

# On the Exchange of Information in Collaborative Product Development

R.S. Sreenivas and A. Yassine

Department of Industrial and Enterprise Systems Engineering  
University of Illinois at Urbana-Champaign  
Urbana, IL 61801, USA  
{rsree,yassine}@uiuc.edu

**Abstract**—We develop a dynamic programming (DP) model of the product development (PD) process. We conceptualize PD as a sequence of decisions: whether to incorporate a piece of information that just arrived (i.e. became available) or wait longer. We utilize this formulation to derive optimal decision rules to determine whether (and when) to incorporate newly available information in an archetypal situation in PD. The managerial implications are discussed with several numerical examples.

## I. INTRODUCTION

The next generation *Product Information Management* (PIM) systems have to deal with the concept of evolving information that is partial and incomplete, but potentially useful to other collaborators if shared at the right times. This paper presents a model for the management and control of such information. We consider the archetypal scenario where a collaborative participant is capable of accessing, at any time during development, unreliable, but related, design information, which has the potential to change as the development endeavor progresses. The participant has to decide what the appropriate action should be in response to this information. For instance, she can choose to ignore the information and continue with her original mission, or, she can incorporate the information and modify her goals appropriately in light of this new, but partial, information. The trade-off involved here is that acting upon such information may improve the quality of her work; however, there is a risk of actually deteriorating the quality due to the unreliable nature of the newly available information.

The rest of the paper is organized as follows. Section 2 presents related literature on modelling collaborative design processes. Section 3 contains a brief description of our model, and the form of the optimal incorporation policy for a commonly found scenario is PD is presented in section 4. The Appendix contains the proofs of the various results presented in the body of the paper.

## II. MODELS AND ANALYSIS OF COLLABORATIVE DESIGN PROCESSES

Many optimization models have been proposed to manage information flows within a design process [4], [6], [10], [1], [9], [5]. These models conceptualize engineering design (ED) as a complex network of information processing nodes (i.e.

design tasks) that make decisions under time- and budget-constraints [7], [3]. The core tradeoff in all these models is to strike a balance between the time spent on working alone on a specific design task versus the time spent communicating with other designers. While communication reduces the chance of future rework, it prevents designers from spending time on actually performing their job. The objective function in all these models is to (1) reduce total development/design lead-time by choosing the optimal number of information exchanges between design tasks, or (2) determine the optimal amount of overlap between tasks. Though these models constitute a solid foundation that we build upon, they suffer from serious disadvantages as all these models are concerned with interactions between only two development activities and they do not scale up to large collaborating groups. It is widely acknowledged that if a design activity proceeds without using information from its predecessor activities, design flaws will arise and development performance will deteriorate [4]. This fact suggests that information benefits the development performance [8]. Hence, if we measure information benefit in terms of its impact on development performance, then each piece of information contributes some potential benefit to the collaborator. In general, the cost of incorporation, or simply rework, can be incurred in two ways, financial cost and time delays. Time delay is inevitable, once it is decided that the information is to be incorporated and the rework is started. However, if overtime were allowed, the time delay would be transformed into an expense.

## III. OUR MODEL

In this paper, we assume that development performance,  $Q$ , is measured as the expected profit that the company would obtain by selling the new product. There is ample empirical evidence relating product performance to profitability [3]. Furthermore, we assume that this performance accumulates or evolves as an increasing function of the time spent performing work on the activity [3], [8]. In general, there are many possible shapes that the performance evolution might follow, four of which are shown in Figure 1. If the performance function has a concave shape, the team works with a diminishing rate. Specifically, the evolution is very fast at the beginning and slows down as the activity progresses.

The concave evolution performance represents those activities that are carried out early and further improvement becomes more and more difficult as performance grows, the team gets tired or the resources become limited. The activities with linear evolution are similar to a manufacturing process. The performance increases at a constant rate. The convex evolution is also frequently seen in PD projects. Some activities need a lot of setup, hard thinking and analysis, so the evolution is slow at first. Once the preparation work is done, the performance then progresses rapidly. Finally, the S-curve evolution has also been used to model an activity progress [2]. In this paper we restrict attention to the linear-evolution case in figure 1. More specifically, as we will see in the next section, we suppose that the rate of increase is characterized by the information received by the team. That is, for the case when performance increases linearly (which is all we consider in this paper), new information presented to the team is equivalent to the performance-rates.

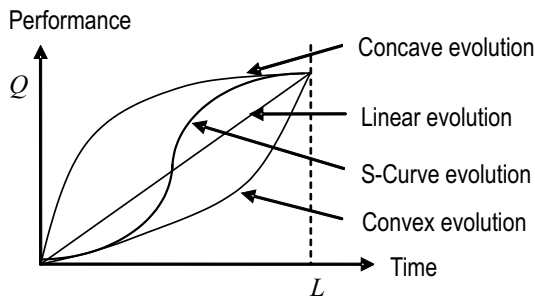


Fig. 1. Sample Performance Evolution Functions.

We consider a single design activity that has a deadline  $L$ . As mentioned in the previous section, we assume that the performance evolution is linear. We divide the time from the beginning of the activity to the deadline into  $N$  equal periods. At the beginning of each period, the designer checks whether the information has changed during the last period, and then makes a decision whether to incorporate it or not. We say that we are at stage  $k$  if there are  $k$  periods remaining. In the beginning (i.e. stage  $N$ ), the design team collects the available information that values  $v_N$ . Based on this information, the team works on their activity at rate  $v_N$ . If the information does not change and no new information arrives, it is the same as the zero-information model, and the deadline performance would be  $v_N L$ . However, the information collected at the beginning may change during the course of development. Or some new information may arrive at a certain stage  $s$ , which can add value to the activity. After evaluating its value  $v_s$ , the team may decide to incorporate the updated (or new) information into the current design, which incurs certain amount of cost to adjust its earlier work. Because of our assumption that rework costs are purely financial, the new information shifts the performance up to  $v_s s$  instantaneously and makes the performance increase at the rate  $v_s$  there after (as shown

in Figure 2). If no incorporation occurs later, the deadline performance would be  $v_s L$ . The team may also decide to ignore this information for the current product and use it in the follow-up products, in order to save the rework cost.

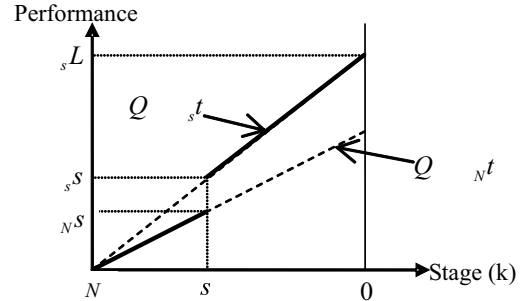


Fig. 2. Performance Evolution with Information Incorporation.

Moreover, we assume that the development performance is measured as the expected profit that the company would obtain by the new product. Then, the designer wants to maximize the expected return (the expected deadline performance minus the total rework cost,  $\sum r_{ki}$ , where  $\sum r_{ki}$  is the cost of rework incurred if the team decides to perform rework at stage  $k$  and the last rework performed at stage  $i$ ). This decision process can be modelled as a dynamic program. Let  $v_k$  be the largest value among all information that has arrived since the latest incorporation until stage  $k$ <sup>1</sup>. The team is at stage  $k$  and in state  $(v_k, v_i)$ , if  $i$  ( $i > k$ ) is the stage at which the last incorporation was done and the value of the last incorporated information was  $v_i$ . There are two options: continue working on the activity based on the information incorporated at stage  $i$  or incorporate the latest information immediately. Suppose the team decides to incorporate the latest information. It costs the team  $r_{ki}$  to perform the rework. Then the team goes to the state  $(v_{k-1}, v_k)$ . If the team decides to ignore the current information and continue working, then the process proceeds to the next stage and the state becomes  $(v_{k-1}, v_i)$ . We are interested in finding an optimal information incorporation strategy that maximizes the expected return. Let  $J_k(v_k, v_i)$  be the *maximum expected return at the deadline* (the expected performance less the total rework cost spent from stage  $k$  onwards), when at stage  $k$  and in state  $(v_k, v_i)$ . Thus, the optimality equation can be written as:

$$J_k(v_k, v_i) = \max\{J_{k-1}(v_{k-1}, v_i), J_{k-1}(v_{k-1}, v_k) - r_{ki}\} \quad (1)$$

Equation 1 can be solved using backward recursion for relatively small number of stages. However, when  $N$  is large, a numerical solution will take a great deal of computation. Instead of a numerical solution, our approach will be to make reasonable assumptions if necessary, to allow for the development of an easy-to-implement policy, in addition to

<sup>1</sup>We imagine  $v_k = \max(\bar{v}_1, \bar{v}_2, \dots, \bar{v}_k)$ , where  $\bar{v}_i$  is the value of the actual information that arrived at stage  $i$ .

reducing the amount of necessary computations. In the next section, we consider the situation where the team is presented with dynamic information, with the proviso that the final piece of information should be incorporated.

#### IV. INFORMATION MODEL THAT REQUIRES A FINAL INCORPORATION

In this section we characterize the optimal incorporation policy when it is mandatory that the final piece of information received by the team be incorporated into the product. The maximization problem in Equation 1 is transformed to a problem that minimizes the total rework cost. We divide the overlapping period into  $N$  stages. We set the state to be  $(i, u)$ , where  $i$  is the number of stages that have elapsed since the last incorporation, and  $u$  indicates whether the current information is the same as the last incorporated information. If the information changes, we let  $u = 1$ ; otherwise,  $u = 0$ . Let  $J_k(i, u)$  denote the minimum expected total rework cost that will be incurred from current stage,  $k$ , to the terminal stage if in state  $(i, u)$ . Then the optimality equation can be written as:

$$J_k(i, 1) = \min\{J_{k-1}(i+1, 1), r(i) + J_k(0, 0)\} \quad (2)$$

$$J_k(i, 0) = p_{k-1}J_{k-1}(i+1, 1), r(i) + (1 - p_{k-1})J_k(0, 0) \quad (3)$$

$$J_o(i, u) = u \times r(i) \quad (4)$$

where  $p_{k-1}$  is the probability that the information will change in stage  $(k-1)$ , and  $r(i)$  is assumed to be a convex function of  $i$ .

**Theorem 4.1:** There exists a set of  $N-1$  integers,  $s_1, s_2, \dots, s_k, \dots, s_{N-1}$ , such that if the process is at stage  $k$  and the last incorporation happened  $i$  stages ago, then it is optimal to incorporate the information as soon as  $i \geq s_k$ .

Theorem 4.1 provides an easy-to-execute policy to the design team. At each stage, the decision maker needs only to check how many stages have passed since the last incorporation. If the number of stages exceeds the cutoff value, the team should incorporate any new information; otherwise, it can wait until the next stage and repeat the process again. Although the explicit formula for  $s_k$  is hard to obtain,  $s_k$  can be computed as long as the specific values of parameters are given. To see how the parameters affect the optimal solution, we use a numerical example where we assume that the cost of rework is  $r(i) = G + \alpha i^\gamma$ .  $G$  is the fixed cost for each incorporation and  $\alpha i^\gamma$  is the variable cost that depends on the number of stages since the last incorporation.  $\alpha$  and  $\gamma$  reflect how much the information and the activity are related and the sensitivity of the design activity, respectively. For simplicity, we assume the probability  $p$  that information changes in stage  $k$ , to be constant. We calculate the optimal cutoff values,  $s_1, s_2, \dots, s_5$  for the last five stages by changing one parameter at a time and fixing the others. We investigate the sensitivity to changes in  $p, G, \alpha$ , and  $\gamma$ .

As shown in Table I,  $s_k$  increases as  $p$  increases. This is intuitive because, if  $p$  is small, the designer is more willing to incorporate information when  $i$  is small in order to reduce the variable incorporation cost, as the chance that the information change again is small. If  $p$  is large (i.e. the information is likely to change later), then the designer would like to accumulate

$p$	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$
0.0	1	1	1	1	1
0.2	3	3	3	3	4
0.4	5	4	4	4	5
0.6	8	6	4	4	5
0.8	10	6	5	4	5

TABLE I

OPTIMAL CUTOFF VALUES AS A FUNCTION OF  $p$ , FOR  $G = 10, \alpha = 0, 4$  AND  $\gamma = 2$ .

$i$  to share the fixed cost across many stages. When  $p = 0$ , the information stays constant, and the stationary information model applies. The team should incorporate the information immediately, that is,  $s_k = 1$  for all  $k$ .

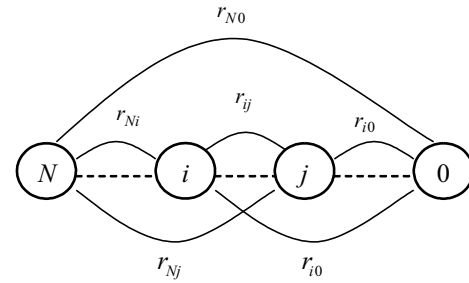


Fig. 3. The Shortest Path Problem for  $p = 1$ .

If  $p = 1$ , the information changes at each stage, the problem becomes a classical shortest path problem as shown in Figure 3. The distance between node  $i$  and node  $j$  is the rework cost incurred if the information is incorporated at stage  $j$ , given the most recent incorporation happens at stage  $i$ . The objective is to find the shortest path from node  $N$  to node  $0$ , which is equivalent to minimizing the total rework cost.

Table II suggests that the optimal cutoff value is also increasing in  $G$ . When  $G = 0$ , the rework cost  $r(i) = \alpha i^\gamma$ , and the cost function is convex if  $\gamma \geq 1$ . To minimize the total rework cost, the designer will incorporate every change immediately after it arrives. When  $G > 0$ , it is not economic to incorporate the information at every stage. The designer would wait until  $i \geq s_k$ , and as a matter of fact, as  $G$  increases, the cutoff value  $s_k$  becomes greater.

Table III demonstrates the relationship between the optimal cutoff value and the sensitivity of the activity. If the task is more sensitive (has a larger  $\alpha$ ), it would cost the designer more to do rework. Note that if  $\alpha = 0$ ,  $s_k$  becomes infinite, which means the team will always incorporate the information at each stage as long as a change arrives. While  $\alpha$  increases, the variable cost becomes more significant such that the optimal cutoff value decreases. Similarly, the coefficient  $\gamma$  has the same impact on the cutoff values (See Table IV). When  $\gamma \leq 1, s_k = \infty$ . Once  $\gamma$  exceeds 1, the cutoff values decrease in  $\gamma$ .

$G$	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$
0	1	1	1	1	1
5	3	3	3	3	3
10	5	4	4	5	5
20	10	8	7	6	6

TABLE II

OPTIMAL CUTOFF VALUES AS A FUNCTION OF  $G$ , FOR  $p = 0.4, \alpha = 0, 4$   
AND  $\gamma = 2$ .

$\alpha$	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$
0.0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
0.2	10	8	7	6	6
0.4	5	4	4	5	5
0.6	4	3	3	4	4
0.8	3	3	3	3	3

TABLE III

OPTIMAL CUTOFF VALUES AS A FUNCTION OF  $\alpha$ , FOR  $G = 10, p = 0, 4$   
AND  $\gamma = 2$ .

### A. Fixed $S$ Policy

We had shown that it is optimal for the team to ignore new information at stage  $k$  as long as the stages elapsed since the last incorporation,  $i$ , is less than  $s_k$ . The computation of the value of  $s_k$  is essentially done by unfolding the recursion in equations 2, 3 and 4 backwards from the deadline. This can be computationally intensive if the number of stages  $N =$  is large. An alternative approach is to use a constant  $s$  for all stages. We call this the *fixed  $s$  policy*. To provide practical applications, we now focus our attention on the fixed  $s$  policy.

Using a fixed  $s$  policy, the team does not care how many stages are remaining. Once  $s$  stages have elapsed since the latest incorporation, the team checks whether any change occurred during this period. If there was a change in information over this period, the team would incorporate it by initiating a rework; otherwise, the team would wait for the next change and incorporate it immediately. As a matter of convention, any incorporation that is scheduled under the fixed  $s$  policy at stage  $i < s$  will be incorporated at the final-stage. We ran a Monte Carlo simulation for the total rework cost incurred by using the optimal incorporation policy and fixed  $s$  policies for a 10-stage problem (see Figure 4). The dotted line represents the cost generated by using the optimal incorporation policy. We find that the total rework cost, as a function of  $s$ , displays a  $U$ -shape curve, and for this particular example,  $s = 5$  is the best among all  $s$  values. This implies that there would be an optimal  $s$  that minimizes the expected total rework cost.

$\gamma$	$s_1$	$s_2$	$s_3$	$s_4$	$s_5$
0	$\infty$	$\infty$	$\infty$	$\infty$	$\infty$
2	5	4	4	5	5
3	2	2	2	2	3

TABLE IV

OPTIMAL CUTOFF VALUES AS A FUNCTION OF  $\gamma$ , FOR  $G = 10, p = 0, 4$   
AND  $\alpha = 0.4$ .

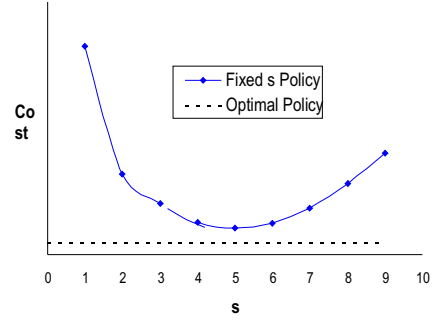


Fig. 4. Comparison of the fixed  $s$ , and the optimal incorporation policy for  $N = 10, G = 7, \alpha = 0.4, \gamma = 2$  and  $p = 0.4$ .

**Theorem 4.2:** If a product development activity has a large number of stages and a fixed  $s$  policy is used, then the following results hold:

- 1) If  $p < \frac{1}{T}$ , then any change should be incorporated immediately after it occurs.
- 2) If  $p \geq \frac{1}{T}$ , then the optimal  $s$ , that minimizes the expected total rework cost, satisfies:

$$s^* + (1 - p)^{s^*} / p = T \quad (5)$$

where  $T = (\frac{G}{\alpha(\gamma-1)})^{1/\gamma}$ .

Part (1) of theorem 4.2 says that if the probability that a change occurs in one stage is small enough, that is, the information revision from the upstream activity is a rare event, then the team does not need to use the fixed  $s$  policy. For such information, once a change occurs, the team should incorporate it immediately. When the information is quite uncertain, part (2) of theorem 4.2 suggests an optimal  $s$  that minimizes the expected total rework cost. Note that the second term on the left hand side of equation 5 is greater than or equal to zero. Thus,  $s^*$  is less than or equal to  $T$ . Also,  $s$  is an integer and must be greater than or equal to 1. Thus, we know that  $s$  could be equal to or  $1, 2, \dots, \lceil T \rceil$ ,  $\lceil T \rceil$  is the smallest integer greater than  $T$ . We calculate the left hand side of equation 5 and compare it to  $T$ . Starting with  $s = 1$  and increasing  $s$  by 1 each time, we pick  $s^*$  to be the smallest value of  $s$  that makes the left hand side of equation 5 greater than  $T$ . Note that  $T$  is constant for any given pair of design activity and feeding information. Figure 45 demonstrates the relationship between the optimal  $s$  and the probability that a change will occur in one stage. It is not surprising that the relationship follows the same pattern as that between the dynamic  $s_k$  and the probability  $p$ . When  $p = 1$ ,  $s^* = T$ , and  $s^*$  decreases as  $p$  decreases. This is quite intuitive: If the probability that a change happens is small, the benefit of waiting for more stages becomes small. Thus, the team would be willing to incorporate the change more frequently.

### B. Policy Comparison

For the dynamic internal information, we provide two policies: the optimal policy that has different cutoff values

at each stage and the fixed  $s$  policy that has constant cutoff value for all stages. As we know, the first one minimizes the total expected rework cost, but the optimal cutoff values are difficult to calculate when the development project is big (e.g. the number of stages is large). Though the fixed  $s$  policy is very simple to obtain, it may incur additional rework cost. The decision maker might be interested in finding out whether the simplification in calculation deserves the extra cost. Hence, we will investigate how much worse the fixed  $s$  policy is compared to the optimal policy.

*Corollary 4.3:* The fixed  $s$  policy is equivalent to the optimal policy when  $p = 1$ .

Corollary 4.3 implies that when the information changes every stage, there is no difference between using the fixed  $s$  policy and the optimal policy. Therefore, the decision maker should just use the fixed  $s$  policy for making decisions regarding whether to incorporate the newest change or not.

## V. CONCLUSION

In this paper, we formulate the decision process of whether or not to incorporate new information in product development as a dynamic programming model. Since we are interested in obtaining managerial insights we have kept the model as simple as possible. In the case where the team has to incorporate the final piece of information that it receives during the course of product development, we show that implementing the optimal incorporation policy reduces to checking if the number of stages that have passed since the last incorporation exceeds a computable threshold.

## ACKNOWLEDGMENTS

This work was supported in part by the National Science Foundation under grants ECS-0000938, ECS-04268321 and CNS-0437415.

## APPENDIX

### Proof of Theorem 4.1

*Lemma 5.1:*  $J_k(i, 1) - r(i)$  is non-decreasing in  $i$ .

*Proof:* (By induction on  $k$ ) For the base case let  $k = 0$ , from equation 4 it is obvious that  $J_0(i, 1) - r(i)$  is non-decreasing in  $i$ , as the induction hypothesis we assume the same for  $J_{n-1}(i, 1) - r(i)$ . Now, from equation 2, we have

$$J_n(i, 1) - r(i) = \min\{(J_{n-1}(i+1, 1) - r(i+1)) + (r(i+1) - r(i)), J_n(0, 0)\}.$$

From the induction hypothesis we know that  $(J_{n-1}(i+1, 1) - r(i+1))$  is non-decreasing. Furthermore,  $(r(i+1) - r(i))$  is non-decreasing due to the convexity of  $r(\bullet)$ . Therefore  $J_n(i, 1) - r(i)$  is also non-decreasing in  $i$ . Thus, the result follows. ■

When at stage  $k$  and in state  $i$ , it is optimal to incorporate the information if  $J_k(i, 1) \geq r(i) + J_k(0, 0)$ . Let:

$$s_k = \min\{s \mid J_k(s, 1) - r(s) \geq J_k(0, 0)\}.$$

By Lemma 5.1, we can see that for all  $i \geq s_k$ ,  $J_k(i, 1) - r(i) \geq J_k(s_k, 1) - r(s_k) \geq J_k(0, 0)$ . Hence, it is obvious that it is

optimal to incorporate the information when at stage  $k$  as soon as  $i \geq s_k$ .

### Proof of Theorem 4.2

*Lemma 5.2:*  $f(s)$ , the average cost for any given  $s$ , is convex in  $s$ .

*Proof:* Let  $h$  be the expected interval between two incorporations. Under the fixed  $s$  policy,  $h(s) = s + (1-p)^s/p$ . It is obvious that  $h(s)$  is increasing in  $s$ . On the other hand, the long-run average cost  $f$  is equal to  $(G + \alpha h^\gamma)/h$ . It follows immediately that  $f$  is convex in  $h$ . Thus,  $f$  is convex in  $s$ . ■

Given that  $f$  is convex in  $s$ , we can find the minimum of  $f$  by setting the first order derivative with respect  $s$  to zero. By doing so, we get equation 5. Note that when

$$p < \frac{1}{\left(\frac{G}{\alpha(\gamma-1)}\right)^{1/\gamma}},$$

equation 5 has no solution. That implies that  $f$  is monotonically increasing. As a result,  $s = 0$  minimizes  $f$ . Part (1) of the theorem 4.2 follows.

### Proof of Corollary 4.3

First, we note that when  $p = 1$ , minimizing the expected value of the total rework cost essentially requires us to solve the shortest-path problem on the rework-cost graph (cf. Figure 4), where the cost associated with any edge that spans  $i$  vertices is  $r(i) = G + \alpha i^\gamma$ . We will now show that the solution to the shortest-path problem is similar in structure to a fixed  $s$  policy when  $\gamma \geq 1$ .

*Claim:* The shortest-path on the rework-cost graph from the  $N$ -th stage to the 0-th stage involves a collection of edges that span a fixed-number (say  $k$ ) of vertices, and at most one edge that spans  $k + 1$  vertices, or  $k - 1$  vertices. Suppose the shortest-path in the rework graph involves  $m$  edges, where the  $i$ -th edge in the path spans  $k_i$  vertices. The cost of this path would be  $\sum_{i=1}^m \alpha(k_i)^\gamma + mG$ , where  $\sum_{i=1}^m k_i = N$ . When  $\gamma \geq 1$ , the optimal solution can take one of three forms:

- 1)  $\lfloor \frac{N}{m} \rfloor = \lceil \frac{N}{m} \rceil = k_i = s$ . In this case the decision-maker performs many incorporations every  $s$  stages.
- 2)  $\lfloor \frac{N}{m} \rfloor \neq \lceil \frac{N}{m} \rceil$  and  $\{(m-1) \times \lfloor \frac{N}{m} \rfloor\} + \lceil \frac{N}{m} \rceil = N$ . In this case the decision-maker performs  $(m-1)$ -many incorporations every  $s = \lfloor \frac{N}{m} \rfloor$  stages, followed by a single (and final) incorporation at the deadline after  $\lceil \frac{N}{m} \rceil$  stages.
- 3)  $\lfloor \frac{N}{m} \rfloor \neq \lceil \frac{N}{m} \rceil$  and  $\{(m-1) \times \lceil \frac{N}{m} \rceil\} + \lfloor \frac{N}{m} \rfloor = N$ . In this case the decision-maker performs  $(m-1)$ -many incorporations every  $s = \lceil \frac{N}{m} \rceil$  stages, followed by a single (and final) incorporation at the deadline after  $\lfloor \frac{N}{m} \rfloor$  stages.

## REFERENCES

- [1] R. Ahmadi and R. H. Wang, "Managing Development Risk in Product Design Processes," *Operations Research*, 47(2) 235-246, 1999.
- [2] G. Barraza, E. Back, and F. Mata, "Probabilistic Monitoring of Project Performance using SS Curves," *Journal of Construction Engineering and Management*, March/April 2000, pp. 142-148.
- [3] M.A. Cohen, J. Eliashberg and T. Ho, "New Product Development: The Performance and Time-to-Market Tradeoff," *Management Science* 42(2), 173-186, 1996.
- [4] A.Y. Ha and E. L. Porteus, "Optimal Timing of Reviews in Concurrent Design for Manufacturability," *Management Science*, 41 (1995), 1431-1447.

- [5] N.R. Joglekar, A. A. Yassine, S. D. Eppinger and D. E. Whitney, "Performance of Coupled Product Development Activities with a Deadline," *Management Science*, 47 (2001), 1605-1620.
- [6] V. Krishnan, S. D. Eppinger and D. E. Whitney, "A Model-Based Framework to Overlap Product Development Activities," *Management Science*, 43 (1997), 437-451.
- [7] T.A. Roemer, R. Ahmadi and R. H. Wang, "Time-Cost Trade-offs in Overlapped Product Development," *Operations Research*, 48 (2000), 858-867.
- [8] S. Thomke and D. E. Bell, "Sequential Testing in Product Development," *Management Science*, 47 (2001), 308-323.
- [9] A.A. Yassine, N. R. Joglekar, D. Braha, S. D. Eppinger and D. E. Whitney, "Information Hiding in Product Development: The Design Churn Effect," *Research in Engineering Design*, 14(3), 2003.
- [10] C.H. Loch and C. Terwiesch, "Communication and Uncertainty in Concurrent Engineering," *Management Science*, 44 (1998), 1032-1048.