

Complex Concurrent Engineering and the Design Structure Matrix Method

Ali Yassine^{1,*} and Dan Braha²

¹Center for Technology, Policy and Industrial Development (CTPID), ²Center for Innovation in Product Development (CIPD),
Massachusetts Institute of Technology, Cambridge, MA 02139, USA

Abstract: Concurrent engineering (CE) principles have considerably matured over the last decade. However, many companies still face enormous challenges when implementing and managing CE practices. This is due to the increased complexity of engineering products and processes, on one hand, and the lack of corresponding CE models and tools, on the other hand.

This paper focuses on four critical problems that challenge management while implementing CE in complex product development (PD) projects. We refer to these problems as: iteration, overlapping, decomposition and integration, and convergence problems. We describe these problems proposing a unified modeling and solution approach based on the design structure matrix (DSM) method, which is an information exchange model that allows managers to represent complex task relationships to better plan and manage CE initiatives.

Key Words: complex product development, concurrent engineering (CE), design structure matrix (DSM), partitioning, clustering, complex systems.

1. Introduction

Engineering systems are increasing in scale, scope, versatility, competitiveness, and complexity. Consequently, an interdisciplinary approach is required that appropriately reacts to increasingly complex situations concerning not only technological, but also factors external to the project (e.g., socio-economic and environmental factors). Similarly, decisions and processes made within a product development environment have important implications not just to customers and competitors but also for the macroeconomic environment.

Concurrent engineering (CE) is an engineering management philosophy and a set of operating principles that guide a product development process through an accelerated successful completion. The overall CE philosophy rests on a single, but powerful, principle that promotes the incorporation of downstream concerns into the upstream phases of a development process. This would lead to shorter development times, improved product quality, and lower development-production costs.

Concurrent engineering is concerned with the timely availability of critical design information to all development participants. For most complex engineering tasks all relevant information required by a specific

development team cannot be completely available at the start of that task. Therefore, CE requires the maximization of such information and the ability to share and communicate useful information on a timely basis.

The last few years have witnessed a resurgence of interest in complex systems [3,6]. Complex systems theory is concerned with the understanding of intrinsic interactions and nonlinear dynamics of systems with many parts. Complex systems theory is a suitable framework that accounts for the complex interactions among the various functions of a manufacturing enterprise, which give rise to a complex behavior that cannot be attributed to a single separate subfunction but it is rather a collective effect. Understanding CE as complex systems may suggest ways for improving the decision-making process, and the search for innovative solutions. It may also lead to the development of guidelines for coping with complexity.

This paper presents four underlying CE principles that are applied to large-scale complex concurrent environments. The CE principles that are elucidated in this paper may be derived from fundamental characteristics of complex systems and design decision-making [4]. Decision-making during the design activity deals with highly complex situations. The traditional methods of decision-making are based on the classical model of pure rationality, which assumes full and exact knowledge about the decision situation being considered. In design, assumptions about the exact knowledge are almost never true. At least to a large

*Author to whom correspondence should be addressed.

measure, the requirements are not comparable and therefore, the preference ordering among them is incomplete. The departure from ‘pure-rationality’ based models to ‘bounded-rationality’ based models [14] is needed in design because of the fact that the designer has a limited information-processing capacity and the information is uncertain, vague, or imprecise. Such limitations may arise in several ways: the designer may not know all the alternative sequence of decisions; or even assuming all the conditions are known, the designer may be unable to decide the best sequence of decisions; or finally, the time and cost of computing the best possible choices may be beyond the bounds of the available resources.

Building on the above observations, the CE principles discussed in this paper are:

- **‘Iteration’ principle:** First, designers are only human and have a bounded rationality. They cannot simultaneously consider every relevant aspect of any given design. As the design process progresses, new information, ideas, and technologies become available that require modifying the design. Second, design systems are limited; there is no known system that can directly input a set of requirements and yield the optimum design. Rather, the designer must iteratively break down the set of requirements into dimensions, constraints, and features and then test the resulting design to see if the remaining requirements were satisfied. Finally, the real world often responds differently than is imagined. The real world is full of chaotic reactions that are only superficially modeled in any design system. All this points to the seemingly undeniable truth that there is an inherent, iterative nature to the design process. Each iteration results in changes that must propagate through the design stages, requiring upstream rework. Consequently, late feedback and excessive upstream rework should be minimized; e.g. by utilizing multifunctional teams and early involvement of downstream activities in upstream stages.
- **‘Parallelism’ principle:** Scalable and complex systems must be highly parallelizable if short development times are required; otherwise valuable development times and resources are wasted. According to Amdahl’s law [2], the system’s performance (i.e., achievable speed-up) of a parallel system can be significantly limited by the existence of a small fraction of inherently sequential tasks, which cannot be parallelized. Consequently, multiple development stages have to be performed in parallel or with some overlap by sharing early preliminary upstream information with downstream stages.
- **‘Decomposition’ principle:** Complex systems are often decomposed into a number of simpler subsystems

that can be controlled independently, and whose individual behaviors yield the performance of the original complex system. Decomposition is also intended to exploit opportunities for parallel execution. The decomposition of a complex system into nearly independent subsystems is an inevitable consequence of ‘bounded rationality’; i.e., the limitations on the cognitive and information-processing capabilities of processors (e.g. the designer’s decision-making). In complex product development, processes are generally divided up into tasks and subtasks. Proper decomposition of design development tasks is concerned with assigning into the same team tasks that are anticipated to require high problem-solving interaction, while assigning to different (“independent”) teams tasks that require low problem-solving interaction. Minimizing the need for problem-solving across teams can have important consequences on product development performance, particularly since much of the activity of innovative product development projects involves problem solving and the creation of new knowledge [5]. Thus, the problem becomes the proper decomposition of the overall system into smaller manageable pieces (which are assigned to multiple development teams) by considering the constraints imposed by bounded rationality.

- **‘Stability’ principle:** At a macroscopic level, complex systems show a coexistence of multiple ground states – or, equivalently, multiple equilibria – which are robust against changes in the internal structure of the system. The system is said to be stable if the state of the system converges to one of the equilibrium states for any initial conditions. A product development process is said to be stable if the total number of design problems being solved remains bounded as the project evolves over time, and eventually falls below an acceptable threshold within a specified time frame. As implied by the iteration principle, design iterations result in changes that must propagate through the design stages, requiring upstream rework. This additional rework might slow down the PD convergence or have a destabilizing effect on the system’s behavior. Several strategies may be utilized in order to mitigate the slow convergence or divergence of PD processes. As a general rule, the rate of problem solving has to be measured and controlled such that the total number of design problems being created is smaller than the total number of design problems being solved.

The above principles are transformed into four fundamental problems. The first principle is concerned with iteration management and control and as such we refer to it as the “Iteration problem”. The second principle deals with overlapping practices in CE and we

refer to the counterpart problem as the “Overlapping problem”. The third principle is concerned with decomposing a large project (or system) into smaller pieces (or subsystems), solving these pieces, and then reassembling them into an overall system solution. We refer to it as the “Decomposition and Integration problem”. Finally, the fourth principle is concerned with the understanding of the intrinsic interactions and dynamics of decomposed product development, and the circumstances under which projects exhibit persistent problems or reach satisfactory performance levels. The corresponding problem is referred to as the “Convergence problem”.

The Iteration, Overlapping, and Decomposition and Integration problems address nontemporal (‘structural’ or ‘static’) issues of product development such as team and product architecture formation, while the Convergence problem is concerned with temporal (‘dynamic’) aspects such as understanding the complex behavior of product development tasks over time.

The above principles and related problems are addressed within a flexible methodology called the Design Structure Matrix (DSM) [9,17,19]. The DSM is shown to share several common methodological themes with complex systems theory including (1) the connectivity effect on product development performance, (2) dynamics of complex product development interaction, and (3) classification of complex collaborative design.

The rest of the paper proceeds as follows. In Section 2 we provide an overview of the DSM method, exposing the different types of DSM models and techniques available in the literature. In Section 3, we discuss the iteration problem, in general, and the different types of iterations as represented by a DSM model. The overlapping problem is discussed in Section 4. Sections 5 and 6 discuss decomposition–integration and convergence of complex systems as viewed by a DSM model. Finally, we conclude the paper in Section 7.

2. Overview of the Design Structure Matrix (DSM) Method

Many of the traditional project management tools (Gantt, CPM, PERT, and IDEF methods, e.g. see [20,21]) do not address problems stemming from project complexity. They allow project and engineering managers to model sequential and parallel tasks but not interdependent tasks, where a set of tasks is dependent on one another.¹ The DSM method provides this representation capability in a simple and elegant manner.

¹Interdependent tasks are characterized by cyclic flows of information. That is, Tasks A and B are said to be interdependent if Task A needs information from Task B and Task B needs information from Task A.

The DSM approach to managing complex development projects is an information exchange model which allows the project or engineering manager to represent important task relationships in order to determine a sensible sequence for the tasks being modeled. In this section, we describe the basic DSM method and how it can be used to model projects, determine the information needed by each task from other tasks, and to more accurately isolate and plan for problem areas taking place in the project.

A DSM is a matrix representation of a directed graph that represents a complex system.² The nodes of the graph correspond to the column and row headings in the matrix, and the arrows correspond to the ‘*’ marks inside the matrix.³ For example, if there is an arrow from Node C to Node A, then a ‘*’ mark is placed in Row A and Column C (see Figure 1). Diagonal elements have no significance and are normally blacked-out.⁴

If the DSM elements represent tasks to be performed, then inspecting the row and column of a task, reveals the inputs and outputs, respectively, for that task. For example, in Figure 1, B feeds C, F, G, J, and K, while D is fed by E, F, and L. If there is a time sequence associated with the position of the elements in the matrix, then all marks above the diagonal are considered feedback marks. Feedback marks correspond to required inputs that are not available at the time of executing a task. In this case, the execution of the dependent task will be based on ‘assumptions’ regarding the status of the input tasks. As the project unfolds these assumptions are revised in light of new information, and the dependent task is reexecuted if needed. It is worth noting how easy it is to determine feedback relationships in the DSM compared to the graph, which makes the DSM a powerful, but simple, graphic representation of a complex system or project.

The matrix can be manipulated in order to eliminate or reduce the feedback marks. This process is called partitioning [17,18]. When this is done, a transparent structure for the network starts to emerge, which allows better planning of the PD project. We can see which tasks are sequential, which ones can be done in parallel, and which ones are coupled or iterative (see Figure 1c). Implementing the CE principles proposed in this paper will facilitate the proper management of coupled tasks, which in turn can have important consequences on product development efficiency, effectiveness, and performance. This is especially important in innovative

²Several researchers proposed matrix methods for system modeling and analysis over thirty years ago [17]. However, it is not until recently that these methods started to attract attention for managing complex product development processes [9].

³There are different ways of building a DSM. For a full description, please refer to the MIT DSM website at <http://web.mit.edu/dsm>.

⁴Some researchers have used the diagonal elements of the DSM to capture task durations or work completion rates [19].

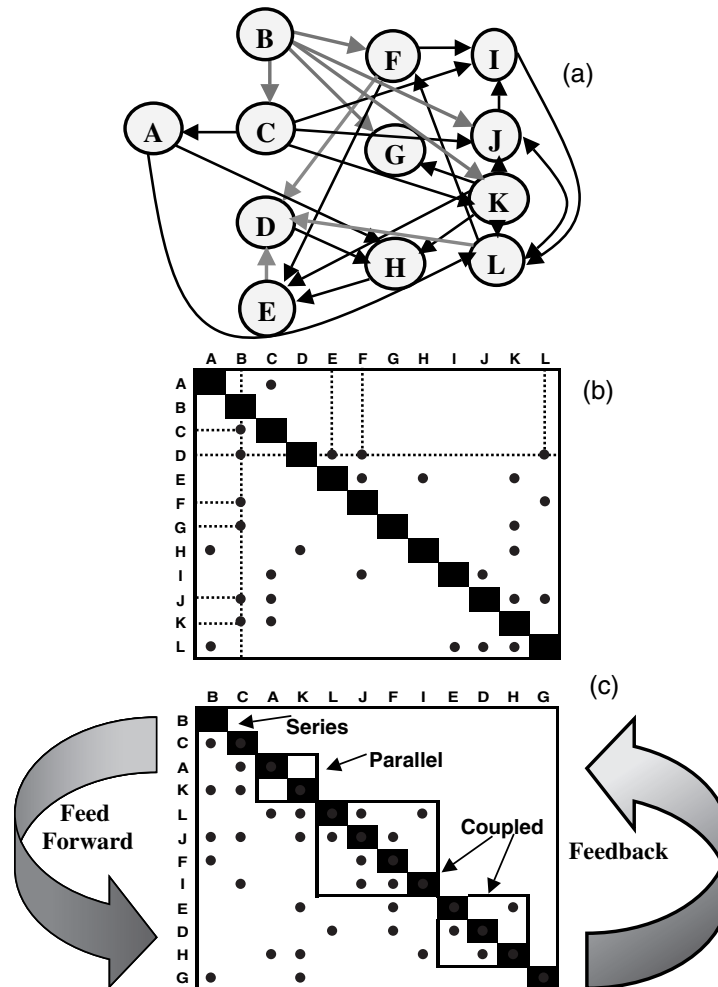


Figure 1. DSM overview: (a) Spaghetti graph; (b) Base DSM; (c) Partitioned DSM.

product development projects, which involve many coupled tasks due to high problem solving and the creation of new knowledge.

Once the DSM is partitioned, tasks in series are identified and executed sequentially. Parallel tasks are also exposed and can be executed concurrently. For the coupled ones, upfront planning is necessary. For example, we would be able to develop an iteration plan by determining what tasks should start the iteration process based on an initial guess or estimate of a missing piece of information. In Figure 1(c), block E-D-H can be executed as follows: Task E starts with an initial guess on H's output, E's output is fed to Task D, then D's output is fed to Task H, and finally H output is fed to Task E. At this point, Task E compares H's output to the initial guess made and decides if an extra iteration is required or not depending on how far the initial estimate deviated from the latest information received from H. This iterative process proceeds until convergence occurs (modeling the convergence process is described in Section 6).

In partitioning, we have seen that the main objective was to move the feedback marks from the above the

diagonal to below the diagonal, given that the DSM elements were tasks to be executed. However, when the DSM elements are people in charge of these tasks or are subsystems and components of a larger system, then we have a different objective for arranging the DSM. The new goal becomes finding subsets of DSM elements (i.e., clusters or modules) that are mutually exclusive or minimally interacting. This process is referred to as clustering. In other words, clusters contain most, if not all, of the interactions (i.e., DSM marks) internally and the interactions or links between separate clusters is eliminated or minimized [1,4,5]. In which case, the blocks become analogous to team formations or independent modules of a system (i.e., product architecture). Furthermore, in this setting, marks below the diagonal are synonymous to marks above the diagonal and they represent interactions between the teams or interfaces between the modules.

As an example, consider the DSM in Figure 2. The entries in the matrix represent the frequency and/or intensity of communication exchanged between the

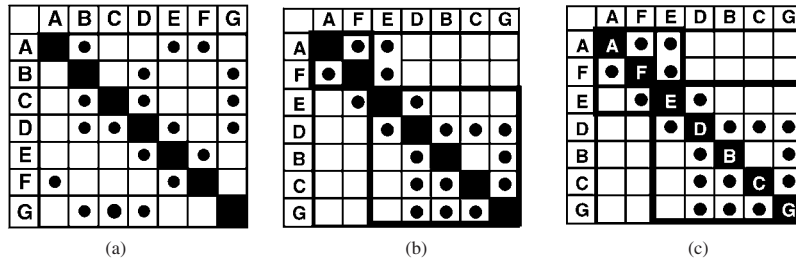


Figure 2. Clustering example: (a) Original DSM; (b) Clustered DSM; (c) Alternative Clustering.

different development participants, represented by person A, person B, ...etc.⁵

As can be seen in Figure 2(b), the original DSM was rearranged to contain most of the interactions within two separate blocks: EF and EDBCG. However, three interactions are still outside any block (i.e., team) and constitute the points of interaction–collaboration between the two teams. An alternative team arrangement is suggested in Figure 2(c). This arrangement suggests the forming of two overlapping teams (i.e., AFE and EDBCG) where one person (i.e., E) belongs to both teams and may act as a liaison person.⁶ The proper decision about how the elements of the product are assigned to chunks is affected by several technical and nontechnical factors. Several computational clustering techniques that search for optimal solutions based on tradeoffs between the importance of capturing intra block dependencies versus inter block dependencies may be able to arrive to optimal solutions under certain assumptions [4,5,11].

We close this section by exposing four different types of DSM models and their application to various levels of abstraction [8]. The ‘Task-based’ DSM model has been discussed earlier (see Figure 1). A variation of the Task-based DSM is the ‘Parameter-based’ DSM, which takes a deeper look into the tasks by decomposing them further into the design parameters that are delivered by each task. Consequently, a parameter-based DSM describes how does the parameters influence each other and in what order they should be determined to minimize guessing and feedback. Thus, partitioning is also used to streamline the sequence of parameter calculations. The third DSM type is called ‘Team-based’ DSM. This DSM model is used for organizational analysis and team formations based on the intensity and frequency of information flow among various organizational entities (i.e., development participants). Finally, the ‘Component-based’ DSM documents interactions between elements in a complex system architecture. For the latter two DSM models, clustering is used to rearrange the DSM

and identify improved team formations or improved product architecture.

3. Iteration

The design and development of complex systems is fundamentally iterative. This process is driven by the repetition of tasks due to the availability of new information such as changes in input, updates of shared assumptions or the discovery of errors. Some iteration is predictable and can be planned in advance. Others are unpredictable, unplanned and impossible to anticipate. Using the DSM method we are able to better manage both anticipated and unanticipated iterations.

Anticipated or planned iterations occur when some development tasks are attempted even though the complete predecessor information is not available or known with certainty. As this missing or uncertain information becomes available, the tasks are repeated to either verify an initial estimate/guess or to come closer to the design specifications. A simple example of a planned iteration is the development of a heat exchanger that depends on three key variables: temperature, heat flux, and the heat transfer coefficient [17]. The value of each of these parameters depends on the values of the other two; therefore, to determine the parameters of the design, the guess of an initial value for one of these parameters is needed to start the process. For example, guessing a temperature value will lead to a heat flux value, which will lead to a heat transfer coefficient value. The heat transfer coefficient value will in turn give a new temperature value and so on.⁷ In the absence of development time or budget constraints, this iteration process will continue until the final output value of the temperature converges to an acceptable value.

Planned iteration is usually of the useful in complex product development since such iterations are used to refine an evolving design and allows for innovation [9].

⁵That is, how frequently teams need to work with other teams (daily, weekly, or monthly).

⁶See also Figure 8.

⁷This planned iteration approach is called “sequential” iteration. An alternative approach would be “concurrent” iteration, which calls for the parallel execution of all coupled tasks in a block. For the heat exchanger example, all three parameters are calculated at once (based on three separate guesses), and then a decision is made whether another round of parallel calculations is required. Concurrent iteration will be revisited in Section 6.

	A	B	C	D	E	F	G	H	I	J
Concept	A	■								
Business Requirements	B	*	■				○	○	○	
System Requirements	C	*	■	■			○	○		
Network Plan	D	*	*	■	*		○	○		
Technical Specifications	E	*	*	*	■		○	○		
Engineering Design	F		*	*	*	■				
Billing Implementation	G		*	*	*	*	■		*	
Operations Engineering	H		*	*	*	*	*	■	*	
Customer Service	I		*	*			*	*	■	
Launch	J					*	*	*	*	■

Figure 3. Telecommunications services development (adapted from [9]).

Once identified by a DSM model, these iterations can be managed by facilitating the fast information exchanges between the tasks involved in iteration. Information technologies play a major role in speeding up these iterative cycles. Alternatively, persons involved in iterative cycles maybe relocated next to each other during that phase of the development to facilitate the fast information exchanges.

Unplanned or unanticipated iterations occur when tasks are repeated due to unexpected failure of the design to meet specifications. This type of failure typically occurs during the execution of testing and integration activities, or due to sudden changes in design objectives driven by changing customer needs–requirements. Identifying these unexpected iterations correspond to identifying potential failure modes of a design process. Unlike planned iteration, unplanned iteration has a bad connotation and managers typically need to avoid it rather than facilitate it. Once this ‘bad’ iteration type is identified, managers need to devise plans to reduce the probability of its occurrence.

For example, consider the development process for data services at a telecommunications company as shown in Figure 3. The unplanned iterations are revealed as feedback marks across multiple phases of development (i.e., the ‘hollow-circle’ marks ‘○’). While the company’s current project management plans identify, realize and organize for iterations within each phase, as depicted by the ‘*’ marks, they fail to consider and integrate the impact of the unplanned iterations, which were realized during execution. Once the company identified the existence of both iteration types, they facilitated the information flows within the development stages and formalized the information exchanges between the stages (which was not originally addressed). This resulted is a robust development process and realistic development schedule/budget.

4. Overlapping

In order to illustrate the basic overlapping model, consider the example of overlapping to sequential development stages: A and B, where Stage B depends

on Stage A for information. These stages can be executed sequentially or with some overlap. The objective of the overlapping problem is to find the best overlapping magnitude that minimizes total development lead time. The trade-off encountered in such formulations can be described as follows: starting the downstream stage earlier (based on preliminary upstream information) gives the downstream stage a head-start, but runs the risk of wasting valuable development resources if excessive downstream rework occurs as a consequence of using early and premature upstream information. So, the rework is a function of how much variability the upstream information has as it evolves from its preliminary form to its final form. The larger the overlap time is, the larger the probability that the upstream information will change during overlap. Consequently, the expected reduction in development time is a function of the amount of overlap. Normally, a convex relationship exists between the amount of overlapping and total development lead time. However, these convex formulations [12,22] do not fully account for the complex reality of product development and require the accurate assessment of several development parameters before arriving at an optimal overlapping policy. Therefore, there is a need for the development of more realistic modeling techniques that can be readily employed by PD managers.

In order to use the above overlapping setup in conjunction with DSM models, it is important to estimate two characteristics: upstream information variability and downstream task sensitivity [18]. The ‘sensitivity’ of a task to one of its predecessors expresses how sensitive a task is to changes in the output of its predecessor. If a task is very sensitive to its predecessor information, then a small change in predecessor information has a huge impact on its final results. Alternatively, if a task is insensitive (or weakly sensitive) to predecessor design changes, then predecessor information has to change drastically before the dependent task is affected by the change. ‘Variability’ is the possible deviation of an estimate, at the time of assessment, from the actual value. Here, we are concerned with the variability of the predecessor task and the impact of this variability on the successor task. A task exhibits a high variability if the team working on the successor task is incapable of guessing a value, or range of values, for the output of the predecessor task. On the other hand, a task is said to exhibit low variability if the team working on the successor task can always provide a good estimate on the output value of its predecessor.

The overlapping potential between pairs of sequential tasks can be evaluated by assessing the above two constructs. Normally, overlapping is beneficial in reducing total lead-time between pairs of sequential tasks that have weak sensitivity and low variability.

Tasks	A	B	C	D	E	F	G	H	I
A Engineering Concept	■								
B Design Concept	●	■							
C Product Design		●	■	■	●	●			
D Design Documentation			●	■	■				
E Design Checking				●	■	■			
F Core Design		●	●	●	●	■	■	■	●
G Soft Tooling						●	■	■	
H Tryout Castings							●	■	■
I Casting Development								●	■

Figure 4. Automotive cylinder block development.

Once the overlapping points are identified, the ‘★’ marks in the DSM are replaced by a ☉ mark signifying the potential of overlapping between tasks.

As an example consider the DSM describing the development of an automotive cylinder block as shown Figure 4. Originally, the three overlapping marks shown in the DSM were represented by a single dependency mark from the ‘design checking’ task to the ‘core design’ task (i.e., Row F, Column E). The original process was sequential and the ‘core design’ task was not started until the complete finish of the iteration block C-D-E. After analyzing the information flows in light of the two measures of sensitivity and variability, it was apparent that the ‘core design’ task could be started earlier and overlaps with block C-D-E [18]. The result was to replace the dependency mark in (Row F, Column E) by the three overlapping marks shown in the figure, and to add an additional dependency mark in (Row F, Column B), which did not exist originally. The three overlapping marks denote partial information flows that are necessary for the early start of the core design task.

5. Decomposition and Integration

In complex PD projects, the development effort is usually distributed among several development teams in order to reduce the technical complexity of development, as no single team has the expertise to tackle the whole development process. However, this decomposition creates a serious managerial challenge due to the complex web of dependencies and interactions between these pieces, which need to be managed carefully.

These teams are called ‘local’ development teams. They are domain experts and in charge of developing a subsystem. They work concurrently on the development project and continuously interact with each other. The efforts of these local teams are coordinated and orchestrated by product integrators that belong to a ‘system’ team. System teams overlook the whole

development process, interact with all local teams, and provide feedback and direction. As an example, in a software development project, like that of MS Excel, the local teams would be feature teams (implementing/coding a set of features like editing or file sharing), and the system team could be testers that compile the different pieces of code together in order to test their compatibility with each other.

To get a better understanding of the complex web of interactions that exist between these local teams, let us consider the development of an automotive engine. The development teams are: engine block, pistons, crankshaft, cylinder head, crankshaft, camshaft, induction, and emissions–electrical. Since many engine components (such as ‘pistons’, ‘cylinder head’, ‘crankshaft’, and ‘camshaft’) interface directly with the engine block, changes in any of these components (that are developed by separate teams) influence changes in block design. For example, in order to keep the engine as compact as possible, the block team tries to minimize the spacing between the cylinder barrels. However, the team needs to accommodate the ‘cylinder head’ bolt pattern. Something they need to negotiate and communicate with the ‘cylinder head’ team. As a matter of fact, the location of all bosses on the block requires coordination with all other development teams. A more complex interaction would be the design of the water passages and oil gallery within the block. These need to be coordinated across the multiple teams to ensure that all passages connect and deliver sufficient oil and water flow. Product integrators put the design of the different components together to ensure fit and functionality.

A component-based DSM can be created for the physical interactions between the components to represent the structure of the system decomposition and integration problem and ultimately allow for better architectural decisions. On the other hand, the communications between the different development teams could also be created, using a team-based DSM, in order to better map people and teams, and decide who should be on the team to address the system problems.

The above automotive engine development process was actually mapped by a DSM as shown in Figure 5(a). The figure shows the original team structure that was in place for this development process. The rearranged DSM in Figure 5(b) provided a better team structure for this process since more of the interactions are now contained within teams, and the overall coordination was assigned to a system team. Notice how the size of the dependency marks was used to reflect varying degrees of communication intensity between the different development participants.

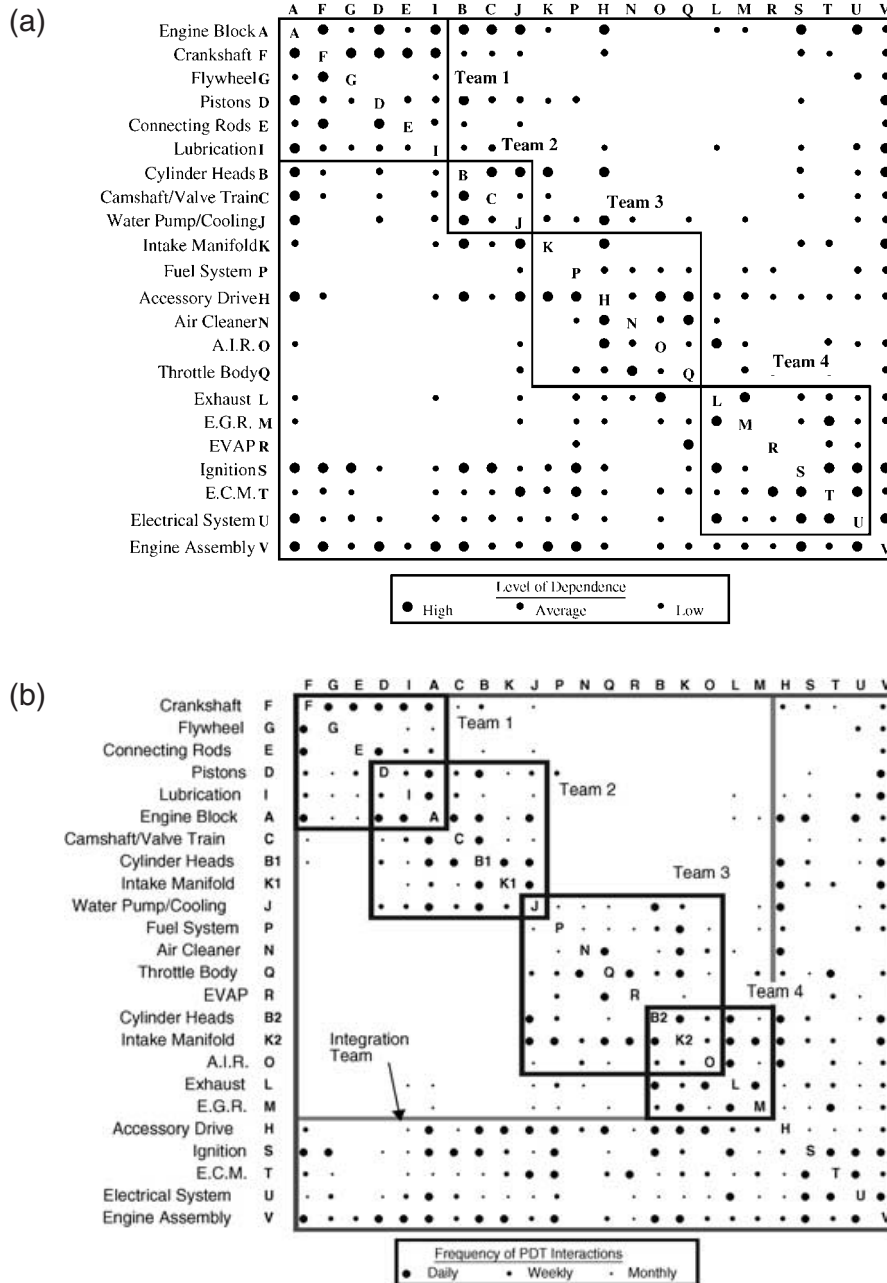


Figure 5. Team-based DSM (sdapted from [7]): (a) Original DSM; (b) Restructured DSM.

6. Convergence

In this section, we consider the Convergence problem, which is concerned with dynamic aspects of product development. Addressing this problem will provide explanations for why CE projects can take so long and cost so much, even in cases where the best design practices are followed. Moreover, mitigation strategies that help accelerate the convergence of product development can be developed.

As explained in Section 5, a complex development project is decomposed into smaller pieces that are assigned to multiple local development teams working

concurrently on the project. Furthermore, the work of such teams is coordinated and orchestrated by product integrators that belong to a system team. Product integrators put the separate development pieces together to ensure fit and functionality. Since late changes in product design are highly expensive, product integrators cannot delay their decisions until the design of all components is finalized. Consequently, they continuously check unfinished component designs and provide feedback accordingly.

Consider a development team that has initially 100 open problems to work on. As the project unfolds, the number of open problems is reduced with time until

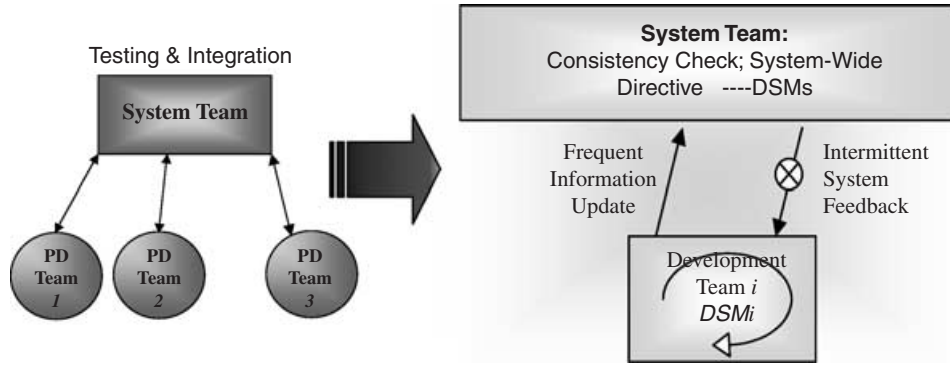


Figure 6. Local and system bifurcation of information.

some new information or feedback is received, in which case previously solved problems are either reopened or new problems are created. This cycle repeats until all the work is finished. We call this oscillatory development progress the ‘Design Churn’ phenomenon.⁸

The flow of information among local development and system integration teams is described in Figure 6. Figure 6(a) shows multiple local development teams that concurrently work on the development project, and continuously interact with each other and with the system team. Figure 6(b) shows how a single local team interacts with a system team. The local team performs work and iterates internally. Also, it provides progress updates continuously to the system team. The system team processes this information, and then provides the local team with periodic feedback. There is some time delay from the moment local teams finish part of their work until the moment they receive feedback on it, which may create new problems or reopen previously solved problems. For example, system testing for various pieces of the development process may take different times to process, and thus system feedback will be provided to local teams at different times. In effect, the inherent delays associated with system tests amount to information hiding since for a while the local teams are unaware of the newly created problems. Thus, this effect is referred to as ‘Information Hiding.’

The delays in receiving the system feedback may be attributed to several causes. For example, the system team may perform an expensive system testing (i.e., crash testing or building a prototype), and would want to batch changes into a single prototype rather than perform the testing on multiple prototypes.

The DSM shown in Figure 7 captures the above development setup. The DSM is composed of blocks that represent several local development teams and a

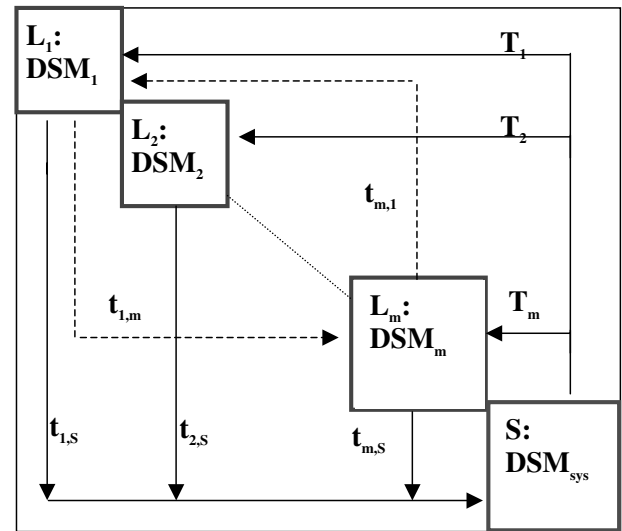


Figure 7. DSM representation of a PD process showing local and system teams (L_i represents a local development team, and S : represents a system team).

system integration team. The system team facilitates interactions between local teams as represented by the solid arrows in the figure. The dotted arrows represent a situation where the local teams are allowed to interact directly (i.e., without the facilitation of the system team). The local DSMs are internally updated and provide status information to others (local and system DSMs) at every time step (ΔT). The system DSM provides updates to the local DSMs at periodic intervals ($N_i \Delta T$). The system update periods (N_i 's) may or may not be synchronous.

To illustrate, consider the following automotive appearance design process at a large automotive company [19]. Appearance design refers to the process of designing all interior and exterior automobile surfaces for which appearance, surface quality and operational interface is important to the customer. Such design items include, for example, exterior sheet metal design and visible interior panels. Appearance design is the earliest of all physical design processes, and changes in this stage easily cascade into later development activities causing

⁸Formally, we define ‘Design Churn’ as a scenario where the total number of problems being solved does not reduce monotonically as the project evolves over time [19].

⁹This industrial design portion is allotted approximately 52 weeks for completion within the context of the overall automotive product development process.

costly rework. This is avoided by allowing ‘stylists’ (from the industrial design team) to work closely with ‘engineers’ (from the engineering design team). While ‘stylists’ are responsible for the appearance of the vehicle, ‘engineers’ are responsible for the feasibility of the design by ensuring that it meets some functional, manufacturing, and reliability requirements. Information exchanges from ‘styling’ to ‘engineering’ take the form of wireframe CAD data generated from clay model scans; referred to as scan transmittals of surface data. Scan transmittals are scheduled at roughly six weeks intervals. Information exchanges between ‘engineering’ and ‘styling’ occur on a weekly basis through a scheduled feasibility meeting. During these meetings various engineers provide feedback to ‘styling’ on infeasible design conditions.

The model in Figure 7 represents the above information transfer scenario; i.e., the local and system teams stand for ‘engineering’ and ‘styling,’ respectively. In addition to the cross-functional information exchanges between ‘styling’ and ‘engineering,’ information flows also occur within functional teams. For example, within ‘engineering,’ a hand clearance study would compile information about the front door trim panel and the front seat to determine whether the two components physically interfere, and whether the space between them meets minimum acceptable requirements.

The study of PD convergence and the ‘Design Churn’ phenomenon – due to time delays and asynchrony in information transfers between the system and different local groups – cannot be treated as a single DSM.¹⁰ Consequently, a dynamic DSM model (using ‘linear systems theory’) has been developed in [19], where the DSM notation has been expanded by replacing the ‘★’ with numerical dependency strength as well as replacing the diagonal elements with the rate of work completion for each task. Analyzing the model shows that the existence of design churn is a fundamental characteristic of the decomposition and integration of design between local and system teams. More specifically, it can be shown that design churn may be attributed to two modes. The first mode reflects the ‘fundamental churn’ of the design process, and the second mode, termed ‘extrinsic churn’, may be present depending on the relative rates of work completion and the rework induced between system and local tasks.

Conditions under which the total number of design problems (associated with the system and local tasks) converges to zero as the development time increases are provided in terms of a particular set of eigenvalues [19].

If the largest magnitude eigenvalue (most slowly converging design mode) is less than one, then the system is stable and the development process converges. Also, the eigenvector corresponding to the largest magnitude eigenvalue provides useful information regarding ‘bottleneck’ design tasks that require significant amount of work. More specifically, the larger the magnitude of an element in that eigenvector, the stronger the corresponding task contributes to the slowly converging design mode.

The dynamic model shows that it is possible for development processes to exhibit churning behavior under both converging and diverging scenarios. Thus, design churn is an unavoidable phenomenon and a fundamental property of a decomposed development process where the product or process architecture dictates delays in feedback information amongst the development groups. Consequently, a significant insight this model brings to managers is to avoid making myopic decisions based on the observation of churn. The fluctuation in development progress cannot be avoided, but can be managed once managers understand its sources. The dynamic model reveals several main sources of churn: (1) hidden, ignored, or forgotten interdependencies of process or product structure that are due to the process decomposition into multiple groups. Fully anticipating, understanding, and accommodating these interdependencies can explain why the tasks prone to regular change, (2) improper planning of feedback flows that drive the process unstable (e.g. generating more rework than the development teams can handle), and (3) feedback delays, which are the main reason why development problems, believed to be solved, tend to reopen at later stages of development.

Although ‘Design Churn’ is a fundamental property of decomposed development processes, managers can still apply several mitigation strategies to combat the above three sources of churn (i.e., interdependency, concurrency, and feedback delays), divergence, or slow convergence. Several strategies, which are effective either individually or collectively, may be considered: (1) Timing-based strategies advocate the minimization of delays for specific tasks (called ‘bottleneck’ tasks), which contribute the most to the slow convergence of the development process. The model presented in [19] provides a quantitative approach to identify these ‘bottleneck’ tasks. Once identified, strategies for reducing the time delays for these tasks should be implemented. The model presented in [19] demonstrates that accelerating the synchronization frequency for all tasks ‘may not’ be as effective as accelerating, by the same amount, the synchronization frequency for the ‘bottleneck’ tasks, (2) Resource-based strategies allow local and system teams to work faster by incorporating more resources. Working faster on all the tasks simultaneously ‘may not’ be as effective as allocating the same amount of

¹⁰In cases where there is no information delay between local and system task execution, Smith and Eppinger [16] have developed a method using linear systems theory to analyze such models and identified controlling features of a unified iteration process.

resources only to the ‘bottleneck’ tasks, and (3) Rework-based strategies suggest that any team (local or system) ‘ignores’ low priority feedback from others (local and system teams). A similar strategy is to reduce the feedback by requiring that teams (local or system) not produce much feedback to others (local or system teams). Rework-based strategies benefit from a modular architecture (see Section 2).

7. Summary

Concurrent Engineering (CE) is a method for managing the development of complex systems. CE is effective; but it requires a set of analytic tools and procedures to operationalize its concepts. Building on fundamental characteristics of complex systems and design decision-making, four principles have been proposed: (1) the ‘Iteration’ principle encapsulates the fact that there is an inherent, iterative nature to the design process. Iteration results in changes that must propagate through the design stages, requiring upstream rework; (2) the ‘Parallelism’ principle exploits the possibility of achieving shorter development times by performing multiple development stages in parallel or with some overlap (e.g. by sharing early preliminary upstream information with downstream stages); (3) the ‘Decomposition’ principle is concerned with the proper decomposition of the overall product development system into smaller subsystems that can be controlled and managed ‘nearly independently’; and (4) the ‘Stability’ principle is concerned with the understanding of the dynamics of decomposed product development, and the circumstances under which the total number of design problems being solved eventually falls below an acceptable threshold within a specified time frame.

These CE principles have been translated into four core problems, and a unified solution approach based on the DSM method has been proposed. The DSM method offers a promising suite of qualitative and quantitative methods capable of bridging the gap between CE theory and practice. DSM models deal with the understanding and formulation of the interactions among the many components of a complex product development project. The strength of DSM models comes from their usefulness in: (1) reducing the complexity of product development by providing managers with a comprehensive analytical tool-set; (2) revealing easily and clearly the information flows between the tasks of a complex system. This allows for a simple visual display and analysis of potential iterations. Resequencing tasks in a DSM minimizes iteration, and thus reduces complexity. The iteration loops that cannot be resolved via resequencing can be strategically planned for in-advance and incorporated in the project schedule; and thus results in more realistic development time and cost estimates; (3) allowing

development managers to discover previously unknown process, product and organizational patterns, which opens new avenues for improvement; and (4) understanding the dynamics of the total number of design problems being solved at any time point; thus, suggesting ways to mitigate the slow convergence or divergence of PD processes.

References

- Alexander, C. (1964). *Notes on the Synthesis of Form*, Harvard Press, Boston, MA.
- Amdahl, G.M. (1967). Validity of the Single-processor Approach to Achieving Large Scale Computing Capabilities, In: *AFIPS Conference Proceedings* (Atlantic City, N.J., April 18–20), Vol. 30, pp. 483–485, AFIPS Press, Reston, VA.
- Bar-Yam, Y. (1997). *Dynamics of Complex Systems*, Addison-Wesley, Reading, MA.
- Braha, D. and Maimon, O.A. (1998). *Mathematical Theory of Design: Foundations, Algorithms, and Applications*, Kluwer Academic Publishers, Boston, MA.
- Braha, D. (2002). Partitioning Tasks to Product Development Teams, In: *Proceedings of the ASME 14th International Conference on Design Theory and Methodology*, Montreal, Canada.
- Cowan, G.A., Pines, D. and Meltzer, D. (1994). *Complexity: Metaphors, Models and Reality*, Addison-Wesley, Reading, MA.
- McCord, K.R. and Eppinger, S.D. (1993). *Managing the Integration Problem in Concurrent Engineering*, Working Paper no.3594, M.I.T. Sloan School of Management, Cambridge, MA.
- Browning, T. (2001 August). Applying the Design Structure Matrix to System Decomposition and Integration Problems: A Review and New Directions, *IEEE Transactions on Engineering Management*, **48**(3).
- Eppinger, S. (2001 January). Innovation at the Speed of Information, *Harvard Business Review*.
- Eppinger, S.D., Whitney, D.E., Smith, R.P. and Gebala, D.A. (1994). A Model-Based Method for Organizing Tasks in Product Development, *Research in Engineering Design*, **6**: 1–13.
- Fernandez, C.I.G. (1998). *Integration Analysis of Product Architecture to Support Effective Team Co-Location*, Master’s Thesis (ME), Massachusetts Institute of Technology, Cambridge, MA.
- Krishnan, V., Eppinger, S.D. and Whitney, D.E. (1997). A Model-Based Framework to Overlap Product Development Activities, *Management Science*, **43**: 437–451.
- Pimmler, T.U. and Eppinger, S.D. (1994 September). Integration Analysis of Product Decompositions, In: *Proceedings of the ASME Sixth International Conference on Design Theory and Methodology*, Minneapolis, MN.
- Simon, H. (1996). *The Sciences of the Artificial*, MIT Press, Cambridge, MA.
- Sosa, M.E., Eppinger, S.D. and Rowles C.M. (2000 September). Designing Modular and Integrative Systems. *ASME Conference on Design Theory and Methodology*, Baltimore, MD.

16. Smith, R.P. and Eppinger, S.D. (1997). Identifying Controlling Features of Engineering Design Iteration, *Management Science*, **43**: 276–293.
17. Steward, D.V. (1981). The Design Structure System: A Method for Managing the Design of Complex Systems, *IEEE Transactions on Engineering Management*, **28**: 71–74.
18. Yassine, A., Falkenburg, D. and Chelst, K. (1999). Engineering Design Management: An Information Structure Approach. *International Journal of Production Research*, **37**(19): 2957–2975.
19. Yassine, A., Joglekar, N., Braha, D., Eppinger, S. and Whitney, D. (2002). *Information Hiding in Product Development: The Design Churn Effect*, MIT Sloan Working Paper No. 4333-02.
20. Spinner, M. (1989). *Improving Project Management Skills and Techniques*, Prentice-Hall, Englewood Cliffs, NJ.
21. Ross, D. (1977). Structured Analysis (SA): A Language for Communicating Ideas, *IEEE Transactions on Software Engineering*, **SE-3** (1): 16–34.
22. Joglekar, N., Yassine, A., Eppinger, S. and Whitney, D. (2001). Performance of Coupled Product Development Activities with a Deadline, *Management Science*, **47**(12): 1605–1620.

Ali Yassine



Ali Yassine is an Assistant Professor at the University of Illinois at Urbana-Champaign (UIUC), Department of General Engineering. His research involves managing the development process of complex engineering systems, design process modeling, and IT-enabled concurrent engineering methodologies. His publications appeared in *Management*

Science, *IEEE Transactions on Engineering Management* and several other international journals. He is a member of INFORMS, ASME and PDMA.

Dan Braha



Dan Braha is a Visiting Professor at the MIT Center for Innovation in Product Development (CIPD), an affiliate of the New England Complex Systems Institute (NECSI), and a senior engineering faculty member at Ben-Gurion University. Prior to that he was a Research Associate in the Department of Manufacturing Engineering at Boston Univer-

sity. One of his primary areas of research is engineering design and manufacturing. His research within engineering design focuses on developing methods to help the designer move from the conceptual phase to the realization of the physical device. He has developed a mathematical theory – the Formal Design Theory (FDT). Dan Braha has published extensively, including a book on the foundations of engineering design with Kluwer Academic Publishers, and an edited book on data mining in design and manufacturing; also with Kluwer. He serves on the editorial board of *AI EDAM*, and was the editor of several special journal issues. He has also served on executive committees and as chair in several international conferences. Currently, he aims to advance the understanding of Complex Engineered Systems (CES); arrive at their formal analysis; as well as facilitate their application.