HBR CASE STUDY
Who Goes, Who Stays?
David A. Light

FIRST PERSON
The Ultimate Creativity Machine: How BMW Turns Art into Profit
Chris Bangle

HBR AT LARGE
Tocqueville Revisited: The Meaning of American Prosperity
Charles Handy

Level 5 Leadership: The Triumph of Humility and Fierce Resolve
Jim Collins

Where Value Lives in a Networked World
Mohanbir Sawhney and Deval Parikh

The Ten Deadly Mistakes of Wanna-Dots
Rosabeth Moss Kanter

Strategy as Simple Rules
Kathleen M. Eisenhardt and Donald N. Sull

The Making of a Corporate Athlete
Jim Loehr and Tony Schwartz

HBR INTERVIEW
Managing for the Next Big Thing: EMC’s Michael Ruettgers
Paul Hemp

BEST PRACTICE
Getting 360-Degree Feedback Right
Maury A. Peiperl

TOOL KIT
Innovation at the Speed of Information
Steven D. Eppinger

DIFFERENT VOICE
What Is Science Good For? A Conversation with Richard Dawkins

BOOKS IN REVIEW
Should Strategy Makers Become Dream Weavers?
John Stopford
Developing a new product involves trial and error, but beyond a certain point, redesign becomes wasteful. A practical and proven tool, the Design Structure Matrix, can help streamline the way a company innovates.

The exchange of information is the lifeblood of product development. When an electronics company's circuit designers know what the casing designers are doing, they design a better-fitting circuit for the casing. And when the casing designers know what the circuit designers need, they design a casing where it's easier to put in a better circuit. Such flows of information allow for experimentation and innovation, and for that reason, many companies encourage feedback and iteration in their product development processes. This practice is known as concurrent engineering.

But excessive iteration can have drawbacks. A continual back-and-forth of work inevitably consumes time and resources. And many of the iterations may turn out to be only marginally beneficial or even wasteful. For example, at a telecommunications company that my colleagues and I advised, there were as
many as 20 regular iterations among the teams working on two key technical specifications. The result was that few people worked carefully on specifications in the early stages, because everyone knew that extensive rework would occur down the line.

The lesson is clear. Iteration must be carefully planned and managed. Good iteration should be encouraged and bad iteration eliminated. To do that, managers need representational tools to help them identify and model iteration.

The Design Structure Matrix differs from conventional project-management tools in that it focuses on representing information flows rather than work flows. Thus, it is better able to depict the key dynamic of innovation processes.

Unfortunately, the standard project-management tool kits such as Microsoft Project don’t contain such tools. The most commonly used project-planning tools—principally PERT and CPM network diagrams—are graphic descriptions of task flows. In them, tasks are usually represented by boxes or circles, and sequencing is represented by arrows. In a complex project, a chart can run to tens or even hundreds of pages, and each page accommodates only so many readable circles and arrows. A boxes-and-arrows depiction of the design process for a car’s suspension, for example, would run to more than 30 pages. If the project is made up of tasks that can be completed in sequence without fear of rework, manageable charts for small chunks of work can be generated.

But the tools become very hard to use if what happens on page 60 forces you to rework a task on page 18, and, thus, many of the subsequent tasks down to page 60 again. It is almost impossible for managers to comprehend, let alone represent the information flows of a project rather than the work flows. As a result, it is better able to depict the key dynamic of innovation processes such as product development. What’s more, it can often provide representations of complex development processes on a single page. In this article, we will explain how the DSM works and show how a manager can use it to make development processes more efficient.

First, though, let’s explore why product development needs a fundamentally different planning tool.

**Product Development Is About Information, Not Tasks**

PERT charts, Gantt charts, and other common scheduling tools were created to help managers plan large construction projects such as building ships or factories. Although these projects can be complex, involving hundreds of different tasks or more, the planning principles are fairly simple: you decide where and when tasks should be carried out. No matter how complicated, all construction projects can be thought of as a sequence of discrete tasks, many of which can be conducted simultaneously but none of which should need reworking as a result of later information.

Imagine you are building a house. Some tasks have to be completed in sequence. You can’t frame the walls until you’ve built the foundation. You can’t put on the roof until you’ve built the walls. Sequential tasks, by definition, rely on information generated by earlier tasks. Other tasks, called parallel tasks, can be carried out simultaneously. You can install windows, run wires, and lay plumbing at the same time. These three jobs need walls, but none needs the other two. Neither sequential nor parallel tasks will need to be reworked as a result of subsequent tasks. You don’t change the foundation after building the walls. You don’t rewire as a result of the window glazing.

Product development is very different. It requires innovation, and innovation requires complex learning (feedback) loops. You repeat prior tasks as you learn from subsequent ones. Interdependent tasks that benefit each other in this way are known as coupled tasks. If you want to come up with a better foundation for future houses, you might rework a foundation after building the walls. The information from such an iteration is precisely what helps you find the improvement, and the presence of coupled tasks is what distinguishes innovation processes such as product development from construction projects.

Conventional tools answer the question, “What other tasks must be completed before I begin this one?” But the planners of a product development process need a tool that answers a very different question: “What information do I need from other tasks before I can complete this one?” This is the question the Design Structure Matrix addresses.

**Drawing the DSM**

Constructing a DSM of your company’s existing product-development process is a relatively straightforward, if sometimes time-consuming, process. The first step, identifying the tasks involved, is easy and is often available as part of the project-management documentation. Companies with an established development process already know the tasks needed to develop a new product. Ford,
What the DSM Can Tell You

A DSM of your development process provides a useful reality check. First, it clearly reveals which information exchanges involve design iteration and which do not. In the example just presented, note the diagonal formed by the dots that divide our matrix. All the X’s below the diagonal denote feedback information exchanges in which information from earlier tasks is available for later tasks. But an X in the upper half denotes feedback in which information from a subsequent task may force a reworking of a prior task. These are the coupled tasks. Task B, for instance, needs information from task G, which is carried out long after B. Executing B requires making a guess about or assuming the missing information from G. When complete and accurate information from task G is finally available, a reworking of task B may be necessary. Then the development process has to begin again from B onward, with intervening tasks also being repeated to reflect the change to B’s output.

A DSM can also help you to see how well your development process is anticipating the need for rework. Here’s how. On the DSM, you simply draw boxes around the tasks that your company performs concurrently, that is, interdependently. These are your company’s planned iterations, the tasks your company recognizes as repeating and therefore organizes so as to facilitate and speed the flow of information among them. If all the X’s above the diagonal are captured within the boxes, your organization has planned for these iterations and has arranged its process to accommodate them as efficiently as possible. Our second example, below, shows that tasks E through I are done concurrently. However, the company has failed to prepare for a fair number of potential iterations: there are still four feedback marks (now represented by O’s) above the diagonal and outside of the boxed tasks. These are unplanned iterations.

In practice, of course, successful product developers are good at recognizing potential iterations, and their DSMs will usually reveal a fairly efficient flow of information. But some companies find that this exercise reveals muddled processes. The telecommunications company whose development process for data services is depicted in the chart “Chaotic Development in Telecommunications” is a case in point. The company’s engineers had long been frustrated by the time and energy it took to develop new services, but it wasn’t until they saw a DSM that they realized just how many feedback loops were built into their process.

Optimizing Information Flows

The DSM is a powerful resource for organizations like the telecom company because it helps managers not only identify problems but also see how to fix them. Below, we examine four ways to improve a company’s information flows.

Rearrange the sequence of tasks. The first step in streamlining a product development process is to determine whether a different sequence of tasks will reduce the number of feedback marks. This involves rearranging the rows of the DSM, a process that Boeing executives call “eliminating out-of-sequence rework.” The objective is to move as many X’s as possible from above the diagonal to below it.

You begin by identifying candidates for the earliest and the latest tasks. Ideally, the first task would require no inputs at all, indicating that it would never...
Chaotic Development in Telecommunications

Drawing a DSM of the development process for this company's data services immediately revealed the degree of unplanned iteration. Iterations within the phases were built into the process (the shaded boxes), but unplanned iterations across the phases (the O's) were the reality.
Here are three ways to reduce the need to exchange information. First, you can transfer key knowledge between teams. In some cases, a company can decouple one task from another simply by adding to each task someone with expertise in the other task. These people must be sufficiently knowledgeable to supply information that would otherwise have been exchanged in one or more iterations between teams.

A similar effect can be achieved by using information technology. Computer-aided engineering software, for example, allows design teams to predict the implications of their designs on each other, again obviating the need for a direct exchange of information. A plastic-parts designer at a manufacturer of cell phones can use mold-flow simulation software to foresee production problems arising from her design. In this way, she can anticipate feedback from the mold-tooling experts that would otherwise force her to go back to the drawing board weeks later.

Second, you can introduce a new task earlier in the process so as to simplify subsequent, time-intensive iterations performed by interdependent teams. The new task typically requires early agreement about aspects common to the coupled tasks. The addition of a new task, carried out by representatives of the coupled teams, can break down some of the iterations. For example, two teams designing coupled parts—say, an electronic control circuit and its user-interface keypad—can agree in advance on the locations of the attachment points and microswitches. This interface specification allows the circuit-design team and the keypad team to proceed in parallel.

Third, you can redefine tasks within coupled groups. Another way to reduce iterations is to eliminate an information exchange within a group by adding an extra task or two to intervene between existing tasks in the group. To see how this can be done, it’s instructive to compare how two suppliers design the same dashboard component, a speedometer cluster for General Motors. (See the chart “Reducing Information Exchange by Decoupling Tasks.”)

A coupled process encourages iterations and the search for creative solutions. But sometimes speed is more important than innovation.

Supplier A adopts a concurrent engineering process, carrying out three tasks (casing design, wiring layout, and lighting details) at once, with three task teams working in close proximity. The three teams go through a number of iterations and take a relatively long time to finalize their plans, but the end result is a fairly advanced prototype. However, if the testing phase reveals problems, up to four steps of the process can be affected, and much of the process may have to be repeated (see O’s in the left side of the exhibit).

Supplier B takes a different approach, reducing the coupled tasks to just two: casing design and lighting details. A wiring circuit is worked out based on output from the first two tasks, and the speedometer is then crudely prototyped. Once the prototype has been tested, wiring revisions are made to produce the final design. Although it involves an extra task—revising the wiring—in fact, this process is faster than the first because there is much less iteration between two task teams than among three. The extra step also anticipates and allows for changes to the pro-

Reducing Information Exchange by Decoupling Tasks

Supplier A has three teams that carry out their tasks concurrently. After a number of iterations, a fairly advanced prototype is developed and tested. But if problems arise during the testing phase, up to four steps of the engineering process can be affected (see O’s).

Supplier B couples only two tasks and creates a wiring circuit based on output from them. A soft prototype is built and tested, and an additional task of wiring revision is introduced. However, this process is faster than Supplier A’s because less iteration occurs between the two initial task teams, and the extra step of planned iteration virtually eliminates unplanned iteration.

Supplier A
Highly innovative, but slow

Supplier B
Fast, but less innovative

Reducing Information Exchange by Decoupling Tasks
Manage unplannable rework. So far, most of our discussion has focused on relatively small and easily managed processes. In our theoretical example, for instance, it was possible to optimize a process so that all of the X’s were moved close to or below the diagonal. All coupled tasks could be carried out concurrently, so all iterations could be planned for.

In real life, however, product development is a large and complicated process. It is rare that a company will be able to design a process in which all coupled tasks can be carried out together. There will generally be tasks that can only be conducted later in the process but that provide information for tasks completed months earlier.

Consider Intel, one of the most consistently successful product developers and a company that regularly leads the way in developing the next generation of microprocessors. The product development process for semiconductor chips at Intel consists of 60 distinct tasks, and the DSM is shown in the chart “Semi-conductor Development at Intel.” Above the diagonal, the X’s denote planned information exchanges, and the O’s signify unplanned iterations.

As the boxed areas on the DSM reveal, Intel groups most of the tasks into 15 concurrently managed stages. Some of the stages overlap, that is, some of the tasks in one group are also included in the next. As you might expect, there is little incentive for a highly successful company like Intel to improve its tried, trusted process through rearranging and reorganizing its tasks. Equally, there is

Improving Communication at General Motors’ Powertrain Division

Within the Powertrain Division of General Motors, we used an interesting variant of the DSM to improve communication during engine development. The company had organized its 22 engine product-development teams (PDTs) into four system teams, each dealing with one of the four engine subsystems: the short block, the valve train, the induction system, and the emissions and electrical system. We suspected, however, that this organization did not adequately accommodate the communication needs of all PDTs.

To check out our hypothesis, we surveyed each of the 22 PDTs about its communication with other teams, asking each to describe how often it needed to meet with other teams. From these data, we drew a DSM to describe those communications. Instead of marking each interaction with an X, we used three symbols to reflect the varying frequency of team interactions: a large circle for daily communication, a midsize circle for weekly meetings, and a small dot for monthly conferences. The results are shown in the chart “Organizing Communication at GM PT: Before.”

The new DSM demonstrated to GM that it needed to introduce a different type of organization. For a start, while many PDTs could be assigned to one system team encompassing their principal interactions, a certain amount of cross-membership was required. Some PDTs—pistons, for example—were assigned to two system teams. Two PDTs, cylinder heads (G) and intake manifold (J), were assigned to three system teams. (For visual clarity in the new matrix, these teams appear as G1 and G2 and J1 and J2, respectively.) The boxed groups on the reconfigured matrix denote the four new overlapping system teams that GM created.
But the greatest challenge facing GM was how to organize the five PDTs at the bottom of the matrix, whose systemwide communication needs could not be met within the structure of a system team. The solution we worked out with GM was to group these teams into a special “system integration team” whose responsibility was to ensure the overall integration of work by the four system teams so that the engine being developed would meet the required performance standards. The system integration team met its communication needs by leading monthly program meetings at which everyone discussed issues related to overall product performance.

These DSMs use symbols of varying size to reflect the frequency of team interactions. The boxes show which product development teams (PDTs) are grouped together into system teams. In the “before” DSM, we see that many PDTs had frequent contacts outside their designated groups.

The reorganization first isolated the PDTs with communication needs across the whole organization and then regrouped the remaining PDTs into new system teams so that most daily and weekly meeting needs would be met. The “after” DSM shows the four new overlapping teams as well as the system integration team that communicated with PDTs throughout the organization.

Organizing Communication at GM PT

Before…

...and After
## Semiconductor Development at Intel

This DSM describes Intel’s proven and successful process for developing and producing semiconductors. This complex DSM clarifies to Intel where to concentrate its efforts to improve the process. As the matrix reveals, the process includes 15 planned stages, most of which involve substantial iteration. There are also a significant number of unplanned iterations (the O’s), which are the subject of Intel’s process-improvement efforts. Three of the iterations occur so late that the company treats feedback from the later tasks (the O’s) as “generational learning” feedback, to be used in the design and development of subsequent products.

---

<table>
<thead>
<tr>
<th>Stage</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>Set customer target</td>
</tr>
<tr>
<td>20</td>
<td>Establish pricing direction</td>
</tr>
<tr>
<td>30</td>
<td>Schedule project timeline</td>
</tr>
<tr>
<td>40</td>
<td>Development methods</td>
</tr>
<tr>
<td>50</td>
<td>Macro targets/constraints</td>
</tr>
<tr>
<td>60</td>
<td>Financial analysis</td>
</tr>
<tr>
<td>10</td>
<td>Establish pricing direction</td>
</tr>
<tr>
<td>20</td>
<td>Set technical requirements</td>
</tr>
<tr>
<td>30</td>
<td>Write customer specification</td>
</tr>
<tr>
<td>40</td>
<td>High-level modeling</td>
</tr>
<tr>
<td>50</td>
<td>Write target specification</td>
</tr>
<tr>
<td>60</td>
<td>Develop test plan</td>
</tr>
<tr>
<td>10</td>
<td>Develop test plan</td>
</tr>
<tr>
<td>20</td>
<td>Develop validation plan</td>
</tr>
<tr>
<td>30</td>
<td>Build base prototype</td>
</tr>
<tr>
<td>40</td>
<td>Functional modeling</td>
</tr>
<tr>
<td>50</td>
<td>Develop product modules</td>
</tr>
<tr>
<td>60</td>
<td>Lay out integration</td>
</tr>
<tr>
<td>10</td>
<td>Integration modeling</td>
</tr>
<tr>
<td>20</td>
<td>Random testing</td>
</tr>
<tr>
<td>30</td>
<td>Develop test parameters</td>
</tr>
<tr>
<td>40</td>
<td>Finalize schematics</td>
</tr>
<tr>
<td>50</td>
<td>Validation simulation</td>
</tr>
<tr>
<td>60</td>
<td>Reliability modeling</td>
</tr>
<tr>
<td>10</td>
<td>Complete product layout</td>
</tr>
<tr>
<td>20</td>
<td>Continuity verification</td>
</tr>
<tr>
<td>30</td>
<td>Design rule check</td>
</tr>
<tr>
<td>40</td>
<td>Design package</td>
</tr>
<tr>
<td>50</td>
<td>Create initial QFD matrix</td>
</tr>
<tr>
<td>60</td>
<td>Develop program map</td>
</tr>
</tbody>
</table>

---

**X** indicates information flows. **☐** indicates planned stages. **O** indicates unplanned iterations. **●** indicates generational learning.

Little to be gained from breaking down the existing coupled groups.

But as the O’s show, a significant number of potential unplanned iterations can occur when errors are discovered during the development process. Results from Intel’s thermal testing (task 54), for example, could force the company to rework package design (task 29) or to rework functional modeling (task 17) or both. This rework would also require the company to redo some intervening tasks. The value of the DSM in cases like this resides principally in making explicit where information exchanges of...
this kind might occur. The company then decides what to do about them. Sometimes, there’s little it can do. The interdependent tasks may be so far apart that a delay caused by incorporating late information effectively means starting the whole process again. These situations usually arise because some fundamental mistake in assumptions was made at the beginning of the project. In Intel’s case, creating a product demonstration (task 48) had the potential to reveal that the company’s estimates of sales volumes and pricing levels were faulty (tasks 2 and 3). If the mistakes were serious, the product would have to be completely redesigned.

In such cases, we recommend treating the negative information as “generational learning feedback,” as Intel did (see ♦’s near the top of the chart). Rather than entirely rework the developed product and come out behind competitors in that product cycle, the company will either abandon the project altogether if the information reveals fatal flaws or launch the product as designed if the flaws are minor. Meanwhile, the information will be fed into the design and development of a product in the next generation.

In most cases, though, development teams prefer to minimize the probability that a later task will generate information that necessitates rework. Thus, it makes sense to transfer key knowledge and create prior tasks, as we discussed above. But individual solutions can emerge unexpectedly. At Intel, for instance, managers found they could reduce the likelihood of failures in thermal testing by having a thermal-testing engineer contribute to package design. Solutions of this kind will never entirely eliminate unplanned iterations, but they will certainly reduce the probability of them.

By stripping away the mystery around the exchange of information during innovation, the DSM can give managers far more control over some of their company’s most risky and expensive projects.

In our experience, the information generated in a DSM analysis has always yielded new insights to improve the ways companies develop products and services. By stripping away the mystery around the exchange of information during innovation, the DSM can give managers far more control over some of their company’s most risky and expensive projects.

For a more in-depth tutorial on how to create DSM models and for links to DSM research and software, go to http://web.mit.edu/dsm.

Reprint R001
To place an order, call 1-800-988-0886.