OPERATIONS RESEARCH
Vol. 52, No. 4, July–August 2004, pp. 606–622
ISSN 0030-364X | EISSN 1526-5463 | 04 | 5204 | 0606

DOI 10.1287/opre.1040.0125
© 2004 INFORMS

Concurrent Crashing and Overlapping in Product Development

Thomas A. Roemer
Sloan School of Management, Massachusetts Institute of Technology, 30 Wadsworth Street, E53-387, Cambridge, Massachusetts 02142, troemer@mit.edu

Reza Ahmadi
Anderson School of Management, University of California at Los Angeles, Los Angeles, California 90095, rahmadi@anderson.ucla.edu

This research addresses two common tools for reducing product development lead times: overlapping of development stages and crashing of development times. For the first time in the product development literature, a formal model addresses both tools concurrently, thus facilitating analysis of the interdependencies between overlapping and crashing.

The results exhibit the necessity of addressing overlapping and crashing concurrently, and exhibit general characteristics of optimal overlapping/crashing policies. The impact of different evolution/sensitivity constellations on optimal policies is investigated, and comprehensive guidelines for structuring development processes are provided.

For the special case of linear costs, an efficient procedure is presented that generates the efficient time-cost trade-off curves and determines the corresponding optimal overlapping/crashing policies. The impact of key parameters and the robustness regarding their estimates is illustrated with a simple two-stage example.

Subject classifications: product development processes; overlapping; crashing; development leadtimes; development costs.
Area of review: Manufacturing, Service, and Supply Chain Operations.
History: Received June 2001; revision received May 2002; accepted August 2003.

1. Introduction

The increasing importance of fast product development has given rise to a large body of literature dedicated to decreasing product development cycles. Traditional discussions have focused primarily on "crashing" that is working with higher work intensities to reduce development times. A more recent body of literature addresses the challenges and benefits of "overlapping" of traditionally sequential stages, yet to date, no literature exists that integrates both concepts in one modeling framework. In this research, we therefore look at the costs of accelerated product development and address both overlapping and crashing. By exhibiting the interdependencies between the two, we illustrate the need to address them concurrently and to provide general guidelines for overlapping and crashing policies. In particular, by adopting a framework from Krishnan et al. (1997), we investigate how upstream evolution and downstream sensitivity interact with crashing decisions. Based on a simple example, we provide an in-depth discussion of the benefits of optimal policies for a variety of different scenarios and provide a comprehensive toolset for managers to structure their processes, to allocate and plan required resources, and to make budget and timing decisions for new product introductions.

1.1. Example

The central notion of this research is presented in the simplified two-stage problem depicted in Figure 1. Consider a process that consists of two stages, up- and downstream, where downstream depends on the output information from upstream. If no measures are taken to reduce the completion time, then the natural process structure is to first complete the upstream task and then proceed with the downstream task under complete information (Figure 1a). When crashed, the work intensities during the stages are increased (as indicated by the wider bars in Figure 1b) and the duration of each task is reduced from say T_i to $\widetilde{T}_i \in [\bar{r}_i \cdot T_i, T_i]$, where $0 < \bar{r}_i \leqslant 1$ expresses the maximum level of crashing at stage i.

Alternatively, to reduce the lead time, the stages may be performed partially in parallel or overlapped (Figure 1c). Now, however, because downstream starts without complete information, additional work might be necessary to accommodate unforeseen upstream developments. Consequently, total downstream time increases to say $T_i + h_i(y_i)$, where the amount of additional work $h_i(y_i)$ is a function of the overlap y_i between the stages. The dark gray area in Figure 1c indicates the additional downstream work caused by overlapping. Finally, the two approaches can be combined as in Figure 1d, where both methods are concurrently employed. For the remainder of the paper, we will refer to reducing development lead times, be it through crashing, through overlapping, or through a combination of both, as *process compression.*

If the functional form for h_i is given and the associated crashing costs are known, then under fairly general

Figure 1. Different compression strategies.

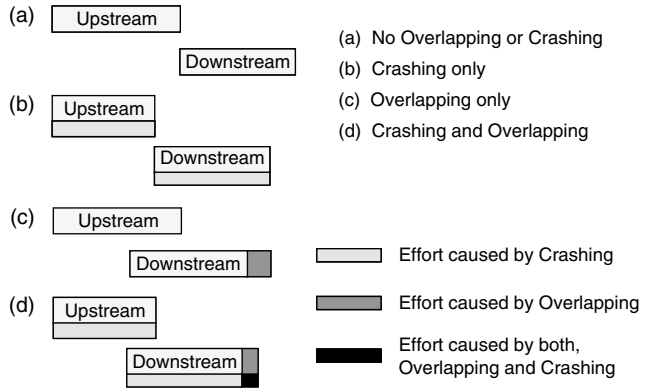
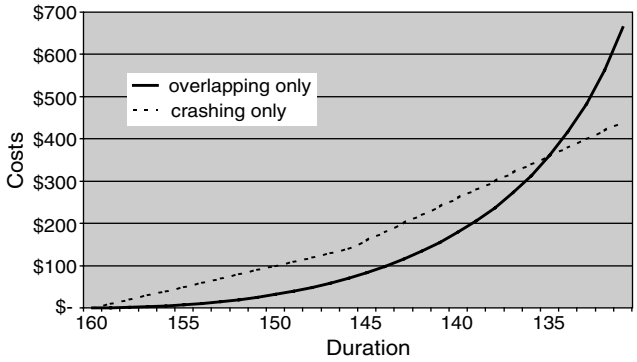


Figure 2. Overlapping vs. crashing costs.



conditions, the procedure suggested in Roemer et al. (2000) can be employed to calculate the time-cost trade-off curves for overlapping. Deriving the corresponding efficient frontier for crashing is straightforward if crashing costs are known.

To illustrate, suppose crashing costs K_i for stage i are linear in the amount of time crashed; i.e., $K_i = k_i \cdot (T_i - \widetilde{T}_i)$ and overlapping costs C are linear in h, i.e., $C = ch$. Furthermore, let $h = y - (1 - e^{-\alpha y})/\alpha$, with α being a constant. For this scenario, Figure 2 depicts the efficient frontiers assuming the values in Table 1. Figure 2 shows that both curves are convex increasing over the development time decrease. The range of feasible development times is between 160 units of time (days), the standard duration, and 131 days for either overlapping or crashing. Due to the linearity of crashing costs, overlapping is initially the more lucrative alternative, but eventually crashing will become preferable for large decreases in development time.

After reviewing some of the related literature in the next section, we will formally model the problem of jointly overlapping and crashing for a multistage process in §3. Our focus will be on structuring the development process in terms of overlapping and crashing, under particular consideration of time and budget constraints. Section 4 addresses the interdependencies between overlapping and crashing and demonstrates the necessity for an integrative framework. Properties for optimal solution policies are derived in §5, and a solution procedure for the special case of linear costs is presented. General insights and a concluding discussion follow in the subsequent sessions.

Table 1. Two-stage example.

	Upstream	Downstream
Standard duration T_i	70	90
Maximum crashing $\bar{r}_i$	0.80	0.83
Crashing costs k_i	10	20
Rework costs c		17
Constant α		0.03

2. Literature Review

Management of product development projects has received considerable attention during the last decade from both the academic community and practitioners (Krishnan and Ulrich 2001). One focal point of this literature is the structure of development processes (e.g., Eppinger 2001) and corresponding product architectures (e.g., McCord and Eppinger 1993) and modularization (e.g., Baldwin and Clark 2000 or Hoedemaker et al. 1999). A second focal point of the literature is how to narrow down design choices over the course of the development project. Recommendations include clearly defined sequential stage-gate processes (e.g., Cooper and Kleinschmidt 1996), early problem solving or "*frontloading*" (Thomke and Fujimoto 2000), and highly flexible development processes with delayed real-time definition of product attributes (Iansiti and MacCormack 1997, Kalyanaram and Krishnan 1997, Bhattacharya et al. 1998).

Common to all of these approaches is that they address the overall structure of the development process, and that they therefore take a relatively high-level look at product development. Implementation of these approaches typically involves upper management and spans across functional and organizational borders. Rewards can be ample and more efficient development processes often lead to shorter development times, lower costs, and higher product quality. We therefore strongly recommend exploring the full potential of these approaches before employing the methods suggested here. Indeed, in an earlier and related paper (Ahmadi et al. 2001), we have proposed such a global approach to (re-)structure a product development process, yielding drastic improvements in both development time and costs. However, at some point management will have to decide on an overall approach to the product development process. Once this decision stands, managers typically must (at least in the short term) operate within the parameters of the established process structures. At this point, flexibility in accelerating development processes is often limited to the tools identified by Graves (1989), particularly overlapping and crashing. In this paper, we take the principal view that the overall structure of the process has been established and—for the time

being—cannot be challenged. As such, we are more concerned with the operational approach of *when* to schedule work, as opposed to the more strategic approach of *how* to schedule *what* type of work in the extant literature.

Implicitly, such a distinction already exists within the overlapping literature, giving rise to two somewhat dissimilar views of overlapping. In the first view, overlapping is considered necessary to avoid costly late changes or product obsolescence (e.g., MacCormack et al. 2001). Thus, in addition to accelerating the development process, overlapping also leads to an overall reduction in costs. The findings by Clark and Fujimoto (1989), Imai et al. (1985), and Takeuchi and Nonaka (1986) greatly contributed to this view. The alternative view considers overlapping "*a calculated risk*" (Smith and Reinertsen 1995), where inherently sequential activities are performed in parallel to reduce development time. Mansfield et al. (1972) provide early empirical support for this view. Whereas in the first view overlapping is particularly promising if the outside world is highly dynamic and characterized by uncertainty, this view considers overlapping most appropriate for stable environments (e.g., Cordero 1991). Two of the most influential and rigorous empirical studies of development projects (Eisenhardt and Tabrizi 1995, Terwiesch and Loch 1999) lend support to this claim.

Common to almost all surveys is the conclusion that observed overlapping levels were typically not aligned with overall goals such as profit maximization or lead time minimization. Evidently, overlapping levels are commonly determined on an ad hoc basis, rather than on solid analytical grounds, yielding inefficient overlapping strategies. In part, this may be due to the lack of analytical approaches to overlapping, because to date only a few analytical models of overlapping exist. Ha and Porteus (1995) discuss how to optimally schedule design reviews between two overlapping stages under informational interdependencies between the stages. Krishnan et al. (1997) provide a model-based framework for overlapping that identifies conditions under which various types of overlapping are appropriate for a pair of coupled activities. Loch and Terwiesch (1998) study the impact of upstream engineering changes on optimal overlapping policies, and Roemer et al. (2000) discuss time-cost trade-offs for multistage overlapping processes.

In this research we will also address time-cost trade-offs in a multistage setting, but we will explicitly model the impact of upstream changes on downstream progress. More central, though, to this research is the question of how overlapping and crashing of development processes interrelate. To keep our analysis simple and aligned with the extant literature, our model of crashing closely resembles those initially introduced by Kelley and Walker (1959) and Berman (1964). In particular, we will assume that crashing costs are either linear or convex, and that crashing is limited both from above and below. In particular, working below minimum work intensity implies increasingly inefficient usage of available resources. Conversely, there is also ample evidence that excessive crashing can become counterproductive, leading to Brooks' (1995) famous statement that "…adding manpower to a late software project makes it later." Other researchers make similar observations (e.g., Abdel-Hamid and Madnick 1990), and either suggest strongly convex crashing costs (e.g., Putnam and Myers 1992) or assume explicit upper limits on productivity (e.g., Boehm 1981). As in the extant modeling-based crashing literature, we will assume that the level of crashing beyond which it becomes counterproductive is known and limit our analysis to crashing strategies below this level.

As with overlapping, we thus consider crashing as a tool that helps compress product development time at additional cost. This does not imply that any attempt to reduce development times will invariably lead to an increase in development effort. In fact, findings in Clark and Fujimoto (1991), that Japanese car manufacturers spend fewer person hours on development projects, yet have shorter lead times than their competitors, seem to indicate the opposite. Similarly, Pisano (1997) shows that development hours and lead times are positively correlated in a sample of 23 chemical and biotechnology companies. These results reconfirm our earlier statement that finding an appropriate *structure* for the product development process is paramount. By no means do they suggest that within the same overall process structure, increased effort levels will invariably lead to longer development times.

This, then, is the principal setting of our paper. We investigate how, in the short term, a company can decrease time to market by employing a mixed strategy of overlapping and crashing. Thus, our view is primarily an operational one concerned with time-cost trade-offs, rather than a long-term strategic one that aims to improve overall process efficiency. Under fairly general assumptions, we will show how crashing and overlapping are interdependent and provide general guidelines for compressing development times. Because our definition of overlapping is based on the notion of working under uncertainty in a mostly sequential process, our analysis presupposes largely sequential processes. Because these have been shown to be particularly successful in stable environments, our analysis is naturally better suited for stable environments than for highly dynamic ones.

3. The General Model

Throughout the paper we assume a product development process that consists of $n+1$ sequential stages, where the principal information exchange between consecutive design stages is unidirectional—from upstream to downstream. In addition, the regular duration T_i of stage i is known, where the regular duration of stage i is the expected time it takes to perform stage i without overlapping and crashing. Ahmadi et al. (2001) show how to obtain such an overall process structure and how to estimate stage durations.

In contrast, the actual stage duration depends on both crashing and overlapping. In particular, the actual stage

duration decreases under crashing. To keep the model as general as possible and to reflect existing crashing practices more accurately, we explicitly allow for different crashing levels during a single stage. In particular, we assume that any given time interval dt of stage i can be crashed to $r_i(t) \cdot dt$, with $r_i(t) \in [\bar{r}_i, 1]$. Note that crashing is essentially a consequence of increased work intensities and that both terms are directly related. We therefore say that if interval dt has been crashed to length $r_i dt$, then the *work intensity* during interval dt was r_i^{-1}. For the remainder of the paper we will largely use the terms work intensities and crashing interchangeably, unless the context requires a more precise distinction. Means to increase work intensities throughout a project include working overtime, adding staff, or utilizing more experienced staff or better equipment.

In contrast to crashing, the actual stage duration increases under overlapping. Because overlapping between stages $i-1$ and i increases uncertainty in stage i (downstream), some of the downstream work during the overlap may be obsolete due to unforeseen upstream developments. Consequently, the overall downstream duration increases by the amount of obsolete work, henceforth referred to as rework H_i. Throughout the model-based overlapping literature, downstream uncertainty is considered to decrease as upstream progresses. Therefore, H_i should be an increasing function of the overlap. Moreover, because upstream progress and downstream *work content* depend on the respective work intensities, H_i should also be a function of the work intensities during stage i and stage $i-1$. We will formally establish this functional relationship in §4.

Similar to most models in the crashing literature, we will assume that an earliest project-starting time S is given and that the project must be completed by a given and fixed due date D. Similarly, to allow for a better comparison between project structures, we will hold the quality of the project outcome constant. Finally, we assume that the decision maker is risk neutral, strictly focusing on development costs. The objective is therefore to find feasible crashing and overlapping policies that minimize total development costs subject to a due-date constraint. For ease of exploration, we impose the additional constraints that no stage may commence or terminate before its predecessors—that is, the principal process structure must remain intact—and that at most two stages can be processed concurrently. The latter restriction enforces a classical Sashimi-style solution (Imai et al. 1985), where each stage overlaps only with its immediate predecessor. This structure often obtains naturally if the principal information flow is unidirectional and is therefore not overly restrictive. Imposing it provides a clearer picture of the interdependence between crashing and overlapping and helps isolate different phenomena. To formulate the problem, we introduce the following notation:

$n+1$: number of process stages.

i: indices for the process stages, $i = 0, 1, \ldots, n$.

T_i: time required to complete design stage i without overlapping and crashing.

y_i: overlap duration between design stages $i-1$ and i; $y_0 \equiv 0$.

$r_i^{-1}(t)$: work intensity of stage i at time t.

$\bar{r}_i^{-1}$: maximum work intensity.

$H_i(y_i, r_i, r_{i-1})$: expected time increase at stage i as a function of overlapping and crashing; $H_0 \equiv 0$.

$c_i(H_i)$: rework costs at stage i as a convex increasing function of the design time increase.

$k_i(r_i^{-1})$: crashing costs for stage i as a convex increasing function of the work intensity.

s_i: starting time of stage i.

d_i: completion time of stage i; $d_{-1} \equiv s_0$.

S: earliest project starting time.

D: project deadline.

C: project cost.

We define the **Overlapping-Crashing Problem (OCP)** as follows:

$$\min C = \sum_{i=1}^{n} c_i(H_i) + \sum_{i=0}^{n} \int_{s_i}^{d_i} k_i(r_i^{-1}(t))\, dt \tag{1}$$

subject to

$$\int_{s_i}^{d_i} \frac{dt}{r_i(t)} = T_i + H_i \quad \text{for } i = 0, 1, \ldots, n, \tag{2}$$

$$y_i = d_{i-1} - s_i \quad \text{for } i = 1, 2, \ldots, n, \tag{3}$$

$$s_0 \geqslant S, \tag{4}$$

$$d_n \leqslant D, \tag{5}$$

$$0 \leqslant y_i \leqslant \min[d_{i-1} - d_{i-2}, d_i - s_i] \quad \text{for } i = 1, 2, \ldots, n, \tag{6}$$

$$1 \leqslant r_i^{-1}(t) \leqslant \bar{r}_i^{-1} \quad \text{for } i = 0, 1, \ldots, n. \tag{7}$$

The objective function (1) minimizes the additional costs caused by rework and crashing over the decision vectors $\mathbf{y} \equiv [y_1, \ldots, y_n]$ and $\mathbf{r}^-(t) \equiv [r_0^{-1}(t), \ldots, r_n^{-1}(t)]$. The first term in the objective function addresses the costs caused by the additional rework at each stage. We assume that the costs of rework are a convex function of the amount of rework and note that linear relationships prevail in many applications. The second term captures crashing costs. Following the general consensus in the crashing literature (e.g., Foldes and Soumis 1993), we model crashing costs as a convex, increasing function $k_i(r_i^{-1})$ of the work intensity. Diminishing returns, labor regulations on overtime, or additional training and communication requirements (e.g. Brooks 1995) are all contributing factors for convex increasing crashing costs. Assuming that $k_i(r_i^{-1})$ is Riemann integrable, crashing costs for each stage can be determined by integration over the stage duration.

Constraints (2) and (3) recursively define the start and completion times for the stages. Note that the difference between the boundaries on the left-hand side of (2), $d_i - s_i$, yields the actual stage duration, whereas the right-hand side is the stage duration without crashing. The overlaps y_i are defined in (3) as the difference between the completion time of stage $i-1$ and the starting time of stage i. Constraints (4) and (5) maintain that the project is not started

prematurely and that it completes on time. Note that any feasible solution to the OCP with $s_0 > S$ can be replaced by an otherwise identical solution, where $s_0 = S$. Because the resulting solution is also feasible but finishes earlier at the same costs, we can without loss of generality limit ourselves to solutions where constraint (5) is tight. Constraints (6) preserve the Sashimi-style character of the solution and warrant in conjunction with constraints (3) that no stage commences or terminates before any of its predecessors.

In accordance with the extant literature (e.g., Berman 1964) and without loss of generality, we define the regular work intensity as the cost-minimizing work intensity. Working below standard intensity leads to higher costs because of inefficiencies such as underutilization of equipment or suboptimal numbers of setup times. Consequently, constraints (7) bound the work intensities from below by unity. In the classical crashing literature, crashing is either explicitly bounded from above (e.g., Kanda and Rao 1984 or Leachman and Kim 1993) or a bound is implied by the shape of the cost curves (e.g., Berman 1964 or Falk and Horowitz 1972). In this research, we follow the prior approach and assume that crashing costs are convex in the work intensity, which is explicitly bounded from above. Such limits occur, for example, if technical systems reach their capacity limits, if personnel requirements exceed the available person power, or if adding additional personnel becomes counterproductive due to increased communication requirements.

Allowing flexible work intensities during each stage does not just model reality more accurately, but also allows the study of the impact of different crashing strategies on the amount of rework. Because the amount of rework at a stage depends on how much work has been accomplished upstream, timing of crashing within a stage impacts the amount of rework. The next section formally explores the interrelationship between crashing and rework and establishes a functional relationship.

4. Interdependencies Between Crashing and Overlapping

To identify optimal policies, it is essential to determine the extended rework H_i and how it is impacted by the decision vectors $\mathbf{y}$ and $\mathbf{r}^-(t)$. In the overlapping literature, the uncertainty resolution between up- and downstream has been modeled in several closely related ways. Krishnan et al. (1997) introduce the concept of *upstream evolution* and *downstream sensitivity* to capture how uncertainty decreases during the period of overlapping. A similar concept is employed by Loch and Terwiesch (1998), who study the effects of upstream evolution and *downstream dependence*, a concept akin to that of sensitivity, on the optimal overlap duration. In their model, the influence of upstream modifications on downstream rework is reflected in an *impact function*, a concept also employed by Carrascosa et al. (1998) in conjunction with their *probability of change* function as a measure of upstream evolution. Similarly, Roemer et al. (2000) introduced the *probability of rework* as a (nondecreasing) function P_i of the overlap between two stages. While this framework directly addresses upstream evolution through the shape of P_i, downstream sensitivity is addressed only indirectly by its magnitude and assumed to be constant over the progress of downstream. In this research, we will integrate the probability of rework function with the impact function to more generally address the interdependencies between upstream evolution, downstream sensitivity, rework, and work intensities.

In particular, we assume that *under normal work intensities*, if stage i makes a prediction at time $t \in [s_i, d_{i-1}]$, to be updated dt units of time later, then with probability $P_i(t)$ this prediction will turn out to be wrong and some portion $q_i(t) \cdot dt$ of the work performed between prediction and update must be performed again. We refer to the functions $P_i(t)$ and $q_i(t)$ as the *standard probability* and *standard impact function*, respectively, indicating that those functions apply to the case of standard work intensities. In accordance with the extant literature, we assume that $P_i(t)$ is nonincreasing and $q_i(t)$ nondecreasing. Note that the total rework $h_i(y_i)$ under standard work intensities can be obtained by integrating the product of impact and rework probability (*PI-curve*) over the entire interval $[s_i, d_{i-1}]$, that is, over the overlap y_i as indicated in Figure 3.

Thus, whereas Roemer et al. (2000) assume that invariably all work under erroneous assumptions was futile, we acknowledge that some fraction of the work performed might be salvageable, and that this amount might change (decrease) over time, as more and more downstream work has been completed already. In a second critical departure, we explicitly take work intensities into account as a measure for work accomplished. The implications are threefold. First, because the impact of upstream changes is predominantly dependent on how much downstream work has been performed already (e.g., Loch and Terwiesch 1998), and because with work intensities $r_i^{-1}(\tau)$ the amount of work performed by time t is $\int_{s_i}^{t} d\tau / r_i(\tau)$, the impact $Q_i(t)$ at time t given work intensities $r_i^{-1}(\tau)$ must be

$$Q_i(t) = q_i\left(\int_{s_i}^{t} \frac{d\tau}{r_i(\tau)}\right). \tag{8}$$

Figure 3. Probability of rework, impact, and PI-functions.

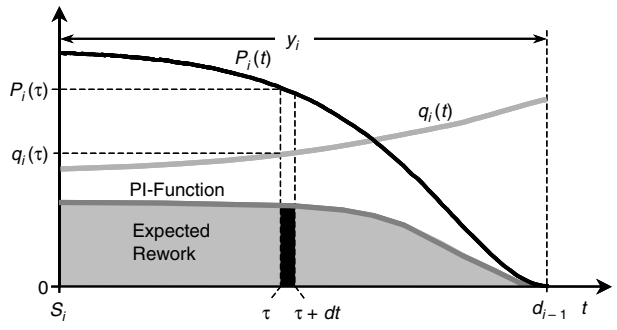
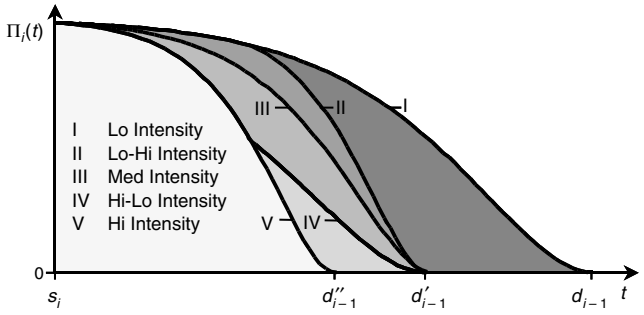


Figure 4. Impact of crashing strategies on probability of rework.



Second, the probability of rework is no longer simply a function of time, but rather of how much upstream work still has to be accomplished. In particular, at time $t \in [s_i, d_{i-1}]$, $d_{i-1} - t$ units of time are left until upstream terminates. However, because the work intensity during the remaining time might be above regular intensity, the remaining *work content* is $\int_t^{d_{i-1}} d\tau / r_{i-1}(\tau)$. Because unfinished upstream work is the primary cause of downstream uncertainty, the probability of rework with upstream crashing becomes

$$\Pi_i(t) = P_i\left(\int_t^{d_{i-1}} \frac{d\tau}{r_{i-1}(\tau)}\right), \tag{9}$$

where $P_i(\xi)$ is the standard probability of rework function for regular work intensity.

Figure 4 demonstrates how the probability function (I) for standard work intensities changes with the work intensity of stage $i-1$. In all five scenarios, stage i starts at the same time, but the upstream (stage $i-1$) work intensities differ after that point. The average work intensities in Scenarios II–IV are the same, yielding identical upstream termination times. The different shapes are caused solely by different timing patterns of work intensities. Scenario II starts out with regular work intensity before eventually proceeding with maximum work intensity. Conversely, Scenario IV begins with maximum work intensity and then returns to regular work intensity. Finally, in Scenario III the work intensity remains constant throughout the stage. Note how the cumulative probability—that is, the area under the curve—becomes smaller the earlier work intensities increase. When working with maximum work intensity throughout, as in Scenario V, the probability curve is pushed towards the origin, further reducing the area under the curve. Note also that the overall shape remains the same for Scenarios I, III, and IV, where the work intensities are constant over the stage, whereas the shapes in Scenarios II and IV are different due to the varying work intensities.

The final implication of changing work intensities is to note that if the probability of rework at time $t \in [s_i, d_{i-1}]$ is $\Pi_i(t)$, and if the work intensity of stage i at this time is $r_i^{-1}(t)$, then in terms of *work content* at normal work intensity, the expected amount of rework caused during the interval $[t, t+dt]$ is $q_i(t)\Pi_i(t) \cdot r_i^{-1}(t)dt$. Thus, the total expected amount of rework is

$$H_i = \int_{s_i}^{d_{i-1}} \frac{Q_i(t) \cdot \Pi_i(t)}{r_i(t)} dt, \tag{10}$$

and substituting (9) and (8) into (10) yields

$$H_i = \int_{s_i}^{d_{i-1}} \left[\frac{q_i\left(\int_{s_i}^{t} d\tau / r_i(\tau)\right)}{r_i(t)} \cdot P_i\left(\int_t^{d_{i-1}} \frac{d\tau}{r_{i-1}(\tau)}\right)\right] dt. \tag{11}$$

We thus have shown that the expected rework H_i at stage i is a function of its overlap with stage $i-1$ and the work intensities during stages $i-1$ and i. Assuming that the probability of rework and impact functions can be obtained for standard work intensities, rework can easily be computed for a given overlapping-crashing policy.

5. Optimal Overlapping-Crashing Strategies

Based on the derived expression for the rework (11), this section characterizes optimal solution structures that can be exploited to solve special instances of the OCP. In any Sashimi-style approach to overlapping, three distinct phases arise within each stage as depicted in Figure 5. In phase $i/1$, stage i overlaps with stage $i-1$ and, according to (11), the crashing policy during this interval affects only the amount of rework of stage i. In phase $i/2$, stage i is the only stage in process. As can be seen from (11), the crashing policy during this interval has direct impact on the amount of rework at any stage. Finally, in phase $i/3$, stage $i+1$ is concurrently being processed. The crashing policy during this interval now affects exclusively the amount of rework at stage $i+1$.

This structure enables us to investigate the effects of the crashing policies on the amount of rework in isolation. In particular, we can isolate the impact of downstream crashing (during phase $i/1$) from that of upstream crashing (during phase $i/3$). Focusing on these three phases, we will characterize optimal solution structures for the OCP in this section. For the special case of linear costs we develop an efficient solution procedure that easily extends to the case of piecewise linear (convex) cost curves.

Figure 5. Three phases of a stage.

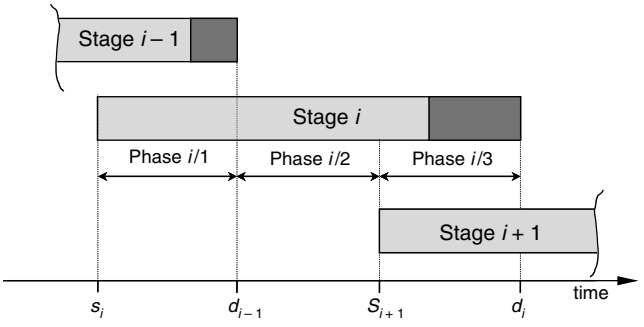


Figure 6. Delayed downstream crashing.

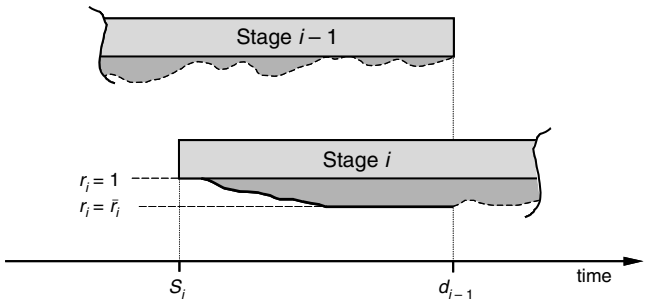
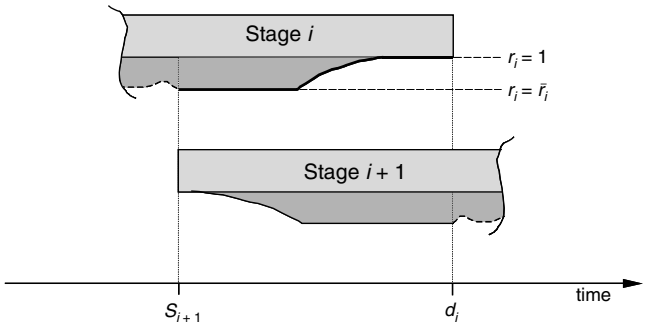


Figure 7. Upstream upfront crashing.



5.1. Optimal Solution Structure

To develop a general structure for optimal crashing policies, we next investigate the impact of crashing during each phase on the amount of rework. Note that at the beginning of stage i, that is, during phase $i/1$, there is uncertainty regarding the output of stage $i-1$. The harder stage i works under this uncertainty, the more work is wasted if it turns out to be based on wrong assumptions. Consequently, during phase $i/1$, stage i has an incentive to delay as much work content as possible, that is, to work with low intensity. We will refer to this phenomenon as *delayed downstream crashing*. In Proposition 1 we will show that this intuition is indeed correct, and that in any optimal solution, the work intensity is nondecreasing during phase $i/1$ as depicted in Figure 6.

PROPOSITION 1 (DELAYED DOWNSTREAM CRASHING). *If $s_i \leqslant u < v \leqslant d_{i-1}$, then $r_i^{-1}(u) \leqslant r_i^{-1}(v)$ in any optimal solution. Moreover, if $r_i^{-1}(u) = r_i^{-1}(v)$ in any optimal solution, then (at least) one of the following conditions must hold*: (i) $r_i^{-1}(v) = \bar{r}_i^{-1}$, (ii) $r_i^{-1}(u) = 1$, *or* (iii) *the first derivative of $k_i(r_i^{-1}(u))$ is discontinuous at $r_i^{-1}(u)$.*

To enhance readability of this paper, all proofs are presented in Appendix 1. Next, we turn our attention to phase $i/3$. During this phase, stage $i+1$ is working under uncertainty and requires information, in the form of completed work, from stage i (upstream) as fast as possible. Thus, the earlier that work is completed during phase $i/3$, the less rework will be required at stage $i+1$. We refer to this phenomenon as *upstream upfront crashing* and demonstrate in Proposition 2 that the work intensity is nonincreasing in the last phase of each stage, as depicted in Figure 7.

PROPOSITION 2 (UPSTREAM UPFRONT CRASHING). *If $s_{i+1} \leqslant u < v \leqslant d_i$, then $r_i^{-1}(u) \geqslant r_i^{-1}(v)$ in any optimal solution. Moreover, if $r_i^{-1}(u) = r_i^{-1}(v)$ in any optimal solution, then (at least) one of the following conditions must hold*: (i) $r_i^{-1}(v) = \bar{r}_i^{-1}$, (ii) $r_i^{-1}(u) = 1$, *or* (iii) *the first derivative of $k_i(r_i^{-1}(u))$ is discontinuous at $r_i^{-1}(u)$.*

Finally, we turn our attention to the intermediate phase $(i/2)$. During this phase, only stage i is being processed, and neither of the forces that impact work intensities in the other two phases is active. As a consequence, we should expect two properties of the optimal crashing policies during phase $i/2$. First, work intensities should strictly exceed those during the other two phases unless they are at the extreme points or where marginal crashing costs are discontinuous. Second, work intensities during this phase should comply with the established results in the literature, i.e., should (must) be kept constant (for strictly convex cost curves). The following proposition formalizes our expectations.

PROPOSITION 3. *Let $u \in [d_{i-1}, s_{i+1}]$. If $v \notin [d_{i-1}, s_{i+1}]$, then $r_i^{-1}(u) \geqslant r_i^{-1}(v)$ in any optimal solution. Moreover, if $r_i^{-1}(u) = r_i^{-1}(v)$ in any optimal solution, then (at least) one of the following conditions must hold*: (i) $r_i^{-1}(v) = \bar{r}_i^{-1}$, (ii) $r_i^{-1}(u) = 1$, *or* (iii) *the first derivative of $k_i(r_i^{-1}(u))$ is discontinuous at $r_i^{-1}(u)$. Else, if $v \in [d_{i-1}, s_{i+1}]$, then $r_i^{-1}(u) = r_i^{-1}(v)$ if $k_i(r_i^{-1})$ is strictly convex; otherwise, an optimal solution exists where $r_i^{-1}(u) = r_i^{-1}(v)$.*

We have thus shown that we can limit our attention to remarkably simple crashing policies. In each stage the work intensity initially increases monotonically, and after reaching a plateau of constant intensity, will eventually decrease again. This has several implications. First, implementing strategies where intensities increase and decrease gradually is usually smoother than trying to force rapidly alternating work intensities onto people or machines. Second, many successful projects follow such a pattern naturally. Personnel is built up after ground rules have been determined in each stage and is reduced towards work wrap-up. At Toyota, for example, most design engineers are only temporarily assigned full time to a single project during peak periods, but rotate otherwise between different projects (Ward et al. 1995). Conversely, they point to the pitfalls of the "end-of-term-syndrome" where work is delayed as much as possible towards a deadline. Our results indicate that in this case the increased work intensities towards the end of the stage (Phase 3) will adversely impact total development costs. The result thus emphasizes the importance of setting proper incentive structures for stage managers that aim at minimizing system costs, rather than stage costs that managers are typically held responsible for. Third, even if the probability of rework functions cannot be estimated reliably, the results provide general guidelines to structure

the work intensities over the course of the stages. If the probabilities can be estimated, though, then it is still difficult to compute exact levels of work intensities, because this requires explicit functional forms for all $r_i^{-1}(t)$. In the next section, we will therefore restrict our focus to linear crashing costs and show how to solve the problem under this assumption. Because most managerially relevant cost curves can be approximated by piecewise linear functions, the generality of our results suffers little from this additional assumption.

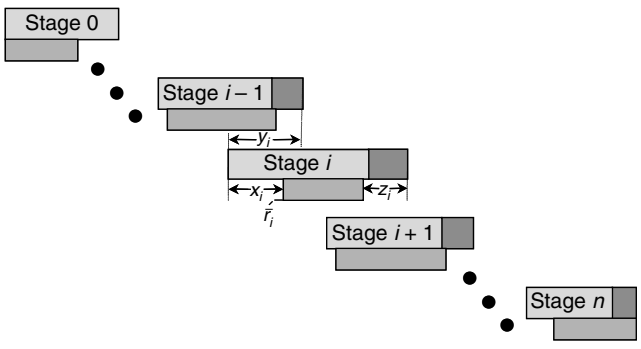
5.2. The Linear Cost Case

In this section, we will discuss the case where crashing costs k_i are linear in the time crashed, and rework costs c_i are linear in the amount of rework H_i. We will first show that under these conditions, remarkably simple optimal solution structures obtain and will subsequently exploit them to reformulate the OCP as the OCP/Lin. In Appendix 2, we provide an efficient solution for the OCP/Lin. We also note that the extension to piecewise linear costs is straightforward, albeit at considerable notational tedium. The observation is nevertheless important, because most managerially relevant cost curves can be approximated by piecewise linear functions. Thus, the methods described here also constitute the basis for solving the general case discussed before. The following proposition illustrates the simplicity of optimal crashing policies in the case of linear costs.

PROPOSITION 4 (LINEAR COSTS). *There is an optimal policy $r_i^{-1}(t)$ and $x_i, z_i \geqslant 0$ associated with it, such that $r_i^{-1}(t) = \bar{r}_i^{-1}$ for $t \in [s_i + x_i, d_i - z_i]$ and $r_i^{-1}(t) = 1$ otherwise.*

Thus, under a linear cost regime, each stage switches the level of work intensity at most twice, from regular intensity at the beginning, to maximum intensity during an intermediate period, back to regular intensity towards the end (see Figure 8). With the principal structure of the optimal solution known, it is now sufficient to determine the optimal overlap for each stage, and the optimal timing for the two changes in work intensity. With x_i and z_i as defined in the corollary, we can thus formulate the **Overlapping Crashing Problem with Linear Costs (OCP/Lin)**, as follows:

Figure 8. Optimal crashing policy for piecewise linear costs.



$$\min C = \sum_{i=1}^{n} c_i \cdot H_i + \sum_{i=0}^{n} k_i \cdot (1 - \bar{r}_i) \cdot (T_i + H_i - x_i - z_i), \tag{12}$$

$$\bar{r}_i \cdot (T_i + H_i) + (1 - \bar{r}_i) \cdot (x_i + z_i) = d_i - s_i \quad \forall i, \tag{13}$$

$$x_i + z_i \leqslant T_i + H_i \quad \forall i, \tag{14}$$

$$x_i, z_i \geqslant 0 \quad \forall i, \tag{15}$$

and (3) to (6).

The objective function (12) is the sum of rework and crashing costs, whereas constraint (13), similarly to constraint (2), defines the stage durations and thus, in unison with constraint (3), their start and completion times. Constraints (14) bound the durations of regular work intensity by the respective maximum stage durations and (15) are the nonnegativity constraints. Given the simple optimal solution structure, the expected rework in Equation (10) can now be expressed as

$$H_i = \int_0^{x_i} Q_i(t) \cdot \Pi_i(t)\, dt + \frac{1}{\bar{r}_i} \int_{x_i}^{y_i} Q_i(t) \cdot \Pi_i(t)\, dt \tag{16}$$

with probability of rework function

$$\Pi_i(t) = P_i\left(\frac{\max[y_i - z_{i-1}, t] - t}{\bar{r}_{i-1}} + y_i - \max[y_i - z_{i-1}, t] \right) \tag{17}$$

and impact function

$$Q_i(t) = q_i\left(\min[t, x_i] + \frac{\max[0, t - x_i]}{\bar{r}_i} \right). \tag{18}$$

The above expressions greatly facilitate solving the OCP, because only three variables per stage must be computed, rather than the infinite vector $r_i^{-1}(t)$. Moreover, with (17) and (18), rework probabilities and impact can simply be expressed in terms of the standard probabilities and impacts. To compute solutions, we define the marginal costs $\lambda(\xi) = (\partial C/\partial \xi) \cdot (\partial d_n/\partial \xi)^{-1}$. It follows by the Kuhn-Tucker conditions that all free variables in an optimal solution must yield the same marginal costs λ. After some algebra, this yields for given marginal costs λ the following necessary conditions for optimality:

$$\frac{\partial H_i}{\partial y_i} = \frac{\lambda}{r_i \cdot (\lambda + k_i) - k_i - c_i}, \tag{19}$$

$$\frac{\partial H_i}{\partial x_i} = -\frac{(1 - r_i) \cdot (\lambda + k_i)}{r_i \cdot (\lambda + k_i) - k_i - c_i}, \tag{20}$$

$$\frac{\partial H_i}{\partial z_i} = -\frac{(1 - r_{i-1}) \cdot (\lambda + k_{i-1})}{r_i \cdot (\lambda + k_i) - k_i - c_i}. \tag{21}$$

Table 2. The decision variables as a function of the Lagrangian multiplier λ.

Multiplier λ	Overlap y_i	Start crashing x_i	End crashing z_{i-1}
$-\lambda \leqslant \min[k_{i-1}, k_i]$	$q_i \cdot P_i(y_i) = \dfrac{\lambda}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$x_i = y_i$	$z_{i-1} = T_{i-1} + H_{i-1} - x_{i-1}$
$k_i < -\lambda < k_{i-1}$	$q_i \cdot P_i(y_i) = \dfrac{\bar{r}_i(\lambda + k_i) - k_i}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$q_i \cdot P_i(y_i - x_i) = \dfrac{\bar{r}_i(\lambda + k_i)}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$z_{i-1} = T_{i-1} + H_{i-1} - x_{i-1}$
$k_{i-1} < -\lambda < k_i$	$q_i \cdot P_i\left(\dfrac{y_i - z_{i-1}}{\bar{r}_{i-1}} + z_{i-1}\right) = \dfrac{\lambda}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$x_i = y_i$	$q_i \cdot P_i(z_{i-1}) = \dfrac{-k_{i-1}}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$
$\max[k_{i-1}, k_i] \leqslant -\lambda \leqslant k_{i-1} + k_i$	$q_i \cdot P_i\left(\dfrac{y_i - z_{i-1}}{\bar{r}_{i-1}} + z_{i-1}\right) = \dfrac{\bar{r}_i(\lambda + k_i) - k_i}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$q_i \cdot P_i(y_i - x_i) = \dfrac{\bar{r}_i(\lambda + k_i)}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$q_i \cdot P_i(z_{i-1}) = \dfrac{-(1 - \bar{r}_i)(\lambda + k_i) - k_{i-1}}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$
$-\lambda \geqslant k_{i-1} + k_i$	$q_i \cdot P_i\left(\dfrac{y_i - z_{i-1}}{\bar{r}_{i-1}} + z_{i-1}\right) = \dfrac{\bar{r}_i(\lambda + k_i) - k_i}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$q_i \cdot P_i\left(\dfrac{y_i - x_i - z_{i-1}}{\bar{r}_{i-1}} + z_{i-1}\right) = \dfrac{\bar{r}_i(\lambda + k_i)}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$	$q_i \cdot P_i(z_{i-1}) = \dfrac{-\bar{r}_i \cdot k_{i-1}}{\bar{r}_i(\lambda + k_i) - k_i - c_i}$

In general, finding a solution to this system of equations depends on the properties of functional forms of (17) and (18) such that uniqueness of solutions is not always warranted. However, constant impact functions are oftentimes suitable approximations (Carrascosa et al. 1998), in which case (18) are constants (q_i) yielding for the expected rework

$$\frac{H_i}{q_i} = \frac{\bar{r}_{i-1}}{\bar{r}_i} \cdot \left[h_i\left(\frac{\max[0, y_i - x_i - z_{i-1}]}{\bar{r}_{i-1}} + z_{i-1} \right) - h_i(z_{i-1}) \right] + h_i(\min[y_i, z_{i-1}]) + \left(\frac{1}{\bar{r}_i} - 1 \right) \cdot h_i(\min[y_i - x_i, z_{i-1}]) + \bar{r}_{i-1} \cdot \left[h_i\left(\frac{\max[0, y_i - z_{i-1}]}{\bar{r}_{i-1}} + z_{i-1} \right) - h_i\left(\frac{\max[0, y_i - x_i - z_{i-1}]}{\bar{r}_{i-1}} + z_{i-1} \right) \right]. \quad (22)$$

Equations (19) to (21) simplify to five cases, as shown in Table 2. In Appendix 2, we present a computationally efficient solution procedure that determines the efficient frontier of time-cost trade-offs through a binary search over the marginal costs λ, utilizing these results. Moreover, from this scenario we can derive insights on how evolution and sensitivity impact crashing and overlapping.

6. Insights and Results

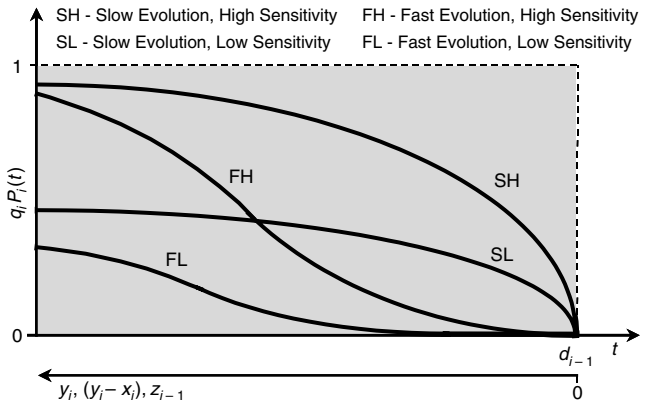
In this section, we will use the previous results to derive generalized guidelines for overlapping and crashing under different scenarios, particularly regarding different patterns of information evolution and stage sensitivity to upstream changes. Moreover, in the context of the simple two-stage example presented earlier, we will discuss how different parameter values impact the value of optimal strategies.

6.1. The Impact of Evolution and Sensitivity

Note that the left-hand side of the entries in Table 2 are the PI-functions, that is, the product of probability and impact. The shape of these functions reflects some of the notions of upstream evolution and downstream sensitivity as defined by Krishnan et al. (1997). In particular, the slope of the curve, that is, the rate at which the probability of rework vanishes, represents the speed at which upstream information evolves. On the other hand, high (low) PI-values, and, in particular, high (low) values for q_i are indicative of high (low) downstream sensitivity. Figure 9 shows generic PI-functions that map onto the four evolution-sensitivity constellations identified by Krishnan et al. (1997). For example, functions SH and SL evolve slowly—that is certainty regarding the upstream output is only obtained when upstream nears finalization. However, sensitivity to upstream changes is much higher for SH, indicated by higher values of the PI-values. Curves FH and FL also differ in their relative sensitivities, but here information evolves much faster than with SH and SL—that is, the probability of rework vanishes faster in these cases.

This interpretation, together with the results from Table 2, helps derive general guidelines for overlapping and crashing for the four different scenarios. First, note that given marginal costs λ, and in particular the optimal marginal costs, uniquely define the $q_i \cdot P_i$ values in Table 2, implying that the degree of overlapping y depends only on

Figure 9. Evolution and sensitivity for different probability curves.



upstream evolution and *downstream* sensitivity. Similarly, the optimal *starting* time for crashing (x) is determined by the stage's sensitivity and by the speed of *upstream* evolution. In contrast, the time to resume normal intensity (z) depends on the stage's evolution and *downstream* sensitivity.

Moreover, Figure 9 implies that overlapping (y_i) and *downstream* crashing during the overlap ($y_i - x_i$) are highest for FL and lowest for SH. In contrast, *upstream* crashing is more extensive for SH (z is smaller) than for FL. Thus, under fast evolution and low sensitivity, the focus should be primarily on overlapping and *downstream* crashing, whereas for the other extreme combination, slow evolution and high sensitivity, the focus should be on *upstream* crashing.

Values for FH and SL will generally be between those for the two extreme constellations, but no particular bias towards overlapping or crashing either stage exists, suggesting a more "balanced" approach. Interestingly, compressing stages with slow evolution and low sensitivity yields benefits primarily if overall process compression is high; otherwise, it is more advisable to compress stages with fast evolution and high sensitivity instead. To see this, note first that $dP_i(y_i)/d(-\lambda)$ and $dP_i(y_i - x_i)/d(-\lambda)$ are strictly positive and that $dP_i(z_{i-1})/d(-\lambda)$ is strictly negative. Therefore, if $-\lambda$ is low, or equivalently, if overall compression is limited, overlap y_i and downstream compression $y_i - x_i$ will be lower for SL than for FH. Similarly (because $P_i(z_{i-1})$ is large for small λ), the value for z in Scenario SL will be larger than that for Scenario FH, so that there will be less crashing during the overlap under the SL regime than under FH. The reverse is true when total development time is compressed considerably. Figure 10 summarizes these findings and provides guidelines for management when the probability of rework function cannot be computed reliably, but if evolution and sensitivity patterns can be assessed roughly.

Figure 10. Crashing and overlapping for different evolution-sensitivity-constellations.

EVOLUTION \ SENSITIVITY	Low	High
Fast	Primarily Overlapping & Downstream Crashing	Balanced Overlapping & Up-/Downstream Crashing
Slow	Balanced Overlapping & Up-/Downstream Crashing Main Relevance: High Compression	Primarily Upstream Crashing

6.2. Results for the Two-Stage Example

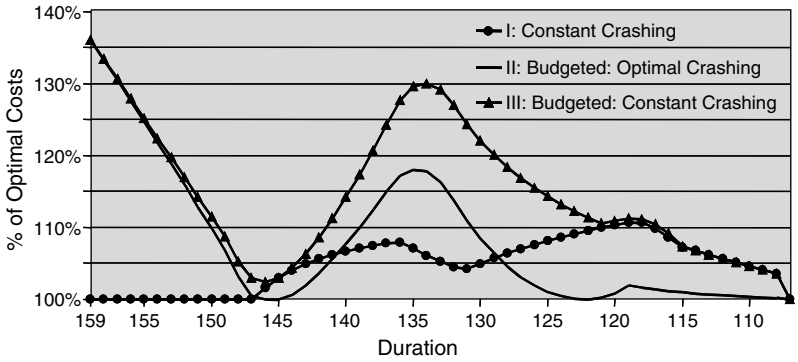
To gain additional qualitative insights, we solved the simple two-stage example presented earlier for a variety of different parameters and with additional constraints to compare optimal results to those that better reflect common industry practices. Furthermore, we evaluated the impact of evolution and sensitivity on the performance of suboptimal solutions and how parameter errors impact product development time and cost.

Mixed vs. Pure Strategies. First, we compared the mixed approach of jointly crashing and overlapping with the two pure approaches of crashing only and overlapping only. To better understand the dynamics of the approaches, we first computed the smallest possible duration ($d_{\min}$) for the joint approach by setting crashing and overlapping levels to their maximum possible values. Not surprisingly, $d_{\min}$ decreased considerably, from 131 days for the pure approaches to 107.3 days for the mixed approach. We then solved the OCP for every integer value between 160 days ($T_0 + T_1$) and 131 days and computed the additional costs of the pure strategies relative to the optimal mixed strategy. On average, pure overlapping was 36% more expensive than the mixed strategy, whereas pure crashing was on average more than four times as expensive as the mixed strategy. However, this comparison is strongly biased towards overlapping, because the linear cost assumption for crashing makes crashing for low process compression relatively unprofitable. For piecewise linear cost functions with initially lower crashing costs, we should therefore expect that pure crashing becomes relatively less costly, whereas pure overlapping will become relatively more costly, because the optimal mixed strategy will employ more crashing earlier on.

Optimal vs. Common Standard Strategies. More interesting than the comparison with the two pure strategies is to benchmark the optimal mixed policy with other mixed policies. Therefore, we looked at three plausible alternatives. Because stable workloads are oftentimes considered desirable, Alternative I keeps crashing at a constant level over the entire duration of each stage. Another typical management tool for development projects is to provide separate budgets for each stage, especially when stages correspond to different organizational units. This is taken into account in Alternative II, where both stages receive a fixed proportion[1] of an available budget. Crashing of the stages is performed optimally according to the results of this paper, but in accordance with the budget constraint. Finally, combining both approaches, Alternative III assumes budget allocation and constant crashing simultaneously.

Figure 11 compares the strategies relative to the optimal solution. The results suggest that in this case penalties for working with constant work intensity (Alternative I) or for budgeting stages (Alternative II) are not overly excessive, averaging 5% over all durations in both cases. Interestingly,

Figure 11. Performance of nonoptimal mixed strategies.



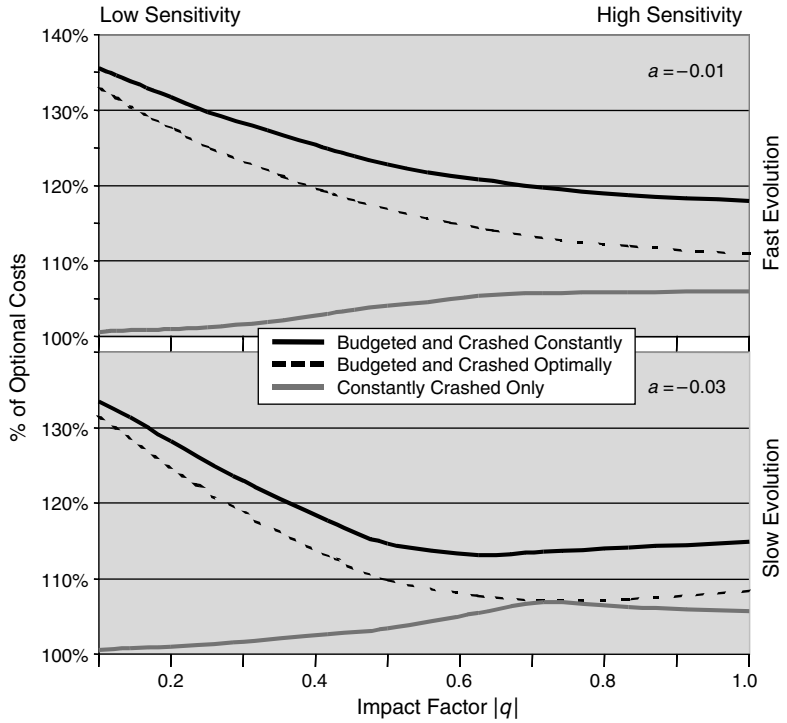
penalties for crashing are strongly dependent on the overall level of process compression and can be as high as 18% for medium levels of overall compression. In comparison, the curve for Alternative II is much flatter, with a maximum penalty below 12%. In Alternative III, repercussions of the two alternatives are compounded, resulting in an average cost penalty of 13%. It is also noteworthy that the penalty can be substantially higher than the sum of the individual penalties.

In general, computing this penalty facilitates a better understanding of the disadvantages of traditional policies. These disadvantages can then be compared with potential benefits. For example, if changing work intensities are facilitated by adding more person power, then there is the risk that additional communication and training requirements actually slow down the process (Brooks 1995). While the model addresses this by imposing an upper bound on the work intensity, overly optimistic estimates of the upper bound may indeed induce the paradox of slowing down work by adding person power. Computing curves such as in Figure 11 help evaluate that trade-off. It is then up to management to decide whether or not to buffer against the risk of overly aggressive work intensities by adopting a (suboptimal) strategy of constant work intensities.

Impact of Upstream Evolution and Downstream Sensitivity. The overall shape in Figure 11 also prevailed when solving the problem with different parameters. In particular, we varied the values for a and q, the latter having so far been implicitly set to unity. Note that a large (small) absolute value for q reflects high (low) sensitivity, whereas large positive (negative) values for a yield fast (slow) evolution patterns. For all evolution-sensitivity constellations the three distinct areas eminent in Figure 11 could be observed. It is noteworthy, though, that fast evolution tends to move the two humps closer together, indicating a consistently worse performance for suboptimal strategies when evolution is fast. The reason for this is that for fast evolution, overlapping becomes more and more lucrative, so that in suboptimal solutions more and more crashing is performed during the overlap, making both humps wider until they eventually merge into one continuous ridge for very fast evolution. Low sensitivity, on the other hand, tended to suppress the first hump in the curve, while pronouncing the second hump more distinctly. The reason for the first phenomenon is that medium compression for low sensitivity can be accomplished almost exclusively by overlapping, thus closely following the optimal strategy. Only for high levels of compression does crashing become lucrative, but now crashing is relatively more expensive because of the large overlaps. In summary, this suggests that suboptimal strategies underperform significantly when processes with low downstream sensitivity must be highly compressed, or for medium compression of processes with high downstream sensitivity.

To investigate the impact of evolution and sensitivity on the average cost increase for the three different mixed policies, we computed the costs for fast ($a = -0.01$) and slow ($a = 0.03$) evolution over a range of sensitivities from $|q| = 0.1$ to 1. The results in Figure 12 suggest that, with decreasing sensitivity, the costs of constant crashing approach those of the optimal policy, whereas the budgeted approaches diverge strongly for low sensitivities. The reason for this behavior is that decreasing sensitivity makes overlapping more and more lucrative for downstream, while budgeting prevents downstream from overlapping more. On the other hand, because for low sensitivity the costs of overlapping become minimal, there is very little benefit in optimizing crashing policies during overlapping, so that constant crashing policies are near optimal for very low sensitivities. Interestingly, for slow evolution extrema occur at an interior point.[2] In this case, for very high sensitivities overlapping is employed minimally, so that the impact of suboptimal crashing during the overlap is overall less pronounced than for lower values of sensitivity where overlapping is more aggressively employed. Similarly, the costs of budgeting are increasing again relative to the optimal strategy, because more downstream budget is spent on overlapping than in the optimal solution. Together, these drivers lead to the inverse relation of the performance of constant crashing and of budgeting stages relative to stage sensitivity as illustrated in Figure 12.

Figure 12. Relative advantage of optimal policies as a function of evolution and sensitivity.



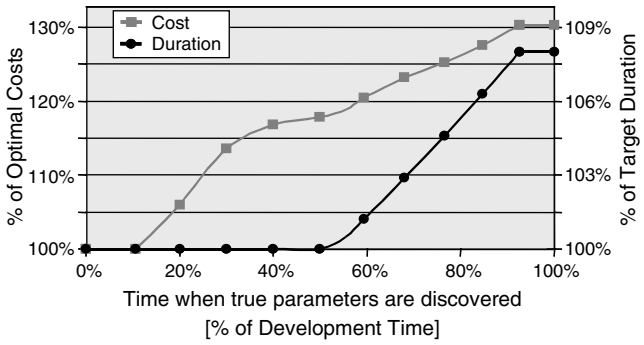
It should be noted that the magnitude of the curves is highly dependent on the cost parameters. However, the general shapes of the functions, and thus the general insights, are quite robust for different cost assumptions. Moreover, in the two-stage example the negative impact of budgeting is likely to be overestimated relative to multistage scenarios. The reason is simply that the degrees of freedom for upstream are reduced to crashing only. Conversely, the impact of constant crashing is underestimated in the two-stage example, because crashing during overlap has adverse effects on downstream only. In multistage environments, constant crashing has adverse impacts on both upstream and downstream for all but the first and the last stages. Again, this impacts chiefly the magnitude of the curves but not their overall shapes. By focusing on the two-stage example, we were therefore able to exhibit the impact of evolution and sensitivity more clearly, without sacrificing general insights.

Sensitivity to Parameter Errors. To evaluate the sensitivity to estimation errors, we looked at the case where the parameter *estimates* were $a = 0.03$ and $q = 0.5$ and computed the corresponding optimal policies for three different target completion times ($D \in \{145, 130, 115\}$). We then computed the actual completion times for these policies, if the *true value* of one parameter at a time or both parameters concurrently differed by $\pm 10\%$, $\pm 20\%$, or $\pm 50\%$ from those estimates. The average *absolute*[3] deviation from the target completion times for the resulting 3×18 scenarios was only 1.6%. However, even in the worst case ($D = 115$, $a = 0.045$, $q = 0.75$), the resulting completion time of 124.2 days was only 8% above the target duration of 115 days. This suggests that the general structure of the optimal schedule is quite robust in terms of completion times. Deviations from the target tended to increase with the degree of overall compression; group averages for $D \in \{145, 130, 115\}$ were 0.3%, 1.3%, and 2.5%, respectively.

To evaluate the costliness of estimation errors, we computed for all 54 scenarios the optimal costs for the resulting actual completion times and compared them to the true costs. For example, for our worst case the cost of finishing in 124.2 days was \$504. Had the true parameters been known at the beginning, then the same completion time could have been accomplished at the cost of \$386. Thus, in this case total costs for the resulting completion time are 30% above optimal costs. Overall, the average costs for all 54 scenarios were 5.4% above optimal costs. Again, higher compression targets were more sensitive to estimation errors; group averages for $D \in \{145, 130, 115\}$ were 1.5%, 5.5%, and 9.1%, respectively.

The Value of Updated Information. The worst case was further analyzed to determine how updating information *during* the development process impacts costs and duration. In particular, we assumed that the process starts with a compression policy based on the estimates $a = 0.03$ and $q = 0.5$, and that at some time during the development process, the true values $a = 0.045$ and $q = 0.75$ become known. From then on, the process continues optimally for the remaining time. Figure 13 shows how costs and duration depend on the timing of discovering the true parameter

Figure 13. The value of updated information.



values. For instance, even if the true parameters are discovered when 50% of the development process is already completed, the project can still be completed on target, but at costs that are 18% higher than if the true values had been known at the beginning. No additional costs occur when the true parameters are discovered within the first 10% of development time, but discovering them after 90% project completion will no longer yield any benefits in our case. The results indicate that, while overall robustness of solutions to estimation errors is already fairly high, updating estimates during the development process and adjusting the compression policy for the remaining time can further increase robustness.

7. Conclusion

This research has discussed two common tools for expediting product development processes, overlapping and crashing. By exhibiting the interdependencies between these two tools, we have bridged the gap between two separate literature streams and demonstrated that the decisions to overlap and crash should be made jointly to exploit their full potential. For cases where all input information (i.e., cost and rework functions) is obtainable, we introduced an algorithm that computes the efficient frontier of time-cost trade-offs. Knowledge of the frontier serves as a powerful tool to market entry and timing, as well as to budgetary decisions. The efficiency of the algorithm also allows for analyzing results regarding the sensitivity to input parameters. Highly optimistic (pessimistic) assumptions can be employed to bound the efficient frontier from below (above). Also, the reader should note that the underlying problem structure is dynamic and that system feedback evolving over time can be used to resolve the problem with updated information. For multistage problems in particular, the problem can be resolved after the completion of each stage for the remaining stages, with the actual stage completion time as an input parameter, rather than a decision variable. Similarly, the problem can be resolved when updated information regarding the probability of rework emerges. As Figure 13 and the corresponding discussion demonstrate, the value of such updated information can be substantial. Thomke and Bell (2001) suggest resolving technical uncertainty through appropriate testing strategies, which essentially leads to improved probability of rework functions at the cost of additional testing. The algorithm proposed here can be employed to evaluate the benefits of reducing the probability of rework and to compare them to the costs of testing.

The testing strategies in Thomke and Bell may also help determine the probability of rework functions as input for our model. So far no rigorous research exists that addresses the question of how to best estimate probability of rework functions. In our experience, design engineers typically felt quite comfortable in providing probability functions. Other researchers (e.g., Carrascosa et al. 1998) report similar observations. In several projects in the aerospace industry, we found that development projects evolved around a comprehensive list of parameters *to be determined* ("TBDs"). At the beginning all parameters are considered TBDs, and an essential part of project planning is to establish when TBDs are supposed to be locked in. The pattern of how TBDs vanish is often a good starting point for estimating probability of rework functions. Future research should explore best practices in establishing rework functions and investigate their robustness.

However, the insights derived in the paper help in structuring the overall process even if the probability of rework cannot be assessed properly. In particular, we have shown that the work intensities for each stage should always follow a certain pattern, from initially increasing to decreasing towards the end. Moreover, if costs are (piecewise) linear, then changes are stepwise, and therefore particularly suitable for implementation. Such policies should be implemented even if the necessary data for optimization are unavailable.

Additional questions arise from this insight. First, does this pattern remain optimal if the assumption of Sashimi-style solutions is abandoned? It is straightforward to show that Propositions 1 and 2 remain valid in that case as well. However, in the second phase of each stage, the desire to delay action now opposes that of generating information for downstream, so that erratic patterns of changing work intensities may prevail.

Second, implementing strategies where stages concentrate their work efforts towards the middle may face some resistance, particularly when stages are performed by independently managed teams. It is partly human nature to delay important decisions. For stage managers, this means that they might tend to delay work content towards the end of each stage. Unfortunately, such behavior is also rational, if only in a myopic, stage-focused manner. Equation (11) reveals why. Given the choice of increasing work intensity at time u or v $(u < v)$, the amount of rework is less if the intensity is increased at time v, because $P_i(v) \leqslant P_i(u)$. Consequently, the "rational" stage manager who minimizes effort and costs at the stage level will try to reap the benefits of delaying decisions, thus robbing downstream of the advantage of earlier information. The results are suboptimal intensity structures on the process level that are not

in accordance with the findings in this paper. Conventional incentive structures tend to reward stage managers that keep their costs low and thus encourage suboptimality. Finding incentive structures that result in crashing and overlapping structures as suggested in this research is an interesting and challenging problem for future research.

Third, a related problem is that of resource availability during the progress of each stage. Implicitly, it was assumed that resources are readily available whenever needed. This may not be the case if personnel is still involved in a previous project or must be withdrawn to serve on future ones. Consequently, consecutive projects should be looked at concurrently. The insights derived here can serve as the basis to study such a comprehensive dynamic view of product development processes.

The results from the previous sections provide further guidelines for compressing development processes. The roadmap in Figure 10 shows general overlapping and crashing patterns for different evolution and sensitivity constellations under low and high overall compression. In particular, it exhibits that the optimal time to start crashing a stage depends on its sensitivity to changes and the speed of upstream evolution. Conversely, when to stop crashing a stage depends on the speed of its evolution and downstream sensitivity. Thus, even if the probability of rework and impact functions cannot be quantified, the general structure of crashing can be determined based on a simple assessment of the relative values of evolution and sensitivity.

The generalized insights from the two-stage example illustrate that under low sensitivity, optimal crashing patterns provide few additional benefits over constant work intensities, but that stage budgeting may have particularly damaging effects. As Figure 12 shows, the converse is not necessarily true for slow evolution. Moreover, the suboptimality gap tends to be widest (in relative terms) when evolution is fast. These results indicate, therefore, when a more thorough investigation of crashing and overlapping patterns is warranted.

In conclusion, we would like to point out four possible limitations and pitfalls to the approaches presented here. First, we have implicitly assumed that all stages are on the critical path of the project. Development projects that exhibit a network structure are not fully captured by our approach. Berman (1964) introduces the concept of complex junctions, where more than one activity originates and/or terminates and shows that in any optimal solution the sum of time-cost slopes of activities leading into a junction must be equal in value and be opposite in sign to the sum of time-cost slopes emanating from a complex junction. In principal, we could simply adopt this approach, but this would require assessing joint probability of rework and impact functions for multiple overlaps. Given the difficulties in determining these, we see limited value in extending our approach to this case. Instead, we encourage managers to apply the general insights derived in this paper to determine overlap and crashing for complex junctions.

Second, some evidence suggests that the benefits of overlapping vanish if the pace of technical progress within an industry is high or if innovations are radical (e.g., Cordero 1991, Eisenhardt and Tabrizi 1995, or Terwiesch and Loch 1999). While we do not have enough empirical evidence to back these studies, they are both intuitively appealing to us and in accordance with our own observations. As reported in an earlier paper (Roemer et al. 2000), probability of rework-based approaches worked well for the development of rocket engines, particularly after structuring the process as suggested in Ahmadi et al. (2001) and Ahmadi and Wang (1999). The methods suggested in these papers and the present one require extensive knowledge of the existing process. In incremental innovations with only moderate technical progress and in situations where the underlying fundamentals are well understood and tested, these data are typically available and the suggested methods are appropriate. In fast-paced environments or for completely new products, this information is often not available and the approaches suggested here may fail. Fortunately, this is not a major limitation of this research, because the overwhelming majority of product development projects are in the realm of incremental innovations and moderate to low technical progress (Whitney 1990).

Third, our model is deterministic and therefore does not directly address risk. While many facets of risk can be addressed by running sensitivity analyses on the input data and by dynamically applying the OCA as more information evolves over the course of the development process, our approach does not discriminate between overlapping and crashing. However, because the former's principal tenet is to work under uncertainty, overlapping can be considered as inherently more risky than crashing (at least as long as crashing is limited to "reasonable" work intensities). Risk-averse project managers may thus prefer costlier solutions with less overlapping (and hence less risk) than our approach suggests. To some extent, simple modifications of our model can remedy this. For example, if risk can be characterized by the maximum level of uncertainty in a project, then the probability of rework can be limited from above in the OCP/Lin, and only minor changes to the OCA are necessary to solve the resulting problem. Similarly, schedule and cost uncertainties can be mitigated by imposing upper bounds on H_i and $c_i \cdot H_i$, respectively. Repeated solving with different upper limits thus generates a time-cost-risk profile for a project. Contingency risks, on the other hand, remain outside the model's scope. Of course the model can *reactively* be applied to the emerging postcontingency scenario, but it is unable to *predict* or *proactively* plan for contingencies.

Finally, our model should not serve as an excuse to neglect proper upfront work—that is, determining suitable process structures and product development strategies first. On the contrary, their full potential can only be realized if based on solid upfront work. Moreover, by no means should the reader conclude from our model that crash-

ing will always lead to shorter development times. The "crashing paradox" (Brooks 1995) is real and should not be ignored. Any overly optimistic estimate of the maximum work intensity presents the danger of inducing this paradox. While our model helps to determine the potential benefits from changing work intensities, it is up to management to make sure that overall crashing levels remain within reason.

Appendix 1. Proofs of Propositions

PROOF OF PROPOSITION 1. Define the Lagrangian multiplier λ_w as

$$\lambda_w = \frac{\partial \sum_{j=0}^{n} \left(c_j(H_j) + \int_{s_j}^{d_j} k_j(r_j^{-1}(t))dt\right)}{\partial r_i(w)} \cdot \left[\frac{\partial \sum_{j=0}^{n}(d_j - s_j - y_j)}{\partial r_i(w)}\right]^{-1}. \quad (23)$$

Note that the first factor in (23) is the change of the objective function value due to a small change in the work intensity at time w. Similarly, the second term is the change of the completion time d_n, derived by recursively solving (3). Intuitively, (23) expresses the cost of decreasing the completion time by changing the work intensity at time w. Clearly, if that cost is higher at some time, say u, than at another time, say v, and if both variables are "free"—that is, at neither bound, then by (slightly) increasing work intensity at time v and (slightly) decreasing work intensity at time u, the same completion time can be obtained at lower cost. More formally, Lagrangian Multiplier Theory requires that $\lambda_u = \lambda_v$ for all free variables. After simplifying (23), we will exploit this fact and show that for free variables, the necessary condition can only hold if $r_i^{-1}(u) < r_i^{-1}(v)$ for $u < v$. The first term in the numerator of (23) simplifies to

$$\frac{\partial \sum_{j=0}^{n} c_j(H_j)}{\partial r_i(w)} = \frac{\partial c_i(H_i)}{\partial r_i(w)} = c_i'(H_i)\frac{\partial H_i}{\partial r_i(w)} \quad (24)$$

with

$$\frac{\partial H_i}{\partial r_i(w)} = -\frac{dt}{r_i^2(w)} \cdot \left[Q_i(w)\cdot\Pi_i(w) + \int_w^{d_{i-1}} Q_i'(t)\cdot\frac{\Pi_i(t)}{r_i(t)}dt\right] \equiv -\frac{dt}{r_i^2(w)}\cdot\Psi_i(w), \quad (25)$$

whereas if $k_i'(r_i^{-1}(w))$ exists, the second term yields

$$\frac{\partial \int_{s_j}^{d_j} k_j(r_j^{-1}(t))dt}{\partial r_i(w)} = -\frac{dt}{r_i^2(w)} \cdot \left[k_i'(r_i^{-1}(w)) + \frac{r_i'(d_i)}{r_i(d_i)} \cdot (1 - \Psi_i(w)) \cdot k_i'(r_i^{-1}(d_i))\right]. \quad (26)$$

The denominator in turn becomes

$$\frac{\partial \sum_{j=0}^{n}(d_j - s_j - y_j)}{\partial r_i(w)} = \frac{\partial(d_i - s_i)}{\partial r_i(w)}. \quad (27)$$

To evaluate the right-hand side of (27) we take the derivative with respect to $r_i(w)$ on both sides of (2), which yields for the left-hand side of (2)

$$\frac{\partial \int_{s_i}^{d_i} dt/r_i(t)}{\partial r_i(w)} = -\frac{dt}{r_i^2(w)} + \frac{\partial(d_i - s_i)}{\partial r_i(w)} \cdot \frac{1}{r_i(d_i)} \quad (28)$$

and for the right-hand side

$$\frac{\partial(T_i + H_i)}{\partial r_i(w)} = -\frac{\Psi_i(w)}{r_i^2(w)}\,dt \quad (29)$$

so that

$$\frac{\partial(d_i - s_i)}{\partial r_i(w)} = \frac{r_i(d_i)}{r_i^2(w)} \cdot (1 - \Psi_i(w)) \cdot dt. \quad (30)$$

Inserting these results into (23) yields then for $\lambda_u = \lambda_v$,

$$\frac{c_i'(H_i)\cdot\Psi_i(u) + k_i'(r_i^{-1}(u))}{1 - \Psi_i(u)} = \frac{c_i'(H_i)\cdot\Psi_i(v) + k_i'(r_i^{-1}(v))}{1 - \Psi_i(v)}. \quad (31)$$

Note that the Lagrangian multipliers must be positive in any optimal solution; otherwise, costs and project duration could be reduced concurrently, contradicting optimality. It follows from (31) that $\Psi_i(w) < 1$. Consequently, because $r_{i-1}(t)$ is strictly positive, $u < v$ implies $\Pi_i(u) > \Pi_i(v)$ yielding for any nondecreasing impact function

$$\Psi_i(u) - \Psi_i(v) = Q_i(u)\cdot\Pi_i(u) - Q_i(v)\cdot\Pi_i(v) + \int_u^v Q_i'(t)\cdot\frac{\Pi_i(t)}{r_i(t)}dt > \Pi_i(v)\cdot\left[Q_i(u) - Q_i(v) + \int_u^v Q_i'(t)\cdot dt\right] = 0. \quad (32)$$

Thus, (31) can only hold if $k_i'(r_i^{-1}(u)) < k_i'(r_i^{-1}(v))$, which, by the convexity of $k_i(r_i^{-1})$, implies $r_i^{-1}(u) < r_i^{-1}(v)$. □

PROOF OF PROPOSITION 2. As in the proof of Proposition 1, $\lambda_u = \lambda_v$ must hold for the Lagrangian multipliers as defined in (23) whenever neither of the first two conditions hold. In this case, however, the derivatives are different and the first term in the numerator simplifies to

$$\frac{\partial \sum_{j=0}^{n} c_j(H_j)}{\partial r_i(w)} = \frac{\partial c_{i+1}(H_{i+1})}{\partial r_i(w)} = c_{i+1}'(H_{i+1})\frac{\partial H_{i+1}}{\partial r_i(w)} \quad (33)$$

with

$$\frac{\partial H_{i+1}}{\partial r_i(w)} = \frac{dt}{r_i^2(w)} \cdot \int_w^{d_i} \frac{Q_{i+1}(\tau)\cdot\Pi_{i+1}'(\tau)}{r_{i+1}(\tau)}d\tau; \quad (34)$$

whereas, if $k_i'(r_i^{-1}(w))$ exists, the second term yields

$$\sum_{j=1}^{i+1} \frac{\partial \int_{s_j}^{d_j} k_j(r_j^{-1}(t))dt}{\partial r_i(w)} = \left(k_{i+1}(r_{i+1}^{-1}(d_i)) + \frac{\partial d_{i+1}}{\partial r_i(w)} - \frac{dt}{r_i^2(w)} \left[k_i'(r_i^{-1}(w)) + \frac{r_i'(d_i)}{r_i(d_i)} \cdot k_i(r_i^{-1}(d_i)) \right] \right). \quad (35)$$

The denominator in turn becomes

$$\frac{\partial \sum_{j=0}^{n}(d_j - s_j - y_j)}{\partial r_i(w)} = \frac{\partial d_{i+1}}{\partial r_i(w)} = r_{i+1}(d_{i+1}) \cdot \frac{\partial H_{i+1}}{\partial r_i(w)}. \quad (36)$$

Inserting these results into $\lambda_u = \lambda_v$ yields, finally,

$$\frac{k_i'(r_i^{-1}(u)) + [r_i'(d_i)/r_i(d_i)] \cdot k_i(r_i^{-1}(d_i))}{\int_u^{y_{i+1}} [Q_{i+1}(\tau) \cdot \Pi_{i+1}'(\tau)/r_{i+1}(\tau)]\, d\tau} = \frac{k_i'(r_i^{-1}(v)) + [r_i'(d_i)/r_i(d_i)] \cdot k_i(r_i^{-1}(d_i))}{\int_v^{y_{i+1}} [Q_{i+1}(\tau) \cdot \Pi_{i+1}'(\tau)/r_{i+1}(\tau)]\, d\tau}. \quad (37)$$

Note that $\Pi_{i+1}(t)$ is increasing, and both $Q_{i+1}(\tau)$ and $r_{i+1}(t)$ are strictly positive. Hence, for $u < v$ the denominator on the left-hand side of (37) is strictly larger than that on the right-hand side. Equation (37) can therefore only hold if $k_i'(r_i^{-1}(u)) > k_i'(r_i^{-1}(v))$ which implies by the convexity of $k_i(r_i^{-1})$, that $r_i^{-1}(u) > r_i^{-1}(v)$. □

Proof of Proposition 3. Let $v < d_{i-1}$. In that case, (31) must still hold (with $\Psi_i(u) = 0$) and the proposition follows. Similarly, if $v > s_{i+1}$, then (37) must hold, where the lower boundary on the integral on the left-hand side is now 0 instead of u, and again the proposition follows directly. Finally, let $v \in [d_{i-1}, s_{i+1}]$. In that case, both (31) and (37) simplify to $k_i'(r_i^{-1}(u)) = k_i'(r_i^{-1}(v))$, and $r_i^{-1}(u) = r_i^{-1}(v)$ follows for strict convexity, whereas otherwise the cost function is linear between $r_i^{-1}(u)$ and $r_i^{-1}(v)$ and both values can be changed to the arithmetic middle without changing completion time or costs. □

Proof of Proposition 4. It suffices to show that there is an optimal solution such that $r_i^{-1}(t) \in \{1, \bar{r}_i^{-1}\}\ \forall t$. The proposition then follows directly from the preceding propositions. Suppose not, and let there be an interval W such that $1 < r_i^{-1}(t) < \bar{r}_i^{-1}$ for all $t \in W$. If this interval is during phase $i/1$, then (31) is violated for all $u, v \in W$, because $k_i'(r_i^{-1}(u)) = k_i'(r_i^{-1}(v)) = k_{i,j}$. Likewise, if the interval is during phase $i/3$, (37) is violated and such a policy cannot be optimal. Note that during phase $i/2$, $k_i'(r_i^{-1}(u)) = k_i'(r_i^{-1}(v)) = k_{i,j}$ holds for any $r_i^{-1}(t)$. Thus, optimal solutions with $r_i^{-1}(t) \notin \{1, \bar{r}_i^{-1}\}$ are possible during this interval, but an equivalent solution must exist where $r_i^{-1}(t) = \bar{r}_i^{-1}$ over some continuous interval and $r_i^{-1}(t) = 1$ for the remaining time of the entire stage. □

Appendix 2. Solution Procedure

In this appendix we present a solution procedure (**OCA**) for the OCP/Lin. To that end, we observe that the marginal costs strictly decrease with y_i. For x_i and z_{i-1}, costs are strictly increasing as long as $x_i \geqslant y_i$ and $z_{i-1} \geqslant y_i$, respectively. Beyond these points they remain constant at $\lambda(x_i) = -k_i$ and $\lambda(z_{i-1}) = -k_{i-1}$. This property allows us to map marginal costs onto efficient solutions, thus deriving in each iteration of the algorithm a point on the optimal time-cost trade-off curve.

Overlapping Crashing Algorithm (OCA)

Step 0-1 (Initiation). $\underline{\lambda} = H_0 = x_0 = y_0 = y_{n+1} = d_{-1} = 0$.
Step 0-2 (Feasibility Check). $x_i = z_i = 0$,
$y_i = \min[\bar{r}_{i-1} \cdot (T_{i-1} + H_{i-1}), \bar{r}_i \cdot (T_i + H_i)]$.
If $d_n > D$: Stop, "The Problem is Infeasible."
Else, if $d_n < D$: set $-\bar{\lambda} = M$ "a large number."
Do While $d_n \neq D$
Step 1. $\lambda = (\bar{\lambda} + \underline{\lambda})/2$, $I = \{i \mid k_i = -\lambda\}$.
Step 2. For $i = \leqslant 1, 2, \ldots, n$ do:
For the current value of λ determine z_{i-1}, y_i, and x_i according to Table 2.
If $y_i > \min[d_{i-1} - d_{i-2}, d_i - s_i]$, then $y_i = \min[d_{i-1} - d_{i-2}, d_i - s_i]$.
If $-\lambda \leqslant k_n$, then $z_n = T_n + H_n - x_n$, else $z_n = 0$.
Step 3. Determine d_n.
If $d_n < D$, then $\bar{\lambda} = \lambda$.
Else If $d_n > D$, then $\underline{\lambda} = \lambda$,
$z_i = y_{i+1}\ \forall\ i \in I$; Determine d_n.
If $d_n < D$, then
$$z_i = y_{i+1} + \frac{(D - d_n) \cdot (T_i + H_i - y_i - y_{i+1})}{\sum_{j \in I} (1 - r_j) \cdot (T_i + H_i - y_i - y_{i+1})}\ \forall i \in I;$$
Loop

After initiating and checking for feasibility, the main loop performs a binary search over the marginal costs, computing the values of the decision variables from Table 2. After comparing the resulting due date with the required one, the marginal costs are adjusted appropriately and a new iteration begins until the resulting due date coincides with the required due date. Because the binary search can be performed in $\log M$ steps (with M being a sufficiently large number), and because the values for x_i, y_i, and z_i can be computed in at most $\log D$ steps, the overall computational effort of the algorithm is $O(n \cdot \log \bar{\lambda} \cdot \log D)$. An important characteristic of the OCA is that it generates an optimal schedule for d_n in each iteration, thus yielding the efficient time-cost frontier for a project rather than a single solution for a given due date D.

Endnotes

1. In this case, we determined the proportions relative to the standard durations and crashing costs of the stages, such that of a given budget B, Stage 1 receives $b_1 =$

$BT_1k_1/(T_0k_0 + T_1k_1)$ and Stage 0 receives the remainder, $b_0 = B - b_1$.

2. All three cost curves for fast evolution also have an extreme point, but for values of $q > 1$. Because this would require that any work performed erroneously causes more time to redo than it initially took, we cut the curve off at $q = 1$.

3. Note, that over- (under-)estimating of the parameters leads to shorter (longer) than planned development times.

Acknowledgments

The authors are indebted to two anonymous referees whose invaluable feedback on the paper "Time-Cost Trade-Offs in Overlapped Product Development" (Roemer, Ahmadi, and Wang 2000) planted the seed for this research. The authors also gratefully acknowledge the contributions of the referees to this paper, which led to a much improved final version.

References

Abdel-Hamid, T., S. E. Madnick. 1990. *Software Project Dynamics: An Integrated Approach.* Prentice-Hall, Englewood Cliffs, NJ.

Ahmadi, R., R. H. Wang. 1999. Managing development risk in product design processes. *Oper. Res.* **47**(2) 235–246.

Ahmadi, R., T. A. Roemer, R. H. Wang. 2001. Structuring product development processes. *Eur. J. Oper. Res.* **130** 539–558.

Baldwin, C. Y., K. B. Clark. 2000. *Design Rules*. MIT Press, Cambridge, MA.

Berman, E. B. 1964. Resource allocation in a PERT network under continuous activity time-cost functions. *Management Sci.* **10**(4) 734–745.

Bhattacharya, S., V. Krishnan, V. Mahajan. 1998. Managing new product definition in highly dynamic environments. *Management Sci.* **44** S50–S64.

Boehm, B. W. 1981. *Software Engineering Economics.* Prentice-Hall, Englewood Cliffs, NJ.

Brooks, F. P. 1995. *The Mythical Man Month.* Addison-Wesley, Reading, MA.

Carrascosa, M., S. D. Eppinger, D. E. Whitney. 1998. Using the design structure matrix to estimate product development time. *Proc. ASME Design Engrg. Tech. Conf.*, Atlanta, GA.

Clark, K. B., T. Fujimoto. 1989. Overlapping problem solving in product development. K. Ferdows, ed. *Managing International Manufacturing.* Elsevier Science, Amsterdam, The Netherlands.

Clark, K. B., T. Fujimoto. 1991. *Product Development Performance.* Harvard Business School Press, Boston, MA.

Cordero, R. 1991. Managing for speed to avoid product obsolescence: A survey of techniques. *J. Product Innovation Management* **8**(4) 283–294.

Cooper, R. G., E. J. Kleinschmidt. 1996. Winning businesses in product development: The critical success factors. *Res. Tech. Management* **39**(4) 18–29.

Eisenhardt, K. M., B. N. Tabrizi. 1995. Accelerating adaptive processes: Product innovation in the global computer industry. *Admin. Sci. Quart.* **40** 84–110.

Eppinger, S. D. 2001. Innovation at the speed of information. *Harvard Bus. Rev.* **79**(January) 3–11.

Eppinger, S. D., D. E. Whitney, R. P. Smith, D. A. Gebala. 1994. A model-based method for organizing tasks in product development. *Res. Engrg. Design* **6** 1–13.

Ettlie, J., S. A Reifeis. 1987. Integrating design and manufacturing to deploy advanced manufacturing technology. *Interfaces* **17**(6) 63–74.

Falk, J. E., J. L Horowitz. 1972. Critical path problems with concave cost-time curves. *Management Sci.* **19** 446–455.

Foldes, S., F. Soumis. 1993. PERT and crashing revisited: Mathematical generalizations. *Eur. J. Oper. Res.* **64** 286–294.

Fulkerson, D. R. 1961. A network flow computation for project cost curves. *Management Sci.* **7**(2) 167–178.

Graves, S. B. 1989. The time-cost tradeoff in research and development: A review. *Engrg. Costs Production Econom.* **16** 1–9.

Ha, A. Y., E. L. Porteus. 1995. Optimal timing of reviews in concurrent design for manufacturability. *Management Sci.* **41**(9) 1431–1447.

Hoedemaker, G. M., J. D. Blackburn, L. N. Van Wassenhove. 1999. Limits to concurrency. *Decision Sci.* **30**(1) 1–17.

Iansiti, M., A. MacCormack. 1997. Developing products on Internet time. *Harvard Bus. Rev.* **75**(5) 108–117.

Imai, K., I. Nonaka, H. Takeuchi. 1985. Managing the new product development process: How the Japanese companies learn and unlearn. K. B. Clark, R. H. Hayes, C. Lorenz, eds. *The Uneasy Alliance*. Harvard Business School Press, Boston, MA.

Kalyanaram, G., V. Krishnan. 1997. Deliberate product definition: Customizing the product definition process. *J. Marketing Res.* **34**(2) 276–285.

Kanda, A., U. R. K. Rao. 1984. A network flow procedure for project crashing with penalty nodes. *Eur. J. Oper. Res.* **16** 174–182.

Kelley, J. E., M. R. Walker. 1959. Critical-path planning and scheduling. *Proc. Eastern Joint Comput. Conf.*, 160–173.

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

Krishnan, V., S. D. Eppinger, D. E. Whitney. 1997. A model-based framework to overlap product development activities. *Management Sci.* **43**(4) 437–451.

Leachman, R. C., S. Kim. 1993. A revised critical path method for networks including both overlap relationships and variable-duration activities. *Eur. J. Oper. Res.* **64** 229–248.

Loch, C. H., C. Terwiesch. 1998. Communication and uncertainty in concurrent engineering. *Management Sci.* **44**(8) 1032–1048.

MacCormack, A., R. Verganti, M. Iansiti. 2001. Developing products on "Internet time": The anatomy of a flexible development process. *Management Sci.* **47**(1) 133–150.

Mansfield, E., J. Rapaport, J. Schnee, S. Wagner, M. Hamburger. 1972. *Research and Innovation in the Modern Corporation*. Norton, New York.

McCord, K. R., S. D. Eppinger. 1993. Managing the integration problem in concurrent engineering. Working paper #3594-93-MSA, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA.

Pisano, G. P. 1997. *The Development Factory.* Harvard Business School Press, Boston, MA.

Putnam, L. H., W. Myers. 1992. *Measures for Excellence: Reliable Software on Time, Within Budget*. Yourdon Press, Englewood Cliffs, NJ.

Roemer, T. A., R. Ahmadi, R. H. Wang. 2000. Time-cost trade-offs in overlapped product development. *Oper. Res.* **48**(6) 858–865.

Smith, P. G., D. G. Reinertsen. 1995. *Developing Products in Half the Time*, 2nd ed. Van Nostrand Reinhold, New York.

Takeuchi, H., I. Nonaka. 1986. The new product development game. *Harvard Bus. Rev.* (Jan.–Feb.) 137–146.

Terwiesch, C., C. H. Loch. 1999. Measuring the effectiveness of overlapping development activities. *Management Sci.* **45**(4) 455–465.

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

Thomke, S., T. Fujimoto. 2000. The effect of "front-loading" problem-solving on product development performance. *J. Product Innovation Management* **17**(2) 128–142.

Ward, A., J. K. Liker, J. J. Cristiano, D. K. Sobek. 1995. The second Toyota paradox: How delaying decisions can make better cars faster. *Sloan Management Rev.* **36**(Spring) 43–61.

Whitney, D. E. 1990. Designing the design process. *Res. Engrg. Design* **2** 3–13.

Copyright 2004, by INFORMS, all rights reserved. Copyright of Operations Research is the property of INFORMS: Institute for Operations Research and its content may not be copied or emailed to multiple sites or posted to a listserv without the copyright holder's express written permission. However, users may print, download, or email articles for individual use.