10

Design Structure Matrix (DSM) Method

This chapter introduces the Design Structure Matrix (DSM) method for project analysis and design.

10.1 DSM Introduction

In Chapter 9 we learned about ways in which projects can be modeled and planned as networks of tasks. The main method in use today is the Critical Path Method (CPM), as well as the Program Evaluation and Review Technique (PERT). PERT assumes probabilistic task durations, while CPM only uses deterministic values.

Table 10.1 reviews the advantages and disadvantages of these methods.

Table 10.1. Advantages and Disadvantages of CPM and PERT

Advantages	**Disadvantages**
Focuses attention on a subset of critical tasks	Doesn’t capture task iterations, in fact
Determine effect of shortening/lengthening tasks	Prohibits iterations = called ’cycle error’
Evaluate costs of a ’crash’ program	
Links task durations to schedule	

One of the main features of projects is that tasks sometimes have to be repeated, either entirely or in part. This is sometimes also referrred to as *rework*. This can be the case because of a variety of reasons. In product and system development projects in particular, it is very frequent that certain analysis steps are repeated multiple times, but the number of such iterations may not be known exactly ahead of time. This is less true in construction projects, but nevertheless it is essential that a method used to analyze and plan projects be able to handle iterations. This is one of the main motivations of the Design Structure Matrix (DSM) method.

Let us consider, once more the very simple CPM chart in Fig. 10.1. As before the critical path is identified, but now a dashed arrow indicates that task B has not only task A as a prerequisite, but also task D. But D also depends on B, so that there is a circular precedence relationship. So, while the CPM and PERT methods are simple to understand and use, they cannot represent coupled/iterative task relationships.

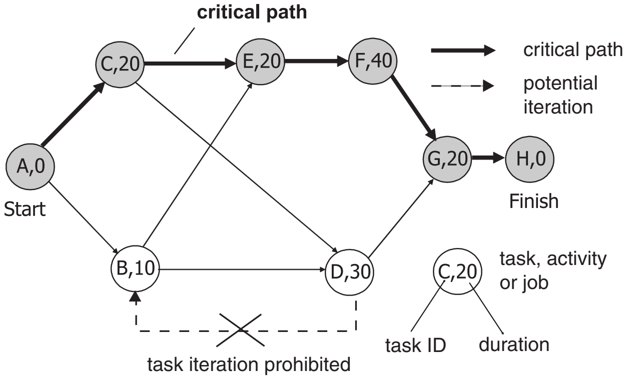


Fig. 10.1. Basic CPM network with prohibited iteration

Thus, there are really three possible sequences for two tasks to consider (Fig. 10.2), beyond what was shown in Fig. 9.1.

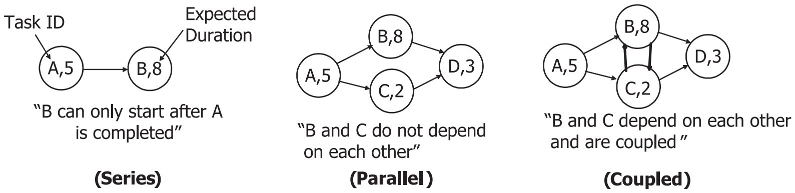

Task ID
Expected Duration
A,5
B,8
"B can only start after A is completed"
(Series)
A,5
B,8
C,2
D,3
"B and C do not depend on each other"
(Parallel)
A,5
B,8
C,2
D,3
"B and C depend on each other and are coupled"
(Coupled)

Fig. 10.2. Basic task dependencies in a project, including coupling (iteration)

There are a number of reasons why a task may have to be repeated or why two tasks may be coupled. There is a subtle distinction between *planned* and *unplanned* reasons for repeating tasks:

- Two tasks may depend on each other for mutual input information
- Two tasks are partially overlapped in time and incomplete information was given from one task to the other, which has to be later updated
- A mistake was made during the execution of a task which impacted subsequent tasks. The offending task has to be repeated to correct the mistake, potentially impacting downstream tasks.

In Systems Engineering, the classical case of coupled tasks is also referred to as a *chicken-and-egg* problem. In other words, two tasks depend on each other and are in an iteration loop. A specific example is given below.

Example: Design of a Spacecraft Attitude Control System (Fig. 10.3). Coupling between spacecraft mass and inertia and the control actuators. In this example the designers of the attitude control system request a value for the total mass and inertia matrix of the spacecraft so that they can size the reaction wheel assembly (RWA) torque and momentum capacity requirement for a given slew rate performance requirement. The reaction wheels themselves, however, are quite heavy, on the order of 20-50 kg/wheel. The weights and balance engineers say that they cannot generate an estimate for mass and inertia of the spacecraft unless they include precisely the mass of the RWA, but in order to size the RWA the mass and inertia of the spacecraft are needed, etc., etc.

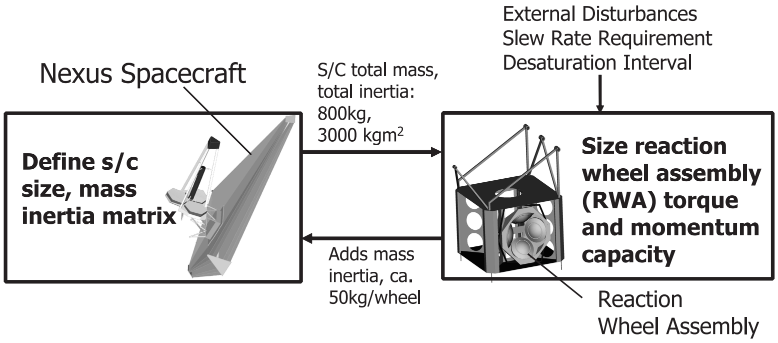


Fig. 10.3. Coupling between spacecraft inertia and control actuators.)

In order to run an effective project, one has to try to identify these iteration loops and manage them effectively. One strategy is to break a loop deliberately at a certain point by introducing an assumption, and then iterating until convergence is achieved. This will be discussed below in more detail.

Reflection Point: Have you experienced 'chicken-and-egg' problems in your engineering work before? What were these problems? How were they resolved?

In this context a number of questions arise:

- How can iterations be represented?
- How do you enter the loop?
- What are the going-in assumptions?
- When do you exit the loop?
- When has design convergence been achieved?
- How many iterations are needed?
- How do the iterations impact ...
  - Project schedule and cost?
  - Product/Design quality?

- How can iterations be planned strategically? (product development process ‘meta-design’)

The Design Structure Method (DSM) is one method to answer such questions.

10.2 DSM Basics

A general definition of DSM is given below:

Definition: A Design Structure Matrix (DSM) is a two-dimensional matrix representation of the structural or functional relationships of objects, variables, tasks or teams

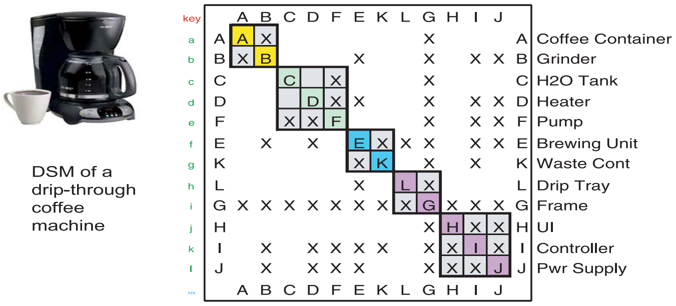
There are a number of synonyms that essentially denote the same concept:

- Design Structure Matrix (DSM)
- N^2-Diagram
- ‘N-squared’ Matrix
- Dependency Structure Matrix
- Adjacency Matrix (graph theory)

There are really five types of DSM’s that are of interest to Systems Engineers: object-based, variable-based, team-based, task-based, and hybrids. A recent review of the history and research into DSMs has been given by (Browning 2004). Examples for each type of DSM are given below.

Object-based DSM

In an object-based DSM, the rows and columns represent physical and/or virtual objects and the entries in the matrix represent structural links or (suppressed) procedural links (see OPM chapter). An example of a simple object-based DSM is shown in Fig. 10.4 along with a photograph and simple object-process diagram.



key		A	B	C	D	F	E	K	L	G	H	I	J		
a	A	A	X							X				A	Coffee Container
b	B	X	B				X			X		X	X	B	Grinder
c	C			C		X				X				C	H2O Tank
d	D				D	X	X			X		X	X	D	Heater
e	F			X	X	F				X		X	X	F	Pump
f	E		X		X		E	X	X	X		X	X	E	Brewing Unit
g	K						X	K		X		X		K	Waste Cont
h	L						X		L	X				L	Drip Tray
i	G	X	X	X	X	X	X	X	X	G	X	X	X	G	Frame
j	H									X	H	X	X	H	UI
k	I		X		X	X	X	X		X	X	I	X	I	Controller
l	J		X		X	X	X			X	X	X	J	J	Pwr Supply
zzz		A	B	C	D	F	E	K	L	G	H	I	J		

Fig. 10.4. Design Structure Matrix, object-based, of a simple drip-through coffee machine

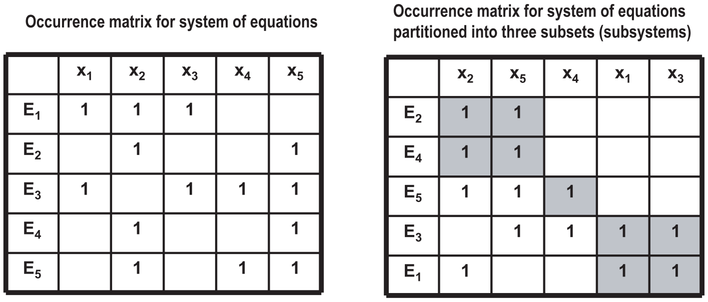
In this example the internal structure of a simple drip-through coffee machine is represented. The rows and columns have been reordered to group tightly coupled components together. Implicitly various functions are carried out by the components: the coffee container and grinder hold and grind the coffee beans, the water tank, heater and pump ensure the flow of water at the right temperature and pressure, the brewing units combines the water and coffee beans and manages the waste. The frame is the most connected component, providing structural support and integrity. The user interface, controller and power supply control the whole machine and distribute the necessary electrical power. The entries in the object-based DSM are marked with an 'X' to indicate when there is a direct relationship between two components. This relationship can be further broken down if necessary. Typically, object-based DSMs are symmetrical when relationships are represented at an aggregate level.

Variable-based DSM

A system of equations (linear or non-linear) expressed in a vector of variables **x** can be represented in matrix form. If the system is linear of the form **Ax-b** = **0**, then the A matrix contains the actual coefficients and contains the structure of the system of equations (Strang 199X). If the system is non-linear, then an incidence or occurrence matrix can be found that indicates what variable participates in what equation. The variable based matrix representation (Steward 1965) is at the origin of the DSM method. An example is given below (Eq. 10.1). Consider the system of equations:

$$\begin{aligned} E_1 &: \; x_1x_2 - 2x_3 + 2 = 0 \\ E_2 &: \; x_2 + 3x_5 - 9 = 0 \\ E_3 &: \; x_1 - x_4x_5 - x_3 + 10 = 0 \\ E_4 &: \; 9x_5 - 3x_2 + 7 = 0 \\ E_5 &: \; x_2x_5 - x_2x_4 + x_2 - 9 = 0 \end{aligned} \tag{10.1}$$

Here there are 5 independent variables and 5 equalities (equations). The original occurrence matrix is shown in Fig. 10.5 (left), showing which variable x_i, participates in which equation, $E_1, E_2, \ldots, E_5$. One may solve this large system all-at once, which requires iterations since the system is nonlinear.



Occurrence matrix for system of equations

	x_1	x_2	x_3	x_4	x_5
E_1	1	1	1		
E_2		1			1
E_3	1		1	1	1
E_4		1			1
E_5		1		1	1

Occurrence matrix for system of equations partitioned into three subsets (subsystems)

	x_2	x_5	x_4	x_1	x_3
E_2	1	1			
E_4	1	1			
E_5	1	1	1		
E_3		1	1	1	1
E_1	1			1	1

Fig. 10.5. Occurrence matrix for systems of equations: (left) unordered, (right) ordered

However, a more clever approach is to reorder the system into three subsets, which are irreducible (i.e. they cannot be decomposed further), see Fig. 10.5 (right). In that case we can solve the first block E_2, E_4 for x_2, x_5, then solve the second block E_5 for x_4 using this information, and finally solve the third block, E_3, E_1 for x_1, x_3 using the previously obtained solutions for the other variables. This solution of smaller decomposed problems speeds up the solution by close to an order of magnitude relative to solving the system all-at-once. The key insight comes from considering the structure of the system in the form of the DSM-occurrence matrix (Fig. 10.5, right). Automatic algorithms for clustering of large system have been developed, such as DeMAID (Rogers 1996).

Team-based DSM

A third popular use of DSMs is to represent organizational structures. Fig. 10.6 shows the structure of an engine development team at General Motors using DSMs (Eppinger 199x). The team structure in so called Integrated Product Teams (IPTs) is shown, along with the importance of the integration team. The DSM was found by interviewing team members in terms of their frequency of interaction (daily, weekly, monthly) and then reordering the people responsible for specific components into tightly coupled teams.

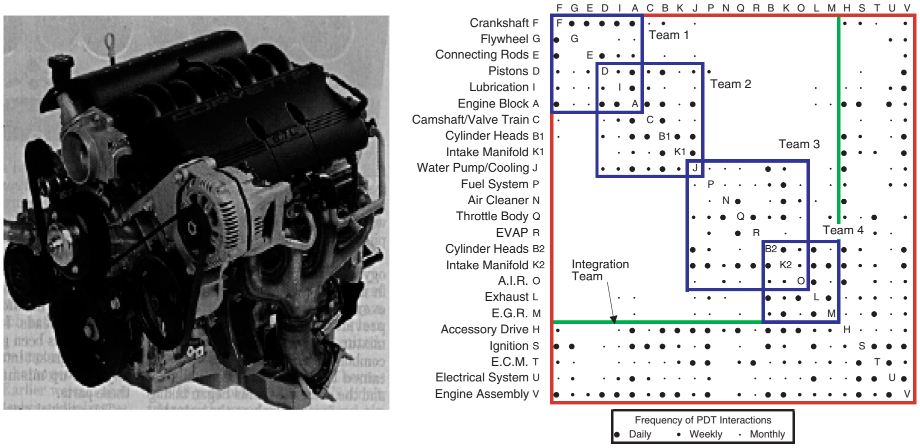


Fig. 10.6. Team-based DSM to represent Integrated Product Teams (IPT) for engine development

Task-based DSM

The form of the DSM that is most useful for modeling and analyzing systems engineering projects, is the task-based DSM. In this form the network of tasks, as depicted in Fig. 9.5 is translated into a matrix format. In order to do this we need to first define two conventions:

- Tasks are essentially information processes that require a finite, known, set of inputs and produce a finite set of outputs
- In the DSM matrix representation, inputs can come from upstream tasks (feedforward) or from downstream tasks (feedback)
- Inputs into a task are shown on the horizontal (rows), while outputs are shown on the vertical (columns) (Fig. 10.7).

The last convention is sometimes reversed, with inputs shown on the vertical. This book, however, uses the more common convention where inputs are rows (Fig. 10.7).

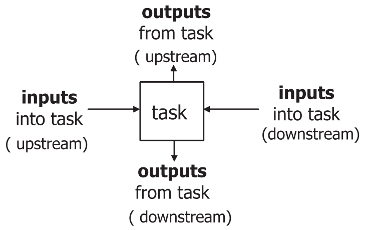
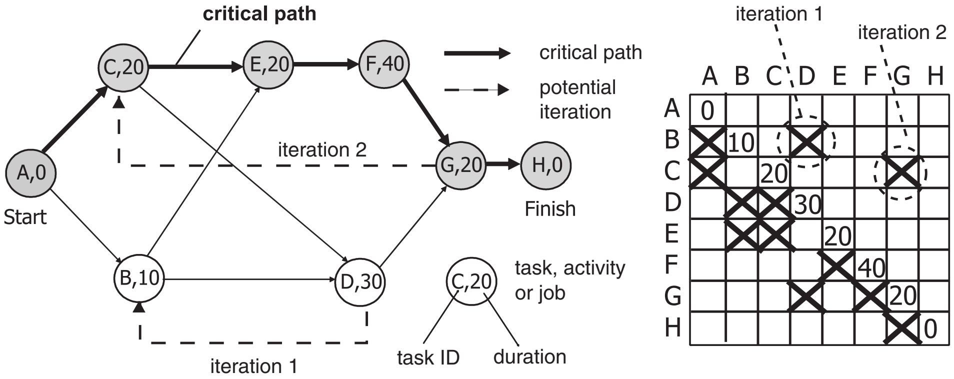


Fig. 10.7. Convention for inputs and outputs of a task-based DSM.

A number of comments are appropriate:

- The task-based DSM models the flow of work products
- In Systems Engineering and Product Development projects the main work product is information
- Therefore task-based DSMs are matrix information flow models
- The information flow is rarely symmetric, i.e. task-based DSMs are usually asymmetric unlike component-based DSMs

A simple CPM graph can now be translated into a DSM by first listing all tasks, A ... H, as rows and columns of a square matrix and then entering a mark 'X' for each arrow in the CPM chart into the appropriate cell of the matrix. It is important to note that coupled relations, e.g. of the form 'A requires information from B requires information from A' can also be represented naturally (Fig. 10.8).

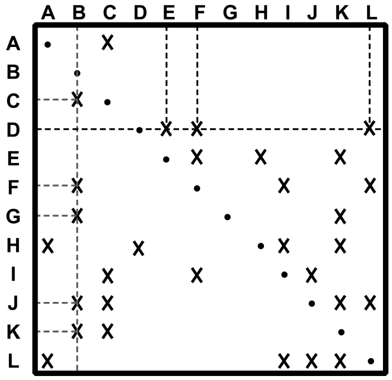


	A	B	C	D	E	F	G	H
A	0							
B	X	10		X				
C	X		20				X	
D		X	X	30				
E		X	X		20			
F					X	40		
G				X		X	20	
H							X	0

Fig. 10.8. Translation of a CPM graph into a task-based DSM

In the figure, we see that all *forward* flow of information is contained in the lower left part of the task-based DSM (below the diagonal), while all *reverse* flow of information is contained in the upper right of the matrix (above the diagonal). Information flows are generally easier to capture than work flows. Also, inputs are easier to capture than outputs.

This can again be seen in Fig. 10.9.



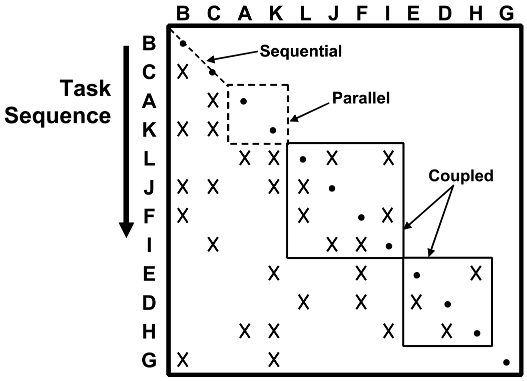
	A	B	C	D	E	F	G	H	I	J	K	L
A	•		X									
B		•										
C		X	•									
D				•	X	X						X
E					•	X		X			X	
F		X				•			X			X
G		X					•				X	
H	X			X				•	X		X	
I			X			X			•	X		
J		X	X							•	X	X
K		X	X								•	
L	X								X	X	X	•

Fig. 10.9. A task-based DSM as information flow model

This can be interpreted as follows:

- Task D requires information from tasks E, F, and L.
- Task B transfers information to tasks C, F, G, J, and K.

The tasks in the DSM (Fig. 10.9) can be reordered into a sequence that reveals structure. We refer to these operations as partitioning or sequencing. Fig. 10.10 shows a partitioned task sequence based the earlier matrix, where tasks were simply listed in alphabetical order.

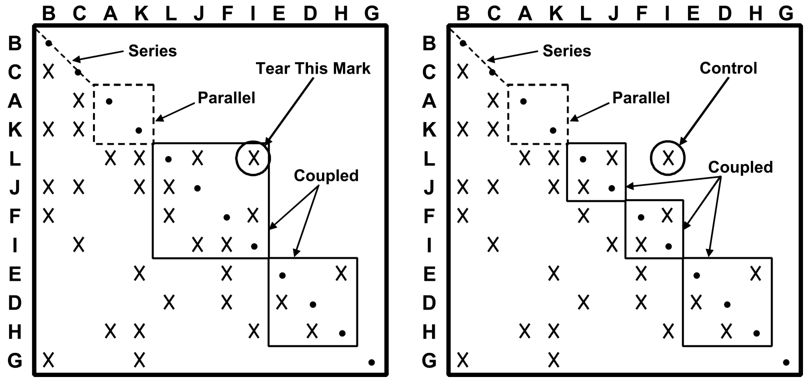


	B	C	A	K	L	J	F	I	E	D	H	G
B	•											
C	X	•										
A		X	•									
K	X	X		•								
L			X	X	•	X		X				
J	X	X		X	X	•						
F	X				X		•	X				
I		X				X	X	•				
E				X			X		•		X	
D					X		X		X	•		
H			X	X				X		X	•	
G	X			X								•

Fig. 10.10. A partitioned, or sequenced DSM reveals structure

We see that tasks B and C are in sequential order, that is, task C requires information from task B for its input. Tasks A and K can be carried out in parallel, since they do not require information from each other. The blocks of tasks, L,J,F,I and E,D,H are coupled within, revealing mutual complex dependencies for information. The sequencing operation of the DSM attempts to minimize the number of marks ‘X’ to those that are irreducible and due to coupling, since some marks can be due purely to the sequence ordering of tasks in the DSM. The display of the matrix can be manipulated to emphasize certain features of the process flow.

For complex projects the DSM can be quite complex, revealing large coupled blocks. These are reflective of the ‘chicken-and-egg’ problems discussed earlier. One way to deal with this is to deliberately break loops, an operation referred to a ‘tearing’. Tearing means temporarily removing a mark, which equates to making assumptions about missing information. There are algorithms to optimally tear a DSM so that the resulting sequenced DSM will be as nearly lower left triangular as possible.



	B	C	A	K	L	J	F	I	E	D	H	G
B	•											
C	X	•										
A		X	•									
K	X	X		•								
L			X	X	•	X		X				
J	X	X		X	X	•						
F	X				X		•	X				
I		X				X	X	•				
E				X			X		•		X	
D					X		X		X	•		
H			X	X				X		X	•	
G	X			X								•

	B	C	A	K	L	J	F	I	E	D	H	G
B	•											
C	X	•										
A		X	•									
K	X	X		•								
L			X	X	•	X		X				
J	X	X		X	X	•						
F	X				X		•	X				
I		X				X	X	•				
E				X			X		•		X	
D					X		X		X	•		
H			X	X				X		X	•	
G	X			X								•

Fig. 10.11. Tear the marks which break the coupled block into smaller ones or make it sequential.

Torn marks may become

- Assumptions
- Feedbacks
- Controls for the process

The presence of coupled blocks in a project makes traditional, purely sequential, engineering ineffective. Concurrent Engineering was discussed earlier (Chapter 6 as one approach to dealing with coupled tasks in practice. We may distinguish between Concurrent Systems Engineering *in the small* and *in the large*:

Concurrent Engineering in the Small:

- Projects are executed by a cross-disciplinary team (5 to 20 people).
- Teams feature high-bandwidth technical communication.
- Tradeoffs are resolved by mutual understanding.
- Design and production issues are considered simultaneously.

On the other hand, some projects exceed this, with hundreds or even thousands of tasks. Concurrent Engineering in the Large:

- Large projects are organized as a network of teams (100 to 1000 people).
- Large projects are decomposed into many smaller projects.
- Large projects may involve development activities dispersed over multiple sites.
- The essential challenge is to integrate the separate pieces into a system or problem solution.

- The needs for integration depend upon the technical interactions among the sub-problems.

The DSM method can be essential in helping organize large scale projects, mainly by visualizing and quantifying the information flows. Fig. 10.12 shows a DSM from an industrial example, the design and manufacture of semiconductors (Intel). This DSM is the result of an extensive industrial project and information flow mapping at Intel (Eppinger 200X).

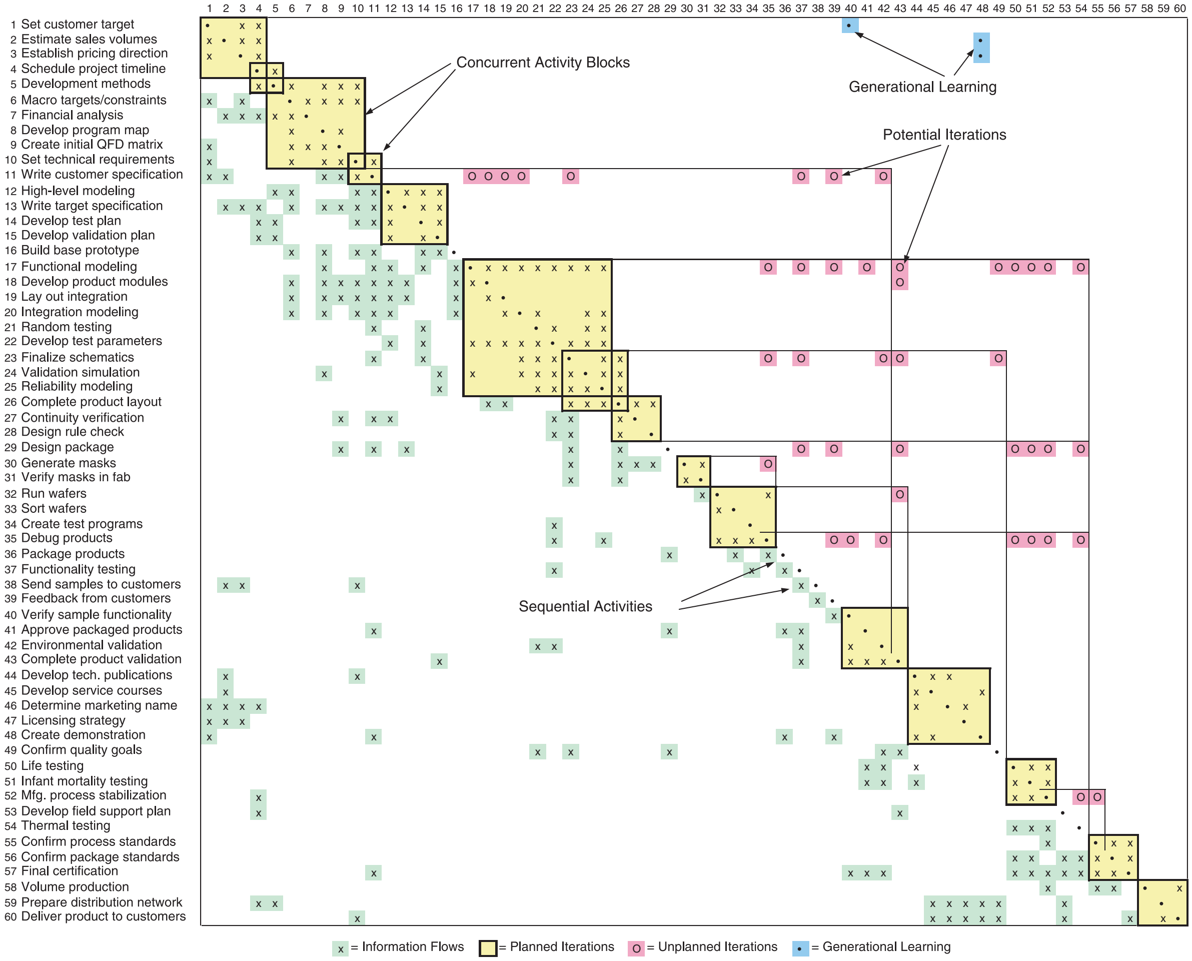


Fig. 10.12. DSM model of semiconductor development project at Intel (Source: Eppinger)

In the example, sequential activities are shown in the lower left below the diagonal. Concurrent activities blocks are shown near the main diagonal. These contain sets of highly tasks that should be executed together, potentially by the same team. The execution of these concurrent activity blocks is characterized by many internal iterations. These iterations are deliberate and *beneficial* and their aim is to converge the results from this set of tasks and to improve the quality of the product.

Above the main diagonal, but *outside* the concurrent activity blocks, there are also potential iterations. However, unlike their counterparts within the concurrent activity blocks, these iterations are *undesired* and *unplanned*. They only occur if mistakes are discovered downstream, which require rework of upstream tasks. The occurrence of rework should be minimized but is a real life fact in large projects. This will be a major topic in the System Dynamics Chapter. A main reason why unplanned iterations are undesired is that they can strongly affect the finish date of a project by requiring rework of tasks that are significantly upstream. An example of this is *task 54: thermal testing* in Fig. 10.12. If a problem is discovered late in the semiconductor development project, e.g. overheating of the circuit, the the following tasks might have to be reworked:

- task 35: product debugging
- task 29: package design
- task 17: functional modeling

This in turn might affect tasks that depend on these and changes might therefore ripple through the project.

A third type of iteration is also indicated in Fig. 10.12 (columns 40,48), indicating generational learning. This is the same as the inter-project learning shown earlier in Fig. 8.3. This captures the information transfer from one project to the next, mainly to avoid the unplanned iterations.

10.3 Creating a DSM model

This section discusses how to create a Task-Based Design Structure Matrix Model. The main steps are as follows:

- Select a project, process or sub-process to model.
- Identify the tasks of the process, who is responsible for each one, and the outputs created by each task.
- Lay out the square matrix with the tasks in the order they are nominally executed.
- Ask the process experts what inputs are used for each task.
- Insert marks representing the information inputs to each task.
- Optional: Analyze the DSM model by re-sequencing the tasks to suggest a new process.
- Draw solid boxes around the coupled tasks representing the planned iterations.
- Draw dashed boxes around groups of parallel (uncoupled) tasks.
- Highlight the unplanned iterations.

Reflection Point: What is really different about the DSM method? ‘In a critical path schedule this simultaneous or iterative situation does not arise or is not explicitly shown because one is dealing only with items to be done. But here we are dealing both with what is to be done and what information is to be determined. Information is needed to determine how a task item it to be done, and information is produced by doing the task. The introduction of information flow raises the possibility, indeed the probability, of being confronted with blocks of more than one item where each item in the block depends on all the other items in the same block. This is usually resolved by making assumptions and iterating to refine the assumptions.’ (Don Steward)

A number of tools for creating, manipulating and visualizing DSMs are available in the part on methods and tools. A few important references are:

- http://www.problematics.com (commercial product PSM32)
- http://www.dsmweb.org/ (this site contains DSM Tutorials, Publications, Examples, Software, Contacts, and Events)

Conclusions are as follows:

- Iterations are an essential part of design
  - Some iterations are desirable
  - They improve quality
  - Some iterations are undesirable
  - These are rework and can cause delay and cost increases
- Differences between CPM/PERT and DSM
  - CPM/PERT is work-flow oriented
  - captures time and schedule flow
  - are useful for planning and tracking detailed execution of project The DSM is information-flow oriented
  - DSM captures iterations
  - DSM shows blocks , i.e. the macro-tasks
  - DSM is useful for analyzing and improving design processes and projects, less for detailed planning of schedules

10.4 Iteration Models

Definition: An iteration is the repetition of tasks due to the availability of new information.

In Fig. 10.12 we saw clearly that we must distinguish between two types of iterations in projects, *planned* and *unplanned* iterations.

- Planned Iterations
  - are caused by needs to 'get it right the first time.'
  - We know where these iterations occur, but not necessarily how much.
  - Planned iterations should be facilitated by good design methods, tools, and coordination.
- Unplanned Iterations
  - are caused by errors and/or unforeseen problems.
  - We generally cannot predict which unplanned iterations will occur.
  - Unplanned iterations should be minimized using risk management methods.

Product development and system project management are fundamentally iterative, yet iterations are hidden. An iteration is the repetition of tasks due to the availability of new information. New information can take the form of changes in input information (upstream), updates of shared assumptions (concurrent), and discovery of errors (downstream). Engineering activities are repeated to improve product or system quality and/or to reduce cost. To understand and accelerate iterations requires visibility of iterative information flows and therefore understanding of the inherent process coupling.

Discussion Point: What is your professional experience with iterations? Can you give technical examples? What are the drivers of iterations (rework, incomplete information)? How are they viewed in the organization? (encouraged, discouraged ...

An example of the difference how firms implement and deal with iterations is shown in Fig. 10.13. Two firms design and manufacture instrument clusters for automobiles. Firm **A** uses a more traditional waterfall approach, and a slower design process with several planned iterations between casing design, wiring layout and lighting details. Firm **A** usually has to do one unplanned iteration. Firm **B** on the other hand uses a process that is more akin to prototyping: fewer planned iterations, a planned revision cycle, usually no unplanned iterations, and extensive use of 'soft' prototypes. Firm **B** is usually faster and can accommodate last minute customer changes with greater ease.

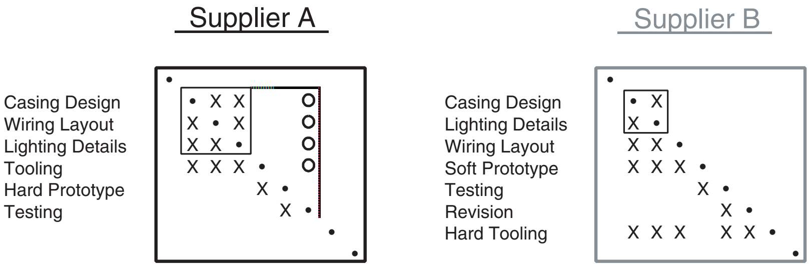


Fig. 10.13. Different use of iteration models between two manufacturers (**A,B** of instrument clusters

Aside from the distinction between planned and unplanned, we can also distinguish two iteration styles: *sequential* and *parallel* (Fig. 10.14:

- Sequential Iteration
  - One activity is executed at a time.
  - Models assume that probabilities determine the next actions.
  - Signal Flow Graph models may be used
- Parallel Iteration
  - Several activities are executed at the same time.
  - Models assume that rework is created for other coupled activities.
  - The Work Transformation Model applies

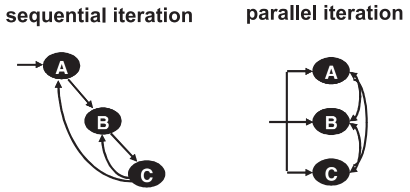


Fig. 10.14. Sequential and parallel iteration models

10.5 Signal Flow Graph Model

In the sequential iteration case, we want to model a project, due to its highly iterative nature to answer the following questions:

- How often is each task carried out on average?
- How long does the project take to complete?
- What are the most effective ways in which the total project time may be shortened?

The signal flow graph method has proven to be an effective way of dealing with such situations. As an example, consider, the die development process shown in Fig. 10.15. Stamping dies are used for a variety of parts fabrication processes, mainly in the automotive industry. Stamping dies are used to make large automotive panels such as the doors, the hood, roof and so forth. Die development is both an art and a science, and typically requires a number of iterations between die design, manufacturability evaluation and surface modeling (Fig. 10.15). This is primarily driven by the complexity and size of parts, the tight tolerances and the need for repeatability and robustness of the stamping process.

The project shown in Fig. 10.15 can be modeled as a signal flow graph, as in Fig. 10.16. Note, that tasks are linked by arrows just as in a CPM graph. However, here once a task is completed, the subsequent task is not exactly known ahead of time, depending on the work result of the just

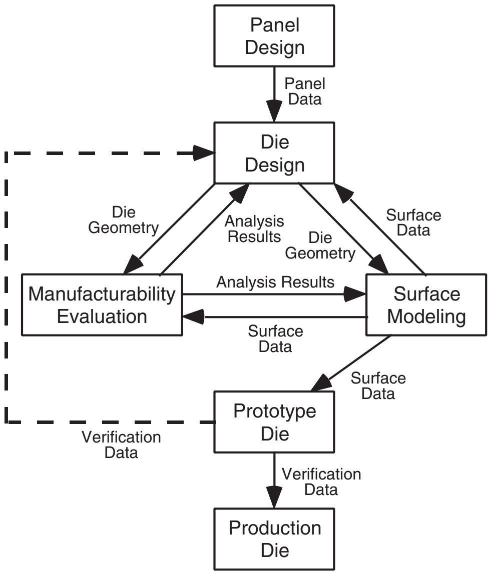


Fig. 10.15. Stamping Die Development Process

completed task. Thus, there is a transition probability from one task to the next, and this can capture the effect of iterations. The probability of transitioning from one task to the next is p, while the task duration is t. For example, once task 7 (initial surface modeling) is completed there is typically a 90% chance that one may proceed with final surface modeling (task 8), however, if a problem is discovered there is a 10% chance that we may have to proceed with (or repeat) task 6: die-design-3.

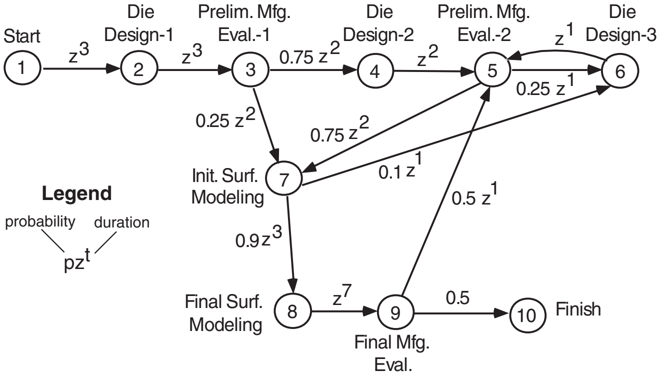


Fig. 10.16. Stamping Die Development Process: Signal Flow Graph model

The transition probabilities and task durations are typically derived from experience with previous projects, or estimated based on expert opinion for entirely new projects. The signal flow graph (Fig. 10.16) can now be used as a basis for a discrete event simulation[1]. The simulation captures the signal flow graph using a state transition probability matrix, **P**, as well as a state transition

[1] Using Matlab or any other simulation tool

duration matrix, **T**. At each transition point, after completion of a task, a uniformly distributed random number is generated, and this is used to determine the next task. This way a large number of probabilistic paths through the signal flow graph model can be generated.

Fig. 10.17 (left) shows the histogram of outcomes for 100 simulations of the project.

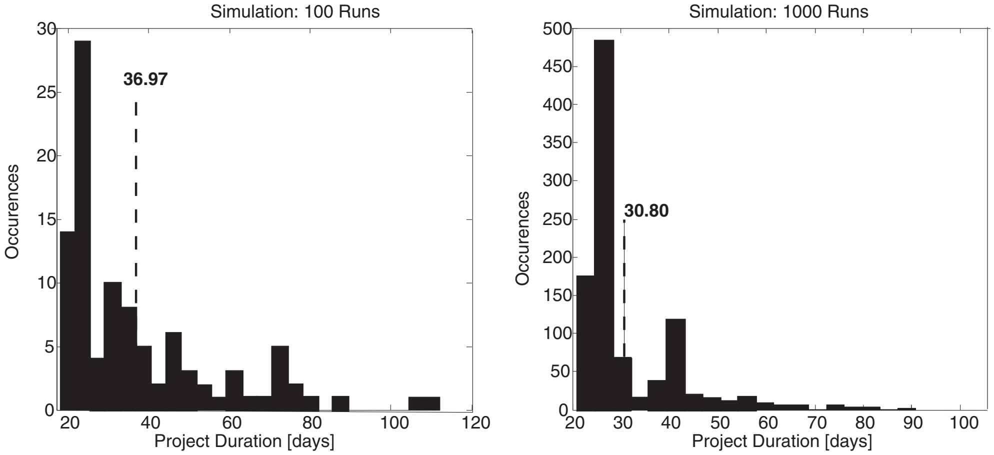


Fig. 10.17. Discrete event simulation of die development project: (left) original model - 100 runs , (right) modified process with more time spent on final surface modeling (task 8) - 1000 runs

We can see that the fastest possible execution of the die development project is 18 days. This corresponds to path: 1-2-3-7-8-9-10. Note that everything has to go right in this path, and in this case the initial die-design (task 2) is immediately implemented. Unfortunately, this is not the *most likely* path. The most likely path is found by taking the most likely transitions after each task. The most likely path is: 1-2-3-4-5-7-8-9-10. This path requires 22 days to complete, and also corresponds to the most likely occurrence in Fig. 10.17 (highest bar in the histogram). Note however, that after task 9, there only is a 50% chance that the project is completed, since there is a 50% chance that one will have to redo task 5, presumably due to errors detected during quality control. This path then requires repeating some tasks that were already done. Thus, even though the most likely path takes 22 days to complete, the mean (average) or expected duration of the project is 36.97≈37 days. This is due to the high likelihood of sequential iterations. What is also noteworthy is that the distribution in Fig. 10.17 (left) is very asymmetric with a long tail. Thus, project durations over 100 days can occur is some cases.

How could the project be changed to decrease the expected (mean) project duration? According to the saying ‘get it right the first time’ we could do the following:

- Spend more time on die design 1 (task 2)
- Increase time spent on Initial Die Design-1 from 3 to 6 days
- Increase the likelihood of going to Initial Surface Modeling (task 7) from 0.25 to 0.75
- Is this worthwhile doing?

We would expect that with this change the likelihood of taking the shortest path is significantly improved. The original project had an expected finish time of E[F]=37 days. After simulating the new project with more emphasis on upfront design (task 2), the new expected finish time is *also E[F]= 37 days*, the change had no measurable effect on the expected completion time.

How can this be?

We attempt a different change:

- Spend more time on final surface modeling (task 8)
- Increase time for that task from 7 to 10 days
- Increase the likelihood of finishing from 0.5 to 0.75
- The new expected Finish is E[F] = 30.5 days
- The three extra days spent on task 8 will save 6.5 days for the project duration overall (on average)

Why is this happening?

The distribution of outcomes for the modified project (with lengthened task 8) is shown in Fig. 10.17 (right). An analysis of actual paths reveals that what drives the average project duration is the instances of projects in the long tail. How can the tail be shortened? Looking at Fig. 10.16 we see that projects that take very long to complete are driven by *late rework.* This is happening when the final die design is not accepted during final manufacturing evaluation (task 9) but is sent back for rework with a 50% probability. Thus, being able to increase the likelihood of acceptance of the work late in the project is the most effective way of shortening project duration.

This is an important and somewhat counterintuitive result. Conventional wisdom would have been to spend more time on design upfront (‘get it right the first time’), but in this particular situation the design is so complex, that some information can only be gained through prototyping, and making and discovering mistakes quickly upfront. Thus here, many fast upfront iterations, combined with a careful ‘polishing’ of the result late in the project appears to be a more successful strategy.

It is in this way that DSM modeling, combined with signal flow graph modeling and discrete event simulation can provide insights that are valuable in understanding the information flow in a project, as well as in helping redesign and streamline projects, sometimes taking counterintuitive steps. Other examples are provided by Isaksson et al. (2000).

10.6 Work Transformation Model

For the second type of iteration model (Fig. 10.14, right), the *parallel* iteration model, we cannot use the signal flow graph model effectively, because it assumes that tasks are performed sequentially. In the parallel iteration model the assumption is that all coupled tasks are being worked on simultaneously.

The Work Transformation Model has been developed to analyze this situation (*add ref*). Fig. 10.18 shows a variable-based DSM with a total of 105 variables involved in the design of an automotive brake system.

The middle block represents a large coupled block of variables that influence each other:

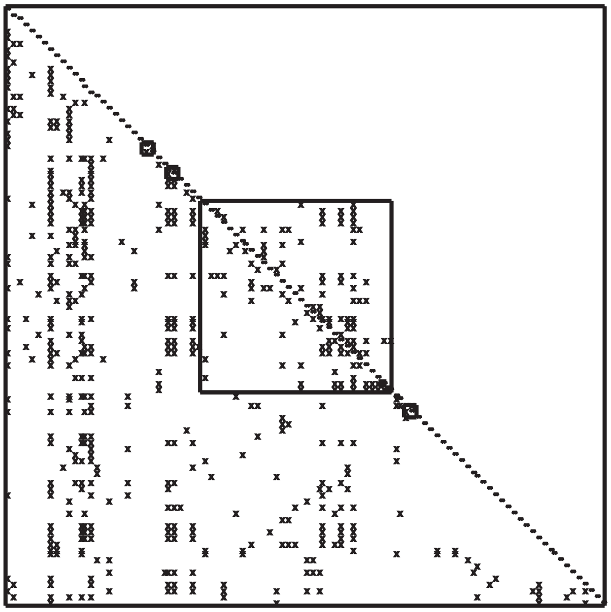


Fig. 10.18. Variable-based DSM with 105 parameters involved in automotive break system design

- Which of these variables are more coupled than others?
- What is the sequence in which these variables need to be determined during design?
- Can we group certain variables together to create work packages, or so classed *design modes*?
- What is the rate of convergence of the design process?
- How many iterations are needed to reach convergence?

To clarify this we first zoom in on the large coupled block in Fig. 10.18. This is shown in Fig. 10.19, with a block of 28, coupled variables. Each variable is called out in a row with a unique ID number and variable name. Interactions between variables are classified as 'none' (empty cell), 'weak', 'medium' and 'strong' based on expert opinion.

The work transformation model assumes that the *amount* of work to be done on a task at a particular instance in time is u_t, and that it is reduced in one time step into an updated work vector u_{t+1} when multiplied by the work transformation matrix, A. Eventually, as t gets very large the work vector u_t should approach zero as the design converges.

$$u_{t+1} = Au_{t+1} \tag{10.2}$$

A number of assumptions underlie this model:

- All coupled tasks are attempted simultaneously.
- Off-diagonal elements of A correspond to fractions of each task's work which must be repeated during subsequent iterations.
- The objective of the work transformation model is to characterize the nature of design iterations.

		33	34	35	37	40	44	45	46	48	49	50	51	52	53	54	55	56	57	58	59	60	61	62	63	64	65	66	04
33	Knuckle envelope & attach pts	■																•									•		
34	Pressure at rear wheel lock up		■	x																		X			•		•		
35	Brake torque vs. skidpoint			■	x																	x			x		x		
37	Line pressure vs. brake torque				■																	•			x		•		
40	Splash shield geometry—front	•				■		•				•			x	x											•	x	
44	Drum envelope & attach pts	•					■																						
45	Bearing envelope & attach pts	•						■										•									•		
46	Splash shield geometry—rear	•					•		■			•			•														
48	Air flow under car/wheel space					x			x	■		x																	
49	Wheel material										■	x																	
50	Wheel design									•		■			•														
51	Tire type/material										•		■	•															
52	Vehicle deceleration rate		X	X	X									■								•			x		•		
53	Temperature at components									x					■							•			x				•
54	Rotor cooling coeficient									x		x	•			■		x									x		
55	Lining—rear vol and area				x										•		■					x							
56	Rotor width									x						x		■									x	x	•
57	Pedal attach pts																		■	x	•								
58	Dash deflection																		x	■	x								
59	Pedal force (required)																			x	■	•	x		X	x	•		
60	Lining material—rear																•					■	•			x	•		
61	Pedal mechanical advantage																				x		■			x	•		
62	Lining—front vol & swept area				x										x									■	•				
63	Lining material—front																						x	•	■	X	x		x
64	Booster reaction ratio																					•	x		x	■	•		
65	Rotor diameter																					•	•		•	•	■	x	•
66	Rotor envelope & attach pts	x																										■	
104	Rotor material														x			•									•	•	■

• Weak
x Medium
X Strong

Fig. 10.19. Large coupled block of 28 parameters in automotive break system design

The mathematics behind the work transformation model are as follows. Eq. 10.2 is called the work transformation equation. The total work vector, U, captures the total amount of work done on all tasks over time:

$$U = \sum_{t=0}^{\infty} u_t = (\sum_{t=0}^{\infty} A^t) u_0 \tag{10.3}$$

Insight into the dynamics of the concurrent iterations can be gained by doing an eigenvalue decomposition of the work transformation matrix, A:

$$A = S \Lambda S^{-1} \tag{10.4}$$

We can then substitute the result into Eq. 10.3.

$$\mathrm{U} = \mathrm{S} \left(\sum_{\mathrm{t}=0}^{\infty} \Lambda^{\mathrm{t}} \right) S^{-1} u_0 \tag{10.5}$$

whereby Λ is the diagonal matrix of eigenvalues. Based on the infinite series

$$\sum_{k=0}^{\infty} aq^k = a + aq + aq^2 + \ldots = \frac{a}{1-q} \text{ for } |q| < 1 \tag{10.6}$$

we can write

$$\left(\sum_{\mathrm{t}=0}^{\infty} \Lambda^{\mathrm{t}}\right) = (I - \Lambda)^{-1} \tag{10.7}$$

This allows rewriting the total work equation as follows:

$$\underbrace{U}_{\substack{\text{total}\\\text{work}}} = \underbrace{S}_{\substack{\text{eigenvector}\\\text{matrix}}} \underbrace{\left[(I - \Lambda)^{-1} S^{-1} u_0\right]}_{\text{scalingvector}} \tag{10.8}$$

This tells us that *total work is a scaling of the eigenvectors* of the work transformation matrix. Each of the eigenvectors of A, are columns in the eigenvector matrix S and represent a particular *design mode*. The eigenvalues λ_i corresponding to the brake system modes tell us about the rate of convergence of a particular design mode, i.e. the rate of convergence scales with λ. Fig. 10.20 shows the rate of convergence of the 28 design modes of the brake system (Fig. 10.19).

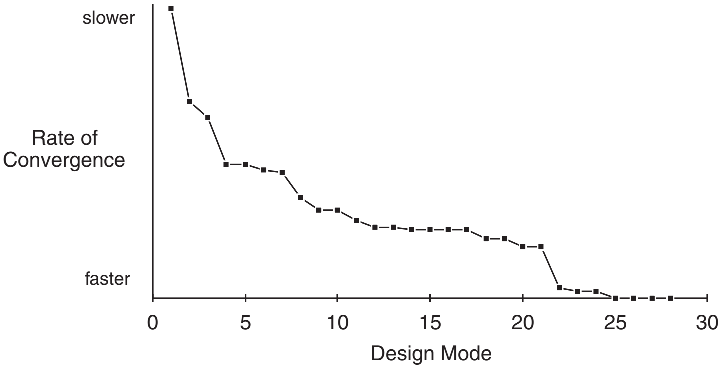


Fig. 10.20. Rate of convergence (eigenvalues) of 28 brake system design modes

We can see that the first 2-3 design modes converge much more slowly than the others. They are the ones that drive the total time required to converge the brake system design project. Consequently particular attention needs to be paid to these design modes (= sets of coupled tasks). Also, understanding what tasks (or variables) have large eigenvector entries in each mode, helps understand what variables are related to each other, what the nature of each design mode is and organizational measures might be taken to promote fast convergence of those modes.

The two dominant (slow) modes (Fig. 10.21) in the brake system design problem are the *stopping performance design mode* as well as the *thermal design mode.* The goal of the first mode is to meet the stopping requirements for the brake system which ensure that the vehicle will decelerate and stop within a specified distance. The second mode ensures that the breaks don't overheat and fail. Both, designing for stopping performance and thermal criteria are highly iterative tasks.

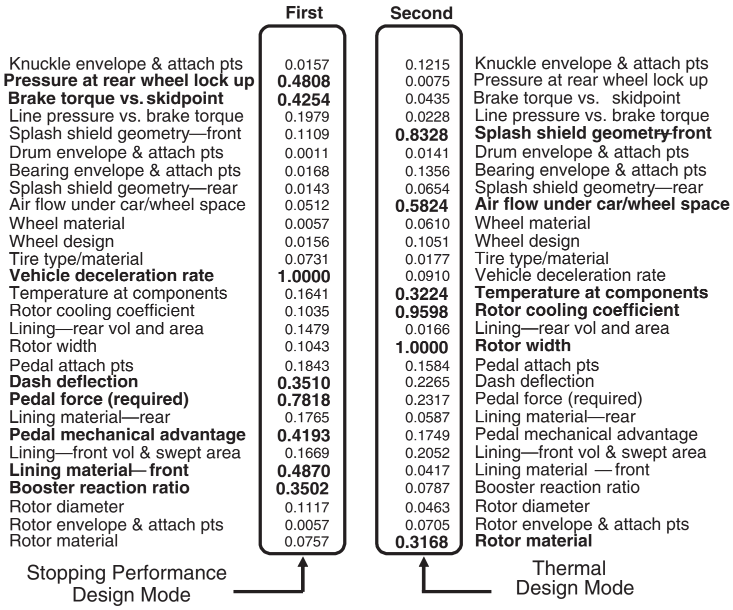

First
Second
Knuckle envelope & attach pts
0.0157
0.1215
Knuckle envelope & attach pts
Pressure at rear wheel lock up
0.4808
0.0075
Pressure at rear wheel lock up
Brake torque vs. skidpoint
0.4254
0.0435
Brake torque vs. skidpoint
Line pressure vs. brake torque
0.1979
0.0228
Line pressure vs. brake torque
Splash shield geometry—front
0.1109
0.8328
Splash shield geometry—front
Drum envelope & attach pts
0.0011
0.0141
Drum envelope & attach pts
Bearing envelope & attach pts
0.0168
0.1356
Bearing envelope & attach pts
Splash shield geometry—rear
0.0143
0.0654
Splash shield geometry—rear
Air flow under car/wheel space
0.0512
0.5824
Air flow under car/wheel space
Wheel material
0.0057
0.0610
Wheel material
Wheel design
0.0156
0.1051
Wheel design
Tire type/material
0.0731
0.0177
Tire type/material
Vehicle deceleration rate
1.0000
0.0910
Vehicle deceleration rate
Temperature at components
0.1641
0.3224
Temperature at components
Rotor cooling coefficient
0.1035
0.9598
Rotor cooling coefficient
Lining—rear vol and area
0.1479
0.0166
Lining—rear vol and area
Rotor width
0.1043
1.0000
Rotor width
Pedal attach pts
0.1843
0.1584
Pedal attach pts
Dash deflection
0.3510
0.2265
Dash deflection
Pedal force (required)
0.7818
0.2317
Pedal force (required)
Lining material—rear
0.1765
0.0587
Lining material—rear
Pedal mechanical advantage
0.4193
0.1749
Pedal mechanical advantage
Lining—front vol & swept area
0.1669
0.2052
Lining—front vol & swept area
Lining material—front
0.4870
0.0417
Lining material —front
Booster reaction ratio
0.3502
0.0787
Booster reaction ratio
Rotor diameter
0.1117
0.0463
Rotor diameter
Rotor envelope & attach pts
0.0057
0.0705
Rotor envelope & attach pts
Rotor material
0.0757
0.3168
Rotor material
Stopping Performance Design Mode
Thermal Design Mode

Fig. 10.21. First and second design modes in brake system development project

10.7 Summary

The DSM method is useful for analyzing projects in terms of their (actual or desired) information flow. The main advantage of DSM over CPM and PERT is that it allows to model and understand the effect of iterations:

- System and product development is inherently iterative.
- An understanding of the coupling of tasks is essential.
- Not everything should be concurrent in concurrent engineering.
- Deliberate (planned) iterations result in improved quality.
- Convergence in a project can be accelerated through:
  - information technology (faster iterations)
  - coordination techniques (faster iterations)
  - decreased coupling (fewer iterations)
- There are two fundamental types of iteration:
  - planned iterations (getting it right the first time)
  - unplanned iterations (fixing it when its not right)
- Mature processes have more planned and fewer unplanned iterations.

DSMs can also be used as a starting point for project simulations. DSM Cost and Schedule simulations allow to more effectively manage cost and schedule risk in projects (Cho 2001, Browning 2001):

- Start with a task-based project network model using DSM representation
- The following effects can be modeled:
  - Probabilistic iteration and rework
  - Learning effects
  - Stochastic task durations and costs
  - Resource constraints
  - Overlapping tasks
  - Risk-based control of parallel rework
- Discrete-event Monte-Carlo simulation
- Latin Hyper Cube sampling
- Simulation outputs:
  - Cost and schedule outcome distributions
  - Budget and deadline risk factors

Discussion Point: Schedule and Cost Risk: How is it handled in your project experience? Is the probabilistic nature of the project acknowledged by management? What can be done to affect schedule and cost risk?