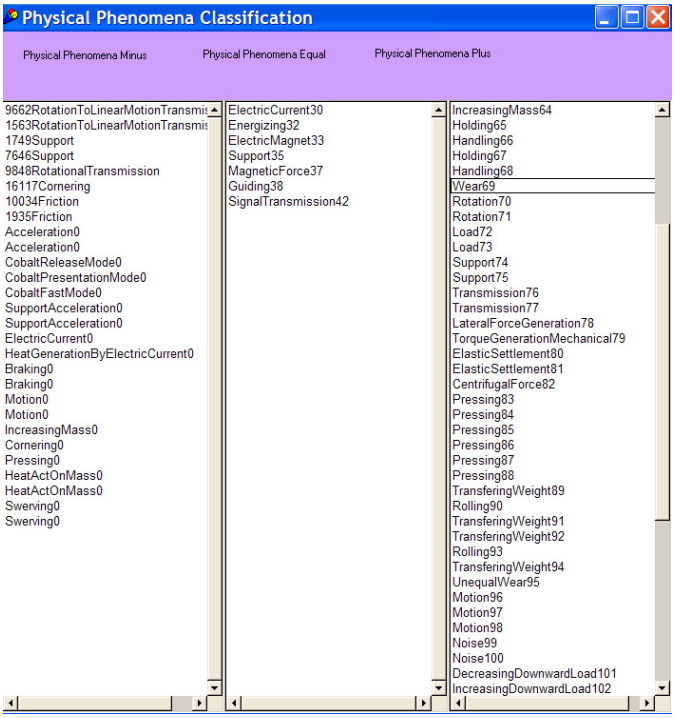

Figure 5. A DID screen hardcopy showing categorized behaviors with the contrast method [50]

Basically DID reasons out all possible phenomena that a design may exhibit and checks if it entails unexpected (or unpredicted), undesirable phenomena. However, since qualitative physics models and reasons about physical systems without quantitative information, it generates too many spurious phenomena and cannot reject ambiguities in reasoning results. While the effort necessary to build a qualitative model is much simpler than that for quantitative (numerical) models, engineers cannot apply qualitative reasoning to practical situations, because of these problems of spurious and ambiguous solutions. Therefore, DID is equipped with the following three functionalities to reduce those spurious and ambiguous solutions. Fig. 5 shows the filtering results obtained for the AGV transport system.

- The prioritization method qualitatively prioritizes the reasoning results (e.g., one scenario is more likely to happen, because one component is heavier than the other).
- When there is a small design change to an existing design solution, the old design model and the new model are qualitatively compared. The interaction method is similar to the contrast method, but analyzes the interactions of two previously isolated models. These two methods classify the reasoned out solutions and categorize them into predicted (or known to happen), negligible, and unpredicted. DID then reduces negligible solutions of qualitative reasoning using heuristics.
- The ambiguity solver method eliminates ambiguous behaviors from the reasoned out results at the parameter level based on intelligent user interventions that finds that most influential parameter.

D. Integration of Workflow Modeling and Function Modeling [59, 60]

During the product development of complex multi-disciplinary systems that involve a wide range of stakeholders, it is critical to capture user requirements as correctly as possible in the form of common stakeholder understanding. On one hand, such common stakeholder understanding should be formally captured and described domain-independently, so that it facilitates understanding of all the stakeholders without any mistakes or ambiguities. On the other hand, too much formalism may hamper the expressiveness of the method. In addition, those stakeholders, among others, the end users are sometimes domain experts (e.g., in case of medical equipment, medical professionals who do not have sufficient time to work with engineers). Therefore, there is a risk that these experts do not use the method at all, if it requires too much time to learn to use it.

We employ a workflow modeling method for these purposes. Fig. 6 depicts an example of workflow modeling applied to a medical equipment to be used in an operation room. The method captures the high level abstraction of user needs. Combined with the FBS modeling [30], it allows connecting user requirements to technical functions. In this sense, this method is similar to QFD (Quality Function Deployment) [61] but is much more visual than QFD and helps to easily identify, for example, necessary procedures and conditions to operate the machine (such as states of the machine and auxiliary equipment, operator's actions, preparation procedures, etc.).

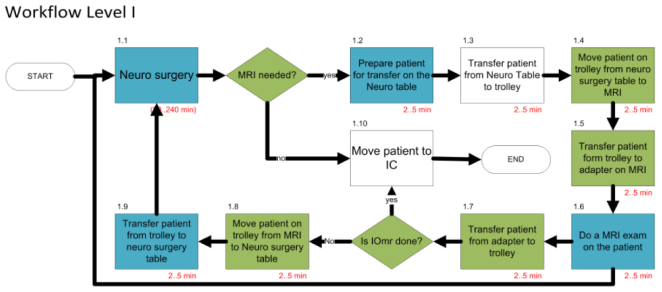

Figure 6. An example of workflow model [59]

V. CONCLUSIONS

This paper reviewed the current practices to develop complex multi-disciplinary systems in industry. Such products and their product development are complex for a variety of reasons; among others, multi-disciplinarity is the central issue. To tackle the complexity during the product development process, we examined several key concepts of product development and their associated problems, including systems engineering, concurrent engineering, and model-based

systems engineering. We also observed that more emphasis should be placed in the conceptual phase (i.e., systems architecting phase) in order to reduce design failures.

To this end, we proposed the architecture-centric model-based product development paradigm. Within this paradigm, the concept of systems architecture acts as an integration framework that facilitates cross-disciplinary communication and mutual understanding. We also discussed that architectural descriptions should be based on an ontology which includes functions, behaviors, states, structure, physical phenomena, and other relevant concept types. These architectural descriptions facilitate communication and mutual understanding of stakeholders from different disciplines.

The paper illustrated some research results from our group including AM tool, SA-CAD, DID, and workflow modeling. It is also important to develop formalized methods, such as the systems decomposition method demonstrated in Section IV-B.

In concluding this paper, some additional remarks can be made. Observing different industry sectors, one might be astonished by the differences in the attitude toward "systems design". For instance, aerospace industry has a completely different approach to the conceptual design stage. They even have a sophisticated collaborative design environment in which multi-disciplinary participants can make decisions during conceptual design stage [60]. Compared with this, other industry sectors pay insufficient attention to conceptual design and systems architecting. In some mechanical industry, the concept of systems design is almost non-existing, although ironically their products are also becoming multi-disciplinary ever.

One another observation is the parallelism in different domains. People in mechanical engineering, systems engineering, control engineering, electrical engineering, and even industrial design speak the same language, use almost identical terminology, taxonomy, and ontology about product development, but they still misunderstand each other. Joint efforts between academia and industry to integrate and unify these concepts are desirable.

ACKNOWLEDGMENT

I would like to thank Dr. Hitoshi Komoto (AIST, Tsukuba, Japan), Dr. Valentina D'Amelio (Hukselux, Delft, The Netherlands), Dr. Andrés Alberto Alvarez Cabrera (Sonion, Amsterdam, The Netherlands), and Thom van Beek (TU Delft, Delft, The Netherlands) for providing me with research materials and help to write this article.

REFERENCES

[1] http://www.3ds.com/products/catia/portfolio/dymola/overview/ (visited on 23/7/2012).

[2] http://www.plm.automation.siemens.com/en_us/products/teamcenter/systems-engineering-requirements-management/ (visited on 23/7/2012).

[3] T.W. Simpson, Z Siddique, J. Jiao, Eds. Product platform and product family design: Methods and applications. Berlin: Springer, 2006.

[4] T.W. Simpson, J.R. Maier, F. Mistree, "Product platform design: Method and application," Res. Eng. Design, Vol. 13 (1), pp. 2-22, 2001.

[5] G. Sohlenius, "Concurrent engineering," CIRP Annals–Manuf. Tech., Vol. 41 (2), pp. 645-655, 1992.

[6] A. Kusiak, Ed., Concurrent Engineering: Automation, Tools, and Techniques, New York: John Wiley & Sons, 1993.

[7] G.Q. Huang, Ed., Design for X: Concurrent Engineering Imperatives, Berlin: Springer, 1996

[8] M.L. Dertouzos, R.K. Lester, R.M. Solow, The MIT Commission, Made in America: Regaining the Productive Edge, Cambridge, MA: MIT Press, 1989.

[9] T. Tomiyama, "A Japanese view on concurrent engineering," AI EDAM, Vol. 9(2), pp. 69-71, 1995.

[10] For example, see http://cscwd.org for IEEE Technical Committee on Computer Supported Cooperative Work in Design (visited on 23/7/2012).

[11] S.D. Eppinger, D.E. Whitney, R.P. Smith, D.A. Gebala, D.A., "A model-based method for organizing tasks in product development," Res. Eng. Design, Vol. 6(1), pp. 1-13, 1994.

[12] V. Krishnan, S.D. Eppinger, D.E. Whitney, "A model-based framework to overlap product development activities," Management Science, Vol. 43(4), pp. 437-451, 1997.

[13] For example, see http://www.dsmweb.org (visited on 23/7/2012).

[14] T.R.Browning, "Applying the design structure matrix to system decomposition and integration problems: A review and new directions," IEEE Trans. Eng. Management, Vol. 48(3), pp. 292–306, 2001.

[15] K. Forsberg, H. Mooz, "The relationship of system engineering to the project cycle," Proc. Joint Conference National Council On Systems Engineering and American Society for Engineering Management, Chattanooga, TN, 21–23 October 1991.

[16] INCOSE, INCOSE Systems Engineering Handbook v3.2.2, 2010.

[17] G. Muller, Systems Architecting: A Business Perspective, CRC Press, 2011.

[18] T. Tomiyama, V. D'Amelio, J. Urbanic, W. ElMaraghy, "Complexity of multi-disciplinary design," CIRP Annals–Manuf. Tech., Vol. 56(1), pp. 185-188, 2007.

[19] A. Yassine, D. Braha, "Complex concurrent engineering and the design structure matrix method," Concurrent Eng., Vol. 11(3), pp. 165–176, Sep. 2003.

[20] D. Bradley, "Mechatronics – More questions than answers," Mechatronics, Vol. 20(8), pp. 827–841, Dec. 2010.

[21] M.K. Chmara, A.A. Álvarez Cabrera, T.J. van Beek, V. D'Amelio, M.S. Erden, T. Tomiyama, "Revisiting the divide and conquer strategy to deal with complexity in product design," Proc. IEEE/ASME Int. Conf. on Mechtronic and Embedded Systems and Applications, 12-15 Oct. 2008, Beijing, China, pp. 393–398.

[22] V. D'Amelio, M.K. Chmarra, T. Tomiyama, "Early design interference detection based on qualitative physics," Res. Eng. Design, Vol. 22(4), pp. 223–243, Apr. 2011.

[23] O.C.Z. Gotel, A.C.W. Finkelstein, "An analysis of the requirements traceability problem," Proc. the 1st Int. Conf. on Requirements Eng. (ICRE 1994), IEEE Computer Society Press, Colorado Springs, Colorado, USA, 18–22 Apr. 1994, pp.94–101.

[24] N. Suh, The Principles of Design, New York: Oxford University Press, 1990.

[25] C. Eckert, P.J. Clarkson, W. Zanker, "Change and customisation in complex engineering domains," Res. Eng. Design, Vol. 15(1), pp. 1-21, 2004.

[26] P.J. Clarkson, C. Simons, C. Eckert, "Predicting change propagation in complex design," Journ. Mech. Design, ASME, Vol. 126(5), pp. 788-797, 2004.

[27] ISO: ISO 26262 Road vehicles–Functional safety, 2011.

[28] G. Pahl, W. Beitz, H.-J. Schulz, U. Jarecki, Engineering Design: A Systematic Approach, 3rd Edn., Berlin: Springer, 2007, Trans. L.T.M. Blessing, K. Wallace.

[29] R.B. Stone, K.L. Wood,"Development of a functional basis for design," Journ. Mech. Design, ASME, Vol. 122(4), pp. 359-370, 2000.

[30] Y. Umeda, M. Ishii, M. Yoshioka, Y. Shimomura, T. Tomiyama, "Supporting conceptual design based on the function-behavior-state modeler," AI EDAM, Vol. 10(4), pp. 275-288, 1996.

[31] INCOSE: INCOSE Systems Engineering Vision 2020, INCOSE-TP-2004-004-02, Sept. 2007.

[32] J.A.Estefan, "Survey of model-based systems engineering (MBSE) methodologies," Incose MBSE Focus Group, vol. 25, 2007.

[33] T. Tomiyama, Y. Umeda, T. Kiriyama, "A framework for knowledge intensive engineering," in T.I. Ören, G.J. Klir, Eds., Computer Aided Systems Theory–CAST '94, Ottawa, Ontario, Canada, Lecture Notes in Computer Science, Vol. 1105, Berlin: Springer-Verlag, pp. 123-147, 1996.

[34] M. Yoshioka, Y. Umeda, H. Takeda, Y. Shimomura, Y. Nomaguchi, T. Tomiyama, "Physical concept ontology for the knowledge intensive engineering framework," Adv. Eng. Informatics, Vol. 18(2), pp. 69-127, 2004.

[35] C.J.J. Paredis, A. Diaz-Calderon, R. Sinha, P.K. Khosla, "Composable models for simulation-based design," Eng. with Comp., Vol. 17(2), pp. 112-128, 2001.

[36] K. Thramboulidis, "Model-integrated mechatronics–Toward a new paradigm in the development of manufacturing systems," IEEE Trans. Ind. Inf., Vol. 1(1), pp. 54–61, Feb. 2005.

[37] P. Hehenberger, F. Poltschak, K. Zeman, W. Amrhein, "Hierarchical design models in the mechatronic product development process of synchronous machines," Mechatronics, Vol. 20(8), pp. 864–875, Dec. 2010.

[38] R. Scheidl, B. Winkler, "Model relations between conceptual and detail design," Mechatronics, Vol. 20(8), pp. 842–849, Dec. 2010.

[39] Y. Cao, Y. Liu, C.J.J. Paredis, "System-level model integration of design and simulation for mechatronic systems based on SysML," Mechatronics, Vol. 21(6), pp. 1063-1075, 2011.

[40] ISO: ISO/IEC/IEEE 42010 Systems and software engineering—Architecture description, 2001.

[41] P. Borst, H. Akkermans, J. Top, "Engineering ontologies," Int. Journ. Human Comp. Stud., Vol. 46(2-3), pp. 365-406, 1997.

[42] Y. Kitamura, R. Mizoguchi, "Ontology-based description of functional design knowledge and its use in a functional way server," Expert Sys. with Applications, Vol. 24(2), pp. 153-166, 2003.

[43] Z. Li, K. Ramani, "Ontology-based design information extraction and retrieval," AI EDAM, Vol. 21(2), pp. 137-154, 2007.

[44] A.K. Goel, S. Rugaber, S. Vattam, "Structure, behavior, and function of complex systems: The structure, behavior, and function modeling language," AI EDAM, Vol. 23(1), pp. 23-35, 2009.

[45] H. Takeda, S. Hamada, T. Tomiyama, H. Yoshikawa, "A cognitive approach to the analysis of design processes," in J.R. Rinderle, Ed., Proc. Design Theory and Methodology–DTM '90–, Chicago, IL., 16-19 Sept. 1990, DE-Vol. 27, New York: ASME, pp. 153-160, 1990.

[46] A.A. Alvarez Cabrera, K. Woestenenk, T. Tomiyama, "An architecture model to support cooperative design for mechatronic products: a control design case," Mechatronics, Vol. 21(3), pp. 534-547, 2011.

[47] A.A. Alvarez Cabrera, T.J. van Beek, H. Komoto, T. Tomiyama, "Architecture-centric design approach for platform development", in T.W. Simpson, R.J. Jiao, Z. Siddique, and K. Hölttä-Otto, Eds., Advances in Product Family and Product Platform Design: Methods & Applications, Berlin: Springer, unpublished.

[48] M.J. Foeken, A.A. Alvarez Cabrera, M. Voskuijl, M.J.L. van Tooren, "Enabling control software generation by using mechatronics modeling primitives," Adv. Eng. Informatics, Vol. 26(2), pp. 196–206, Apr. 2012.

[49] A.A. Alvarez Cabrera, Architecture-Centric Design: Modeling and Applications to Control Architecture Generation, PhD Thesis, Delft University of Technology, Oct. 2011.

[50] H. Komoto, T. Tomiyama, "A system architecting tool for mechatronic systems design," CIRP Annals–Manuf. Tech., Vol. 59(1), pp. 171–174, 2010.

[51] H. Komoto, T. Tomiyama, "A framework for computer-aided conceptual design and its application to system architecting of mechatronics products," Comp. Aided Design, Vol. 44(10), pp. 931–946, Oct. 2012.

[52] V. D'Amelio, Design Interference Detection for Multi-Disciplinary Product Development, PhD Thesis, Delft University of Technology, Dec. 2010.

[53] K.D. Forbus, J. de Kleer, Building Problem Solvers. Cambridge, MA: MIT Press, 1993.

[54] C.J. Price, L. Travé-Massuyès, R. Milne, L. Ironi, K. Forbus, B. Bredeweg, M.H. Lee, P. Struss, N. Snooke, P. Lucas, M. Cavazza, G.M. Coghill, "Qualitative futures," Knowl. Eng. Rev., Vol. 21(4), pp. 317–334, 2006.

[55] M.P.J. Fromherz, D.G. Bobrow, J. de Kleer, "Model-based computing for design and control of reconfigurable systems," AI Magazine, Vol. 24(4), pp. 120-130, 2003.

[56] M. Klenk, J. de Kleer, D.G. Bobrow, J. Hanley, W.C. Janssen, "Placing qualitative reasoning in the design process," in Proc. 26th Int. Workshop on Qualitative Reasoning, Playa Vista, CA, 16-18 July 2012.

[57] K.D. Forbus, "Qualitative process theory", Artif. Intelligence, Vol. 24(1-3), pp. 85-168, 1984.

[58] V. D'Amelio, M.K. Chmarra, T. Tomiyama, "A method to reduce ambiguities of qualitative reasoning for conceptual design applications," AI EDAM, 2012, in press.

[59] T.J. van Beek, T. Tomiyama, "Workflow modelling of intended system use", in P. van de Laar, T. Punter, Eds., Views on Evolvability of Embedded Systems, Berlin: Springer, pp. 153-170, 2011.

[60] T. J. van Beek, T. Tomiyama, "Structured workflow approach to support evolvability," Adv. Eng. Informatics, Vol. 26(3), pp. 487–501, Aug. 2012.

[61] S. Mizuno, Y. Akao, QFD: The Customer-Driven Approach to Quality Planning & Deployment, Tokyo: Asian Productivity Organization, 1993.

[62] J. Osburg, D. Mavris, "A collaborative design environment to support multidisciplinary conceptual systems design," SAE Technical Paper, vol. 114, pp. 1508–1516, 2005.