

# Managing the Development of Complex Product Systems—What Managers Can Learn From the Research

—ALI YASSINE 

Department of Industrial Engineering and Management,  
American University of Beirut, Beirut  
11-0236, Lebanon

IEEE DOI 10.1109/EMR.2018.2870401

**Abstract**—Product development (PD) in the 21st century is becoming more complex due to fast paced technological advances and demanding customers. In this paper, I presents a preview of the various models and tools that can be used to manage the development of such complex product systems. These models and tools are organized around three PD domains: people, product, process, and the information that flows between them. The intention is not to identify all publications in any single domain, but instead to expose PD and engineering managers to a set of useful tools to manage the development complexity in each domain and more importantly at the intersection of the three domains.

**Key words:** Product decomposition, product architecture, product development teams, project planning and scheduling, multi-domain analysis

## 1. INTRODUCTION

THE design and development of modern engineering products is a complex endeavor, which can involve hundreds of development participants with various backgrounds, over many years during which requirements may change, evolve, or emerge. The complexity of such an endeavor does not merely stem from the technical challenge of the design but also from managerial aspects. Orchestrating the various flows of information, requirements and dependencies crossing disciplinary and organizational boundaries is a daunting managerial task, which requires sophisticated design management principles.

The generic product development (PD) process is composed of four phases as shown in Figure 1 (Ulrich and Eppinger, 2008). It starts after upfront planning and ends with full production. Before the actual PD team is formed, the project is launched, and the company starts

spending development money, the company must decide which PD projects to pursue from the many available opportunities; which is the essence of the planning phase. The planning phase also includes assessment of technology and market objectives. The main output is a product plan and project mission statements.

In the conceptual development phase, the needs of the target market are identified, and alternative product concepts are generated and evaluated. Following conceptual design, the selected system concept (which is represented in a function diagram) is partitioned into subsystems according to a specified architecture. A major outcome of system design includes the definition of the system (or product) architecture, identification of subsystems, and a communication plan among the various subsystem teams. In the detailed design phase, the subsystems are assigned to various development teams within

(or outside) the organization. These teams are tasked with completing the detailed design for each subsystem, which includes the complete specification of geometry, material and tolerances for all subsystems and components. Testing and integration begin once the detailed subsystem design phase is completed. This phase includes the construction and evaluation of many prototypes and pre-production versions of the product. Design changes and refinements are made based on test results. Integration testing of subsystems into systems and problems of fit and function as a system are detected and corrected in this phase.

From the above description, PD is a process (to be properly managed), which involves various developers (i.e., people) collaborating to achieve a common goal (i.e., a successful product). Therefore, in this paper, I consider PD as the interaction of various analysis domains: process, people, and product. Investigating these domains separately, although useful, may have a limited impact on the PD project. This paper does not attempt to identify all publications in a specific domain but instead is focused on providing PD and engineering managers with a set of useful tools to manage the development complexity in each domain and more importantly at the intersection of the three domains.

The rest of the paper proceeds as follows. In Section 2, I start by exposing the problems, issues, and research questions that may arise from an 'isolated' approach to the management of complex PD using a single domain. Then, in the next three sections (Sections 3, 4 and 5),

I proceed to describe and discuss the existing literature (i.e., models and tools) that can be usefully applied in each domain. This discussion is followed by an overview of the literature and problems that lie at the intersection of these three domains (Sections 6). In Section 7, I present my concluding remarks.

## 2. MANAGING COMPLEX PD PROJECTS: ISSUES AND RESEARCH QUESTIONS

Traditionally, the technical complexity of engineered systems has been addressed through decomposition (Alexander, 1964). The overall system is decomposed into subsystems and modules (according to a specified architecture) that can be dealt with individually. Consequently, many interesting research and practical questions come to mind, such as: What is the proper decomposition? What is the optimal product architecture? And do we choose a modular architecture with few well-defined interfaces or an integral architecture with more complex interfaces? The complexity of the product decomposition stems primarily from the fact that some of these interactions are unknown in nature and magnitude, and their implications on product and process performance can also be unknown. These questions, among others, have been addressed in the literature and in this paper, I review many of these models highlighting opportunities for managers to use these models.

Although decomposition may reduce the technical complexity of development; however, it opens a whole new set of challenges in terms of managing the dependencies and

interfaces between these modules (Rechlin, 1991). These modules are assigned concurrently to various development teams and these teams can have their own communication (or social) network (which may or may not be reflected in a formal organizational chart) (Krackhardt and Hanson, 1993). It would be beneficial to discover from the social network which nodes are the most prominent in terms of supplying or receiving information, and which are not. But more importantly, we would like to know if this social network supports the communication patterns and coordination mechanisms required for a specific decomposition or product architecture. So, do we design the organization and then the products or vice versa (MacCormack et al., 2012). Do products look like organizations that make them (Hoetker, 2006)? Do they co-evolve? Do and how changes in architectures conflict with rigidities in organizational structures, communication patterns, and coordination mechanisms (Henderson and Clark, 1990).

To complicate things further, there is the process (or project) domain. A PD project schedule is not the usual schedule realized in project management, where traditional (i.e., classical) techniques such as CPM and PERT can be used directly to manage the PD project (Kerzner, 2013). In PD projects, there can be repetitions of activities, as you learn more downstream (or as some downstream tests fail), which might force us to revisit earlier design decisions. In PD projects, activities can also overlap based on preliminary information sharing, providing downstream activities a head start. In such an environment,

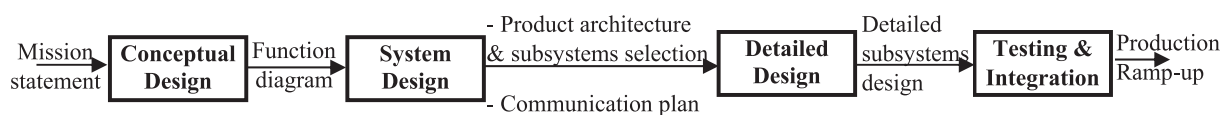


Figure 1. Phases of the PD process.

project management and scheduling must investigate the impact of rework and overlapping on project duration, cost, and product quality/performance. Furthermore, a process requiring frequent bi-directional communication among various development teams must be supported by the existing social network of the development organization. Moreover, this social network itself is evolving in response to different project plans that may have lasting implications (beyond the finish of the development project) on the social characteristics of the development organization. Does the social network support the required preliminary/partial information flows (because some people fear to share partial information not to be blamed later if it changes)? Finally, does the partial information flows or rework cross module boundaries as specified by the chosen architecture?

To illustrate, consider the design and development of a new automotive

control panel. Figures 2a and 3a present two different designs for the control panel: an existing modular design (Figure 2a) and a new integrated design (Figure 3a). There are two separate teams at this automotive company that are involved in the development of this control panel: the climate control team and the electronics team. Traditionally, these two teams worked separately and have had limited interactions with each other. To create the new integrated panel, the project required a lot more interaction between the two teams; more than what traditionally existed. These interactions are also different than what they normally used to be. In this example, the chosen architecture of the panel (Figures 2b and 3b) dictates a specific information exchange requirement between the two teams. This new requirement would be difficult to implement successfully unless it is supported by the social network of the organization (Figures 2c and 3c). Finally, Figures 2d and 3d are

concerned with the project schedule, which also involves a communication plan for how frequently the design teams should exchange information (and about what) under the two different architectural scenarios. An underestimation of the augmented and increased amounts of coordination required for the new integrated design may jeopardize the success of the new product.

The above development scenario, highlights the problems with isolated management procedures that do not consider the strong interdependencies *within* and *between* the three domains (as exemplified in Figures 1, 2, and 3). These problems manifest themselves in several ways (Yassine and Braha, 2003):

- A. **Integration problems:** Integration is the process of aggregating the decomposed subsystems (or product modules) back together to deliver the final product (or system)

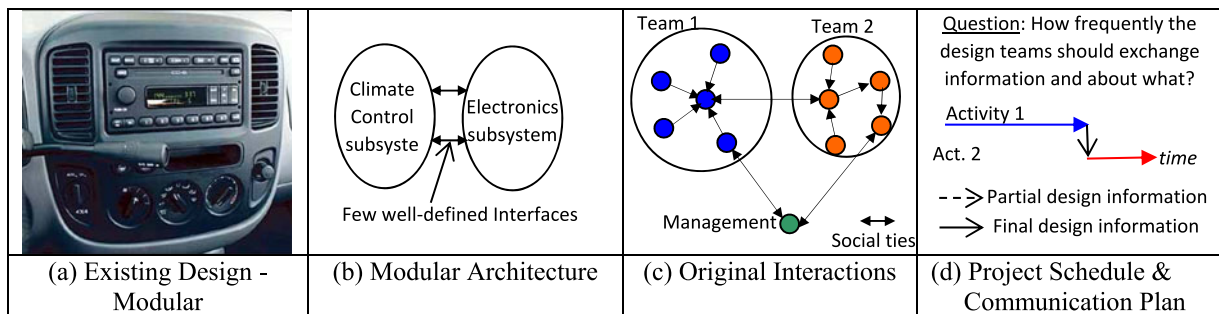


Figure 2. Existing panel architecture and corresponding social network and communication plan.

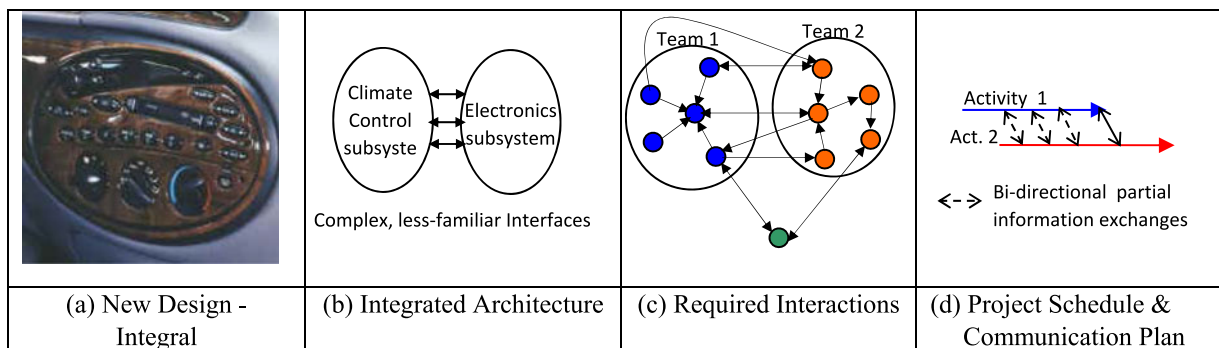


Figure 3. New panel architecture and corresponding required interactions and communication.

design. Carefully decomposed products have 'clean' (i.e., well-defined) module boundaries and interfaces which will facilitate system testing and integration. Alternatively, poor decompositions translate into overlapped responsibilities and deteriorated product performance (Helmer et al., 2010).

- B. **Excessive iteration:** The interdependence between subsystems makes engineering design fundamentally iterative, driven by the repetition of tasks (i.e., rework) due to the availability of new information (generated by other teams) such as changes in input, updates of shared assumptions or the discovery of errors. As this uncertainty is resolved, portions of the design work are repeated to either verify an initial estimate/guess or to meet design specifications (Meier et al., 2007).
- C. **Churn behavior:** Iteration not only slows down design convergence, but also may have a destabilizing effect on the system's behavior. It is not hard to see that this interdependence could result in a "chain-reaction" of updates in the form of new information "flooding" the system. In this pathological situation, most of the developmental effort of the team will be spent updating plans-of-action, only to have them changed later, and no real progress is made (i.e., design churn) (Yassine et al., 2003a).
- D. **Coordination problems:** The decomposed subsystems (or product modules) are assigned to different development participants (or teams), which can be distributed in space and time (Allen, 2007). The flow of information (especially partial information), required for

successful coordination between the participants, presents a serious managerial challenge that needs to be planned throughout the development timeline (Tripathy and Eppinger, 2013).

- E. **Locus of innovation and decision-making:** Another consequence of decomposition is the distributed locus of innovation and decision-making in the decomposed (but interlinked) development network. As such, no single designer/team has full knowledge of the overall system and identifying potential sources of architectural/system innovation becomes difficult unless careful analysis techniques are employed (Batallas and Yassine, 2006).
- F. **Misalignment inefficiencies:** The three decompositions simultaneously coexist: product decomposition into modules, process decomposition into design tasks, and organizational decompositions into design teams. If these decompositions are not well coordinated across the various domains, inefficiencies may arise resulting in increased development time/cost and reduced product quality/performance (Sosa et al., 2004).

### 3. DEFINING THE PRODUCT ARCHITECTURE

**Research Questions:** What is the proper decomposition? What is the optimal product architecture? Do we choose a modular architecture with few well-defined interfaces or an integral architecture with more complex interfaces?

**Objective:** To establish a product architecture composed of modules and interfaces between them.

**Issues:** Decomposition granularity, documenting interfaces, clustering metrics, and clustering methods or procedures.

**Tools for Managers:** Design (or Dependency) Structure Matrix (DSM), Function Structure Diagrams (FSD), Modular Function Deployment (MFD).

Several tools exist for decomposing a product system into its constituent elements to identify subsystems and modules. This section will briefly introduce three such tools: Design Structure Matrix (DSM), Function Structure Diagrams (FSD), and Modular Function Deployment (MFD). In general, these tools are limited to guidelines and heuristics, lack an explicit metric to measure modularity, rely on undirected search for modules, and their outcome greatly depend on users' background and expertise (e.g., (Stone et al., 2000)). Consequently, these methods are likely to result in non-optimal and/or non-unique product decompositions (Höltkä-Otto et al., 2012).

Pimpler and Eppinger (1994) proposed a systematic procedure for product decomposition: (1) decompose the system into elements, (2) document the interactions between elements, and (3) cluster the elements into chunks. Generally, this procedure is used in conjunction with a DSM of the product decomposition. Although the proposed procedure is useful and attracted numerous applications, it suffered from subjective implementation of the various steps. For instance, in the decomposition step, the issue of granularity was not addressed clearly: what is the proper level of decomposition (Chiriac et al., 2011)? In the document interactions step, there are countless ways of specifying these interactions depending on the application (Helmer et al., 2010). Finally, in the

clustering step, no obvious repeatable technique was proposed but simply a visual inspection of the interactions and then subjective identification of the clusters (Hölttä-Otto et al., 2012). However, more recently, clustering approaches based on genetic algorithms (GA) are the predominant approach (Yu et al. 2007).

Functional decomposition using Function Structure Diagrams (FSD) is another proposed approach. Stone et al. (2000) defined a FSD as a graphical representation of the functions a product performs on its inputs and outputs. In a FSD, the overall functionality of the product system is broken down into elemental sub-functions. Each sub-function cannot be broken down further and is solution neutral. The sub-functions are connected by “flows” on which they operate. Similar to Pimmler and Eppinger’s (1994) dependency scheme, flows are materials, energy or information that is used by or affects the product (Ulrich and Eppinger, 2008). By following each of the flows we can determine what operations are required to link, connect or transform inputs into outputs. Three heuristics were proposed to group sub-functions into modules: dominant Flow, branching flow, and conversion of flow (Stone et al., 2000). Similar to Pimmler and Eppinger’s (1994) decomposition procedure, this approach also suffers from subjectivity in assigning interaction values and in delineating module boundaries.

Modular Function Deployment (MFD) is a module identification method that builds on elements of both the FSD and DSM methods (Erixon, 1998). Similar to FSDS and DSM, the MFD method is also based on functional decomposition, but unlike FSD and DSM, however, MFD uses modularity drivers to identify modules rather than through function based heuristics. The basic MFD method maps into a

matrix product functions against the modularity drivers. There are twelve modularity drivers that include considerations of how modules would impact: function carryover to different products, technology evolution, planned product changes, different specifications across product variants, styling and aesthetics, common units, process and/or organization, separate testing, supplier availability, service and maintenance, upgrading, and recycling. The functions dominated by the same modularity drivers are good candidates to make up a single module.

#### 4. ANALYZING THE ORGANIZATION AND ITS COMMUNICATION PATTERNS

**Research Questions:** What is the proper teaming arrangement for the PD participants? Which PD players are the most prominent in terms of supplying or receiving information, and which are not?

**Objective:** To form a development team charged with carrying out the development project, and the assignment of team members to development activities.

**Issues:** Team composition (task execution cost), interfaces, communication/coordination cost, team leader.

**Tools for Managers:** Design (or Dependency) Structure Matrix (DSM), Social Network Analysis (SNA), Agent-based simulation models using Computational Organizational Theory (COT).

There is a large body of literature on teams spanning areas from product

and software development, management and organization sciences, to social psychology. These studies normally involve team size and composition, methods for the selection of team members, and characteristics (personality) of successful team members and leaders (Nasrallah et al., 2015). In this section, I mainly focus on the quantitative PD literature that includes models and methodologies to form and analyze development teams using optimization models, simulation and computational organizational models, network-based models (i.e., DSM-based models).

Organizational analysis using team (or people) DSM has been popular in the literature for forming development teams (McCord and Eppinger (1993). Team-based DSM models sought to quantify important relationships in organizations and to find team clusters that optimize communication and work flow among various organizational entities. A Team-based DSM is constructed by identifying the required communication flows and representing them as connections between organizational entities in the matrix. It is important to specify what is meant by communication flow among organizational entities: frequency (daily vs. weekly), medium used (face-to-face, email), timing (early, late). The matrix can be manipulated to obtain clusters of highly interacting teams and individuals while attempting to minimize inter-cluster interactions. The obtained groupings represent a useful framework for organizational design by focusing on the predicted communication needs of different players. For an application example, refer to McCord and Eppinger (1993). They proposed a team-based DSM to analyze the organizational structure necessary for an improved automobile engine development process.

Although people often rely on organization charts to describe how functions and responsibilities are distributed in a given enterprise, organization charts are notoriously unhelpful when it comes to explaining other critical organizational attributes, such as, how work gets done or where the centers of influence really are in a specific PD project (Krackhardt and Hanson 1993). Under time/cost pressure, information must flow through channels outside the organization chart. Therefore, the study of the social interactions within a development organization can be much more revealing than organization charts. Many studies have investigated the implication of social ties or relationships within the development organization on the choice and performance of alternative teaming arrangements. This kind of analysis is usually performed using established social network analysis (SNA) techniques.

To capture the social network of the development organization, we first identify the various functional groups within the organization and then we survey members of each group to capture the type and frequency of the interactions within the group and across groups. Once this social network has been identified, it can be used to collect various node and network metrics that can be very informative in choosing the right development team members and their preferred roles. For example, degree centrality can be calculated for each node, which signifies the amount of information received or delivered by a specific node in the network. Another way of measuring centrality is by focusing on nodes that lie in the path between other nodes (i.e., betweenness centrality). These nodes have control over knowledge flow since information must travel through them. Central nodes become powerful gatekeepers that regulate the amount of information transmitted in a network. Finally, another way to

understand actor position in a social network is to identify brokerage actors. Specifically, a broker is a player that lies between two others, who do not have direct communication, and acts as a channel by which they can relate. Battalas and Yassine (2006) presented one of the early applications of SNA to analyze the team interactions for a jet engine development project and found that these measures were very informative in identifying design teams that receive and share large amounts of information through established communication channels in the organization allowing such teams to become natural systems integrators.

Finally, simulation approaches developed detailed models of how project teams perform work within the organization and the supporting intra-organizational communication flows (Rouse and Boff, 2005; Harrison et al., 2007). Additionally, agent-based approaches have used successfully to simulate complex team practices and interactions in development organizations (Carley, 2002; Miller, 2015). Millhiser et al. (2011) proposed a simple that relates team output to both individual effort and peer effects. Peer effects are the contributions everyone (in the team) makes to the output of fellow team members. Without peer effects, a team's output is simply a sum of individual outputs.

## 5. MANAGING THE DEVELOPMENT PROJECT (DESIGN PROCESS)

**Research Questions:** What is the impact of work policy decisions (activity sequencing, partial information sharing, overlapping) on process and project performance (i.e., duration / cost)?

**Objective:** To establish a project schedule for the development endeavor, specifying the execution sequence of development activities and resource allocation.

**Issues:** Information exchanges: complete versus partial information, iteration, overlapping, crashing, work policy.

**Tools for Managers:** Design (or Dependency) Structure Matrix (DSM), Mathematical optimization models, Simulation models.

PD projects are iterative (i.e., cyclical) due to various reasons. First, bounded rationality prevents designers from considering all aspects of the design from the outset (Simon, 1969). As the development process progresses, new information and constraints start to unravel (e.g., design requirements evolve/change), which forces designers to revisit earlier design decisions. Finally, the experimentation and exploration nature of PD projects make these types of projects iterative. Iterations force changes that propagate to various design stages and require rework (Yasine and Braha, 2003). In this section, we discuss how to streamline the PD process, in order to minimize iteration and rework, using a process (or project) DSM. We also discuss ways of reducing development time by overlapping nominally sequential development activities. Overlapping requires the sharing and exchange of incomplete (or partial) information. Models of overlapping using partial information are either mathematical optimization models or simulation models.

A project DSM is a representation of the adjacency matrix of the project network. The potential for iteration, and thus increased process time and cost, can be minimized by resequencing or partitioning the

project DSM (Meier et al., 2007). Partitioning a project DSM is an attempt to transform the DSM into a lower triangular form by reordering the DSM rows and columns such that the new arrangement does not contain any super-diagonal marks (also called feedback marks). In case the DSM cannot be rendered lower-triangular, then the number of feedback marks remaining represent a measure of goodness for the partitioned DSM. More advanced measures can include the length of the feedback loop among other details described in Meier et al. (2007).

Many mathematical models of overlapping utilized the concept of incomplete or partial information exchange (Krishnan and Ulrich, 2001). Earliest models proposed an overlapping framework for two sequential activities based on a downstream rework formulation that depends on upstream information evolution and downstream sensitivity (Krishnan et al., 1997). Based on these two constructs (upstream evolution and downstream sensitivity), optimal time to start overlapping can be determined. The concepts of evolution and sensitivity became at the core of the discussion of overlapping upstream and downstream activities within the PD literature. Sensitivity refers to how sensitive the downstream activity (i.e., how much rework is necessary) due to changes in upstream information (e.g., a design change). However, evolution is related to the rate of design information generation from the start of the upstream activity until its completion.

Simulation-based models have been used to analyze PD processes; mainly to evaluate the impact of iteration and overlapping practices on development cycle time and cost. Early discrete event simulation models treated PD processes as a stochastic processing network where

development resources are workstations and development tasks are jobs flowing between these workstations (and processed by them) (Adler et al., 1995). Few years later, the DSM-based simulation model was introduced (Browning and Eppinger, 2002). The basic DSM simulation model characterizes the development process as being composed of activities that depend on each other for information. Changes in information cause activity rework. Rework in one activity can cause a chain reaction effecting the work of finished activities. Rework is a function of rework risk, which combines the probability of a change in inputs and the impact this change might have on the dependent activities (Browning and Eppinger, 2002). The impact measure used in DSM simulation represents the fraction of the original work (e.g., activity duration) that needs to be repeated. Both probabilities and impacts are numbers between 0 and 1 and they replace the marks in the binary DSM. Thus, the simulation model is represented by two identical DSMs, one containing probabilities of information change and the other containing the impacts of change. In the probability DSM, the super-diagonal marks are replaced by first-order rework probabilities, and the lower-diagonal marks are replaced by second-order rework probabilities. The simulation could also account for stochastic activity duration by using three-point estimates (optimistic, most likely, and pessimistic) to form a triangular distribution.

## 6. ENTERPRISE MANAGEMENT MODELS—INTEGRATING THE THREE DOMAINS

**Research Questions:** How to choose the product architecture in relationship to process characteristics? Does the social network support the

communication patterns and coordination mechanisms required for a specific decomposition or product architecture? Does the social network (and product architecture) support the required information exchanges (final and partial)?

**Objective:** To uncover multi-domain modeling and analysis methods and tools to fully understand and capture the complexity of the PD environment.

**Issues:** Interdependence between the modeling and analysis of the three domains is not properly and formally accounted for.

**Tools for Managers:** Domain Mapping Matrix (DMM), Multi-domain Matrix (MDM), Multi-mode SNA, Multi-domain optimization.

Streamlining information flows or optimizing the product and process architecture independently can be useful, but their impact is limited as these models are confined to a single domain (e.g., people, product, or process). Multi-domain modeling and analysis (also referred to as enterprise management models) are necessary to fully understand and capture the complexity of the PD environment (Yassine et al., 2003b; Danilovic and Browning, 2007). Research efforts in this area are divided into two main streams. First, there is a growing amount of research relating one domain to another by simply investigating and analyzing the properties of one domain on the behavior and properties of another domain. The other stream of research, which is less common, is focused on optimizing several types of domains concurrently. The main objective of this stream is to find a global optimal solution for the overall PD problem.

Along the first stream relating the organizational and product domains together, Conway (1968) was the first to note that organizations will produce product and system designs that are a copy of the organization's communication patterns. He argued that the existing organizational structure dictates (or at least facilitates or impedes) the kinds of product architectures possible. Later, Henderson and Clark (1990) reinvestigated this link, which formed the basis for what became later known as the 'mirroring hypothesis'. The mirroring hypothesis states that the organizational structure corresponds to the technical patterns of dependency (i.e., product architecture) in a system or product under development (Colfer and Baldwin, 2016). The direction of the mirroring is unclear, and evidence is scattered among studies (Hoetker, 2006). While some researchers suggest that organizational structure influence product architecture (Henderson and Clark, 1990), others suggest that organization structure should follow product architecture (Sorkun and Furlan, 2017).

To investigate the mirroring hypothesis, some researchers examined the alignment between organizational structure (as represented by communication between product development team members) and the technical interdependencies between components in a system (as represented in a product architecture). For example, Sosa et al. (2004 and 2007) compared the team interactions for a jet engine development process with the product architecture of the jet engine and discovered that there is a 67% chance if two subsystems physically interface (or have shared design variables) in the product architecture, their corresponding teams in the social network exchange information. This comparison can be readily performed using a simple matching

algorithm between team interaction (i.e., social network) and product architecture matrix. This is performed automatically using UCINET software (Borgatti et al. 2002).

Analysis of such discrepancies between two or more domains can be helpful for managers to turn attention to areas that need more coordination between developers as suggested by the product architecture, or alternatively to areas where coordination is unnecessary. Particularly, these observations suggest that there is an informal network of communication that could possibly be impacting the PD process, either negatively or positively. Also, there may be communication channels missing that should be present in the network. There are components that interface or have dependencies, but their design teams are not exchanging information. This could suggest a re-evaluation of the current assignment of components to the different design teams and/or perhaps a better product architectural configuration.

Once a product architecture has been selected and subsystems have been identified, teams in charge of detailed subsystem designs are formed. These teams are related to each other through shared design variables as represented by the resultant interfaces from the product architecture choice. Although these teams will, in general, work concurrently with each other, they need to occasionally share information (about the shared design variables) to avoid (a) expensive rework during detailed design and (b) severe integration problems during "testing and Integration". It is worth noting that sharing too much information may not necessarily be beneficial to other teams as it may hinder the rate of progress due to frequent evaluations of the received information. On the other hand,

withholding information from other teams until the completion of a detailed subsystem design may result in a complete redesign for some subsystems. So, there must be some tradeoff formulated to strike the required balance and achieve optimal process performance (e.g., minimum development time and cost) for all subsystem teams (Yassine et al., 2008 and 2013b).

Resource-constrained scheduling problems (RCSP) is an area that maps the organizational domain into the project domain. RCPS involves task assignment to various members of a product development team (i.e., single-model and multi-mode resource-constrained project scheduling problems) (Hartmann and Briskorn, 2010). Considering the complexity of solving these problems optimally, heuristic and meta-heuristic techniques have been successfully used (Browning and Yassine, 2010; Browning and Yassine, 2016). However, these resource-constrained formulations do not explicitly consider the available set of skills and competencies (i.e., human resources) to be assigned to various project activities, which in turn require specific skill sets (Wilde, 2010). Accordingly, Acuna and Juristo (2004) consider the positions as roles and push the mapping of positions to skills further by considering psychological traits suitable for playing the role and compute a correspondence between software development roles and a likelihood of needed personality features.

Along the second stream, general matrix mapping approaches has been formalized by Yassine et al. (2003b) by introducing the notion of a relationship map, which relates two domains to each other, and the concept of connectivity map, which essentially combines two relationship maps into a single matrix. Similarly,

Danilovic and Browning (2007) introduced the domain mapping matrix (DMM) as an approach to map two different analysis domains. So, a DMM is a rectangular ( $m \cdot n$ ) matrix relating two DSMs of sizes  $m$  and  $n$ , respectively, which is not very different from the matrix mapping approach suggested by Yassine et al. (2003b). Around the same time, Lindemann and Maurer (2007) recommended to arrange the domains in square matrix, named the multi-domain matrix (MDM), which arranges the domains and illustrates the relations between them (i.e., the meaning of domain linking written in the matrix cells).

Another analysis approach suggested by Yassine and Bradley (2013) is the use of multi-mode social networks. Their model analyzed virtual PD organizations using a multitude of interacting networks (e.g., product architecture network, project activity network, team/social network). The interaction between these networks allowed them to

convert design and product information into knowledge, which ultimately manifests itself in a product architecture network.

Finally, when considering a process DSM arrangement, for example, one cannot isolate it from the rest of its environment (i.e., people and product DSMs) (Eppinger and Salminen, 2001). The impact (on development time, cost, and quality) of two different feedback marks can be different depending on how these marks map onto the people performing these tasks or the components they affect in the product DSM. This fact must be taken into consideration to penalize feedback marks differently depending on the situation in the other two domains. Based on these observations, Yassine et al. (2013a) devised a set of relational rules that relate the three domains together and proposed to formulate a global optimization objective function for the three domains simultaneously.

## 7. CONCLUSION

Most research in managing complex PD projects dealt with understanding and modeling each domain individually. Few models existed that attempt to tie all three domains together. Figure 4 provides a summary of the issues and research questions addressed in this paper for each domain and at the intersection of the three domains. Isolated management procedures that do not consider these strong interdependencies *within* and *between* the three domains result in serious problems that appear in several ways: integration problems, excessive iteration, and misalignment inefficiencies. Mitigating the risks of rework, integration, and misalignment requires proper single- and multi-domain modeling and analysis tools. This paper presented an overview of these single and multi- domains approaches in a way that allows engineering and PD managers to utilize these models and tools. This will help expedite the diffusion and transfer of PD research into practice.

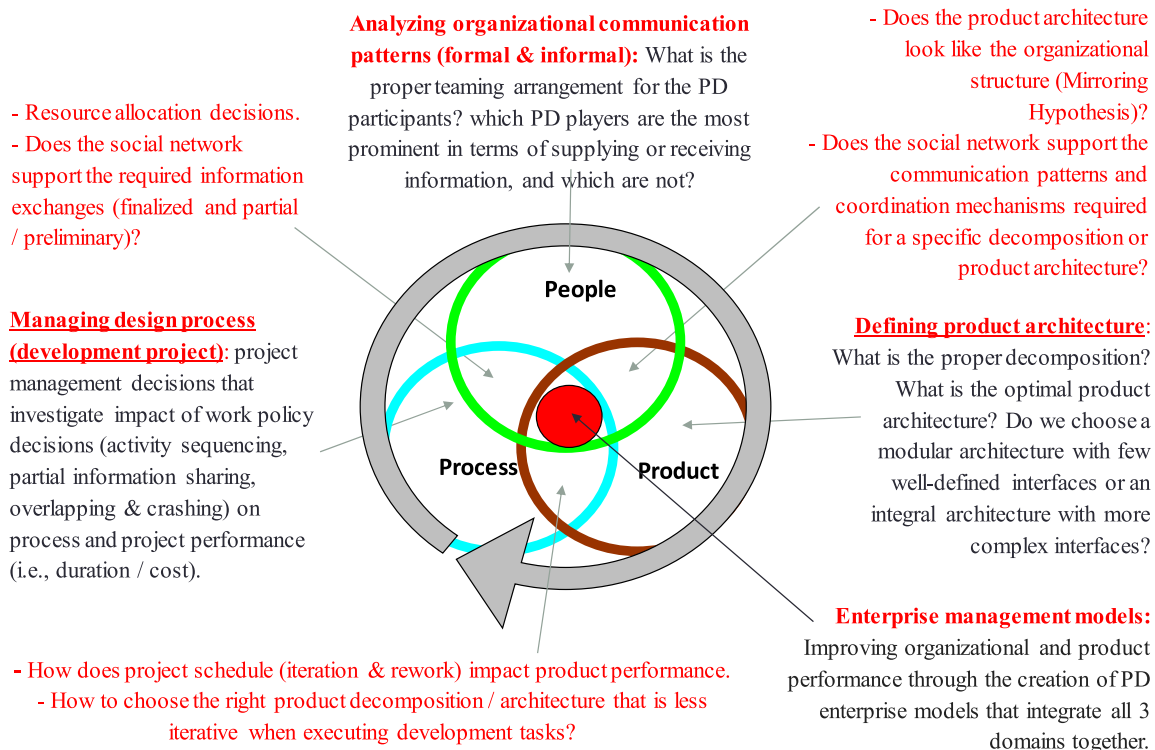


Figure 4. Summary of issues in each domain and at the intersection of domains.

## REFERENCES

- Acuna, S. T., & Juristo, N. (2004). Assigning people to roles in software projects. *Software Practice and Experience*, 34, 675–696.
- Adler, P. S., Mandelbaum, A., Nguyen, V., & Schwerer, E. (1995). From project to process management: An empirically-based framework for analyzing product development time. *Management Science*, 41(3), 458–484.
- Alexander, C. (1964). *Notes on the Synthesis of Form*. Cambridge, MA, USA: Harvard University Press.
- Batallas, D. A., & Yassine, A. A. (Nov. 2006). Information leaders in product development organizational networks: Social network analysis of the design structure matrix. *IEEE Transactions on Engineering Management*, 53(4), 570–582.
- Borgatti, S. P., Everett M. G., & Freeman L. C. (2002). *Ucinet for windows: Software for social analysis*. Analytic Technologies, Harvard, MA, USA.
- Browning, T. R., & Eppinger, S. D. (Nov. 2002). Modeling impacts of process architecture on cost and schedule risk in product development. *IEEE Transactions on Engineering Management*, 49(4), 428–442.
- Browning, T., & Yassine, A. (2010). A comprehensive study of the resource constrained multi-project scheduling problem. *International Journal of Production Economics*, 126(2), 212–228.
- Browning, T. R., & Yassine, A. A. (2016). Managing a portfolio of product development projects under resource constraints. *Decision Sciences*, 47(2), 333–372.
- Carley, K. (2002). *Smart Agents and Organizations of the Future, in the Handbook of New Media: Social Shaping and Consequences of ICTs*, In L. Lievrouw, & S. Livingstone (Eds.). Thousand Oaks, CA, USA: Sage Publications, ch. 12, pp. 206–220.
- Chiriac, N., Hölttä-Otto, K., Lysy, D., & Suh, E. S. (2011). Level of modularity and different levels of system granularity. *Journal of Mechanical Design*, 133(10), 101007-1–101007-10.
- Colfer, L. J., & Baldwin, C. Y. (2016). The mirroring hypothesis: Theory, evidence, and exceptions. *Industrial and Corporate Change*, 25(5), 709–738.
- Conway, M. E. (1968). How do committees invent. *Datamation*, 14(4), 28–31.
- Danilovic, M., & Browning, T. (2007). Managing complex product development projects with design structure matrices and domain mapping matrices. *International Journal of Project Management*, 25, 300–314.
- Eppinger, S., & Salminen, V. (Aug. 2001). Patterns of product development interactions. *Proc. Inter. Conf. Eng. Des.*, 1–8.
- Erixon, G. (1998). MFD-modular function deployment, a systematic method and procedure for company supportive product modularization. Doctoral dissertation, Ph.D. thesis, Department of Manufacturing Systems, The Royal Institute of Technology, Stockholm, Sweden.
- Harrison, J. R., Lin, Z., Carroll, G. R., & Carley, K. M. (2007). Simulation modeling in organizational and management research. *Academy of Management Review*, 32(4), 1229–1245.
- Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1), 1–14.
- Helmer, R., Yassine, A., & Meier, C. (2010). Systematic module and interface definition using component design structure matrix. *Journal of Engineering Design*, 21(6), 647–675.
- Henderson, R. M., & Clark, K. B. (1990). Architectural innovation: The reconfiguration of existing product technologies and the failure of established firms. *Administrative Science Quarterly*, 35(1), 9–30.

- Hoetker, G. (2006). Do modular products lead to modular organizations? *Strategic Management Journal*, 27(6), 501–518.
- Hölttä-Otto, K., Chiriac, N. A., Lysy, D., & Suk Suh, E. (2012). Comparative analysis of coupling modularity metrics. *Journal of Engineering Design*, 23(10/11), 790–806.
- Kerzner, H. (2013). *Project Management: A Systems Approach to Planning, Scheduling, and Controlling*. Hoboken, NJ, USA: Wiley.
- Krackhardt, D., & Hanson, J. R. (1993). Informal networks: The company behind the chart. *Harvard Business Review*, 71, 104–111.
- Krishnan, V., & Ulrich, K. T. (2001). Product development decisions: A review of the literature. *Management Science*, 47(1), 1–21.
- Krishnan, V., Eppinger, S. D., & Whitney, D. E. (1997). A model-based framework to overlap product development activities. *Management Science*, 43, 437–451.
- Lindemann, U., & Maurer, M. (2007). Facing multi-domain complexity in product development. in *The Future of Product Development*, Springer. Berlin, Heidelberg, 351–361
- MacCormack, A., Baldwin, C., & Rusnak, J. (2012). Exploring the duality between product and organizational architectures: A test of the “mirroring” hypothesis. *Research Policy*, 41(8), 1309–1324.
- McCord, K. R., & Eppinger, S. D. (1993). Managing the integration problem in concurrent engineering. M.S. thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, Cambridge, MA, USA.
- Meier, C., Yassine, A., & Browning, T. (2007). Design process sequencing with competent genetic algorithms. *Journal of Mechanical Design*, 129(6), 566–585.
- Miller, K. D. (2015). Agent-based modeling and organization studies: A critical realist perspective. *Organization Studies*, 36(2), 175–196.
- Millhiser, W. P., Coen, C. A., & Solow, D. (2011). Understanding the role of worker interdependence in team selection. *Organization Science*, 22(3), 772–787.
- Nasrallah, W. F., Ouba, C. J., Yassine, A. A., & Srour, I. M. (2015). Modeling the span of control of leaders with different skill sets. *Computational and Mathematical Organization Theory*, 21(3), 296–317.
- Pimmler, T. U., & Eppinger, S. D. (1994). Integration analysis of product decompositions. Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA, USA, Tech. Rep. 3690-94.
- Rechtin, E. (1991). *Systems Architecting: Creating and Building Complex Systems*. Englewood Cliffs, NJ, USA: Prentice-Hall.
- Rouse, W. B., & Boff, K. R. (2005). *Organizational Simulation*. Hoboken, NJ, USA: Wiley.
- Simon, H. (1969). *The Sciences of the Artificial*. Cambridge, MA, USA: MIT Press.
- Sorkun, M. F., & Furlan, A. (2017). Product and organizational modularity: A contingent view of the mirroring hypothesis. *European Management Review*, 14(2), 205–224.
- Sosa, M. E., Eppinger, S. D., & Rowles, C. M. (2004). The misalignment of product architecture and organizational structure in complex product development. *Management Science*, 50(12), 1674–1689.
- Sosa, M. E., Eppinger, S. D., & Rowles, C. M. (2007). Are your engineers talking to one another when they should? *Harvard Business Review*, 85(11), 133–142.
- Stone, R., Wood, K., & Crawford, R. (2000). A heuristic method for identifying modules for product architectures. *Design Studies*, 21(1), 5–31.
- Tripathy, A., & Eppinger, S. D. (2013). Structuring work distribution for global product development organizations. *Production and Operations Management*, 22(6), 1557–1575.
- Ulrich, K. T., & Eppinger, S. D. (2008). *Product Design and Development* (4th ed.). New York, NY, USA: McGraw-Hill.

- Wilde, D. (2010). Personalities into teams: We take different approaches to problems, and the best solutions are achieved by the greatest diversity. *Mechanical Engineering*, 132(2), 22–26.
- Yassine, A. A., & Bradley, J. A. (2013). A knowledge-driven, network-based computational framework for product development systems. *Journal of Computing and Information Science in Engineering*, 13(1), 011005.
- Yassine, A. A., Chidiac, R. H., & Osman, I. H. (2013a). Simultaneous optimisation of products, processes, and people in development projects. *Journal of Engineering Design*, 24(4), 272–292.
- Yassine, A., & Braha, D. (2003). Four complex problems in concurrent engineering and the design structure matrix method. *Concurrent Engineering Research Applications*, 11(3), 165–176.
- Yassine, A., Joglekar, N., Eppinger, S. D. & Whitney, D. (2003a). Information hiding in product development: The design churn effect. *Research in Engineering Design*, 14(3), 145–161.
- Yassine, A., Maddah, B., & Nehme, N. (2013b). Optimal information exchange policies in integrated product development. *IIE Transactions*, 45(12), 1249–1262.
- Yassine, A. A., Sreenivas, R. S., & Zhu, J. (2008). Managing the exchange of information in product development. *European Journal of Operational Research*, 184(1), 311–326.
- Yassine, A., Whitney, D., Daleiden, S., & Lavine, J. (2003b). Connectivity maps: Modeling and analyzing relationships in product development processes. *Journal of Engineering Design*, 14(3), 377–394.
- Yu, T. L., Yassine, A. A., & Goldberg, D. E. (2007). An information theoretic method for developing modular architectures using genetic algorithms. *Research in Engineering Design*, 18(2), 91–109.