2010 NDIA GROUND VEHICLE SYSTEMS ENGINEERING AND TECHNOLOGY SYMPOSIUM
SYSTEMS ENGINEERING & INTEGRATION (SE&I) MINI-SYMPOSIUM
AUGUST 17–19, DEARBORN, MICHIGAN

A SYSTEMS ENGINEERING & INTEGRATION METHODOLOGY FOR COMPLEX SYSTEMS

Frederick P. Samson
Associate
Booz Allen Hamilton
Troy, MI 48084

Troy Peterson
Senior Associate
Booz Allen Hamilton
Troy, MI 48084

ABSTRACT

Understanding the intricacy of a complex system is a major challenge in today's highly dynamic environment. In addition, managing the lifecycle processes and schedule risks of these complex systems adds to the challenge of achieving program objectives. Complexity is ever present in today's military system of systems architectures, acquisition lifecycle management processes, and supporting organizational structures. This complexity is often an impediment to the successful development, integration, and transition of capability gap-closing technologies to support our Warfighters' needs. Over the years, methods to reduce system complexity have taken many forms. The Design Structure Matrix (DSM) is one methodology that has proven very effective in the analysis, management, and integration of complex system architectures, organizational structures, and densely networked processes. DSM enables the user to model, visualize, and analyze the dependencies among the entities of any system—and derive suggestions for system optimization.

INTRODUCTION

The intent of this paper is to introduce Design Structure Matrix (DSM), a simple and insightful yet powerful Systems Engineering and Integration (SE&I) methodology for managing and developing complex systems. DSM is a matrix-based system modeling methodology that may be applied to the three critical domains in design and development of systems: product (Product Breakdown Structure), process (Work Breakdown Structure), and organization (Organizational Breakdown Structure). Delivering successful complex systems design and management through the use of DSM requires a deep understanding of system element interactions. DSM can assist by providing a compact and clear representation of a complex system and a capture method for the interactions, interdependencies, and interfaces between system elements.

This paper will address several applications of DSM to optimize system structures (architectures) in the domains mentioned above. It will also specifically address task-based DSMs (Process Architecture), component-based DSMs (System Architecture), and team-based DSMs (Organizational Architecture).

BACKGROUND

The use of matrices in system modeling can be traced back to the 1960s and '70s with Donald Stewart and John Warfield. However, it wasn't until the 1990s that the method received widespread attention. Much of the credit in its current popularity is accredited to MIT's research in the design process modeling arena by Dr. Steven Eppinger.

DSM—also known as the dependency structure matrix, dependency source matrix, and dependency structure method—is a square matrix that shows relationships between elements within a system. Since the behavior and value of many systems is largely determined by interactions between its elements, DSMs have become increasingly useful and important in recent years.

The DSM is related to other square-matrix–based methods, such as: a dependency map, a precedence matrix, a contribution matrix, an adjacency matrix, a reachability matrix, and an N-square diagram, and also related to non-matrix–based methods such as directed graphs, systems of equations, and architecture diagrams and other dependency models.

Relative to other system modeling methods, DSM has two main advantages that differentiate it from the others:

- DSM provides a simple and concise way to represent a complex system.
- DSM is capable of powerful analyses techniques—which will be discussed in subsequent sections of this paper.

UNDERSTANDING COMPLEXITY

The key to understanding and managing complexity is through system decomposition. Figure 1 may represent the decomposition of a complex:
– System into subsystems and components
– Process into subprocesses and tasks
– Organization into teams and individuals.

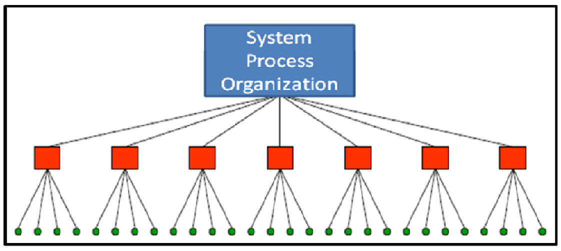


Figure 1: Decomposition of a System

The relationship or pattern of interactions between the decomposed elements defines the architecture:
– System architecture
– Process architecture
– Organization architecture.

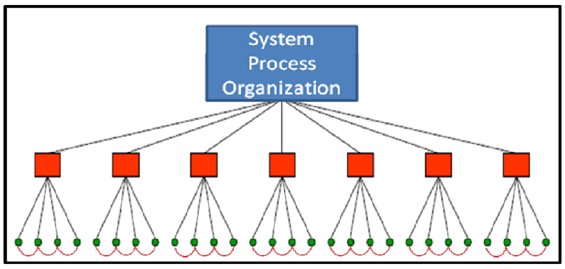


Figure 2: Decomposed Simple Architecture

Note the simple architecture in Figure 2 compared to the complex architecture in Figure 3. This is evident by the lack of pattern in the interactions between the lowest level elements of the system.

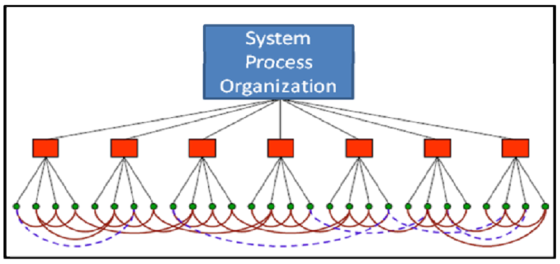


Figure 3: Decomposed Complex Architecture

DESIGN STRUCTURE MATRIX

There are two major types of DSMs: static (relationships between the system elements are not time-based) and temporal (relationships between the system elements are time-based). [2] For that reason, they are analyzed differently. Static DSMs are analyzed by clustering elements of the matrix, and temporal DSMs are analyzed by sequencing elements of the matrix.

The general DSM modeling approach consists of the following steps:

1) Define the system boundary
2) Describe important interfaces
3) Decompose the system into simpler elements
4) Define the characteristics of the elements
5) Characterize the element interactions
6) Analyze the system architecture (structure):
   a) System model behaviors
   b) Potential element arrangements/integrations.

In the next section, we will go into greater detail regarding the use of DSM for managing complex projects (task-based DSM).

Process Domain: Task-Based DSM

Before we introduce the use of the DSM method of analysis for program management tasks, we will briefly discuss the three possible types of task sequences.

Consider a system (or project) consisting of two elements (or tasks): Task A and Task B. As illustrated in Figure 4, there are three basic sequences for describing the relationship between the tasks: sequential (or dependent tasks), parallel (or independent tasks), or coupled (or interdependent tasks).

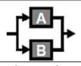
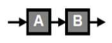
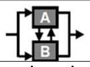
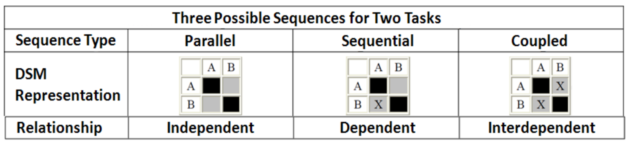
Three Possible Sequences for Two Tasks			
Sequence Type	Parallel	Sequential	Coupled
Graphic Representation			
Relationship	Independent	Dependent	Interdependent

Figure 4: Task Sequences and their Relationships

The matrix representation of the task sequences in Figure 4 is shown in Figure 5. The matrix layout is as follows: The system elements are placed down the side of the matrix as row names and across the top as column headings in the same order. If there is an interaction, it is marked with an X, and if there is no interaction, it is left empty. In a matrix representation of a system, the diagonal of the matrix does not assist in describing the system.



Three Possible Sequences for Two Tasks			
Sequence Type	Parallel	Sequential	Coupled
DSM Representation	$\begin{array}{c\|cc} & A & B \\ \hline A & ■ & \\ B & & ■ \end{array}$	$\begin{array}{c\|cc} & A & B \\ \hline A & ■ & \\ B & X & ■ \end{array}$	$\begin{array}{c\|cc} & A & B \\ \hline A & ■ & X \\ B & X & ■ \end{array}$
Relationship	Independent	Dependent	Interdependent

Figure 5: DSM Representation of Task Sequences

As shown in Figure 5, because there are no interactions in the first matrix, they are independent of each other; thus, the tasks can be done in parallel (or concurrently). In the second matrix, Task B is dependent on information from Task A; therefore, the task sequence should be to complete Task A prior to Task B. Lastly, in the third matrix, Task B depends on Task A. However, Task A also depends on information provided by Task B, which makes them interdependent (coupled), which typically results in iteration.

Product or system development is fundamentally iterative, yet iterations are hidden. Iteration is the repetition of tasks due to the availability of new information. For example:

– Changes in input information (upstream)
– Update of shared assumptions (concurrent)
– Discovery of errors (downstream).

Engineering activities are repeated to improve product quality and/or to reduce cost. To understand and accelerate iterations requires:

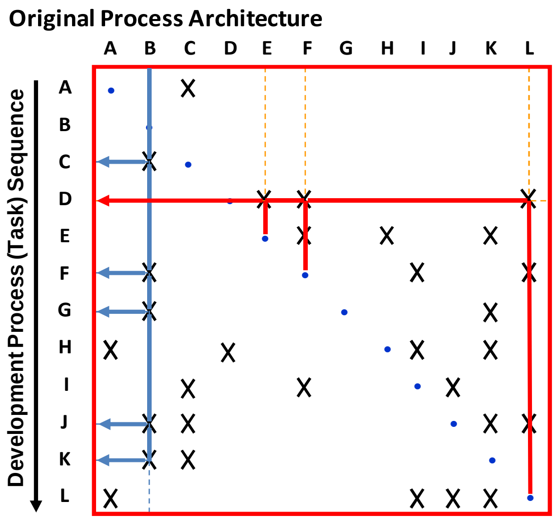
– Visibility of iterative information flows
– Understanding of the inherent process coupling.

A task-based DSM can also be considered an information exchange model. Figure 6 is an example of a task-based DSM; this can represent any project plan in the order of its development sequence

DSM Used for Managing Complex Projects

The DSM is constructed in the following manner:
Each task or process step (in our example, letters) is listed in the order of its development sequence along the side and top. The X's in the matrix represent an information exchange (or interaction) between the tasks, or process steps.

The DSM is read in the following manner: (1) Focus on the vertical blue line aligned with Task B, and follow the arrow to see that Task B transfers information to Tasks C, F, G, J, and K; (2) Focus on the red horizontal line with Task D; Task D requires information from Tasks E, F, and L. Note that information flows are easier to capture than work flows and that inputs (blue lines) are easier to capture than outputs (red lines).


Original Process Architecture
A B C D E F G H I J K L
A B C D E F G H I J K L
Development Process (Task) Sequence
X

Interpretation:
•Task B transfers information to tasks C, F, G, J, and K
•Task D requires information from tasks E, F, and L

Figure 6: Task-Based DSM (Original Sequence)

Figure 7 shows the direction of information flows provided initially in Figure 6 with the upper and lower diagonal sections of the matrix. Information exchange in the upper half of the matrix represents feedback of information that cause process iteration and are sometimes undesirable but sometimes necessary for system optimization. Information exchange in the lower diagonal is desirable, meaning that it should not cause iteration.

Note that each X above the diagonal represents a potential area of iteration and rework resulting in workforce inefficiency. The key is to reorder the tasks such that the number of X's above the diagonal is minimized for risk mitigation.

Note that this convention in some literature shows feedbacks below the triangular and feed-forward of information flows above the diagonal. This is just using the transpose of the matrix. As long a consistent approach is used, the two conventions convey the equivalent information.

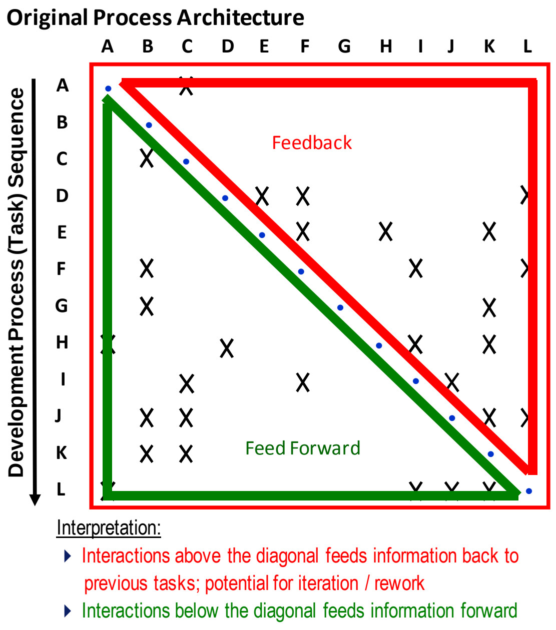

Original Process Architecture
Development Process (Task) Sequence
Feedback
Feed Forward
Interpretation:
Interactions above the diagonal feeds information back to previous tasks; potential for iteration / rework
Interactions below the diagonal feeds information forward

Figure 7: Information Flows

The visualization and analysis depicted in Figures 6 and 7 of the product lifecycle provides insight into the program or system complexity.

Analysis Technique for Process DSMs

Partitioning is the re-sequencing or reordering of the DSM rows and columns such that the new DSM arrangement does not contain any feedback marks in the upper diagonal, thus transforming the DSM into a lower triangular form. For complex engineering systems, it is highly unlikely that simple row and column manipulation will result in a lower triangular form. Therefore, the analyst's objective changes from eliminating the feedback marks to moving them as close as possible to the diagonal (this form of the matrix is known as block triangular).

In Figure 8, the development sequence provided in Figures 6 and 7 was partitioned (or re-sequenced) to minimize the marks in the upper triangle of the matrix. After partitioning—the new development sequence shows an improved order of process steps—also shown are the three different types of process steps previously discussed:

– Sequential tasks (in green) shows Task B must be completed before Task C as information on Task B is required to do Task C.

– Parallel tasks (in blue) have no interaction with each other and can be executed at the same time (e.g., Tasks A and K can be done simultaneously after Task C).

– Coupled (in brown) tasks can be identified uniquely, highlighting iteration and potential rework.

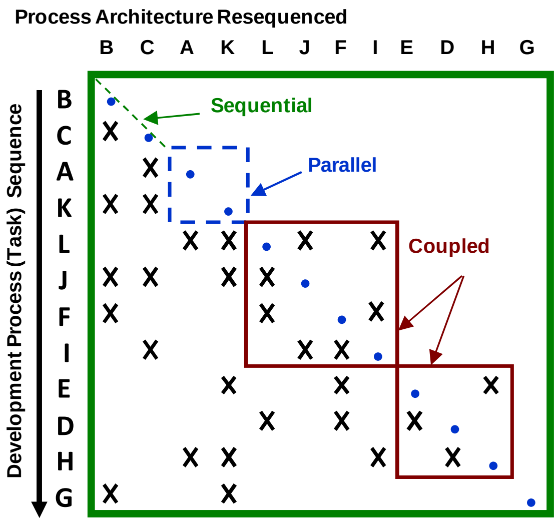

Process Architecture Resequenced
Development Process (Task) Sequence
Sequential
Parallel
Coupled

Figure 8: Partitioned DSM and Sequence Type

Note the large iterative process (or coupled tasks) in the middle of the matrix in Figures 8 and 9. This iterative process can be further reduced to achieve a better process architecture by a method called *tearing* (see Figure 9). Tearing marks in the DSM can break coupled blocks into smaller ones or make them sequential.

Torn marks may become assumptions or controls for the process. Torn marks are usually justified by good assumptions that reduce risk of unwanted iteration. This requires a full understanding of relationships of the interfacing elements. Adding controls in the process is another method to allow tearing of marks.

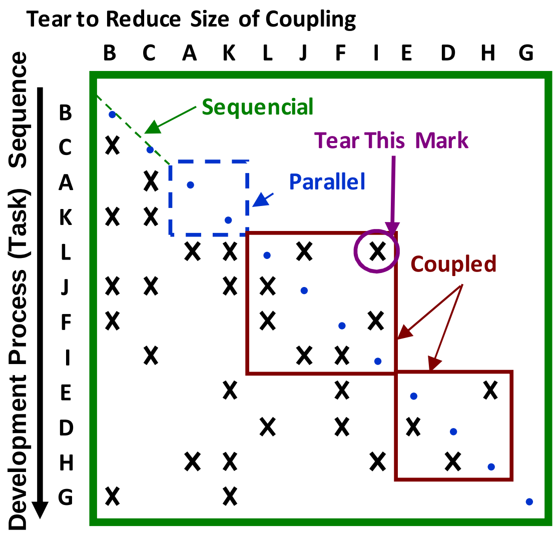

Tear to Reduce Size of Coupling
Development Process (Task) Sequence
Sequencial
Tear This Mark
Parallel
Coupled

Figure 9: Partitioned DSM with Proposed Tearing

Once a mark is torn, the large coupling that existed in Figures 8 and 9 can be represented by two smaller couplings to allow a quicker overall process, as depicted in Figure 10.

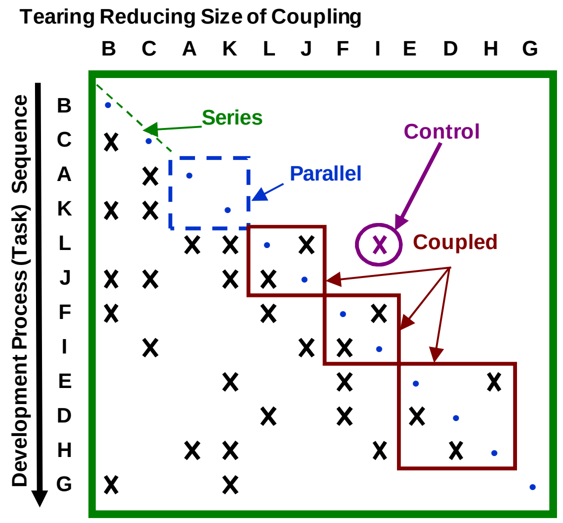

Tearing Reducing Size of Coupling
B C A K L J F I E D H G
Development Process (Task) Sequence
Series
Control
Parallel
Coupled

Figure 10: DSM with Tearing Effects

Process Architecture DSM Approach:

1. Select a process or sub-process to model.
2. Identify the tasks of the process, identify who is responsible, and determine the outputs created.
3. Lay out the square matrix with the tasks in the order they are nominally executed.
4. Ask the process experts what inputs are normally used for each task. Insert marks representing the information inputs to each task.
5. Identify the exceptional (unplanned) process flows and the ways that the process can fail. Include marks representing these unplanned iterations.
6. Analyze the DSM model to re-sequence tasks, suggesting a new process by forcing marks below the matrix diagonal.
7. Draw solid boxes around the coupled tasks representing the task couplings (iterations).
8. Draw dashed boxes around groups of parallel (uncoupled) tasks—opportunities for leaning out process.
9. Highlight the unplanned iterations: risks
10. Further analyze by deep-diving the risks to determine ways to minimize schedule risks.

In summary, the benefits of applying a Process DSM are as follows:

– Visualizing processes or information flows
– Interface management representation
– A means of highlighting iteration and rework
– Enables “leaning” out processes
– Analyzing process cost, schedule, and risks
– A framework for knowledge management.

The next section goes in greater detail regarding the application of static DSMs for both product/system architecture (component-based DSM) and organization (team/people-based DSM) and the analysis technique typically used for managing system complexity.

Product Domain: System Architecture DSM

A definition of System Architecture is “The fundamental organization of a system embodied in its components, their relationships to each other and to the environment, and the principles guiding its design and evolution.” Another definition is, “The arrangement of functional elements into physical modules which become the building blocks for the product of family of products.” [IEEE Std 1471-2000] The modules employ one of more functions. The interaction between modules should be well understood and defined. A modular architecture provides value in its simplicity and reusability for a platform. System integration needs are determined by the chosen decomposition and its resulting architecture. We map the structure of interactions in order to plan for integration.

The system architecture DSM example employs an automobile’s climate control system, as shown in Figure 11.

System Architecture DSM Approach:

1. Decompose the system architecture into its components.
2. Document the interactions between the components using a DSM.
3. Cluster (integrate) the components into “chunks” or subsystem modules.

Clustering is another technique for manipulating a DSM. As seen with partitioning (i.e., re-sequencing) in the task-based DSM, the goal of partitioning was to render the DSM lower triangular as much as possible. The reason was due to the significance of upper-diagonal marks, which represented feedback information flows. This situation arises whenever the matrix elements represent a set of time-based elements. On the other hand, when the DSM elements represent design components (i.e., a component-based DSM), the goal of the matrix manipulation changes significantly from that of partitioning algorithms. The new goal becomes finding subsets of DSM elements (i.e., clusters or modules) that are mutually exclusive or minimally interacting subsets (i.e., clusters as groups of elements that are interconnected among themselves to an important extent while being little connected to the rest of the system). This process is referred to as *clustering.*

In other words, clusters absorb most, if not all, of the interactions internally, and the interactions or links between separate clusters are eliminated or at least minimized.

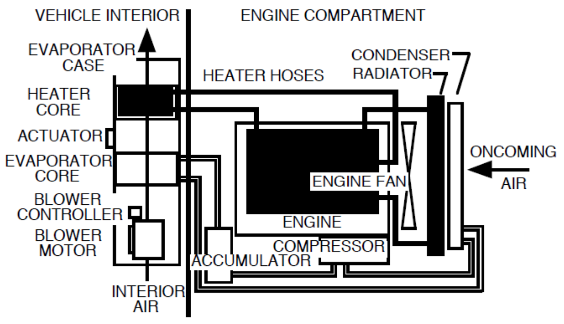
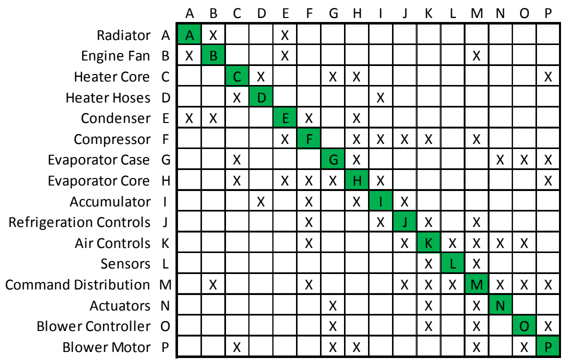


Figure 11: Automotive Climate Control System Component Schematic

Step 2 requires documentation of all component interactions, beginning with a binary DSM.



		A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P
Radiator	A	A	X			X											
Engine Fan	B	X	B			X								X			
Heater Core	C			C	X			X	X								X
Heater Hoses	D			X	D					X							
Condenser	E	X	X			E	X		X								
Compressor	F					X	F		X	X	X	X		X			
Evaporator Case	G			X				G	X						X	X	X
Evaporator Core	H			X		X	X	X	H	X							X
Accumulator	I				X		X		X	I	X						
Refrigeration Controls	J						X			X	J	X		X			
Air Controls	K						X				X	K	X	X	X	X	
Sensors	L											X	L	X			
Command Distribution	M		X				X				X	X	X	M	X	X	X
Actuators	N							X				X		X	N		
Blower Controller	O							X				X		X		O	X
Blower Motor	P			X				X	X					X		X	P

Figure 12: Preliminary Binary DSM

The next step is to classify and quantify the interactions based on type of interaction (i.e., Spatial, Energy, Information, Material) and quantification of each type of interaction (i.e., it could be a scale of -2 {detrimental to system functionality} to 0 {does not affect functionality} up to +2 {required for functionality}).

Analysis of the system architecture identifies functional modules and distributed subsystems. This knowledge can support the formation of your IPTs and plans for system integration. Figure 13 shows the final clustered DSM for our example. When clustering, one objective should be to maximize internal interactions while minimizing external interactions for each module. Another objective when clustering is to consider the size of your subsystem module, such that the modular chunk adds system value.

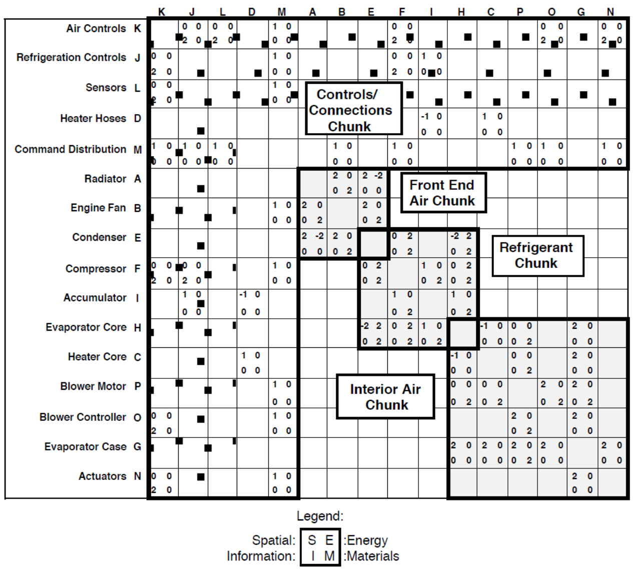


Figure 13: Clustered Interactions Matrix for Automotive Climate Control Example

In summary, the following are insights gained from the application of a System (Product) Architecture DSM:

1) DSM is an effective representation for system components and their relationships.
2) DSM can be analyzed by clustering (integration analysis).
3) Integration analysis:
   a) Can generate alternative views on system architecture.
   b) Help improve architectural understanding.
   c) Facilitates architectural innovation.
4) Architecture DSMs also support the following applications: interface management, functional integration, portfolio segmentation, knowledge capture.

The next section goes in greater detail regarding the application to organizations (team/people-based DSM) and the analysis technique typically used for managing system complexity.

Organization Domain: Organization DSM

Organizations are complex systems. An improved understanding of these complex organizational architectures enables their ability to innovate and continuously improve. Organizational decomposition requires an understanding of the elements and their relationships (interfaces). Relationships among the elements are what give organizations their added value. The greatest leverage in organizational architecting is at the interfaces. Many barriers in system development programs are a result of interfacing problems or the organizations inability to integrate team structures, which leads to either lack of communication or information overload. Another problem complex organizations face is how to design a project or program organization in a way that facilitates and motivates the timely flow of appropriate information and regulates information overload.

Organization Architecture DSM Approach:

1. Decompose the organization into elements or teams with specific functions, roles, or assignments (it is often helpful to map teams to product subsystems, components, etc.).
2. Document the interactions between the elements (teams or people) using a DSM.
3. Cluster (integrate) the elements into higher-level elements (organizational modules).

Example: Automobile Engine Design Organization [7]
This next example will show how DSM can be used to develop organizational team structures for improved communications and systems integration.

This example was an engine development program by GM. Figure 14 shows the original organization breakdown structure for this program.

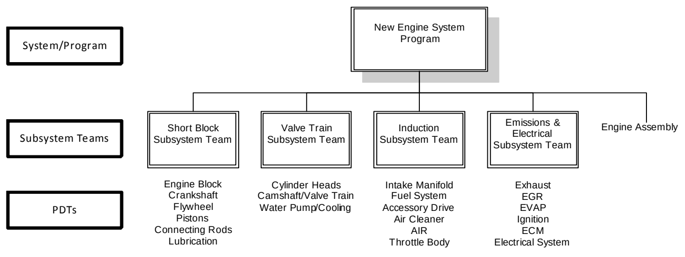


Figure 14: Organization Breakdown Structure

The matrix in Figure 15 shows the original organizational structure and the frequency of interactions of the engine subsystem teams and product development teams (PDT) structures. The subsystem teams are as follows: short block (in red), valve train (in green), induction (in brown), and emissions/electrical (in blue).

Note the legend below the matrix, which describes the frequency of interactions between the system elements; the largest dot depicts daily interactions, the medium size dot depicts weekly interactions, and the smallest dot represents monthly interactions. Also note the number of interactions outside of the team structure and lack of a formal mechanism to ensure communication and integration of the system formally within the current team structures.

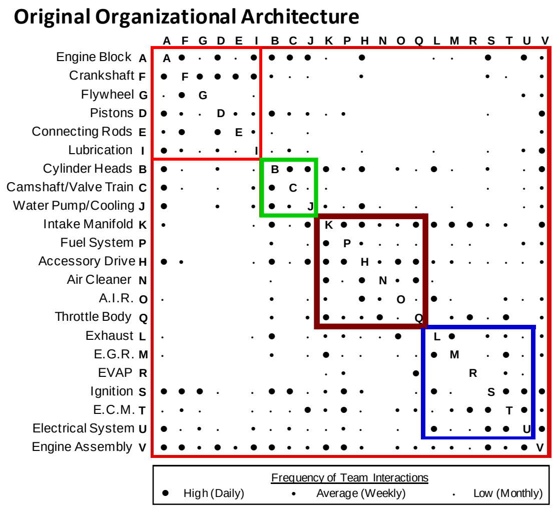


Figure 15: Original Organizational Structure

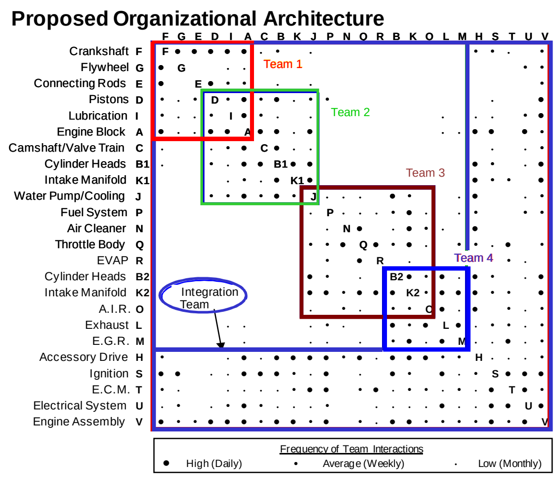


Figure 16: Proposed Organization After Clustering

The matrix in Figure 16 shows the elements rearranged (or "clustered") to minimize interactions outside of the proposed structures. Note that the number of interactions required outside of the team structures has been significantly reduced. Also note the overlapping of teams—requiring that certain team members support multiple teams—and the formation of an integration team.

This proposed organization significantly improved its communication and efficiency and its ability to integrate the engine system with the creation of the integration team.

SUMMARY AND FINAL REMARKS

The DSM methodology supports a major need in engineering design and management of complex systems. The method provides a visually powerful means for capturing, communicating, and organizing engineering design activities and architectural issues such as project team structures and system architecture.

This paper provides an introduction to the DSM method as an alternative approach to classical project management techniques for managing complex systems development. The method is useful by simply building and inspecting the DSM, and even without further analysis, building a DSM model of a project/system improves visibility and understanding of project/system complexity. With the use of a DSM model, one can more easily convey the process to others in a single picture (matrix).

This paper introduced the power of DSM by presenting the application in the three key domains in system design, development, and management. DSMs have been applied in numerous industries including automotive design, aerospace design processes, building construction, microprocessor development, telecom, electronics, and some military applications (e.g., U.S. Air Force and U.S. Navy).

REFERENCES

[1] Browning, T. & Eppinger, S., “Modeling Impacts of Process Architecture on Cost and Schedule Risk in Product Development,” IEEE Trans. on Eng. Mangt, 49(4): 428-442, 2002.

[2] Browning, T. “Applying the Design Structure Matrix to System Decomposition and Integration problems: A Review and New Directions.” IEEE Transactions on Engineering management, Vol. 48, No. 3, August 2001.

[3] Browning, Tyson R., "Use of Dependency Structure Matrices for Product Development Cycle Time Reduction", Proceedings of the Fifth ISPE International Conference on Concurrent Engineering: Research and Applications, Tokyo, Japan, July 15-17, 1998c, pp. 89-96

[4] Carrascosa, Maria, Eppinger, Steven D. and Whitney, Daniel E., "Using the Design Structure Matrix to Estimate Product Development Time", Proceedings of the ASME Design Engineering Technical Conferences Atlanta, GA, Sept. 13-16, 1998.

[5] Eppinger, S., “Innovation at the Speed of Innovation,” Harvard Business Review, Vol. 79, pp. 149-158, 2001.

[6] Eppinger, S.D., Whitney, D.E., Smith, R.P., and Gebala, D.A. “A Model-Based Method for Organizing Tasks in Product Development.” Research in Engineering Design, Vol. 6, pp. 1-13, 1994.

[7] McCord, K.R. and Eppinger, S.D. Managing the Integration Problem in Concurrent Engineering. M.I.T. Sloan School of Management, Cambridge, MA, Working Paper no.3594, 1993.

[8] Pimmler, T.U. and Eppinger, S.D. “Integration Analysis of Product Decompositions.” In Proceedings of the ASME Sixth International Conference on Design Theory and Methodology, Minneapolis, MN, Sept., 1994.

[9] Rinkevich, D. and Samson, F. P., “An Improved Powertrain Attributes Development Process With the Use of Design Structure Matrix”, MIT Master’s Thesis, Cambridge, MA, Feb. 2004

[9] Sharman, D. and Yassine, A., "Characterizing Complex Product Architectures," Systems Engineering, Forthcoming, 2004.

[9] Smith, Robert and Eppinger, Steven, “Identifying Controlling Features of Engineering Design Iteration, Management Science, vol. 43, no. 3, pp. 276-293, March 1997.

[9] Steward, D.V. “The Design Structure System: A Method for Managing the Design of Complex Systems.” IEEE Transactions on Engineering Management, Vol. 28, pp. 71-74, 1981.

[10] Ulrich, K.T. and Eppinger, Steven D., “Product Design and Development,” McGraw-Hill, New York, 1995 (1st edition) and 2000 (2nd edition).

[11] Warfield, John N., "Binary Matrices in System Modeling" IEEE Transactions on Systems, Man, and Cybernetics, vol. 3, pp. 441-449, 1973.

[12] Yassine, A., Falkenburg, D., and Chelst, K. “Engineering Design Management: An Information Structure Approach.” International Journal of Production Research, Vol. 37, no. 19, pp. 2957-2975, 1999.

[13] Yassine, A., Whitney, D., Lavine, J. and Zambito, T., "DO-IT-RIGHT-FIRST-TIME (DRFT) Approach to Design Structure Matrix (DSM) Restructuring", Proceedings of the 12th International Conference on Design Theory and Methodology (DTM 2000), September 10-13, 2000 Baltimore, Maryland, USA.

[14] Yu, T., Yassine, A., and Goldberg, D., "A Genetic Algorithm for Developing Modular Product Architectures," Proceedings from the 2003 ASME International Design Engineering Technical Conference, 15th International Conference on Design Theory & Methodology, September 2-6, 2003. Chicago, Illinois.

[15] Yassine, A., “An Introduction to Modeling and Analyzing Complex Product Development Processes Using the Design Structure Matrix (DSM) Method”, University of Illinois at Urbana-Champain