Available online at www.sciencedirect.com
ScienceDirect
European Journal of Operational Research 184 (2008) 311–326
EUROPEAN JOURNAL OF OPERATIONAL RESEARCH
www.elsevier.com/locate/ejor

O.R. Applications

Managing the exchange of information in product development ☆

Ali A. Yassine *, Ramavarapu S. Sreenivas, Jian Zhu

Product Development Research Laboratory, Department of Industrial & Enterprise Systems Engineering, University of Illinois at Urbana-Champaign Urbana, Urbana, IL 61801, United States

Received 28 November 2005; accepted 4 October 2006
Available online 14 December 2006

Abstract

In the present paper, we develop a dynamic programming (DP) model of the product development (PD) process. We conceptualize product development 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 analyze different situations that depend on the type, and nature of information that is exchanged: stationary versus dynamic information. We derive optimal decision rules to determine whether (and when) to incorporate for each case. An analysis of the model results in several important findings. First, we must not necessarily incorporate all available information that is related to the design activity. Specifically, once the information collection exceeds certain value, the design team should stop collecting further information. Second, only when past design work accumulates to a certain threshold value should the team include the latest information and perform rework. Large uncertainty of the information and large sensitivity of the design activity makes the incorporation of new information less likely. Finally, managerial implications are discussed with several numerical examples.

Keywords: Product development; Overlapping; Design iteration; Concurrent engineering

1. Introduction

In today's fast-changing and highly competitive markets, product development (PD) is a key source of competitive advantage (Wheelwright and Clark, 1992). To improve PD practices, concurrent engineering (CE) principles have shown great promise in simultaneously reducing development lead-time and cost, and improving product quality as well (Clark and Fujimoto, 1989; Rosenthal, 1992; Smith and Reinertsen, 1998). The success of CE rests largely on the ability of an organization to share timely information among members of a cross-functional development team (Yassine and Braha, 2003). Sometimes this information is preliminary and evolving; however, if released early, it could benefit the PD process in two main ways: (a) provide

☆ This work was supported in part by the National Science Foundation under Grants ECS-0000938, ECS-04268321 and CNS-0437415.
* Corresponding author. Tel.: +1 217 333 8765; fax: +1 217 244 6165.
E-mail address: yassine@uiuc.edu (A.A. Yassine).

0377-2217/$ - see front matter © 2006 Elsevier B.V. All rights reserved.
doi:10.1016/j.ejor.2006.10.042

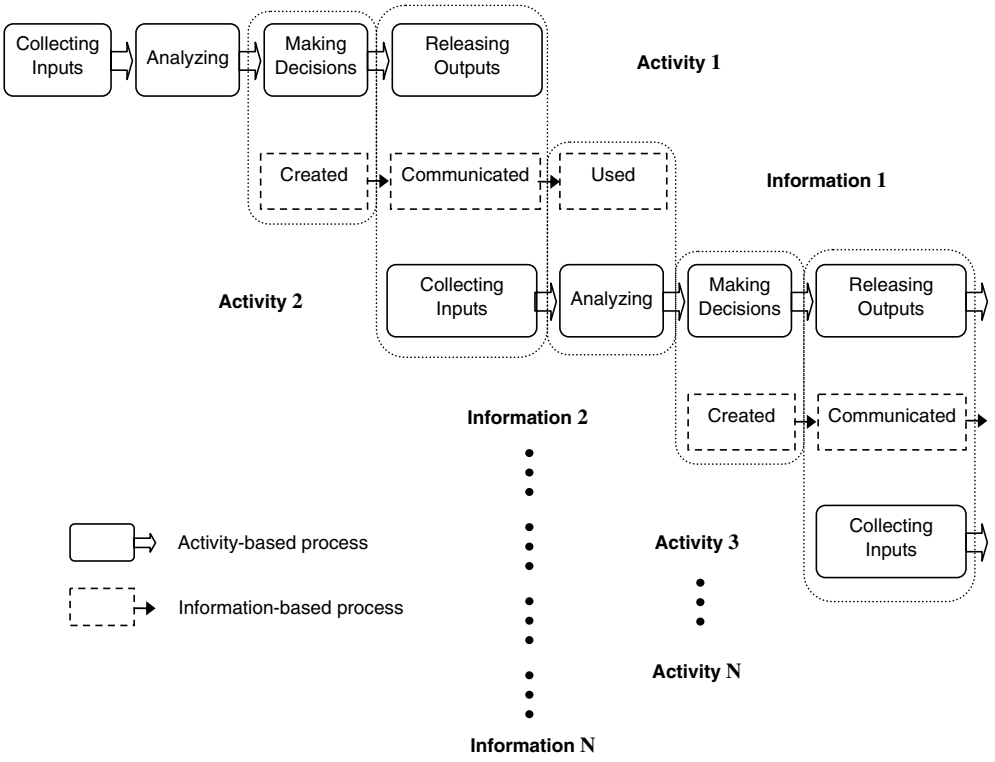

Fig. 1. Product development process from an information processing perspective.

downstream activities with a general sense of direction allowing an earlier start (i.e. compared to a sequential approach) and resulting in an overall time reduction, and (b) provide downstream activities with an early feasibility check resulting in early detection of inconsistencies among the development activities. Therefore, the study of how development information is created, communicated, used, and influence development decisions is instrumental to successful PD practices.

We conceptualize the PD process as a complex information-processing network, punctuated by decision-making (Ullman, 2001). Within this network, each development participant, or team, collects necessary input information, performs analysis, make decisions, and then releases its output (i.e. this output becomes available input information to other development participants or teams in the network).[1] Fig. 1 provides an illustration of this scenario. In this information processing view of PD, information is created when the decisions are made. Then it is communicated to those activities that require this information and it is used when the designers of those activities analyze the problems.

The above scenario is not a pathological case. Many product data management (PDM) systems operate under a similar information-processing configuration (Liu and Xu, 2001; Crow, 2002).[2] For instance, consider the Ford (2002) initiative to coordinate and centralize its distributed product development activities (Bsharah, 2000).[3] This called for flexible information systems and an application for managing and transmitting design documents across various Ford Centers around the world. To support the coordination of design activities worldwide, Ford installed Metaphase, a PDM system from Structural Dynamics Resource Corporation – SDRC. This PDM system organizes the storage

[1] We will use the terms design team and design activity interchangeably throughout this manuscript. The assumption here is that there is a one-to-one correspondence between teams and activities since each design team will be charged with a specific design activity.

[2] The acronym PIM, Product information management, is also used sometimes to mean PDM.

[3] Ford Motor Company launched its Ford (2002) globalization program in January 1994.

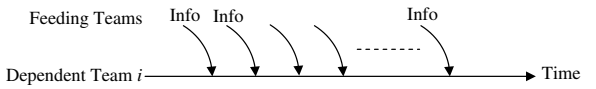

Fig. 2. The continuous arrival of new development information.

and access of design documents generated by the company's existing Computer Assisted Design (CAD) applications, as well as information on specifications and availability of various components of those designs.

Ford's initiative, among other programs at many global organizations, made information access at the fingertips of globally distributed development teams and just a mouse-click away. The main focus of PDM software has been version control and access rights for finalized/released information (Crow, 2002). However, they lack the ability to coordinate, orchestrate or regulate the flow and access of partial/preliminary information and design decisions relating to its use (Bsharah, 2000; Vedapudi, 2000). Developers merely have access rights and can subscribe to services, but acting on the information is yet left out to individual judgment. This paper is about building analytical models of information incorporation in such a distributed and collaborative environment allowing for determining the optimal timing of information exchanges (Yassine et al., 2003).[4]

In this paper, we take the perspective of a single team within the above development network, who is repeatedly faced with the arrival of "somewhat related" information from various sources within and outside the development organization as depicted by Fig. 2. Due to CE practices, some of this information is un-finalized and tends to change in later stages. The team may consider preliminary requisite information that accelerates the development process but offers the potential of rework. We develop a dynamic programming model for this PD environment and analyze several different situations depending on the type and nature of the information being exchanged. An analysis of the model provides several important managerial insights to determine whether (and when) to incorporate new design information. In particular, the main results obtained from the analysis of this model are as follows:

1. If the design information does not change over time (i.e. static), then the development team should either incorporate the information immediately after it arrives or ignore it forever. So, it is not only the information itself that provides value, but the timing of information generation. Therefore, in order to maximize the expected performance at launch, the information that is important input to downstream activities should be frozen early in the development process.
2. If the information evolves over time (i.e. dynamic), then we provide 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 the information; otherwise, it can wait until the next stage and repeat the process again.

The remainder of the paper is organized as follows. In the next section, we present a brief survey of previous studies in the PD area that are most relevant to our work. In Section 3, we discuss the role and benefit of information in PD. Section 4 introduces the general mathematical model, and Sections 5 and 6 analyze the model and characterizes the optimal strategy. Discussions of managerial implications and insights provided by the DP information exchange model are discussed in Section 7. Finally, we summarize the paper and discuss the future research opportunities in Section 8.

2. Literature review

The literature on concurrent engineering (CE) is voluminous. Takeuchi and Nonaka (1986) characterized concurrency as a "rugby team" approach to design compared to the sequential "relay race" approach. Later Clark and Fujimoto (1991) observed that in automobile industry companies with shorter development lead-time frequently overlapped their development activities. Due to these early works, CE has become a core technique for saving development time (Smith and Reinertsen, 1998). In spite of its popularity, the risks associated with CE should not be overlooked. Because it utilizes incomplete information, it is widely recognized that overlapping is more costly due to more frequent communications and increased iterations. Hoedemaker et al. (1999) showed that concurrency is not always a better proposition. Then the

[4] For example, McDaniel (1996) reported that interactions between styling and engineering were only allowed in six weeks intervals.

questions are in what kind of situations CE is favorable and how managers should plan and control CE processes.

Several existing theories and models are developed to address these questions. Ha and Porteus (1995) analyzed a product design for manufacturability project that has a product design phase and a process design phase, which are executed in a parallel fashion. Their purpose was to find a progress review strategy that minimizes the total expected project completion time. A progress review can prevent design flaws, in earlier stages, from moving to later stages, which would result in expensive rework. However, reviewing too frequently consumes too much time, possibly delaying the project completion time. They formulated the problem as a dynamic program and characterized the optimal progress review strategy that minimizes the total expected project completion time. Our model differs from the work of Ha and Porteus by conceptualizing the information much wider, while the information in their model is design flaws.

Krishnan et al. (1997a) investigated the overlapping of two sequentially dependent PD activities. They modeled upstream information as an interval value and introduced two important notions: upstream evolution and downstream sensitivity. Upstream evolution can be interpreted as how close the preliminary upstream information is to its final value, and downstream sensitivity is referred to the relationship between the duration of downstream iteration and the magnitude of the change in the upstream information. Based on these two measures, they formulated a mathematical program to determine the optimal number of iterations and their respective starting times. Although the authors considered uncertainty in the PD process (e.g. the notion of evolution), but they did not model it stochastically.

Similarly, Loch and Terwiesch (1998) presented an analytical model for overlapped upstream and downstream tasks to minimize time-to-market. They modeled preliminary information as engineering changes happen to the upstream task. These changes are released to downstream tasks in batch form and impose rework on the downstream task. They derived an optimal concurrency and communication policy which is affected by uncertainty and dependence.

Our work differs from the above papers in three major aspects. First, all the above papers analyzed the development project without considering development performance. It is the common assumption in these models that the product performance is not influenced by how the development project is organized (Yassine et al., 1999). Krishnan et al. (1997b) discussed how the sequence of decision-making leads to depreciation of product performance. Consequently, in our model, development decisions are made considering their impact on performance. Second, with no exception, all the above models are aimed at time-to-market reduction. However, in practice, most PD projects have pre-specified deadline. Consequently, we consider a development project in a more realistic way and seek to maximize the development performance subject to a deadline constraint (Joglekar et al., 2001).

Third, we focus on a single downstream activity, but those papers investigate the circumstance of two dependent activities, one of which (upstream activity) feeds information to the other (downstream activity). Essentially, the downstream task not only incorporates the information from one upstream task, it also incorporates other information that may or may not come from the development project itself. For example, the fluctuation of interest rate, the change of supplier parts' prices, new customers' preferences, new technology and so on. Upstream design information is just one type of information that is needed to enable downstream tasks.

3. Information in product development

As stated earlier, the present paper views PD as a complex process involving hundreds of decisions in strategy, marketing, finance, engineering, manufacturing and customer service. Hence, information plays a very important role in facilitating and influencing PD decision-making. Krishnan and Ulrich (2001) provided a long list of typical decisions within the context of PD. For example, information about customer needs, available technology and production cost is used in the product concept phase to establish product specifications. Information about these specifications then becomes an important input to further detailed design and prototyping activities. By understanding the benefit and risk of information, better decisions (regarding the use of information) can be made to develop better products. In this section, we discuss the properties of information and performance in PD, which we use, in the next section, to develop models of information flow in PD.

3.1. Development performance

The performance of PD activities can be measured in various ways (O'Donnell and Duffy, 2002). Obviously, development lead-time (and cost) is one measure performance (Roemer et al., 2000). The quality level in terms of the number of open issues remaining (e.g. bugs in software PD) at the time a product is released could be another strong indicator of performance (Yassine et al., 2003). The number of features implemented or supported by this activity also represents performance (Karlsson and Ahlstrom, 1999). The amount of discrepancy between a desired goal and the actual outcome of the activity could also be a measure of performance (O'Donnell and Duffy, 2002).

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 (Zirger and Maidique, 1990; Cohen et al., 1996a,b). Furthermore, we assume that this performance accumulates or evolves as an increasing function of the time spent performing work on the activity (Cohen et al., 1996a,b; Thomke and Bell, 2001). In general, there are many possible shapes that the performance evolution might follow, four of which are shown in Fig. 3 as examples. 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 require a lot of setup work, 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 (Barraza et al., 2000).

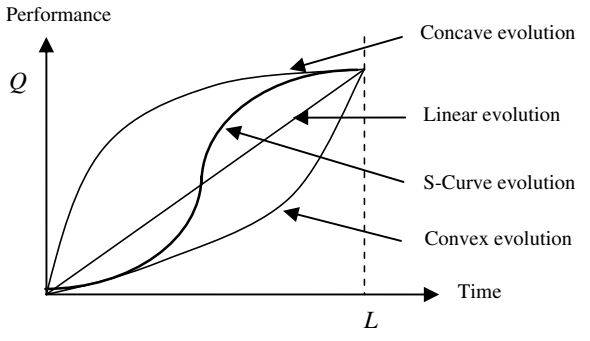

Fig. 3. Sample performance evolution functions.

3.2. Benefit of information

It is widely acknowledged that if a design activity proceeds without incorporating information from its predecessor activities, design flaws will arise and development performance will deteriorate (Ha and Porteus, 1995). This fact suggests that information benefits the development performance (Thomke and Fujimoto, 2000). Hence, to quantify how much a piece of information contributes to a PD activity, we measure the benefit of a piece of information as the rate of development performance evolution over time based on that information. Assuming linear evolution and the benefit of a certain piece of information is υ, then the development performance will be υt if the team works on the development activity based on this information for a period of time t.

Some information does not change in the short term, at least within the duration of the development project. We call this type of information *stationary information*. However, some information can be revised many times, and we define it as *dynamic information*. For example, information from completed activities or information about a proven technology is stationary, while is information from undergoing activities and information about a prospective technology is dynamic. Due to the increasing pressure of competition, designers try to commence design activities earlier by using incomplete information. In light of the interdependence of tasks mentioned earlier, it is very rare that assumptions regarding incomplete information made at the start of an activity are never changed as the activity progresses. As a result, the benefit of the dynamic information changes every time the information is updated.

3.3. Cost of incorporating information

Our model postulates that each time incorporation takes place, a certain amount of cost is incurred. This cost consists of information acquisition cost and incorporation cost. Acquisition cost is the cost that one needs to pay to obtain relevant information. We assume that there is no acquisition cost in our

model. This assumption is practical in large organizations that have established advanced information technology environments. Designers can transfer design information through many media in an almost costless and seamless fashion, for example, telephone, fax, email, web conference etc.

Incorporation cost represents the cost that is required to modify the current design based on the new information if incorporated. 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. Hence, in our model, for simplicity we view the incorporation cost being purely financial.[5]

The cost of rework is determined by three factors. First, as more time is spent on an activity using outdated information, the amount of cumulative work that must be modified will be larger. Hence, rework time is an increasing function of the time between two successive incorporations. Second, rework time is also dependent on how much the information and the activity are related. In general, a change in a major input needs longer rework time than a change in a minor input. The third factor that influences rework time is the degree of sensitivity of the activity. Degree of sensitivity implies the robustness of the activity to changes. In other words, a larger degree of sensitivity indicates a longer rework time. For example, let us consider analytical and physical prototyping. A change in dimension is easy to incorporate in an analytical prototype. By contrast, much more efforts would be required for the same change to a physical prototype. Then we say that physical prototyping has a greater degree of sensitivity than analytical prototyping. Moreover, sensitivity of an activity is also affected by the design method. Toyota's set-based approach (Ward et al., 1995), in which the team works with a set of prospective designs together without making a commitment until necessary, is less sensitive to changes than point-based approaches.

4. The general model

Our work is focused on a PD project that has a fixed deadline (Joglekar et al., 2001). The deadline for the whole project can be the launch date of a new product. For a given design activity, the deadline can be an intermediate milestone.

We consider a single design activity that has a deadline L (see Fig. 3). The team, in charge of this activity, must complete their work by the deadline; otherwise, subsequent activities will be delayed and the total project duration will be increased. Subject to the deadline constraint, the team is trying to maximize its development performance. In some cases, the deadline has to be extended because the minimum performance requirement is not met. The team needs to continue working on the design activity until they achieve the threshold performance. Our model, focusing on maximizing the performance, will still work in this case for the period before the deadline. However, if the development performance does not meet the minimum requirement after the deadline, the problem becomes to minimize the development time to reach the desired performance level, which is not the main objective of this paper. In addition, we assume that the team will not stop before the deadline even if required performance target is achieved. Designers will keep working on their tasks, improving the performance until the due date (Joglekar et al., 2001).

First, we describe how the performance of an activity progresses if no new information arrives during the course of this activity. This model is called the zero-information model. The reader may imagine the zero-information model as an independent activity, which does not need information from other activities. The performance is determined by the parameters which represent the properties of the designer and activity, such as designers' skills and the degree of difficulty of the activity. The performance evolution of such an activity will look like one of the shapes shown in Fig. 3.

Next, we investigate the case where the activity receives new information during the development process. 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 the new information 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 which has a benefit υ_N. Based on this information, the team works on their activity at rate υ_N. If the information does not change and no new information arrives, it is the same as the

[5] This assumption is similar to Thomke and Bell (2001).

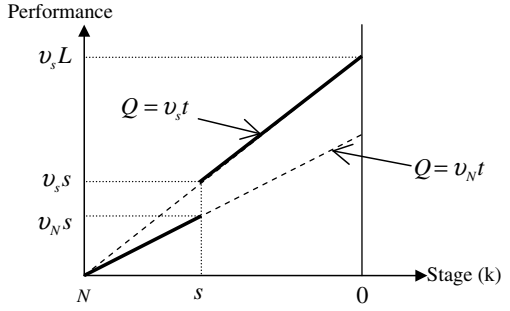

Fig. 4. Performance evolution with information incorporation.

zero-information model, and the deadline performance would be v_NL. However, if at stage s the team incorporates some new information with benefit v_s the new information shifts the performance up to v_ss instantaneously because of our assumption that rework costs are purely financial and makes the performance increases at the rate v_s later on (as shown in Fig. 4).[6] If no incorporation occurs later, the deadline performance would be v_sL.

Finally, the designer's objective is to maximize the expected return (the expected deadline performance minus the total rework cost, $\sum r_{ki}$, where $r_{ki} = r(i-k)$ 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 modeled as a dynamic program. Let $v_k$be the largest benefit among all information that has arrived since the latest incorporation until stage k.[7] 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 benefit 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

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

Eq. (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, we could solve Eq. (1) explicitly (i.e. closed form), or obtain structural results about it (or the optimal policy). The latter option is particularly relevant when we are interested in obtaining managerial insights instead of numerical solutions. Therefore, our approach will be to make reasonable assumptions if necessary, to allow for the development of an easy-to-implement policy, in addition to reducing the amount of necessary computations.

In the next two sections, we will tailor the general model of Eq. (1) to specific models for stationary design information and dynamic design information, respectively. In each case, we will characterize the forms of the optimal incorporation strategy and derive the explicit formulas for the optimal solution to gain managerial insights.

5. Model analysis – stationary information

In this section, we study the case of stationary information. As defined earlier, stationary information does not change after it arrives. To analyze this situation, consider a piece of stationary information that is fed to a design activity and arrives at stage k. Then, its benefit to the activity is v_k, which means the deadline performance of the activity will be v_kL, if the information is used. If the team decides to incorporate the information later at stage s, $k \leqslant s \leqslant 0$, it will cost the team $r_{sN} = r(N-s)$ for rework.

Theorem 1. *The following results hold for each stage k.*

(a) *The optimal incorporation policy for the team receiving stationary information is given as follows*: *If* $(v_k - v_N)L \geqslant r(N-k)$, *the team should incorporate the information immediately*; *otherwise, ignore the information forever.*

[6] We assume if the designer had the $v_s$information in the beginning (instead of v_N), then the performance would have been v_ss by time s. The performance jump at stage s represents the fact that the development team would instantaneously rework its prior work.

[7] We imagine $v_k = \max(\overline{v}_{i-1}, \ldots, \overline{v}_k)$, where $\overline{v}_i$ is the actual benefit of the information that arrived at stage i, as a bundle of information, possibly from various sources. Information is lumped so that a single rework can accommodate many information updates/changes.

(b) *The return at the deadline (the performance less the rework cost) is increasing in k.*

The proof of Theorem 1 is straightforward. Since $r(N-s)$ is decreasing in s and the information will not change again, incorporating the information earlier is always better than incorporating later. Therefore, the team should either incorporate the information immediately after it arrives ($s=k$) or ignore it forever. The benefit of incorporating the information is $(\upsilon_k-\upsilon_N)L-r(N-k)$. If the benefit of incorporating is greater than zero, the team should incorporate the information immediately; otherwise, just ignore the information.

Furthermore, it is interesting to note that the net return achieved at the deadline is decreasing (or increasing) in the arrival time (or stage) of the new information. The earlier the information arrives, the higher the total return achieved at the deadline. So it is not only the information itself that provides benefit, but the timing of information generation as it can change the performance of a development project. The analysis indicates that the earlier the new information becomes available, the more benefit it offers to the activity. In order to maximize the expected return at the deadline, the information that is important input to downstream activities should be frozen early in the development process (Krishnan et al., 1997a).

6. Model analysis – dynamic information

Dynamic information occurs frequently in practice. To accelerate the PD process, upstream activities often release preliminary information to enable downstream activities to start earlier. This information keeps on evolving until it is finalized. Consequently, downstream iteration takes place in order to update past work (Krishnan et al., 1997a).

In this section we introduce two models of dynamic information incorporation. In the first model, we assume it is not mandatory that the requisite design information be incorporated by the dependent activity, but it could be incorporated once only if it increases development performance. In the second model, we impose an additional constraint where requisite information must be incorporated in its final form.

6.1. Dynamic information model 1 – optional incorporation

In this model, we consider development situations where there is no guarantee that the requisite dynamic information will be finalized or frozen before the end of the development process.[8] The development team (in need of such information) has the option of not incorporating this information at all during the current development process, unless they perceive a potential increase in performance.[9]

In this model, the team checks the information that has arrived at the beginning of each stage. Let the benefit of the information arriving at the beginning of any stage, k, be $\overline{\upsilon}_k$, which is given by a random variable with a probability mass function $P(\overline{\upsilon}_k=x)=p_x$ (if no information appears in the current period, we assume that a piece of information of zero benefit has arrived). At the beginning of each stage, the team decides whether to use the new information with the largest benefit (i.e. υ_k) or wait for more information. If, at stage k, the team chooses to incorporate the information, then the deadline performance of the activity would be $\upsilon_k L$ and the team pays $r_{kN}=r(N-k)$ to perform rework. If the team decides to wait and collects more information, the process goes to the next stage $(k-1)$. Therefore, the optimality Eq. (1) can be rewritten as[10]

$$J_k(\upsilon_k,\upsilon_N)=\max\left\{\sum_{\upsilon_{k-1}=0}^{\infty}\left(P_{\upsilon_k,\upsilon_{k-1}}\cdot J_{k-1}(\upsilon_{k-1},\upsilon_N)\right),\upsilon_k L-r(N-k)\right\}\quad\text{for } N\geqslant k\geqslant 1, \tag{2}$$

where $J_0(\upsilon_0,\upsilon_N)=\max\{\upsilon_N L,\upsilon_0 L-r(N)\}$,

where $p_{\upsilon_k,\upsilon_{k-1}}$ is the one-stage transition probability from state υ_k to state υ_{k-1}

[8] This model maybe used especially for external dynamic information. For example, a prospective technology, that may improve development performance but is not fully certain, may be introduced before it is finally proven (Krishnan and Bhattacharya, 2002).

[9] In many cases, once a development decision is made, large amount of resources are committed based on current information and the impact of incorporating future information updates can be prohibitive (e.g. building a comprehensive physical prototype for an automobile). Thus, in these scenarios we assume that the team could incorporate the information once and ignore future updates, which could be used in future product generations.

[10] This is similar to an optimal stopping problem (OSP); however, in our setup we do not forfeit the maximum benefit. That is, unlike OSP, we always hold on to the best offer. In addition, the reward is only collected at the deadline and the problem does not finish when the information is incorporated.

$$p_{v_k,v_{k-1}} = \begin{cases} \sum_{x=0}^{v_k} p_x & \text{if } v_{k-1} = v_k; \\ p_{v_{k-1}} & \text{if } v_{k-1} > v_k; \\ 0 & \text{otherwise.} \end{cases} \quad (3)$$

Theorem 2. *The following results hold for each stage k.*

(a) *The team would only incorporate the newest information that arrives in the most recent period.*
(b) *It is optimal to incorporate the newest information, if its added performance* $(v_k - v_N)L$ *is greater than the rework cost* $r_{kN} = r\ (N-k)$ *and* v_k *is at least as large as a cutoff value* ψ_k*, which is the smallest number that satisfies*

$$E[(v_{k-1} - \psi_k)^+]L \leqslant [r(N-k+1) - r(N-k)], \quad (4)$$

and

$$(v_{k-1} - \psi_k)^+ = \max\{v_{k-1} - \psi_k, 0\}.$$

For easier readability of the paper, the proof of Theorem 2, as well as other proofs, is placed in the appendix. Theorem 2 provides explicit conditions that are sufficient to make the optimal decision. Under this policy, the process terminates at the first stage where condition (4) is met. Condition (4) actually compares incorporating the information at the current stage (resulting in a performance improvement of $(v_k - v_N)L - r(N-k)$) with waiting for exactly one more stage and then incorporating (resulting in an expected performance improvement of $E[(v_{k-1} - v_k)^+]L - r(N-k+1)$). The left hand side of condition (4) can be interpreted as the expected extra benefit of waiting one more stage. Since the state cannot decrease (i.e. the largest benefit ever received cannot decrease in time), the expected extra benefit of waiting decreases because $(v_{k-1} - v_k)^+$ decreases in v_k. The right hand side of condition (4) represents the additional cost required for waiting one more stage, which is increasing in k. It can be intuitively seen that the additional cost is small at the beginning and can be compensated by the extra benefit. The overall gain is positive, and the expected return increases. As the process proceeds, the extra benefit decreases; however, the additional cost increases. At the first stage when the extra benefit is smaller than the additional cost, the team should not wait and must incorporate the information at once.

Theorem 2 implies that, in order to achieve a better outcome of the development activity, it is important to have a good estimate of the requisite information. If the information is underestimated, then the expected extra benefit gained by waiting one more stage is smaller than its actual value. As a result, the team would incorporate the information earlier than the optimal time. If the information is overestimated, the design team will see a false large extra benefit that induces the decision maker to delay incorporation more and leads him to miss the optimal incorporation time. Both situations make the design outcome worse off.

6.2. Dynamic information model 2 – mandatory incorporation

This model differs from the previous dynamic information model in two ways: (1) the requisite information has a specific time to be finalized, and (2) the dependent design team must include/incorporate the finalized requisite information. For example, the deadline of an upstream activity may be the date when the requisite information is finalized. Upon reaching the deadline, the upstream development team stops working on its current activity and moves on to other tasks. As a result, the output information will not change after that. Also, the final upstream information is usually a set of constraints to the downstream activity. So, if the dependent design team does not include the finalized upstream information, its design would be infeasible.

Since the finalized upstream information must be incorporated and the performance is only determined by the benefit of the latest incorporated information, then the benefit of preliminary information does not play any role in this model. Thus, the maximization problem in Eq. (1) is transformed to a problem that minimizes the total rework cost.

In this case, the incorporation strategy will be useful only from the beginning of the feeding activity to the finalization time of its output (we call this the overlapping period). As in the earlier sections, we divide the overlapping period into N stages. We set the state to be (l,u), where l 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 $J_k(l,u)$ denote the minimum expected total rework cost that will be incurred from current stage, k, to the terminal stage if in state (l,u). Then the optimality equation can be written as

$$J_k(l,1) = \min\{J_{k-1}(l+1,1), r(l) + J_k(0,0)\}, \quad (5)$$

$$J_k(l,0) = p_{k-1} \cdot J_{k-1}(l+1,1) + (1-p_{k-1}) \cdot J_{k-1}(l+1,0) \quad (6)$$

$$J_0(l,u) = u \cdot r(l), \quad (7)$$

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

Theorem 3. *There exists a set of $N-1$ integers, $s_1, s_2, \ldots, s_k, \ldots, s_{N-1}$, such that if the process is at stage k and the last incorporation happened l stages ago, then it is optimal to incorporate the information as soon as $l \geqslant s_k$.*

Theorem 3 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 the 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.

6.2.1. Numerical example and sensitivity analysis

To see how the parameters affect the optimal solution, we use an automobile product development example, previously reported by McDaniel (1996) and Yassine et al. (2003), which involves an industrial design activity and an engineering design activity. Industrial design is the earliest of all physical design activities, and changes in this activity can easily cascade into later development activities causing costly rework. Information exchanges from industrial design to engineering design take the form of wireframe CAD data generated from clay model scans, referred to as scan transmittals of surface data. Scan transmittals are scheduled at roughly six-week intervals if any design changes occur. Assuming the industrial design and engineering design start at the same time, the industrial design activity is allotted approximately 52 weeks for completion. With this setup, the engineering design activity can be viewed as a 9-stage mandatory incorporation model with dynamic information.

We assume that the cost of rework is $r(l) = G + \alpha l^{\gamma}$. G is the fixed cost for each incorporation, which is measured as number of hours that the engineering design group needs to review and plan the changes. αl^{γ} is the variable cost required to incorporate the industrial design changes, which depends on the number of stages since the last incorporation. α and γ reflect how much the industrial design information and the engineering design activity are related and the sensitivity of the engineering design activity, respectively. For simplicity, we assume the probability that the industrial design changes in stage k to be constant, p, during the development course. We calculate the optimal cutoff values, $s_1, s_2, \ldots, s_5$ 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, α, and γ.

As shown in Table 1, s_k increases as p increases. This is intuitive because, if p is small, the engineering designer is more willing to incorporate information when l 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 engineering designer would like to accumulate l 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, $s_k = 1$.

If $p = 1$, the information changes at each stage, the problem becomes a classical shortest path problem as shown in Fig. 5. 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 minimize the total rework cost.

Table 2 suggests that the optimal cutoff value is also increasing in G. When $G = 0$, the rework cost $r(l) = \alpha l^{\gamma}$. The cost function is convex if $\gamma \geqslant 1$. To minimize the total rework cost, the engineering designer will incorporate every change immediately after it arrives. When $G > 0$, it is not economic to

Table 1
Optimal cutoff values as a function of p, given $G = 10$, $\alpha = 0.4$, and $\gamma = 2$

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

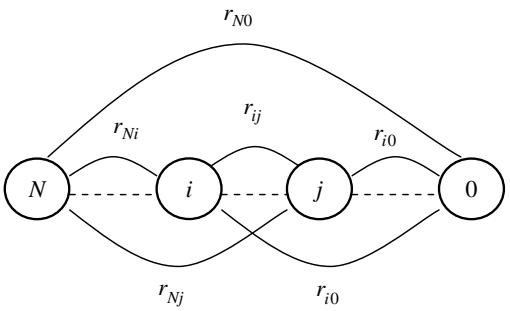

Fig. 5. The shortest path problem for $p = 1$.

Table 2
Optimal cutoff values as a function of G, given $\alpha = 0.4$, $p = 0.4$, and $\gamma = 2$

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

incorporate the information at every stage. The engineering designer would wait until $l \geqslant s_k$, and as a matter of fact, as G increases, the cutoff value s_k becomes greater.

Table 3 demonstrates the relationship between the optimal cutoff value and the sensitivity of the design activity. If it is more sensitive (has a larger α), the engineering design would cost the designer more to do rework. Note that if $\alpha = 0$, s_k becomes infinite, which means the team will always wait until the last stage to incorporate the information. While α increases, the variable cost becomes more significant such that the optimal cutoff value decreases. Similarly, the coefficient γ has the same impact on the cutoff values (see Table 4). When $\gamma \leqslant 1$, $s_k = \infty$. Once γ exceeds 1, the cutoff values decrease in γ.

6.2.2. Fixed s policy

In the previous subsection, we have 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, l, is less than s_k. The computation of the value of s_k is essentially done by unfolding the recursion in Eqs. (5)–(7) 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 fixed s policy. To provide practical applications, we now focus our attention on the fixed s policy.

Table 3
Optimal cutoff values as a function of α, given $G = 10$, $p = 0.4$, and $\gamma = 2$

α	s_1	s_2	s_3	s_4	s_5
0.0	∞	∞	∞	∞	∞
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 4
Optimal cutoff values as a function of γ, given $G = 10$, $p = 0.4$, and $\alpha = 0.4$

γ	s_1	s_2	s_3	s_4	s_5
0	∞	∞	∞	∞	∞
2	5	4	4	5	5
3	2	2	2	2	3

Using 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 $l < s$ 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 the automobile development example (see Fig. 6). 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, $s = 3$ is the best among all $s$values. This implies that there would be an optimal$s$ that minimizes the expected total rework cost.

Theorem 4. *If a product development activity has a large number of stages and a fixed s policy is used, then the following results hold*:

(a) *If $p < 1/T$, then any change should be incorporated immediately after it occurs.*
(b) *If $p \geqslant 1/T$, then the optimal s, that minimizes the expected total rework cost, satisfies*:

$$s* + (1-p)^{s*}/p = T, \tag{8}$$

where

$$T = \sqrt[\gamma]{G/\alpha(\gamma-1)}.$$

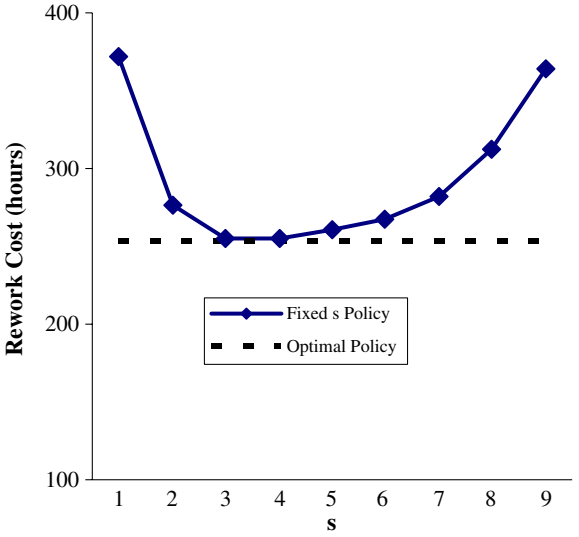

Fig. 6. Total rework cost vs. incorporation policy, given $N = 9$, $G = 40$, $\alpha = 0.5$, $\gamma = 1.1$, and $p = 0.6$.

Part (a) 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 (b) suggests an optimal s that minimizes the expected total rework cost.

Note that the second term on the left hand side of Eq. (8) 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 $1, 2, \ldots$ or $[T]$, $[T]$ is the smallest integer greater than T. We calculate the left hand side of Eq. (8) 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 Eq. (8) greater than T. Note that T is constant for any given pair of design activity and feeding information.

Fig. 7 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. When $p = 1$, $s^* = T$. 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.

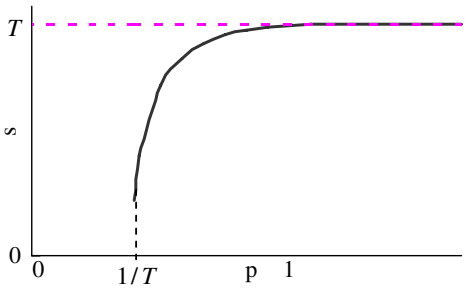

Fig. 7. Optimal s as a function of p.

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

Corollary 1 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.

Corollary 2. *The cost gap between using the fixed s policy and the optimal policy decreases as p increases in the neighborhood of* $p = 1$.

When $p = 1$, the cost gap between using the two policies is zero. As p decreases, the cost gap increases. Therefore, the team should use the fixed s policy instead of the optimal policy, if it is receiving information with high uncertainty, because the cost difference is small. For the case of more certain information, the cost gap is larger and the incentive of using the fixed s policy gets smaller. The team may use the optimal policy to save the total rework cost.

We ran Monte Carlo simulations for different model parameters and plot the relationship between the cost gap and the uncertainty. The curve shows

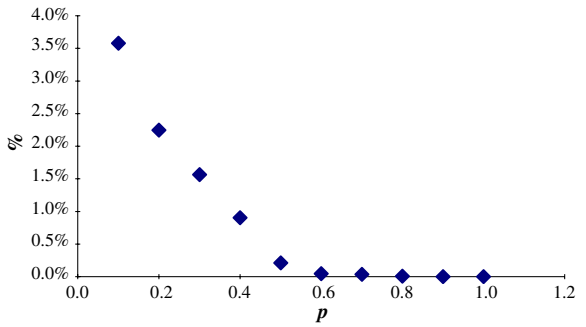

Fig. 8. Cost gap as a function of p.

the decreasing trend that is dictated in Corollaries 1 and 2 (see Fig. 8).

Corollaries 1 and 2 provide a good idea about the difference between the optimal policy and the fixed s policy. When the information is certain to change, the two policies are exactly same. As the uncertainty of the information decreases, the advantage of using the optimal policy increases. Therefore, a particular design team may decide to use the optimal policy when the uncertainty is smaller than a certain value, and use the fixed s policy when the uncertainty is greater than that value.

7. Discussion

In this section, we discuss several important managerial implications based on the proposed information incorporation decision model. The first finding is that the team must not necessarily always incorporate all the relevant available information that may seem useful for the design activity. Clearly, the more information incorporated, the better the new product. Hence, design teams tend to incorporate every piece of information. However, such a strategy may not be optimal as a result of the rework cost that might be incurred. It is not only the information itself that offers value, but also the timing of information arrival. Incorporating a piece of information that arrives approaching the deadline could cost huge amount of resources and time to modify the design. Yassine et al. (2003) study the "design churn" phenomenon and argue that information delay is the major cause of churn. Our result suggests that the team should give up pursuing further information when its current information collection exceeds a critical value. By using this policy, the team sacrifices the prospective design improvement by including further information to avoid unnecessary design churn. As explained in Section 6.2, the critical value depends on estimation of future information and the stage that the decision process is in. If the information arrives in the early stages, it is of little use. Then, the design team would expect a large increment in future information and is more likely to wait longer to collect further information. In contrast, if the information received in the early stages is of great benefit, then the team would expect small improvements that the future information could offer. Thus, the team is unwilling to pay extra cost that cannot be compensated by the benefit.

Second, the analysis of the numerical example gives several interesting insights. The information incorporation policy is mainly determined by two factors: the uncertainty of the information and the cost of incorporating the information. The cost of incorporation can be further divided into fixed cost and variable cost. When the uncertainty of the information is very high (i.e., depending on the type of product being developed and/or technology being used), it is essential to balance the tradeoff between the fixed and variable costs. For the case of high fixed cost and low variable cost, the cutoff value for each stage will be large and, in turn, incorporation happens less frequently. On the other hand, for the case of small fixed cost and large variable cost, we would expect the opposite. That is, the cutoff value will be small and the team incorporates the information more frequently.

If the uncertainty of the information is low, the team has more confidence in the current information. Then, the team is more willing to incorporate information early and save the incorporation cost. Thus, the cutoff values would be smaller than the case when more uncertain information is received. In our model, we set the uncertainty to be constant. However, information produced by different activities has different properties. For the information generated by a design activity with convex evolution, the uncertainty is small at the early stages and become larger in later stages. Therefore, the team using this information is more willing to incorporate in the early stages and more reluctant to incorporate in later stages. On the contrary, if the information is produced by a design activity with concave evolution, changes are more likely to happen in the early stages. Then the cutoff value would be large at the beginning and smaller at the end.

Lastly, we investigate an alternative strategy, fixed s policy. It is interesting that using fixed s policy does not provide any improvement when the

information is quite certain. For this case, the team should simply incorporate any change immediately after it occurs. When the uncertainty of the information is high, we suggest an optimal fixed s policy that minimizes the expected total rework cost, and the optimal s is determined by the structure of the rework cost and the uncertainty of the information. Furthermore, when we compared the total rework cost generated by the optimal policy with that of the fixed s policy, it is interesting that the two policies are equivalent when information changes in every stage. Furthermore, the cost gap of using these two policies increases when the uncertainty of the information decreases. Thus, the design team should use the fixed s policy when the information uncertainty is large, and use the optimal policy when the uncertainty is small.

8. Summary and future work

In this paper, we formulate the decision process of whether or not to incorporate new information in product development as a dynamic programming model. We first develop a general model and then tailor it to three stylized models: stationary information, dynamic information with optional single incorporation, and dynamic information with forced final incorporation.

We have kept the model as simple as possible to obtain managerial insights. Several extensions of the model are possible. First, the model can be modified to be a continuous-time problem. Second, in some cases the time delay due to rework cannot be transformed to a financial cost. Thus, tradeoffs between time, cost, and performance would be more complicated and worthy of further research. Furthermore, in our model, we have only two options vis-à-vis reacting to new information: incorporate or not. There could be other options. For example, one group of the team could continue working with the old information while another could work with the new information. Thus, the model becomes a dynamic resource allocation problem. Finally, it would be interesting to look for the optimal time of incorporations considering a limited and fixed number of incorporations.

In addition, we suggest using the fixed s policy for projects that have a large number of stages and provide the optimal value of s. The cost gap between the optimal incorporation policy and the fixed s policy deserves both further theoretical and empirical research.

Appendix

Proof of Theorem 2. Let $S_k = \{\phi_k : E[(\upsilon_{k-1} - \phi_k)^+] \leqslant [r(N-k+1) - r(N-k)]/L\}$.

From its definition, and Eq. (3), we note that $S_k \subseteq \{0,1,2,\ldots,\}$ is an infinite subset of integers with the following properties:

1. If $\upsilon_k \in S_k$ and if $\upsilon_{k-1} > \upsilon_k$, then $\upsilon_{k-1} \in S_k$ too. That is, the set S_k is *right-closed*, and it has a unique minimal-element, which is denoted by ψ_k.
2. If $\upsilon_k \in S_k$ and $\upsilon_{k-1} \notin S_k$, then from the previous observation, $\upsilon_{k-1} < \upsilon_k$. From Eq. (3), we can infer $P_{\upsilon_k,\upsilon_{k-1}} = 0$.

Claim: For $\upsilon_k \in S_k$ (i.e. $\upsilon_k \geqslant \psi_k$), $J_k(\upsilon_k, \upsilon_N) = \max\{\upsilon_N L, \upsilon_k L - r(N-k)\}$.

This claim can be established by induction on k.

The base-case: $k = 0$. That is, the decision maker is at the final stage, and any incorporation decision at this stage will incur a cost of $r(N)$. Obviously, $J_0(\upsilon_0, \upsilon_N) = \max\{\upsilon_N L, \upsilon_0 L - r(N)\}$, and $\psi_0 = 0$ because $r(N+1) = \infty$. That is, any rework past the deadline has an infinite cost.

As the induction hypothesis let us suppose that for $\upsilon_k \in S_k$, where $0 \leqslant k \leqslant (n-1)$, $J_k(\upsilon_k, \upsilon_N) = \max\{\upsilon_N L, \upsilon_k L - r(N-k)\}$.

To establish the induction step, let $k = n$ where $k \geqslant 1$ and $\upsilon_n \in S_n$. Then, from Eq. (2):

$$
\begin{aligned}
&J_n(\upsilon_n, \upsilon_N) \\
&= \max\left\{ \sum_{\upsilon_{n-1}=0}^{\infty} (p_{\upsilon_n,\upsilon_{n-1}} \cdot J_{n-1}(\upsilon_{n-1}, \upsilon_N)), \upsilon_n L - r(N-n) \right\} \\
&= \max\left\{ \sum_{\upsilon_{n-1} \in S} (p_{\upsilon_n,\upsilon_{n-1}} \cdot J_{n-1}(\upsilon_{n-1}, \upsilon_N)), \upsilon_n L - r(N-n) \right\} \\
&= \max\left\{ \sum_{\upsilon_{n-1} \in S} (p_{\upsilon_n,\upsilon_{n-1}} \cdot \max\{\upsilon_N L, \ \upsilon_{n-1} L - r(N-n+1)\}), \right. \\
&\qquad \left. \upsilon_n L - r(N-n) \right\}.
\end{aligned}
$$

Since $\upsilon_n \in S_n$ it follows that $E[(\upsilon_{n-1} - \upsilon_n)^+]L - r(N-n+1) \leqslant r(N-n)$, which implies:

$$
\begin{aligned}
J_n(\upsilon_n, \upsilon_N) &= \max\{\upsilon_N L, \upsilon_n L + E[(\upsilon_{n-1} - \upsilon_n)^+ \\
&\qquad \cdot L - r(N-n+1), \upsilon_n L - r(N-n)\} \\
&= \max\{\upsilon_N L, \upsilon_n L - r(N-n)\}.
\end{aligned}
$$

We will use the above claim to prove part (b) of the theorem. If $\upsilon_n \in S_n$, then υ_n must be incorporated at the nth stage if $v_n L - r(N-n) \geqslant v_N L \Rightarrow (v_n - v_N)$

$L \geqslant r(N-n)$. Also, $v_n \in S_n \Rightarrow v_n \geqslant \psi_n$. If $v_n \notin S_n$, $E[(v_{n-1}-v_n)^+]L - r(N-n+1) > r(N-n)$, which implies that:

$$\begin{aligned} J_n(v_n, v_N) &= \max\{v_N L, v_n L + E[(v_{n-1}-v_n)^+] \\ &\quad \cdot L - r(N-n+1), v_n L - r(N-n)\} \\ &= \max\{v_N L, v_n L + E[(v_{n-1}-v_n)^+] \\ &\quad \cdot L - r(N-n+1)\}. \end{aligned}$$

Therefore, there will be no incorporations at the nth stage, and decision-maker will proceed to the $(n-1)$th stage. This completes the proof of part (b).

If $n > k$, and the information that arrived at the nth stage was incorporated in the kth stage, it follows from the above result that $v_n \geqslant \psi_k$ and $J_k(v_n, v_N) = \max\{v_N L, v_n L - r(N-k)\}$. Additionally, since there was no incorporation at the nth stage, $v_n < \psi_n$ and $J_n(v_n, v_N) = \max\{v_N L, v_n\ L + E[(v_{n-1} - v_n)^+] \cdot L - r(N-n)\}$. Since $E[(v_{n-1}-v_n)^+] \cdot L \geqslant 0$ and $r(\cdot)$ is convex, it follows that $J_k(v_n, v_N) \leqslant J_n(v_n, v_N)$.This should have resulted in an incorporation at the nth stage itself. This completes the proof of part (a) of the theorem.

Proof of Theorem 3

Lemma 1. *$J_k(l,1) - r(l)$ is non-decreasing in l.*

Proof (*By induction on k*). Base case $k = 0$: From Eq. (7) it is obvious that $J_0(l,1) - r(l)$ is non-decreasing in l, as the induction hypothesis we assume the same for $J_{n-1}(l,1) - r(l)$. Now, from Eq. (5),

$$\begin{aligned} J_n(l,1) - r(l) = \min\{&[J_{n-1}(l+1,1) - r(l+1)] \\ &+ [r(l+1) - r(l)], J_n(0,0)\}. \end{aligned}$$

Because of the induction hypothesis $[J_{n-1}(l+1,1) - r(l+1)]$ is non-decreasing. Furthermore, $[r(l+1) - r(l)]$ is non-decreasing due to the convexity of $r(\cdot)$. Therefore $J_n(l,1) - r(l)$ is also non-decreasing in l. Thus, the result follows. □

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

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

By Lemma 1, we can see that for all $l \geqslant s_k$, $J_k(l,1) - r(l) \geqslant J_k(s_k,1) - r(s_k) \geqslant J_k(0,0)$. Hence, it is obvious that it is optimal to incorporate the information when at stage k as soon as $l \geqslant s_k$. □

Proof of Theorem 4

Lemma 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 to s equal to zero. By doing so, we get Eq. (8). Note that when $p < 1/\sqrt[\gamma]{G/\alpha(\gamma-1)}$, Eq. (8) has no solution. That implies that f is monotonically increasing. As a result, $s = 0$ minimizes f. Part (a) follows. □

Proof of Corollary 1. 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. Fig. 5), where the cost associated with any edge that spans $i$vertices is $r(l) = G + \alpha l^\gamma$. We will now show that the solution to the shortest-path problem is similar in structure to a fixed s policy when $\gamma \geqslant 1$.

Claim: The shortest-path on the rework-cost graph from the Nth stage to the 0th 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 ith 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 $\gamma \geqslant 1$, the optimal solution can take one of three forms:

Form 1: $\lfloor \frac{N}{m} \rfloor = \lceil \frac{N}{m} \rceil = k_i = s$. In this case the decision-maker performs m many incorporations every s stages.[11]

Form 2: $\lceil \frac{N}{m} \rceil \neq \lfloor \frac{N}{m} \rfloor$ and $(m-1)\lfloor \frac{N}{m} \rfloor + \lceil \frac{N}{m} \rceil = N$. 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.

[11] $\lfloor \frac{N}{m} \rfloor$ denotes the *floor* of $\frac{N}{m}$, which is the largest integer that is smaller than $\frac{N}{m}$. Similarly, $\lceil \frac{N}{m} \rceil$ denotes the *ceiling* of $\frac{N}{m}$, which is the smallest integer that is greater than $\frac{N}{m}$.

Form 3: $\lceil \frac{N}{m} \rceil \neq \lfloor \frac{N}{m} \rfloor$ and $(m-1)\lceil \frac{N}{m} \rceil + \lfloor \frac{N}{m} \rfloor = N$. In this case, $s = \lceil \frac{N}{m} \rceil$ stages. □

Proof of Corollary 2. Let $C_1(p)$ and $C_2(p)$ denote the expected total rework cost of using the optimal policy and the fixed s policy for a particular value of p, respectively. For the cost structure in our model, $C_1(p)$ and $C_2(p)$ are both continuous and differentiable. Also, it is obvious that $C_1(p)$ and $C_2(p)$ are both monotonically increasing in p. We know that $C_1(p) \leqslant C_2(p)$ for all p, and $C_1(1) = C_2(1)$. These observations indicate there is a $0 \leqslant p^* \leqslant 1$, defined as the largest number where the following inequality holds: $\forall \bar{p} \in [p^*, 1], \left(\frac{\mathrm{d}}{\mathrm{d}p} C_1(p)\right)_{p=\bar{p}} \geqslant \left(\frac{\mathrm{d}}{\mathrm{d}p} C_2(p)\right)_{p=\bar{p}}$. It follows that the value of $C_2(p) - C_1(p)$ should decrease with an increase in p in the interval $[p^*, 1]$. □

References

Barraza, G., Back, E., Mata, F., 2000. Probabilistic monitoring of project performance using SS curves. Journal of Construction Engineering and Management (4), 142–148.

Bsharah, R., Deploying Advanced PDM Capabilities at Ford Motor Company. In: Information Technology for Engineering and Manufacturing (ITEM 2000) conference, June12–13, 2000. <http://www.mel.nist.gov/div826/msid/sima/item2000/item2000.htm>.

Clark, K.B., Fujimoto, T., 1989. Lead time in automobile product development explaining the Japanese advantage. Engineering and Technology Management 6, 25–58.

Clark, K.B., Fujimoto, T., 1991. Product Development Performance: Strategy, Organization, and Management in the World Auto Industry. Harvard University Press, Cambridge, MA.

Cohen, M.A., Eliashberg, J., Ho, T., 1996a. New product development: The performance and time-to-market tradeoff. Management Science 42 (2), 173–186.

Cohen, M., Eliashberg, J., Ho, T-H., 1996b. New product development: The performance and time-to-market tradeoff. Management Science 42 (2), 173–186.

Crow, K., 2002. Product Data Management/Product Information Management. <http://www.npd-solutions.com/pdm.html> (accessed December 2004).

Ford, D.N., Sterman, J.D., 2002. Iteration Management for Reduced Cycle Time in Concurrent Development Projects, Working paper.

Ha, A.Y., Porteus, E.L., 1995. Optimal timing of reviews in concurrent design for manufacturability. Management Science 41, 1431–1447.

Hoedemaker, G.M., Blackburn, J.D., Van Wassenhove, L.N., 1999. Limits to concurrency. Decision Sciences 30, 1–18.

Joglekar, N.R., Yassine, A.A., Eppinger, S.D., Whitney, D.E., 2001. Performance of coupled product development activities with a deadline. Management Science 47, 1605–1620.

Karlsson, C., Ahlstrom, P., 1999. Technological level and product development cycle time. Journal of Product Innovation Management 16, 352–362.

Krishnan, V., Bhattacharya, S., 2002. Technology selection and commitment in new product development: The role of uncertainty and design flexibility. Management Science 48, 313–327.

Krishnan, V., Ulrich, K.T., 2001. Product development decisions: A review of the literature. Management Science 47, 1–21.

Krishnan, V., Eppinger, S.D., Whitney, D.E., 1997a. A model-based framework to overlap product development activities. Management Science 43, 437–451.

Krishnan, V., Eppinger, S.D., Whitney, D.E., 1997b. Simplifying iterations in cross-functional design decision making. Journal of Mechanical Design 119, 485–493.

Liu, D., Xu, X., 2001. A review of web-based product data management systems. Computers in Industry 44, 251–262.

Loch, C.H., Terwiesch, C., 1998. Communication and Uncertainty in Concurrent Engineering. Management Science 44, 1032–1048.

McDaniel, C.D., 1996. A linear systems framework for analyzing the automotive appearance design process. Master's Thesis (Mgmt./EE), MIT, Cambridge, MA.

O'Donnell, F., Duffy, A., 2002. Modeling design development performance. International Journal of Operations and Production Management 22 (11), 1198–1221.

Roemer, T.A., Ahmadi, R., Wang, R.H., 2000. Time-cost tradeoffs in overlapped product development. Operations Research 48, 858–867.

Rosenthal, S.R., 1992. Effective Product Design and Development. Irwin.

Smith, P.G., Reinertsen, D.G., 1998. Developing Products in Half the Time. Van Nostrand Reinhold.

Takeuchi, H., Nonaka, I., 1986. The new product development game. Harvard Business Review (January–February), 137–146.

Thomke, S., Bell, D., 2001. Sequential testing in product development. Management Science 47 (2), 308–323.

Thomke, S., Fujimoto, T., 2000. The effect of front-loading problem-solving on product development performance. Journal of Product Innovation Management (March).

Ullman, D., 2001. Robust decision making for engineering design. Journal of Engineering Design 12 (1).

Vedapudi, M., 2000. Requirements for a product information management (PIM) infrastructure to support partner programs, System Design and Management Thesis, Massachusetts Institute of Technology, January.

Ward, A., Liker, J.K., Cristiano, J.J., Sobek, D.K., 1995. The second toyota paradox – how delaying decisions can make better cars faster. Sloan Management Review 36, 43–61.

Wheelwright, S.C., Clark, K.B., 1992. Revolutionizing Product Development: Quantum Leaps in Speed, Efficiency and Quality. Free Press, New York.

Yassine, A., Braha, D., 2003. Four complex problems in concurrent engineering and the design structure matrix method. Concurrent Engineering Research & Applications 11 (3).

Yassine, A., Chelst, K., Falkenburg, D., 1999. A decision analytic framework for evaluating concurrent engineering. IEEE Transactions on Engineering Management 46 (2), 144–157.

Yassine, A.A., Joglekar, N.R., Braha, D., Eppinger, S.D., Whitney, D.E., 2003. Information hiding in product development: the design churn effect. Research in Engineering Design 14 (3).

Zirger, B., Maidique, M., 1990. A model of new product development: an empirical test. Management Science 36 (7), 867–883.