Proceedings of DETC'01
ASME 2001 Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
Pittsburgh, Pennsylvania September 9-12, 2001

DETC2001/DTM-21691

PRODUCT DEVELOPMENT PROCESS MODELING USING ADVANCED SIMULATION

Soo-Haeng Cho
Center for Innovation in Product Development
Massachusetts Institute of Technology
Cambridge, MA 02139
Email: soohaeng@mit.edu

Steven D. Eppinger
Center for Innovation in Product Development
Massachusetts Institute of Technology
Cambridge, MA 02139
Email: eppinger@mit.edu

ABSTRACT

This paper presents a product development process modeling and analysis technique using advanced simulation. The model computes the probability distribution of lead time in a resource-constrained project network where iterations take place among sequential, parallel and overlapped tasks. The model uses the design structure matrix representation to capture the information flows between tasks. In each simulation run, the expected durations of tasks are initially sampled using the Latin Hypercube Sampling method and decrease over time as the model simulates the progress of dynamic stochastic processes. It is assumed that the rework of a task occurs for the following reasons: (1) new information is obtained from overlapped tasks after starting to work with preliminary inputs, (2) inputs change when other tasks are reworked, and (3) outputs fail to meet established criteria. The model can be used for better project planning and control by identifying leverage points for process improvements and evaluating alternative planning and execution strategies. An industrial example is used to illustrate the utility of the model.

Keywords: Project Management, Design Structure Matrix, Project Simulation, Iteration

1. INTRODUCTION

1.1 Motivation

Today's competitive market has created a highly challenging environment for product development. Companies are under increasing pressure to sustain their competitive advantages by reducing product development time and cost while maintaining a high level of quality. These needs drive companies to focus on developing a well-coordinated development plan to organize their processes and resources [1-3].

A complex product development project involves a large number of tasks executed by a network of professionals from various disciplines. As complexity increases, it becomes more difficult to manage the interactions among tasks and people; it may be impossible to even predict the impact of a single design decision throughout the development process [4].

Since the introduction of network scheduling techniques such as CPM [5] and PERT [6], many researchers have developed extensions to make these standard techniques more powerful. These improvements include the following:

- allocation of resources across tasks
- modeling of uncertainty in estimated task durations using Monte Carlo methods
- prediction of cost as well as project duration
- analysis of iterative processes including concurrent and/or stochastic rework of tasks

However, there has not been any integrated model that utilizes all of these aspects of real projects.

The goal of this work is to create a generalized process model that can represent realistic behavior of a complex design project. This model can be used to improve the effectiveness and predictability of such processes, accelerate communications among people, and guide project management efforts throughout the development process. The model follows the information-based approach of the design structure matrix (DSM) method and uses advanced simulation techniques such as the Latin Hypercube Sampling (LHS) and parallel discrete event simulation.

1.2 Literature Review

Many models related to resource allocation seek to find the optimal allocation of resources to tasks, minimizing lead time subject to resource constraints. Early efforts concentrated on two areas: one is the formulation and solution of the problem as a mathematical (usually integer) programming problem and the other is the development of heuristic methods to obtain approximate solutions [7]. However, the exact approaches are not computationally viable for large task networks. Cooper [8] proposed heuristic priority rules based on experimental results for assigning resources when tasks are competing for limited resources.

Iteration is a fundamental characteristic of development processes [4, 9, 10]. However, the standard network-based models do not represent iterative task relationships very well. Steward [11] developed the design structure matrix (DSM) to model the information flow of design tasks and to identify their iterative loops. Eppinger et al. [4] extended Steward's work by explicitly modeling information coupling among tasks and investigating different strategies for managing entire development procedures. Several analytical models have been developed to represent iterative design processes. These include sequential iteration models, parallel iteration models, and overlapping models, which are reviewed below.

Sequential iteration models, wherein tasks are repeated one after the other by a probabilistic rule, have been implemented in several approaches. The GERT network is a generalized PERT network that allows probabilistic routing and feedback loops, and Q-GERT is its simulation package for large-scale projects [12-14]. Smith and Eppinger [15] developed a model based on a reward Markov chain using the DSM representation for repetition probabilities and task durations. Ahmadi and Wang [16] extended the sequential iteration model by taking into account dynamic iteration probabilities and learning effects. Eppinger et al. [17] used Signal Flow Graphs to compute the probability distribution of lead time. Andersson et al. [18] extended the SFG model to include learning effects and other non-linearities.

In parallel iteration models, multiple interdependent tasks work concurrently with frequent information exchanges. AitSahlia et al. [19] compared sequential and parallel iteration strategies in terms of time-cost tradeoff. Hoedemaker et al. [20] discussed the limit to parallel iteration due to increasing communication needs. Smith and Eppinger [21] developed a model of fully parallel iteration processes which predict slow and rapid convergence of parallel iteration. Carrascosa et al. [22] included the probability and impact of changes in stochastic processes.

In overlapping models, two development tasks are overlapped to reduce total lead time. Ha and Porteus [23] analyzed a model of concurrent product and process design tasks and explored the optimal review timing to minimize total expected completion time. Krishnan et al. [24] developed a framework for overlapped sequential tasks. They explained appropriate overlapping strategies based on upstream information evolution and downstream iteration sensitivity. Loch and Terwiesch [25] presented an analytical model for optimal communication policy between two sequentially overlapped tasks.

While each of the resource and iteration models mentioned above captures some aspects that earlier methods do not explain very well, each new model also has its own limitations. Due to the limitation of analytical approaches, the sequential iteration models do not handle resource constraints as well as more general project networks having parallel tasks (or paths) and overlapping. The parallel iteration models analyze important aspects of concurrent engineering but use highly simplified assumptions. The two-task overlapping models provide the optimal way to reduce the time of two sequential tasks having the interface of unidirectional information transfer. However, the concept does not easily apply to multiple tasks, in particular, having multiple paths with iteration. In addition, there has not been any significant work resolving resource over-allocation issues in the overlapped and coupled project networks where tasks repeat by a probabilistic rule.

Adler et al. [26] developed a simulation-based framework using queuing principles for a multi-project, shared-resources setting. The model incorporates simple iterative effects on development time but neglects many characteristics of iteration including overlapping (or preemptive) iteration. It also assumes that work can be reallocated between resources with perfect efficiency, which is very difficult to achieve. Browning and Eppinger [10] used simulation to analyze iterative processes based on a DSM model as well as to account for normal variance of development task durations. However, this first DSM simulation method is based on rather restrictive assumptions regarding task concurrency and rework. Roemer et al. [27] discussed time-cost tradeoffs in multiple overlapped tasks. However, his model is limited to a single path, assuming that sequential iterations take place among sub-tasks within a task.

In this paper, we present a second-generation, DSM-based, dynamic process simulation model that can incorporate many more general characteristics of complex product development processes, including the following features:

- modeling dynamic iteration characteristics among sequential, parallel and overlapped tasks
- resolving resource confliction in the iterative project network
- taking into account the normal variance of development time
- assessing schedule risks of iterative processes as they progress

In order to analyze such a rich process model, we must employ numerical simulation methods. The model differs from the Virtual Design Team framework [28] in that the objective of the VDT simulation is to predict organizational breakdowns in performing activities while the goal of ours is to predict dynamic behavior of iterative processes.

2. MODEL INPUTS

We follow the information-based view [9] in which a task is the information-processing unit that receives information from other tasks and transforms it into new information to be passed on to subsequent tasks. The information exchanged between tasks includes both tangible and intangible types such as parts, part dimensions, bill of materials etc. Model inputs characterize behaviors of individual tasks and interactions among the tasks from a schedule perspective. The duration of a task is used to model uncertainty and complexity within the domain of the task. Precedence and resource constraints are used to determine the boundaries of tasks along the time line. Iterations are modeled to predict the patterns of workflows caused by dynamic information exchanges among the tasks.

2.1 Task Durations

The model uses the triangular probability distribution to represent the characteristic of a task duration since it offers comprehensibility to a project planner [29]. For each task, the model receives three estimated durations – optimistic, most likely and pessimistic as in some PERT-based analyses – for the expected duration of *one-time* execution. The expected duration is the duration between the start and end of its continuous work even though the task may iterate more than once afterwards. Remaining duration decreases over time as the model simulates the project's progress. The model also receives actual duration if the task has been in progress.

It has been found that assessing the 10th and 90th percentiles of the expected duration is more reliable than the extremes of the PDF which are typically outside the realm of experience [29, 30]. The model uses the Latin Hypercube Sampling (LHS) method [31] to incorporate the uncertainty of the expected duration of a task based on three estimated durations. After calculating the extreme values of the PDF, it divides the range between them into N strata of equal marginal probability $1/N$ where N is the number of random values for the expected duration representing a known triangular PDF. Then, it randomly samples once from each stratum and sequences the sampled values randomly. Figure 1 illustrates the LHS procedure.

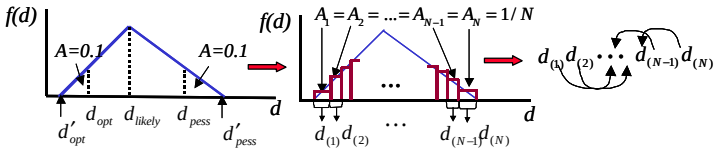


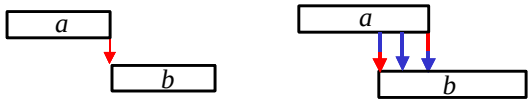
Figure 1. Latin Hypercube Sampling

2.2 Precedence Constraints

From a schedule perspective, there exist two types of *information flow in a task*: (1) information flow at the beginning or at the end of the task, and (2) information flow in the middle of the task. Based on this observation, we define two types of *information flow between two tasks*. The first type represents the case that the task requires final output information from the upstream task to begin its work. The second type represents the case that the task uses final output information in the middle of its process and/or begins with preliminary information but also receives a final update from the upstream task.

The first type of information flow is translated to a "finish-to-start" precedence constraint between two tasks while the second type is translated to a "finish-to-start plus lead" constraint, as illustrated in Figure 2. Note that each of three arrows in Figure 2 (b) represents the second type of information flow while it is modeled as the same precedence constraint. The design structure matrix (DSM) is used to document these information flows and precedence constraints. The notation *DSM(i, j)* for $i, j = 1, \ldots, n$ represents this two-type scheme in the DSM having n tasks where:

- *DSM(i, j)* = 0 when there is no information flow from task *j* to task *i*
- *DSM(i, j)* = 1 when there is the finish-to-start type of information flow from task *j* to task *i*
- *DSM(i, j)* = 2 when there is the finish-to-start-plus-lead type of information flow from task *j* to task *i*



(a) Finish-to-Start *(b) Finish-to-Start + Lead*

Figure 2. Information Flows and Precedence Constraints

2.3 Resource Constraints

The model assumes that there exists a fixed resource pool throughout the entire project duration. It consists of specialized resources and/or resource groups of which constituents exhibit the same functional performance. Each task has its own resource requirement which is assumed to be constant over the entire period the task is processed. When two or more tasks are competing for limited resources in a certain period of time, *i.e.* resources are over-allocated, the model determines priorities by the heuristic rules explained later in the paper.

2.4 Iteration

Eppinger et al. [17] defined iteration as the repetition of tasks to improve an evolving development process. In this paper, iteration is referred to as rework caused by other tasks without including repetitive work within a single task (variance in duration). The model assumes that planned rework of a task is generated due to the following causes (similar to the explanations of Browning and Eppinger [10], Smith and Eppinger [15], Eppinger et al. [17]):

- receiving new information from overlapped tasks after starting to work with preliminary inputs
- probabilistic change of inputs when other tasks are reworked
- probabilistic failure to meet the established criteria

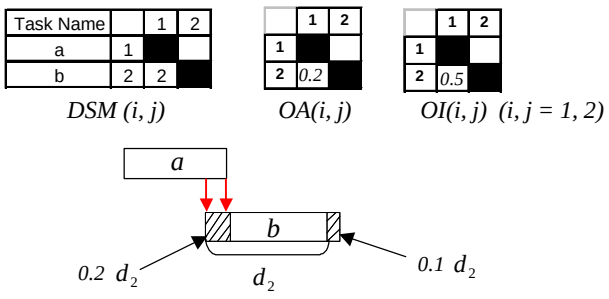
In the proposed model, the first cause gives rise to overlapping iteration, and the second and the third causes give rise to sequential iteration. Parallel iteration of a limited number of tasks is simulated in this model by combining overlapping and sequential iteration.

2.4.1 Overlapping Iteration

Overlapping has been described as a "core technique for saving development time" [32]. It is generally acknowledged that overlapping tasks may save time, but is more costly than the traditional sequential approach. Suppose two dependent tasks are overlapped sequentially and the downstream task starts to work with the preliminary information from the upstream task. As the upstream task evolves, its output information also evolves to its final form while the new information generated since its initial transfer of the preliminary information gets released according to its communication policy. The downstream task may repeat the part of its work to accommodate this new information which is unnecessary if it started to work with the final information from the upstream task. In this model, it is assumed that overlap amounts as well as expected rework impacts between the two tasks can be estimated in the planning stage.

The model uses the DSM representation as shown in Figure 4. The notation *OA(i, j)* is used for maximum overlap amount and *OI(i, j)* for overlap impact for $i, j = 1, \ldots, n$. The former represents the planned overlap amount between tasks *i* and *j*, and it is a fraction of the expected duration of task *i*, d_i. This carries the assumption that the downstream task cannot be completed before the upstream task finishes. The latter represents the expected rework impact when task *i* is overlapped with task *j* by the amount $OA(i, j) \times d_i$ and it is a fraction of that amount. $OI(i, j) = 1$ implies no benefit from overlapping. To implement overlapping strategy, it should be reasonably less than 1 considering additional risk due to the evolution of volatile preliminary information.

In Figure 3, task *b* starts with preliminary information from task *a*. It is planned to begin earlier with preliminary information and expected to finish 20% of its work before task *a* gives a final update. However, it is also expected to rework half of the work done through overlapping to incorporate updated information from task *a*. Thus, lead time is reduced by 10% of d_2 by this overlapping.



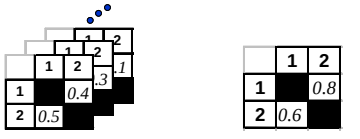
DSM (i, j) *OA(i, j)* *OI(i, j) (i, j = 1, 2)*

Figure 3. Overlap Amount and Impact between Two Tasks

2.4.2 Sequential Iteration

The model takes an approach similar to Browning and Eppinger [10] by explaining sequential iteration using rework probability, rework impact and the learning curve. Rework probability is a measure of uncertainty in sequential iteration. *RP(i, j, r)* represents the probability that task *i* does rework affected by task *j* in *r*th iteration for $i, j = 1, \ldots, n$ and $r = 1, 2, \ldots$ In the case of $i < j$, it represents the *feedback* rework caused by the change of information from downstream task *j* or by the failure of downstream task *j* to meet the established criteria. In the case of $i > j$, it represents the *feedforward* rework that downstream task *i* needs to do since upstream task *j* has generated new information after it has done its own rework. As the development processes converge to their final solutions with iterative rework, there are less chances that new information is generated and errors are discovered. Therefore, rework probability tends to decrease in each iteration.

Rework impact is a measure of the level of dependency between tasks in sequential iteration. *RI(i, j)* represents the percentage of task *i* to be reworked when rework is caused by task *j* for $i, j = 1, \ldots, n$. Rework impact is assumed to be constant in each iteration. The learning curve measures a characteristic of a task when it repeats. $(L_{ori})_i$ for $i = 1, \ldots, n$ represents the percentage of original duration when task *i* does the same work for a second time. The model assumes that the learning curve decreases by $(L_{ori})_i$ percent in each repetition until it reaches $(L_{max})_i$ which is the minimum percentage of original duration when task *i* does the same work repeatedly. Thus, rework amount is calculated as the original duration multiplied by the rework impact and learning curve. Figure 4 shows the rework probability and impact for sequential iteration using the DSM representation, and Figure 5 illustrates the learning curve.



RP(i, j, r) *RI(i, j)* *(i, j=1,2; r = 1,2,…)*

Figure 4. Rework Probability and Impact

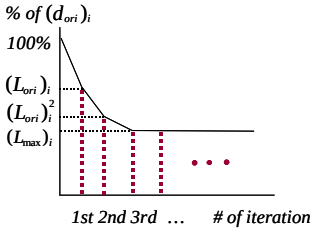


Figure 5. Learning Curve

3. MODEL DESCRIPTION

The model employs a *parallel discrete event simulation* [33, 34] to compute the distribution of lead time. Analytical features are included so that the model can describe the complex behavior of development processes having overlapped tasks and sequential iterations. This section explains the underlying structure of the simulation-based model.

3.1 Model Structure and Algorithm

In the discrete event simulation, events trigger state transitions and time advances in discrete steps by the time elapsed between events. The distinctive feature of the discrete *event* simulation is that no components within a system need to be scanned during times between events (in contrast, the discrete *time* simulation requires that scanning must be performed at each time step). A parallel simulation allows multiple model components to be active and to send their outputs to other components.

The model uses different expected durations of tasks in each simulation run which are initially sampled using the Latin Hypercube Sampling method. With those durations of tasks, it simulates a series of sequential state transitions incorporating iterations in multiple paths. States are determined dynamically based on all the inputs of tasks and task interfaces explained earlier. In each state, it scans all tasks and determines a set of active tasks satisfying both precedence and resource constraints. If the amount of resources required by the tasks satisfying precedence constraints exceeds the resource capacity of the project, resource assignments are made by the heuristic priority rules. It assumes that a task begins to work as early as possible when it has all the necessary inputs from upstream tasks and all the required resources.

An *event* is defined as the completion of a task instead of any information transfer. Thus, when any active task in the current state q is completed, the model makes a transition to the next state q+1. The duration of state q (q = 0, 1, 2, …) is defined as the minimum remaining duration of active tasks in the state. Before making a transition to the next state, the model subtracts the duration of the current state from the remaining durations of all active tasks. If all the remaining durations of tasks are zero, one simulation run is complete and the lead time is calculated as the sum of all the state durations. After N simulation runs, the probability distribution of lead time can be displayed.

Figure 6 summarizes the algorithm to compute lead time in one simulation run. A simulation run starts with initializing model variables from the model inputs at STEP1. It simulates time advance of tasks by following STEP2 through STEP7 in each state until it satisfies the termination condition.

3.2 Overlapping Iteration in Multiple Resource-Constrained Paths

In each state, the model identifies a set of active tasks which have started to work in the current state. For each task in this set, the model simulates that its overlapped part of work has been performed in prior state(s) and the expected impact due to iterations has been added to the projection in the current state. The overlap amount of a task is dynamically determined by both precedence and resource constraints with other tasks in multiple paths. The model assumes that overlapped part of work has a lowest priority for limited resources and does not start to do unless resources are secured during its processing time. When the amount of overlap is different from the planned amount with any information-providing task, it computes its overlap impact, assuming that it is linear to the overlap amount. If a task is overlapped with multiple tasks, the overlap impact is between the maximum of single impacts and the sum of them depending on the amount of duplicate rework caused by those tasks. In this case, the model takes the latter as a default. Finally, the overlap amount is subtracted from the remaining duration of the active task and the overlap impact is added to it.

For each simulation run,

STEP1. Initialize model variables from the inputs at state *0*.

STEP2. Initialize model variables in the current state *q*.

STEP3. Identify a set of concurrent active tasks in the current state satisfying precedence and resource constraints based on the priority rules.

STEP4. Adjust overlapping iteration.

STEP5. Adjust the durations of the active tasks and the lead time.

STEP6. Generate sequential iteration rework.

STEP7. Make a transition to the next state *q*+1 or finish the simulation run if satisfying the termination condition.

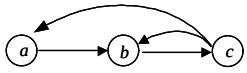
Figure 6. Algorithm to Compute Lead Time

3.3 Rework Generation for Sequential Iteration in Multiple Paths

In each state, the model identifies a set of active tasks which are supposed to be completed in the current state. It is those tasks that cause state-transition events. For each task in the set, the model determines whether it causes feedback and/or feedforward rework to other tasks. Rework decisions are simulated by comparing each rework probability with a random number. When rework occurs, the amount of rework is computed by the original duration of a task doing rework multiplied by a rework impact adjusted for learning curve benefit. Finally, rework amounts are added to the remaining durations of those tasks that are determined to rework. Note that feedback rework in an upstream task can also cause successive feedforward rework in subsequent states to the downstream tasks that have been in process or completed before. The model also simulates that a rework decision can be made before final output information is produced through overlapping.

3.4 Rework Concurrency

When tasks iterate sequentially, a choice of rework policy may allow rework concurrency to shorten the lead time. For illustration, consider the simple example given in Figure 7. The information flow diagram and its corresponding DSM show three sequential tasks with feedback loops. In all other process models for sequential iterations surveyed in the literature, when both tasks *a* and *b* require new information from task *c*, the rework of task *b* cannot begin before the completion of the rework of task *a*. This is based on the underlying assumption that the precedence constraint between tasks *a* and *b* should also be respected when tasks iterate. However, a project manager may prefer a different policy when there is a small chance that task *a* will produce new information also causing task *b* to rework. By performing the rework of both tasks *a* and *b* concurrently, the lead time can be reduced with small additional risk.



Task Name		1	2	3
a	1			1
b	2	1		1
c	3		1	

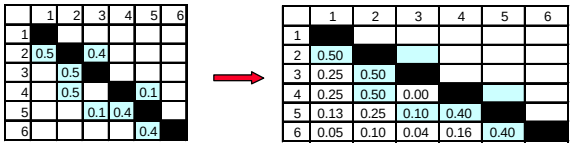
(a) Information Flow Diagram *(b) DSM*

Figure 7. An Example for Task Concurrency

We introduce the *Rework Concurrency (RC)* to model this strategic decision upon task concurrency during iteration. It represents total *direct* and *indirect* feedforward rework probabilities which control the level of concurrency in sequential iteration. The *Rework Concurrency* is a lower-triangular matrix which takes *direct* rework probabilities from *RP(i, j, 1)* ($i = 2,\ldots, n;\ j = 1,\ldots, i\text{-}1$) and adds them with *indirect* rework probabilities. The *indirect* probability *(i, j)* represents the probability of task *i* doing rework caused by task *j* through the intermediary of other tasks between *i* and *j*. For example, *RC(5, 2)* in Figure 8 is computed as the sum of indirect rework probabilities between tasks 2 and 5 through tasks 3 and 4 as intermediaries ($0.5\times0.1 + 0.5\times0.4 = 0.25$).

RC(i, j) (i > j) is computed at STEP3 of the simulation algorithm when determining the concurrency of tasks *i* and *j* when both do rework. The model assumes that a task can rework even though there exists an upstream dependent task doing its own rework if the total rework probability between the two tasks in the *RC* is less than the probability $P_{tolerance}$, a user-specified *rework risk tolerance*. The algorithm to compute the *Rework Concurrency* is explained in the section *A1* of the *Appendix*.

In the above example, if $RC(2, 1) < P_{tolerance}$, the model simulates that both tasks *a* and *b* are reworked concurrently when both must be reworked. Otherwise, task *b* waits until the rework of task *a* is completed, at which time, new information from task *a* becomes available. In some cases, it may not be necessary for task *b* to use any of the new information from the rework of task *a*. However, if the rework of task *a* does create additional rework for task *b*, the total amount of rework of task *b* is between the maximum and the sum of reworks generated by tasks *a* and *c*. The model uses the latter as the default amount of rework required for task *b* and assumes that this quantity cannot exceed the task's original duration diminished by the learning curve effect.



	1	2	3	4	5	6
1						
2	0.5		0.4			
3		0.5				
4		0.5			0.1	
5			0.1	0.4		
6					0.4	

	1	2	3	4	5	6
1						
2	0.50					
3	0.25	0.50				
4	0.25	0.50	0.00			
5	0.13	0.25	0.10	0.40		
6	0.05	0.10	0.04	0.16	0.40	

RP(i, j, 1) (i, j = 1,…, 6) *Rework Concurrency*

Figure 8. An Example of Rework Concurrency

3.5 Measures and Rules for Resource Priorities

The heuristic priority rules that Cooper [8] proposed are no longer applicable in the project network where tasks iterate sequentially in a probabilistic manner. In this paper, a *rework-adjusted rank positional weight* is proposed as a good measure that can help a project manager to determine task priorities. The rank positional weight is one of the measures that Cooper found among the best toward minimizing total lead time in the project network without iteration. (See the section *A2* in the *Appendix* for the definition.) We define the *rework-adjusted duration* of a task as the expected value of the sum of the duration of its first execution and the total amount of successive rework it creates for its predecessors, assuming no resource constraints in the project network. Then, the *rework-adjusted rank positional weight* is computed by replacing the deterministic duration of a task in Cooper's definition with the rework-adjusted duration. This measure of task priority is calculated before computing lead time *with* resource constraints.

The model determines priorities by the heuristic rules whereby a task has a higher priority if:

(1) it has been in process
(2) it has a higher user-specified priority
(3) it has a higher rework-adjusted rank positional weight
(4) it is sequenced more upstream
(from (1) to (4) in order of significance)

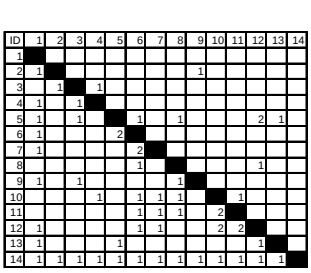
The above priority rule is toward minimizing lead time of a coupled block. Rule (1) implies that a task cannot be interrupted once it has started (non-preemption). Thus, the model does not allow splitting of a task due to its resource constraints. The user-specified rule in (2) can be used when resource priorities should be determined considering different project objectives. Rule (3) stipulates that a higher priority is given to the task that exposes the project to higher schedule risk.

4. SAMPLE APPLICATION

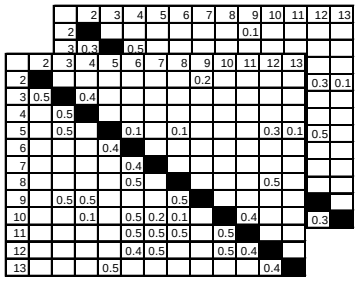
The uninhabited aerial vehicle (UAV) preliminary design process at an aerospace company is presented as a sample application. The data are from Browning [35] and extended to perform further analyses using additional modeling parameters. Figure 9 shows basic model inputs.

ID	Task Name	Exp. Duration			Learning
		Opt.	Likely	Pess.	
1	Prepare UAV Preliminary DR&O	1.9	2.0	3.0	0.35
2	Create UAV Preliminary Design Configuration	4.8	5.0	8.8	0.20
3	Prepare Surfaced Models & Internal Drawings	2.7	2.8	4.2	0.60
4	Perform Aerodynamics Analyses & Evaluation	9.0	10.0	12.5	0.33
5	Create Initial Structural Geometry	14.3	15.0	26.3	0.40
6	Prepare Structural Geometry & Notes for FEM	9.0	10.0	11.0	0.90
7	Develop Structural Design Conditions	7.2	8.0	10.0	0.35
8	Perform Weight & Inertial Analyses	4.8	5.0	8.8	1.00
9	Perform S&C Analyses & Evaluation	18.0	20.0	22.0	0.25
10	Develop Freebody Diagrams & Applied Loads	9.5	10.0	17.5	0.50
11	Establish Internal Load Distributions	14.3	15.0	26.3	0.75
12	Evaluate Structural Strength, Stiffness, & Life	13.5	15.0	18.8	0.30
13	Preliminary Manufacturing Planning & Analyses	30.0	32.5	36.0	0.28
14	Prepare UAV Proposal	4.5	5.0	6.3	0.70

(a) Project Table



(b) DSM(i, j) (i, j = 1,…, 14)



(c) RP(i, j,1) and RI(i, j) (i, j = 2,…, 13)

Figure 9. Model Inputs for UAV Project

4.1 Results Using Basic Modeling Parameters

Under the same conditions of Browning and Eppinger [10] – (1) 0 and 100th percentiles for optimistic and pessimistic duration estimates (2) constant rework probabilities in all iterations, (3) learning curve benefit only in the first iteration, (4) zero rework risk tolerance, $P_{tolerance}$ (5) no overlapping, and (6) no resource conflicts, the average of lead time is 146.8 days with 17.0 days of standard deviation after 2000 simulation runs. The probability distribution shown in Figure 10 is skewed to the right because the lead time becomes longer as more iterations take place. Both the average and standard deviation are higher than those obtained by Browning and Eppinger (avg. 141, s.d. 8). This is mainly because the new model accounts for all the successive feedforward rework while the earlier model does not.

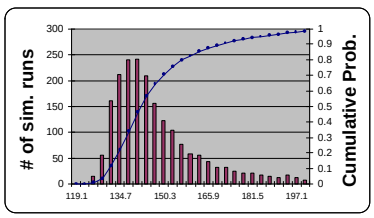


Figure 10. Probability Distribution of Lead Time with Basic Inputs

4.2 Results Using Additional Modeling Parameters

The simulation model allows great flexibility to account for more general features of dynamic development processes. Table 1 summarizes the results using additional modeling parameters as follows:

(1) Optimistic and pessimistic duration estimates are 10th and 90th percentiles, respectively.
(2) Rework probabilities decrease in each iteration by 50%.
(3) Maximum learning curve is 50% of that in the first iteration.
(4) Rework risk tolerance $P_{tolerance}$ = 0.3, i.e. more concurrency is allowed when tasks iterate.

Under the more rigorous assumption about task duration estimates in (1), both the average and standard deviation are greater than those with 0 and 100th percentile estimates. This is mainly due to the right-skewness tendency of task durations. Also, the existence of multiple paths in the project contributes to the increase of the lead time. Incorporating dynamic characteristics of sequential iteration in (2) and (3), and increased task concurrency in (4), the model predicts smaller averages and standard deviations of the lead time. The cumulative effect of the additional modeling parameters from (2) to (4) is a 5.3% decrease from the lead time under the assumption (1). Note that this difference will be more significant with more tasks and iterations.

Basic	(1)	(1), (2)	(1) - (3)	(1) - (4)
(146.8, 17.0)	(152.2, 20.1)	(149.3, 15.8)	(148.5, 14.2)	(144.2, 13.1)

<Note>
(i) (#, #): (average, standard deviation)
(ii) "(1)-(3)" denotes the results of the model with assumptions (1) through (3), and so on.

Table 1. Results Using Additional Modeling Parameters

Ignoring feedback marks in the DSM, *i.e.* assuming no sequential iterations, the path along the tasks 1-2-3-5-6-7-10-11-12-13-14 is a critical path when tasks have most likely durations. This implies that the lead time can be reduced by overlapping the tasks along this path such as the development of preliminary surfaced configuration (task 3) and structural design (tasks 5–7). Other important leverage points for reducing the lead time are the feedback marks. By transferring preliminary review decisions or testing results to upstream tasks, feedback rework can start earlier. This allows for the accelerations of iterative rework. Under the following overlapping scenario, the average lead time is reduced to 137.7 days:

(5) For the six information flows marked by "2" in the DSM in Figure 9 (b), tasks are planned to complete 25% of work with preliminary inputs before receiving final updates, and expected to redo 50% of work completed without final updates.

The above scenario includes the overlapping between tasks 12 and 5 in feedback rework. The following scenario, for example, would cause such overlapping:

As a result of structural evaluation in task 12, it may become necessary to redesign the structural geometry of specific

subsystems before having evaluation results for the entire system. The need to redesign some subsystems (task 5) can be detected early in a series of tests in task 12, whereas the need for additional weight and inertial analyses (task 8) can be determined only at the end of the tests.

All the above analyses are performed under the assumption that there is no resource conflict among the tasks in the UAV project. This assumption is reasonable because tasks that can be performed in parallel are related to different disciplines, therefore, no resource sharing is necessary between those tasks. In multiple project environments, however, the resources belonging to the same functional group may need to get involved in tasks in different projects during a certain period of time. In this case, optimal resource allocation toward project objectives becomes an important issue. For the purpose of illustration, an example situation is tested as follows:

(6) Tasks 7 and 8 compete for limited resources and one of them should be delayed.

Under assumptions (1)-(5), the rework-adjusted rank positional weights of tasks 7 and 8 are 105.4 and 124.5 respectively. Following the resource priority rules, the model assigns resources to task 8 and delays task 7 that can work in parallel with task 8 without this resource constraint. Then, both tasks 7 and 8 become critical and the project is delayed by the duration of task 7. Figure 11 shows an example of a simulated Gantt chart. This is only one of many simulation runs under assumptions (1)-(6). The numbers inside the bars indicate the amounts of overlapped work performed in prior state(s). Even though the scenario shows that tasks rework during states 5 and 14-19, it delays the entire project only in states 5 and 14. This is due to the pre-determined work policy for increased task concurrency during iterations. Since total rework probabilities are less than 0.3 except for task 5, preliminary manufacturing planning and analyses (task 13) are simulated to work concurrently with functional performance analyses (tasks 8-11) after redesigning structural geometry (task 5).

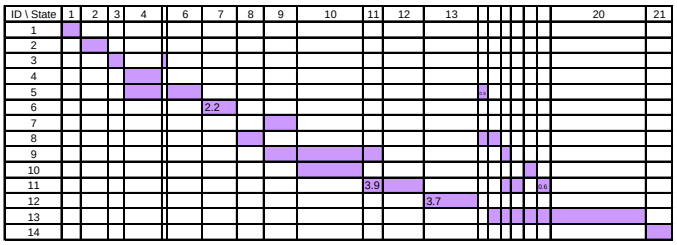


Figure 11. An Example of a Simulated Gantt Chart

5. DISCUSSION

5.1 Applications of the Model

The previous DSM-based process models [10, 15, 17, 21, 22] have provided practical insights for process understanding and improvements. The same types of analyses can be further enhanced using this simulation-based model. The distinctive feature of this model is that it has greater flexibility and generality to describe real development processes. Engineering management can benefit from this rich model through better project planning and control in several ways discussed in this section.

5.1.1 Finding an Optimal Task Ordering

The process model presented in this paper is a performance evaluation model, not an optimization model. However, the model can be easily utilized to find an optimal task sequence given the objective function that might be, for instance, a function of the average and standard deviation of lead time, and schedule risk. Note that this is an O(n!) operation where n is the number of tasks [10] if we allow tasks to be positioned anywhere in the sequence. By fixing some logical precedence relationships between tasks, computation time can be reduced.

5.1.2 Setting Appropriate Due Date and Buffer

By analyzing the probability distribution of lead time, an appropriate due date can be chosen with predictable confidence [10]. When the schedule target is given, the risk that the project fails to meet the target can be assessed via the simulation. When the schedule risk is high, the model can be used to evaluate different improvement efforts such as adding resources, overlapping tasks, executing fewer or faster iterations etc. This perspective may facilitate decision-makings and communications between senior management and the project team. When a project plan is constructed (usually with deterministic task durations), a buffer can be added at the end representing aggregated safety based on the simulation result. This can improve the Critical Chain method [36] as well.

5.1.3 Finding Areas for Process Improvement

The model can be used to evaluate various process improvements through simulation by adjusting model parameters. Below are examples of model applications for this purpose:

- ***Task criticality:*** The sensitivity of project lead time to variation in task duration provides a measure of task criticality [37-39]. This measure can be effectively used in a resource-constrained iterative environment where conventional slack and resource-constrained slack [38] cannot be defined.
- ***Strategic work policy for concurrency:*** The rework risk tolerance defined earlier can be used as guidance for work policy during iterations. Lead time can be reduced by increasing concurrency level strategically, although this may result in increased development costs.
- ***Faster and/or fewer iterations:*** Smith and Eppinger [21] proposed two general strategies for accelerating the iterative processes. Faster iterations can be achieved by increasing learning curve and/or decreasing rework impact, *i.e.* reducing the amount of rework. For instance, the efficient use of information technology such as computer-aided tools could help accomplish this effect. Fewer iterations can be achieved by decreasing rework probabilities. Well-defined interfaces between tasks, and

well-established coordination and communication routes between project members could reduce the number of iterations. The proposed model can be used to identify the most efficient points for such improvements.

5.1.4 Ongoing Risk Assessment

The model inputs can be easily updated to incorporate current information as the project progresses. For instance, initial estimates for task durations, overlap amounts and impacts can be updated or replaced with actual values as they become available. Rework parameters may be updated to represent foreseeable iterations. As uncertainties diminish while the project advances, the variance of lead time becomes smaller. The schedule risk can be identified quantitatively and its assessment can accelerate proactive risk management efforts.

5.1.5 Evaluating Multiple Projects

The model can also be applied in a multi-project environment by representing each project on a different path. In this manner, it is possible to consider resource allocation across the projects and the effect of such constraints on completion of all of the projects.

5.2 Limitations and Extensions

The model simply assumes that the processing time of each task is independent of those of other tasks. However, when uncertainties affect multiple tasks, independent duration distributions cannot be assumed [29]. Various methods have been developed to model interdependencies, from estimation of correlation coefficients between tasks to use of joint distributions. However, difficulties remain in measuring correlation among task durations.

The model does not guarantee optimal resource allocation to minimize lead time. This is a limitation of a richer model incorporating iteration and overlapping in multiple paths. A heuristic resource priority rule can no longer exploit slack (float) under the existing definition in the iterative project network where a probabilistic rule is applied. The following questions illustrate this restriction: if a task has no slack in its first execution while having some slack in its first iteration, would we use the average slack in assigning resources to the task? What if the task repeats twice in another simulation scenario? Thus, the model resorts to the heuristic rule using the rework-adjusted rank positional weight accounting for probabilistic rework, and uses the sensitivity of project lead time to variation of a task duration as a measure of criticality.

In this model, we assume a fixed resource pool for the project and constant resource requirements for tasks. The model can be extended by allowing variable resource capacity and requirements.

The model uses a simple assumption that cumulative overlap impact is the sum of single impacts when a task is overlapped with more than one task. This is not useful to predict the progress of concurrent execution of tightly coupled tasks. More accuracy can be achieved if we could model frequent bi-directional information exchange and consequent impacts between parallel tasks.

Lastly, the model can be further extended by incorporating development cost as in Browning and Eppinger [10]. However, it is very difficult to apply the same cost structure to different iterative process hierarchies in conjunction with development time.

6. CONCLUSION

This paper presents a DSM-based process model using advanced simulation. The model accounts for important characteristics of product development processes, including information transfer patterns, uncertain task durations, resource conflicts, overlapping and sequential iterations, and task concurrency. The model addresses several limitations imposed by previous analytical and simulation-based approaches. It can be applied to a wide range of processes where iteration takes place among sequential, parallel, and overlapped tasks in a resource-constrained project. Increased understanding of realistic behavior of complex design projects can be achieved through modeling information flows and predicting lead times. The model is also useful for evaluating different project plans and for identifying strategies for process improvements. Proactive risk management can be achieved by assessing the status of the project as it progresses.

ACKNOWLEDGMENTS

This research was supported by the Center for Innovation in Product Development at M.I.T. and by the Singapore-M.I.T. Alliance.

REFERENCES

[1] Takeuchi, H. and Nonaka, I., "The New Product Development Game," *Harvard Business Review*, Jan.-Feb., pp. 137-146, 1986.

[2] Clark, K. and Fujimoto, T., *Product Development Performance: Strategy, Organization, and Management in the World Auto Industry*, Harvard Business School Press, 1991.

[3] Wheelwright, S. and Clark, K., *Revolutionizing Product Development: Quantum leaps in Speed, Efficiency and Quality*, Free Press, 1992.

[4] Eppinger, S., Whitney, D., Smith, R. and Gebala, D., "A Model-Based Method for Organizing Tasks in Product Development," *Research in Engineering Design*, **6**, pp. 1-13, 1994.

[5] Kelley, J. and Walker, M., "Critical-Path Planning and Scheduling," *Proceedings of Easter Joint Computer Conference*, pp. 160-173, 1959.

[6] Malcolm, D., Roseboom, J., Clark, C. and Fazar, W., "Application of a Technique for Research and Development Program Evaluation," *Operations Research*, **7**, pp. 646-669, 1959.

[7] Patterson, J., "A comparison of exact approaches for solving the multiple constrained resource, project scheduling problem," *Management Science*, **30**, pp. 854-867, 1984.

[8] Cooper, D., "Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation," *Management Science*, **22**, pp. 1186-1194, 1976.

[9] Alexander, C., *Note on the Synthesis of Form*, Harvard University Press, 1964.

[10] Browning, T. and Eppinger, S., "Modeling the Impact of Process Architecture on Cost and Schedule Risk in Product

Development," Working Paper No. 4050, Sloan School of Management, MIT, 2000.
[11] Steward, D., "The Design Structure System: A Method for Managing the Design of Complex Systems," *IEEE Transactions on Engineering Management*, **EM-28**, 1981.
[12] Neumann, K., "GERT Network and the Time-Oriented Evaluation of Projects," *Lecture Notes in Economics and Mathematical Systems*, **172**, 1979.
[13] Pritsker, A., *Modeling and Analysis Using Q-GERT Networks*, 2nd Edition, John Wiley/Halsted Press, 1979.
[14] Taylor, B. and Moore, L., "R&D Project Planning with Q-GERT Network Modeling," *Management Science*, **26**, pp. 44-59, 1980.
[15] Smith, R. and Eppinger, S., "A Predictive Model of Sequential Iteration in Engineering Design," *Management Science*, **43**, pp. 1104-1120, 1997.
[16] Ahmadi, R., and Hongbo, W., "Rationalizing Product Design Development Process," Working Paper, Anderson Graduate School of Management, UCLA, 1994.
[17] Eppinger, S., Nukala, M. and Whitney, D., "Generalized Models of Design Iteration Using Signal Flow Graphs," *Research in Engineering Design*, **9**, pp. 112-123, 1997.
[18] Andersson, J., Pohl, J. and Eppinger, S., "A Design Process Modeling Approach Incorporating Nonlinear Elements," *Proceedings of ASME Design Engineering Technical Conferences*, DETC98-5663, 1998.
[19] AitSahlia, F., Johnson, E. and Will, P., "Is Concurrent Engineering Always a Sensible Proposition?," *IEEE Transactions on Engineering Management*, **42**, pp. 166-170, 1995.
[20] Hoedemaker, G., Blackburn, J. and Wassenhove, L., "Limits to Concurrency," Working Paper, Vanderbilt University Owen School of Management, 1995.
[21] Smith, R. and Eppinger, S., " Identifying Controlling Features of Engineering Design Iteration," *Management Science*, **43**, pp. 276-293, 1997.
[22] Carrascosa, M., Eppinger S. and Whitney D., "Using the Design Structure Matrix to Estimate Product Development Time," *Proceedings of ASME Design Engineering Technical Conference*, DETC98/DAC-6013, 1998.
[23] Ha, A. and Porteus, E., "Optimal Timing of Reviews in the Concurrent Design for Manufacturability," *Management Science*, **41**, pp. 1431-1447, 1995.
[24] Krishnan, A., Eppinger, S. and Whitney, D., "A Model-Based Framework to Overlap Product Development Activities," *Management Science*, **43**, pp.437-451, 1997.
[25] Loch, C. and Terwiesch, C., "Communication and Uncertainty in Concurrent Engineering," *Management Science*, **44**, pp. 1032-1048, 1998.
[26] Adler, P., Mandelbaum, A., Nguyen V. and Schwerer E., "From Project to Process Management: An Empirically-based Framework for Analyzing Product Development Time," *Management Science*, **41**, pp. 458-484, 1995.
[27] Roemer, T., Ahmadi, R. and Wang, R., "Time-Cost Trade-Offs in Overlapped Product Development," Operations Research, **48**, pp. 858-865, 2000.
[28] Levitt, R., Cohen, G., Kunz, J., Nass, C., Christiansen, T., and Jin, Y., *The virtual design team: Simulating how organization structures and information processing tools affect team performance*, in Carley, K.M. and M.J. Prietula, editors, Computational Organization Theory, Lawrence Erlbaum Associates, Hillsdale, NJ, 1994.
[29] Williams, T., "Practical Use of Distributions in Network Analysis," *Journal of the Operational Research Society*, **43**, pp 265-270, 1992.
[30] Keefer, D. and Verdini, W., "Better Estimation of PERT Activity Time Parameters," *Management Science*, **39**, pp. 1086-1091, 1993.
[31] McKay, M., Beckman, R. and Canover, W., "A Comparison of Three Methods for Selecting Values of Input Variables in the Analysis of Output from a Computer Code," *Technometrics*, **21**, pp. 239- 245, 1979.
[32] Smith, P. and Reinertsen, D., *Developing Products in Half the Time*, 2nd Edition. Van Nostrand Reinhold, NY, 1995.
[33] Pritsker, A. and O'Reilly, J., *Simulation with Visual SLAM and AweSim*, 2nd Edition, John Wiley & Sons, 1999.
[34] Zeigler, B., Praehofer, H., and Kim, T., *Theory of Modeling and Simulation – Integrating Discrete Event and Continuous Complex Dynamic Systems*, 2nd Edition, Academic Press, 2000.
[35] Browning, T., "Modeling and Analyzing Cost, Schedule, and Performance in Complex System Product Development," Ph.D. Thesis (TMP), MIT, Cambridge, MA, 1998.
[36] Goldratt, E., *Critical Chain*, The North River Press, MA, 1997.
[37] Williams, T., "Criticality in Stochastic Networks," *Journal of the Operational Research Society*, **43**, pp. 353-357, 1992.
[38] Bowers, J., "Criticality in Resource Constrained Networks," *Journal of the Operational Research Society*, **46**, pp 80-91, 1995.
[39] Christian, A., "Simulation of Information Flow in Design," Ph.D. Thesis (ME), MIT, Cambridge, MA, 1995.

APPENDIX

A1. Algorithm to Compute the Rework Concurrency

The *Rework Concurrency* between tasks *a* and *b* is computed as follows:

For $i = a + 1, \ldots, b$,
(1) Set $RC(i, j) = RP(i, j, 1)$ for $j = a, \ldots, b - 1$.
(2) For $i > a + 1$, execute the following loop:
  for $j = i - 2$ to 1 decreasing by 1
    for $k = j + 1$ to $i - 1$
      $RC(i, j) = RC(i, j) + RP(k, j, 1) \times RC(i, k)$
    next k
  next j

A2. Definition of the Rank Positional Weight

Cooper [8] defined a rank positional weight of task *i* as follows:

$$rpw_i = d_i + \sum_j d_j$$

where, d_i : expected duration of task *i*

$\sum_j d_j$: sum of all expected durations over all successors of task *i*

(Note: a set of successors includes all downstream tasks that receive outputs from the task.)

By adding the summation part in the above definition, the RPW can measure the global importance of a task.

