The Cost of Speed: Work Policies for Crashing and Overlapping in Product Development Projects

Christoph Meier
Institute of Astronautics
Technische Universität München
85748 Garching, Germany
meier@combinatic.com

Tyson R. Browning*
Neeley School of Business
Texas Christian University
TCU Box 298530
Fort Worth, TX 76129
USA
t.browning@tcu.edu

Ali A. Yassine
Engineering Management
American University of Beirut
Beirut, Lebanon
ali.yassine@aub.edu.lb

Ulrich Walter
Institute of Astronautics
Technische Universität München
85748 Garching, Germany
walter@tum.de

This is a preprint of the final version, which appeared in:

***IEEE Transactions on Engineering Management*, Vol. 62, No. 2, pp. 237-255, 2015.**

*Corresponding author

The first author would like to thank the Bavarian Science Foundation. The second author is grateful for support from the Neeley Summer Research Award Program from the Neeley School of Business at TCU and a grant from the U.S. Navy, Office of Naval Research (grant no. N00014-11-1-0739). The third author is grateful for support from the University Research Board (URB) program at the American University of Beirut.

The Cost of Speed: Work Policies for Crashing and Overlapping in Product Development Projects

Abstract

Project management theory provides insufficient insight on the effects of crashing and overlapping in product development (PD) projects. The scholarly literature largely assumes that networks of project activities are acyclical. PD projects are iterative, however, and the time-cost implications of managerial actions, such as activity crashing and overlapping, differ in this context. We build a rich model that accounts for activity overlapping, crashing, and iteration in the project's activity network. Using a bank of thousands of artificial but realistic test problems, we test the performance of several alternative *work policies*—managerial rules about when to start work and rework, and when to allow crashing and overlapping—and compare their time-cost implications. We find that the time-cost benefits of crashing and overlapping increase with increasing iteration. However, whereas crashing works well in combination with an aggressive work policy that does not seek to minimize iteration and rework, overlapping ironically works better with a work policy that defers rework. No single work policy dominates in all situations, although an aggressive policy that precipitates rework may often be worth its added cost. We distill our findings into a set of propositions that pave the way for further theory development in this area. Our results also illuminate useful heuristics for managers of PD projects.

Keywords: *Project management*; *Iteration*; *Crashing*; *Overlapping*; *Work policy*; *Time-cost tradeoffs*; *Product development*.

1. Introduction

In today's competitive business landscape, improved management of product development (PD) projects can provide significant competitive advantage [1, 2]. In particular, managerial decisions about when to start and stop activities affect project duration and cost significantly. Although a substantial literature exists on project scheduling [e.g., 3], that work assumes an *acyclical* network of activities—i.e., that each activity will occur only once. However, PD projects are notorious for being highly iterative (cyclical) [e.g., 4, 5-7] due to the presence of (often uncertain) feedback loops. Managerial decisions in this context are subject to the challenges of understanding and managing complex systems, such as non-linear feedback and unintended consequences [e.g., 6, 8, 9]. Thus, project management theory regarding scheduling and crashing activities, developed for acyclical processes, can be misleading in the context of iterative PD projects. Moreover, the theory regarding activity overlapping has been developed mainly for activity dyads, not for entire activity networks: thus, again, the theory could also be misleading at the overall process level. Challenges such as these remain poorly understood. Managers of PD projects require further insights into the effects of their decisions.

This paper explores managerial decisions about when to start and stop activities—i.e., when to do work and rework, and when to crash or overlap activities—on the outcomes of *cyclical* processes with probabilistic iterations, which are representative of those encountered in PD projects. From the project management literature, we know that activity crashing (i.e., reducing activity duration at some added expense) provides a way to trade off project time and cost [e.g., 10, 11]. However, crashing is only effective at the overall process level when applied to the subset of activities that actually determines the project duration (i.e., the critical activities). In a dynamic, iterative process, such activities are more difficult to identify because they often change. *How efficient is crashing in the presence of dynamic activity iteration?* Activity overlapping can also affect project duration [e.g., 11, 12, 13]. While crashing attempts to shorten the PD process with intra-activity changes, overlapping looks at inter-activity relationships. Overlapping two sequentially-dependent activities will cause the second to proceed with preliminary instead of final output from the first, thus creating the potential for iteration and rework to reconcile any discrepancies. When an activity starts, and what information it uses in doing so, thus has major implications. Loch and Terwiesch [14] explored this basic dilemma as "rush and be wrong" (potentially causing an iteration) or "wait and be late" (either extending project duration or pushing the same dilemma onto subsequent activities). Yet, most of the research on overlapping has occurred at the two-activity level [15]. *How efficient is overlapping with respect to the entire activity network (process) in the presence of iteration?* Overall, both crashing and overlapping have significant effects on project duration and cost, and it is important to study them at the process level (not just the activity level), especially in the contexts of iteration and uncertainty that characterize PD projects.

Ultimately, both crashing and overlapping boil down to managerial decisions about when to start and stop activities. We call a set of rules about when to start and stop work a *work policy*. A work policy can include criteria for crashing and overlapping activities, and for when to wait for final inputs versus when

to proceed with preliminary ones (or only assumptions) instead. Other researchers have illuminated aspects of work policy by building and analyzing process models. For example, Roemer and Ahmadi [16] built an analytical model to demonstrate the effects of overlapping and crashing on time-cost tradeoffs. However, this work assumes a deterministic, acyclical process and only investigates overlapping between two consecutive activities (or stages). Browning and Eppinger [17] simulated a stochastic model to explore the effects on duration and cost in iterative projects of waiting versus proceeding with assumptions. They allowed for a limited amount of overlapping but not crashing. Research has not yet explored the effects of varied work policies with respect to crashing and overlapping in processes with stochastic iteration.

This paper increases our understanding of the effects of work policies on the cost and duration outcomes of PD projects in such a context. We provide a rich model that enables exploration of the effects of crashing and overlapping at both the individual activity and the overall process levels. The model accounts for stochastic iterations and allows activity attributes to vary across iterations (due to learning curve effects and rework impacts that vary with the cause of rework). The model provides a new lens for insights into project performance and can be used to identify specific leverage points for improvement (e.g., particular activities or relationships to change, although that is not our focus in this paper). The paper's main contribution is to develop and evaluate the model to investigate the implications of alternative work policies, such as which activities should wait for missing inputs, begin without all of their inputs (by making assumptions), or be overlapped or crashed. Based on tests with thousands of artificial (but realistic) projects, we find that *significant time-cost tradeoffs emerge* not only from variations in the structure of the activity network (i.e., the process architecture) but also *from a manager's choice of work policy*. Furthermore, we find that *crashing and overlapping are often quite advantageous in iterative processes*: as process dynamics increase, the benefits of time compression strategies usually justify their increased costs. Crashing and overlapping also become more attractive as the overall, natural activity concurrency (enabled by the dependency network) increases—a finding that runs counter to the conventional wisdom on crashing in acyclic networks. And whereas crashing works well in combination with an aggressive work policy that does not seek to minimize iteration and rework, *overlapping ironically works better with a work policy that defers rework*. No single work policy dominates in all situations: work policies invite significant time-cost tradeoffs and therefore should be considered by managers of iterative projects. Since a lack of existing theory on this topic makes it premature to develop specific hypotheses on many of these effects, we present our findings as a set of propositions that build theory with respect to PD project management and serve as a basis for further empirical research. The results thus contribute towards a theoretical framework to address a highly important and practical topic, the management of work in PD projects.

2. Foundational Concepts and Related Literature

Iteration and overlapping have been explored extensively in the PD literature, as has crashing (albeit only with respect to acyclic processes) in the project management literature. This section provides a brief background on these topics, concluding with some review of time-cost tradeoffs in projects.

2.1 Activity Crashing

In the project management literature, the standard way to reduce project duration is to expedite critical activities by assigning additional resources (e.g., more personnel, overtime, or faster or better tools), an approach known as "crashing" [10, 11]. Crashing shortens an activity's duration at some added cost,[1] thereby decreasing the overall process duration if the activity is on the critical path, and thus providing a time-cost tradeoff opportunity [16, 19]. The new, shorter process retains the original dependency structure, so the process architecture itself remains unchanged. Kelley and Walker [20] first computed the least-cost curve for a project composed of "crash-able" activities as a parametric linear problem. Kelley [21] and Fulkerson [22] independently proposed two different algorithms based on network flow methods to solve this problem with a linear cost function. Later publications investigated more complex cost functions exhibiting convexity [23] or concavity [24], assigning penalties to pivotal activities [25], or representing different types of costs [26] and advanced optimization techniques [e.g., 27]. Despite its prevalence in the literature, however, crashing has not been explored in iterative processes such as PD. In the context of a PD project, a key question is: Should work be sped up if it is just going to have to be redone anyway?

2.2 Activity Overlapping

The overlapping of dependent activities is one of the basic principles of concurrent engineering [28, 29].[2] In a manufacturing process, an assembly activity cannot occur unless its requisite inputs are physically present: the dependencies are mainly "hard." In projects, however, the inputs needed by an activity may be information (data, results of decisions, etc.), and often much or even all of an activity can be done based on assumptions in lieu of this actual information: many dependencies are "soft." Yet this opportunity is a double-edged sword, because sometimes it helps and sometimes it hurts the overall process. If the assumptions are good, if the inputs are fairly stable (not volatile), if changes to the inputs will have a relatively small impact, or if the activity can be reworked much faster than its original duration (due to a large set-up time or learning curve), then letting an activity begin without all of its input information can reduce the overall time and cost of the process [17]. Yet, if some of these conditions are not met, then starting an activity without all of its inputs can do more harm than good, often by causing unnecessary iterations [13] that can consume up to 50% of design capacity [30], and sometimes even leading to perverse situations such as "design churn" [8, 9]. So, which activities should be allowed to begin without all of their inputs, and which should be forced to wait? Beginning early allows some activities to get off the critical path, albeit at the expense of some additional rework thereon [17]. Will this rework take less time than the full activity, and will the time savings justify the extra cost [14]? Overlapping therefore creates another time-cost tradeoff for managers, but this has not been considered in most of the literature (see [31] and below in section 2.4 for exceptions).

2.3 Activity Iteration

PD and innovation processes are inherently iterative (or cyclical) [4-6]. Doing something novel and

[1] Brooks [18] noted that adding resources can sometimes have the opposite effect.

[2] The parallel execution of independent activities ("natural concurrency") does not cause rework; it is the parallel execution of dependent activities, despite their dependency relationships, which we refer to as overlapping ("artificial concurrency").

complex with a multi-disciplinary project team requires some degree of experimentation [32], information transfer [33], and convergence on a satisfactory solution [34]. Abstractly, iteration represents the rework of an activity due to deficiencies with its prior results. These deficiencies could be planned (e.g., an activity deliberately defers work, as in "spiral development"), or they could be unplanned problems due to failure to meet requirements, detected errors (signaled via feedback from later activities), or changes in the inputs upon which the work was done (e.g., requirements changes, design changes [35], or erroneous outputs from other activities), to name but a few of the common causes [36]. Since iteration and rework drive a significant portion of PD time and cost [37, 38], their effective management is required to plan and control project cost, duration, quality, and risk [15]. Iteration can be managed in many ways, including by improving the sequence of design activities to streamline information flows [39, 40], developing new engineering automation tools to perform iterations faster [41], adding resources at bottleneck activities [9], and front-loading iterations to discover and eliminate risky design issues more quickly [42]. It is important to recognize that the successful application of these methods essentially requires an appropriate model of the PD process, one that accentuates the activities, the information flows that drive their dependencies, and the resulting iterations [43].

Some PD process models utilize a design structure matrix (DSM) representation, a square matrix showing activities along its diagonal, and, in one convention, activity outputs in its columns and inputs in its rows [44]. Here, typical feed-forward dependencies appear as marks below the diagonal, while marks in super-diagonal cells indicate feedbacks. As the number of activities and relationships grows, the DSM provides advantages over digraphs for visualizing the paths of iterations.

A design process can often be improved by reordering the activities such that the amount of feedback (number of super-diagonal marks in the DSM) and the extent of feedback (distance of marks from the diagonal in the DSM), and thus the amount and impact of iteration, are minimized. An ideal sequence of activities is initially considered as one with no feedback marks, represented by a lower-triangular DSM. In a real PD process, however, such a sequence is unlikely to exist [45], in which case one may seek the sequence of design activities that minimizes feedback. It has been shown that this problem is an NP-hard combinatorial optimization problem, usually approached with heuristics or meta-heuristics [39, 46]. However, the fastest PD processes may not actually be the ones with minimal iteration [17], because the additional iteration caused by appropriate activity overlapping can often be advantageous [13]. Thus, iterations also contribute to managers' time-cost tradeoff opportunities.

Meanwhile, the project management literature [e.g., 10, 11] assumes that project processes are acyclical (no iteration) and that all dependencies are "hard," meaning that the possibilities and implications of replacing inputs with assumptions are not considered. What are the effects of iteration on the efficacy of managerial decisions about work?

2.4 Time-Cost Tradeoffs

Even with the well-documented importance of speed-to-market for successful PD [e.g., 47, 48], there are limits to the amount of money a firm can pump into its projects. Firms often have a predefined budget

for a PD project, based on a business case, or a hard deadline based on an external event. A time-cost tradeoff arises as organizations seek the fastest PD process within a predefined budget, or the cheapest PD process for a given project duration.

In the project management literature, the time-cost tradeoff has been treated mainly as a crashing problem, and research to date has assumed an acyclic process. Recently, Roemer et al. [49], Roemer and Ahmadi [16], and Gerk and Qassim [50] bridged the gap between the literature on overlapping and crashing by exploring both strategies jointly. However, these models neglect dynamic iteration, which significantly increases the complexity of analytical models and inhibits the derivation of closed-form solutions.

Meanwhile, the PD literature has explored the time-cost implications of iterative projects, particularly through the concept of iterative overlapping [13], where particular activities on the critical path are started earlier (overlapped with other activities) so that only the additional rework created (instead of the entire activity) occurs on the critical path [17]. This approach increases cost but saves time, and it should only be applied to critical activities and where the rework time is likely to be much less than the original time. At an abstract level, Kalyanaram and Krishnan [51] modeled an overall PD process in terms of general amounts of crashing and concurrency and weighed those options against the benefits of improvements in product quality. Overall, however, the models in the PD literature ignore crashing (or account for it only in a general sense) and account for overlapping only in a very general or limited way.

Thus, while the project management literature has addressed the time-cost tradeoff in terms of crashing acyclical processes, and a small amount of work has addressed the tradeoff with respect to iterative overlapping, no study exists on both crashing and overlapping in iterative processes. Without a more comprehensive, process-level view, it is possible for the conventional managerial guidance at the activity level to be misleading and even detrimental to projects. However, gaining such a comprehensive view requires departing from the limitations of analytical models and utilizing carefully designed simulations.

3. A Rich Model of Project Cost and Duration with Crashing and Overlapping

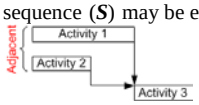
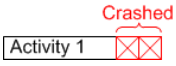
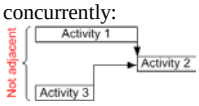
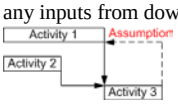
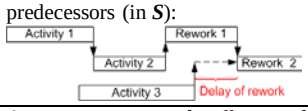
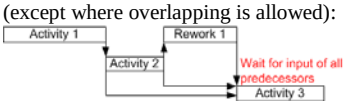
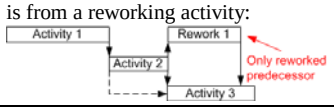
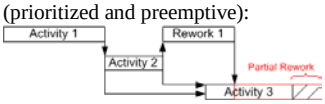
A thorough investigation of time-cost tradeoffs in PD processes requires a rich model that addresses both intra- and inter-activity effects. This section presents the model and its motivations, starting with a discussion of process architecture and work policy and then addressing overlapping and crashing.

3.1 Process Architecture

We model a PD process as an activity network, where the dependencies result from input-output relationships among the activities [15]. A process consists of n_A disjoint activities and their dependencies, which are captured in a DSM, $\boldsymbol{M}$, in a sequence given by the vector $\boldsymbol{S}$. Multiple layers of the DSM each capture a particular attribute of the dependencies. The first layer, M_1, with elements $m_{1,ij}$ ($i \neq j$), records the probability of activity j causing rework for activity i. Should rework occur, M_2, with elements $m_{2,ij}$ records the fraction of activity i that must be redone. For instance, $m_{1,ij} = 0.2$ and $m_{2,ij} = 0.9$ implies that the outputs from activity j have a 20% chance of causing 90% of activity i to be redone. For most PD processes, any sequence of the activities will result in some of the dependencies falling below the diagonal (feed-forward) and others occurring above the diagonal (feedback). By altering the order of the activities

in the DSM (i.e., by changing ***S***), various dependencies will change from feed-forward to feedback or vice-versa. Since an activity may react differently (in the decision to wait versus proceeding with assumptions) depending on whether a particular input is coming from an upstream or downstream activity, varying ***S*** manipulates the *process architecture* [17, 44] in terms of the way the activities interact. We can also explore alternative process architectures by comparing processes with different relationship structures among the activities, such as numbers of feed-forward and/or feedback dependencies (additional dependencies place more constraints on activity scheduling) and the positioning of those dependencies in the activity network, as we will discuss in §4.

Table 1: Overview of the managerial rules assigned to each work policy.

Rule	P_1	P_2	P_3	P_4	P_5	Rule	P_1	P_2	P_3	P_4	P_5
Adjacent, independent activities in the activity sequence (***S***) may be executed concurrently: 	✓	✓	✓	✓	✓	An activity may be crashed: 	×	×	✓	✓	✓
Non-adjacent activities in ***S*** may be executed concurrently: 	×	✓	✓	✓	✓	Adjacent activities (in ***S***) may be overlapped: 	×	×	✓	✓	✓
Activities may begin with assumptions about any inputs from downstream (in ***S***) activities: 	✓	✓	✓	✓	✓	Downstream rework is deferred until the completion of any rework by all predecessors (in ***S***): 	×	×	×	✓	×
Activities must wait for all inputs from upstream activities (in ***S***), even if reworking (except where overlapping is allowed): 	✓	✓	✓	✓	×	Activities must wait for all inputs from upstream activities (in ***S***), *unless* that input is from a reworking activity: 	×	×	×	×	✓
Upstream rework is accomplished immediately (prioritized and preemptive): 	✓	✓	✓	✓	✓						

3.2 Work Policy

A work policy is a set of *a priori* rules (management decisions) about when to start and stop work [17]. It can include rules for crashing and overlapping activities, and for waiting on inputs versus proceeding based on assumptions. We investigate five alternative work policies for iterative processes, designated P_1-P_5 and described in Table 1. Policy P_1 was adopted from Browning and Eppinger [17] and can be regarded as a conservative policy, because it limits the amount of activity concurrency by allowing only independent activities in M_1 (i.e., activities for which $m_{1,ij} = 0$) that are also adjacent in ***S*** to work simultaneously. P_2 permits non-adjacent activities in ***S*** to work concurrently, thereby getting more work done earlier, albeit sometimes at a greater risk of having to be reworked. P_3 adds crashing and overlapping options to P_2. To analyze the impact of additional rules for the timing of rework, we defined a more sequential, rework-deferring policy, P_4, and a more concurrency-oriented policy, P_5. P_4 presumes that it is beneficial to delay rework if any of an activity's predecessors (in ***S***) have not yet finished (see Table 1), because these predecessors might deliver new results that could cause further rework. Delaying the

activity's start until the completion of all of its predecessors obviates this problem, but this policy wastes time if an activity's predecessors do not in fact alter its inputs. Hence, P_5 relaxes this restriction, allowing an activity to proceed despite ongoing rework by its predecessors. This policy would be expected to decrease process duration even further while increasing rework and cost. Table 2 summarizes the five work policies. We will compare the results obtained with these varied work policies to provide practical insights for mangers about which policy to use in various situations.

Table 2: Summary of work policies studied.

P_1	Most conservative; limits activity concurrency to that specified in planned order of activities; no crashing or overlapping
P_2	Increases "natural concurrency" by identifying all opportunities to start activities as early as possible; no crashing or overlapping
P_3	P_2 with allowance for crashing and overlapping ("artificial concurrency")
P_4	P_3 with delay of activity rework until all of its predecessors are finished with known rework
P_5	Most aggressive; P_3 with performance of activity rework even if its predecessors are unfinished

3.3 Activity Cost and Duration

Respectively, c_i and t_i represent the base cost and duration of activity i. The modified cost and duration in iteration k_i due to any learning curve and the rework impact are then:

$$\tilde{c}_i(k_i) = c_i l_i(k_i) m_{2,ij} \quad (1)$$

$$\tilde{t}_i(k_i) = t_i l_i(k_i) m_{2,ij} \quad (2)$$

where $m_{2,ij}$ is the rework impact and $l_i(k_i)$ is the learning curve effect, modeled as a percentage of the activity's original duration. Three more effects may alter an activity's duration and cost. $\Delta c_{C_i(k_i)}$ and $\Delta t_{C_i(k_i)}$ record changes in cost and duration due to crashing activity i in iteration k_i, $\Delta c_{O_i(k_i)}$ and $\Delta t_{O_i(k_i)}$ capture changes as a result of overlapping, and $\Delta c_{R_i(k_i)}$ and $\Delta t_{R_i(k_i)}$ account for partial rework. Collectively:

$$c_i(k_i) = \tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} + \Delta c_{C_i(k_i)} + \Delta c_{R_i(k_i)} \quad (3)$$

$$t_i(k_i) = \tilde{t}_i(k_i) + \Delta t_{O_i(k_i)} + \Delta t_{C_i(k_i)} + \Delta t_{R_i(k_i)} \quad (4)$$

We explain these terms in the following subsections. The finish time for each iteration of each activity is determined by its start time and its duration: $T_{f_i(k_i)} = T_{s_i(k_i)} + t_i(k_i)$.

3.4 Activity Overlapping

An overlapping decision schedules an activity's start time with consideration of the inputs received by that point. Normally, inputs come from predecessors, but overlapping creates "artificial concurrency." Furthermore, in cyclic processes the source of an input may actually be a downstream activity providing feedback. Thus, the efficacy of overlapping decisions will depend on process architecture.

3.4.1 Base Case: Overlapping Based on Finish-To-Start Relationships

Consider the overlapping of any two sequentially dependent activities, referred to as upstream activity i and downstream (dependent) activity j. As the amount of overlap between i and j increases, the overall duration decreases, but coordination costs and the likelihood of feedback from j causing rework for i both increase [13]. The main objective then is to find the optimal overlap of i and j that minimizes total duration, including rework, perhaps while accounting for costs as well.

Initially, we assume finish-to-start relationships between dependent activities *i* and *j*. In this case, all output from *i* occurs at $T_{f_i(k_i)}$, all input for *j* is needed at $T_{s_j(k_j)}$, and $T_{s_j(k_j)} = T_{f_i(k_i)}$. However, with preliminary output from activity *i*, activity *j* may overlap with *i* and $T_{s_j(k_j)} < T_{f_i(k_i)}$. While this reduces the overall time span, $T_{f_j(k_j)} - T_{s_i(k_i)}$, it may elongate *j* by a penalty time $\Delta t_{O_j(k_j)}$, since *j* began with imperfect inputs, thus adding some rework to correct any faulty assumptions (Figure 1). Thus, the amount of overlap between two dependent activities depends on the point in time when activity *i* delivers output in the k_i^{th} iteration to activity *j* in the k_j^{th} iteration, $T_{D_{ij}(k_i,k_j)}$, where $T_{D_{ij}(k_i,k_j)} \le T_{f_i(k_i)}$, and the point in time when *j* receives that input from *i*, $T_{R_{ij}(k_i,k_j)}$, where $T_{R_{ij}(k_i,k_j)} \ge T_{s_j(k_j)}$. Accordingly, with respect to the base case, overlapping occurs if $T_{s_j(k_j)} < T_{D_{ij}(k_i,k_j)} < T_{f_j(k_j)}$ or $T_{f_i(k_i)} > T_{R_{ij}(k_i,k_j)} > T_{s_i(k_i)}$ holds.

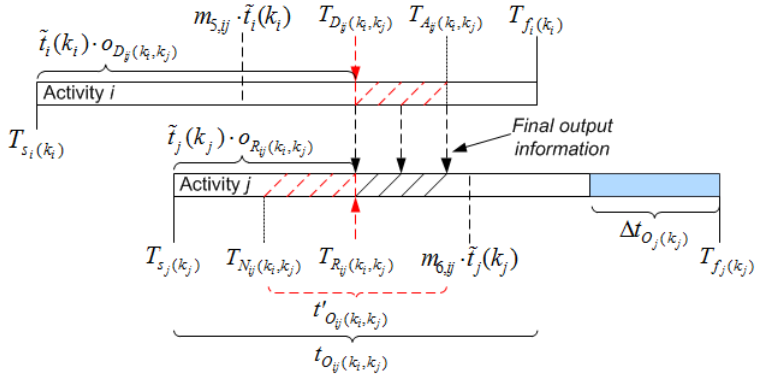


Figure 1: Example of overlapping two activities, displaying most of the overlapping parameters.

3.4.2 Extending the Base Case with Points in Time for Information Available and Needed

Some of an activity's final outputs may be available before the entire activity is finished, and some of an activity's inputs may not be needed until the activity is already underway. Instead of assuming finish-to-start relationships between all dependent activities, we can more generally assume that the complete and *final* output of activity *i* for activity *j* is available at $T_{A_{ij}(k_i,k_j)}$, where $T_{A_{ij}(k_i,k_j)} \le T_{f_i(k_i)}$, and, similarly, that activity *j* needs input at $T_{N_{ij}(k_i,k_j)}$, with $T_{N_{ij}(k_i,k_j)} \ge T_{s_i(k_i)}$. $T_{D_{ij}(k_i,k_j)} \le T_{A_{ij}(k_i,k_j)} \le T_{f_i(k_i)}$, $T_{A_{ij}(k_i,k_j)}$ constitutes an upper bound for $T_{D_{ij}(k_i,k_j)}$, while $T_{N_{ij}(k_i,k_j)}$ constitutes a lower bound for $T_{R_{ij}(k_i,k_j)}$ due to $T_{R_{ij}(k_i,k_j)} \ge T_{N_{ij}(k_i,k_j)} \ge T_{s_j(k_j)}$. To determine $T_{A_{ij}(k_i,k_j)}$, we must find the percentage of completion required of activity *i* before its final output is available for *j*. We capture these parameters in $M_{3,ij}$ and add this to its start time: $T_{A_{ij}(k_i,k_j)} = T_{s_i(k_i)} + \tilde{t}_i(k_i) m_{3,ij}$. Similarly, $T_{N_{ij}(k_i,k_j)}$ is calculated by adding the percentage of work of *j* that can occur before it requires complete input from *i*, recorded in $M_{4,ij}$: $T_{N_{ij}(k_i,k_j)} = T_{s_i(k_i)} + \tilde{t}_i(k_i) m_{4,ij}$. In a pure finish-to-start relationship, $m_3 = 1$ and $m_4 = 0$.

3.4.3 Bounds for Preliminary Information Needed and Available

To bound the time period in which $T_{D_{ij}(k_i,k_j)}$ and $T_{R_{ij}(k_i,k_j)}$ can occur, we also need to define a lower bound for $T_{D_{ij}(k_i,k_j)}$ and an upper bound for $T_{R_{ij}(k_i,k_j)}$. The lower bound for $T_{D_{ij}(k_i,k_j)}$ is a point in time constituting the percentage of activity *i*'s duration required before its *preliminary* output is available for a downstream activity *j*. In contrast, $T_{R_{ij}(k_i,k_j)}$ is bounded above by a point in time representing the percentage of activity *j* that can occur before it can benefit from any preliminary input from *i*. The former percentage is recorded in M_5, and $T_{s_i(k_i)} + \tilde{t}_i(k_i) m_{5,ij} \le T_{D_{ij}(k_i,k_j)}$. Any rework due to incomplete predecessor information is assigned to activity *j* if $T_{D_{ij}(k_i,k_j)} \in \left[T_{s_i(k_i)} + \tilde{t}_i(k_i) m_{5,ij},\ T_{A_{ij}(k_i,k_j)} \right]$. The percentage values for the upper bound of $T_{R_{ij}(k_i,k_j)}$ are recorded in M_6, so $T_{R_{ij}(k_j)} \le T_{s_j(k_j)} + \tilde{t}_j(k_j) m_{6,ij}$ holds and activity *j* is penalized with rework if $T_{R_{ij}(k_i,k_j)} \in \left[T_{N_{ij}(k_i,k_j)},\ T_{s_j(k_j)} + \tilde{t}_j(k_j) m_{6,ij} \right]$.

3.4.4 Generalized Determination of Overlapping Duration

To summarize, $m_{3,ij}$ is the percentage of activity *i* required to be done before its *final* output is available for *j*, $m_{4,ij}$ is the percentage of *j* which can occur without penalty before it requires *complete* input from *i*, $m_{5,ij}$ is the percentage of *i* that can occur without penalty before it can produce any *preliminary* output for *j*, and $m_{6,ij}$ is the percentage of *j* that can occur without penalty before it must receive *preliminary* information from *i*. With these bounds established, we can compute $T_{D_{ij}(k_i,k_j)}$ and $T_{R_{ij}(k_i,k_j)}$ in any iteration using the parameters $o_{D_{ij}(k_i,k_j)} \in \left[m_{5,ij}, m_{3,ij} \right]$ and $o_{R_{ij}(k_i,k_j)} \in \left[m_{4,ij}, m_{6,ij} \right]$ to represent the percentage of an activity which can be processed until it delivers/receives information. The following equations hold:

$$T_{D_{ij}(k_i,k_j)} = T_{s_i(k_i)} + \tilde{t}_i(k_i) o_{D_{ij}(k_i,k_j)} \quad \text{, if activity } j \text{ is not yet active} \tag{5}$$

$$T_{D_{ij}(k_i,k_j)} = T_{A_{ij}(k_i,k_j)} \quad \text{, if activity } j \text{ is already active} \tag{6}$$

$$T_{R_{ij}(k_i,k_j)} = T_{s_j(k_j)} + \tilde{t}_j(k_j) o_{R_{ij}(k_i,k_j)} \tag{7}$$

Equation (6) accounts for the special case when activity *i* is supposed to deliver preliminary output to an already active activity *j*—i.e., activity *j* began before $T_{D_{ij}(k_i,k_j)}$. In this event, it makes sense to wait until the final output of activity *i* is available (i.e., $T_{D_{ij}(k_i,k_j)} = T_{A_{ij}(k_i,k_j)}$) instead of feeding *j* with preliminary output. (Even though using preliminary output would be possible, its poor quality would provoke additional rework of *j* without any temporal benefits.)

Now we can compute the duration of overlapping for the k_i^{th} iteration of activity *i* and the k_j^{th} iteration of activity *j*. $t_{O_{ij}(k_i,k_j)}$ represents the entire overlapping duration between activities *i* and *j*, and $t'_{O_{ij}(k_i,k_j)}$ constitutes the subset of the overlapping duration which causes rework for activity *j* due to an imperfect

information flow (i.e., when j has to begin with preliminary inputs):

$$t_{O_{ij}(k_i,k_j)} = \min\left\{T_{f_i(k_i)}, T_{f_j(k_j)}\right\} - \max\left\{T_{s_i(k_i)}, T_{s_j(k_j)}\right\} \tag{8}$$

$$t'_{O_{ij}(k_i,k_j)} = \begin{cases} \min\left\{T_{A_{ij}(k_i,k_j)}, T_{f_j(k_j)}\right\} - \max\left\{T_{s_i(k_i)}, T_{N_{ij}(k_i,k_j)}\right\} & \text{, if activity } j \text{ is not yet active} \\ 0 & \text{, if activity } j \text{ is already active} \end{cases} \tag{9}$$

3.4.5 The Effect of Overlapping on Activity Rework

To determine the amount of rework for activity j caused by overlapping with activity i (since j starts with imperfect information and may have to be partially reworked), we define an overlapping function (here, a linear one, although it need not be):

$$h_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) = \alpha_{ij} t'_{O_{ij}(k_i,k_j)} \tag{10}$$

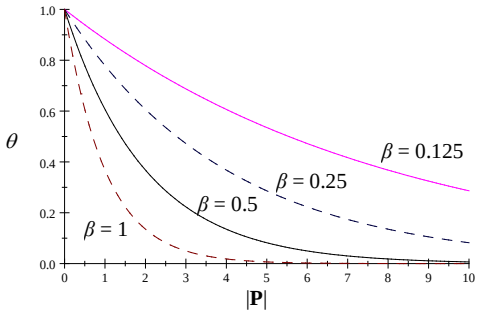
where $\alpha_{ij} > 0$ is a rework scaling factor and t' comes from equation (9). A typical value is $\alpha = 0.5$, which implies that the duration of the rework due to overlapping requires half of the duration of the overlapping. When $0 < \alpha < 1$, then the rework time is shorter than the overlapping time, whereas $\alpha > 1$ implies a greater rework duration than overlapping duration. $\alpha \geq 1$ seems unlikely to occur in practice, since this means that additional cost would be spent without any temporal benefit.

It is possible that two or more chronological predecessors of an activity j overlap with it and deliver their information to it at the same time. In this event, instead of calculating the individual rework durations between j and any $i \in \boldsymbol{P}_j$ (the set of predecessors of j), we reduce the sum of every pair-wise rework between an activity j and all its overlapping predecessors by a factor θ_j depending on the cardinality of $\boldsymbol{P}_j$, where θ_j is calculated as:

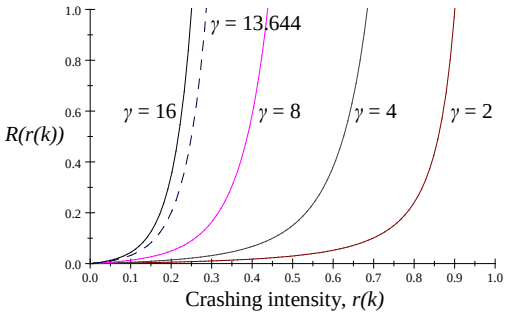
$$\theta_j = e^{-\beta_j |\boldsymbol{P}_j|} \tag{11}$$

where $\beta_j \geq 0$ is a cumulative rework factor. θ_j is plotted in Figure 2(a) for various values of β_j. A typical value is $\beta_j = 0.5$, which in the (base) case of $|\boldsymbol{P}_j| = 1$ implies that only about 60% of the cumulative overlapping duration is considered for the computation of overlapping-related rework. Increasing β_j will lead to a higher rework reduction for a certain number of predecessors—i.e., θ_j will become smaller and hence the influence of $|\boldsymbol{P}_j|$ on overall overlapping duration will decrease. In contrast, decreasing β_j implies higher θ_j values and therefore a longer overall overlapping duration. As we assume the maximal pair-wise rework between j and any $i \in \boldsymbol{P}_j$ to be the minimal amount of cumulative rework, we must furthermore define a predecessor $\hat{i} \in \boldsymbol{P}_j$ with maximal rework time $\hat{h}_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) = \max\left\{h_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) \mid \forall i \in \boldsymbol{P}_j\right\}$. Hence, the overall cumulative change in duration for a downstream activity j due to overlapping is given by:

$$\Delta t_{O_j(k_j)} = \hat{h}_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right) + \theta_i \cdot \left(\sum_{\forall p \in \boldsymbol{P}_j, p \neq \hat{i}} h_{ij}\left(t'_{O_{ij}(k_i,k_j)}\right)\right) \tag{12}$$



(a) Plot of equation (11) for four values of β



(b) Plot of equation (17) for several values of γ

Figure 2: Plots of equations (11) and (17).

We model the increase in activity cost due to rework caused by overlapping as proportional to its increase in duration:

$$\Delta c_{O_j(k_j)} = \frac{c_j(k_j)}{t_j(k_j)} \Delta t_{O_j(k_j)} \tag{13}$$

In summary, we model overlapping in terms of an overlapping function (h_{ij}), a rework scaling factor (α_{ij}), and any constraints on particular activity overlaps (governed by M_3 to M_6), from which we can determine the points in time in which information is delivered (T_D) and received (T_R). These terms are used to calculate the changes in time ($\Delta t_{O_j(k_j)}$) and cost ($\Delta c_{O_j(k_j)}$) due to overlapping for each iteration of each activity.

3.4.6 Economic Efficiency of Overlapping

To evaluate the efficiency of overlapping, we define an index ψ as a ratio of time to cost changes for the overall process:

$$\psi = \frac{\Delta t_{t_{tot}/o}}{\Delta c_{c_{tot}/o}} \tag{14}$$

where $\Delta t_{t_{tot}/o} = \frac{t_{tot/o} - t_{tot}}{t_{tot}} \cdot 100$ and $t_{tot/o}$ and t_{tot} (determined via simulation, as described in §4) are the overall process durations with and without any overlapping, respectively. The calculation of $\Delta c_{c_{tot}/o}$ is analogous. In most cases, the time changes will be negative while the cost changes are positive, giving ψ negative values, where $\psi < -1$ implies a greater time savings than cost increase. (If desired, ψ can be weighted by a constant to account for any known relationship between time and cost, such as the cost of project delay per unit of time [52].)

3.5 Activity Crashing

We define $r_i(k_i) \in [0, \hat{r}_i]$ as the *crashing intensity* of activity i in iteration k_i, where $\hat{r}_i \in [0,1)$ represents the maximum crashing intensity allowed. Crashing intensity specifies the minimum activity duration (as a percentage of its normal duration) achievable through the use of crashing (e.g., 70% of its

normal duration). Crashing the k_i^{th} iteration of activity i reduces its duration[3] by:

$$\Delta t_{C_i(k_i)} = -r_i(k_i)\left[\tilde{t}_i(k_i) + \Delta t_{O_i(k_i)}\right] \tag{15}$$

Crashing an activity increases its cost. In the k_i^{th} iteration, this cost increase, $\Delta c_{C_i(k_i)}$, depends on a function (continuous or discrete) of the crashing intensity, $R_i(r_i(k_i))$:

$$\Delta c_{C_i(k_i)} = \left(\tilde{c}_i\left(k_i\right) + \Delta c_{O_i(k_i)}\right) R_i\left(r_i\left(k_i\right)\right) \tag{16}$$

where $\tilde{c}_i\left(k_i\right)$ is the cost considering learning and rework impact (equation 1), $\Delta c_{O_i(k_i)}$ is the cost due to any overlapping (equation 13), and, in the continuous case:

$$R_i\left(r_i(k_i)\right) = \frac{\left(1 - r_i(k_i)\right)^{-\gamma_i} - 1}{100} \tag{17}$$

where $\gamma_i \geq 0$ represents the return on crashing efforts for activity i. As γ increases, the return on crashing investment increases, as shown in Figure 2(b) for five example values of γ. For example, if $\gamma = 2$, then crashing the activity by 25% will require an 80% increase in cost. Or, if we know that crashing activity i by 20% requires a 20% increase in cost, we can set $R_i(r_i(k_i)) = r_i(k_i) = 0.2$ and solve for $\gamma = 13.644$.

The costs of crashing may differ between original and reworked activities. While irrelevant in acyclic processes, the *payment mode* for crashing costs affects $\Delta c_{C_i(k_i)}$ in cyclic processes. We distinguish three payment modes:

- **Payment Mode A**: a *set-up resource cost* (e.g., setting up a facility or model) is paid once per full version (initial execution or full iteration) of the activity; it is not paid in instances of partial rework.
- **Payment Mode B**: a *one-time resource cost* (e.g., purchasing a new tool) applies only to the initial execution of the activity and adds no further costs to successive iterations: i.e., $\Delta c_{C_i(k_i)} = 0$ if $k_i > 0$.
- **Payment Mode C**: a *proportional resource cost* (e.g., staff) must be paid continuously over the activity's duration in any iteration.

We model a crashing strategy in terms of a specified crashing intensity ($r_i(k_i)$) for each activity. To compare crashing strategies, we define the index ϕ akin to equation (14):

$$\phi = \frac{\Delta t_{t_{tot}/c}}{\Delta c_{c_{tot}/c}} \tag{18}$$

As with ψ, ϕ will usually be negative, where $\phi < -1$ implies a greater time savings than cost increase.

3.6 Partial Rework of Activities

Activity overlapping may also generate *partial rework* for an activity, which we distinguish from regular rework. Partial rework handles the situation when a downstream activity j delivers its output to an upstream activity i after the latest point in time when activity i needed it: $T_{D_{ji}\left(k_j, k_i\right)} > m_{6,ji} \cdot \tilde{t}_i\left(k_i\right)$ (see

[3] Although we do not use it here, the crashing intensity also allows for the inclusion of a "Brooks factor" which is either equal to 1 or else negative. A negative Brooks factor models activities that resist crashing and are elongated by the assignment of additional resources [18].

Figure 1). Such a situation typically emerges if $T_{f_j(k_j)} < T_{f_i(k_i)}$ and $m_{1,ij}$ with $j > i$ holds, which occurs only in iterative processes, such as in the case of iterative overlapping [13, 17]. In this event, the duration $T_{D_{ij}(k_i,k_j)} - T_{N_{ij}(k_i,k_j)}$ of activity i must be reworked, because it constitutes the time span between information delivery and the lower bound for information needed (see Figure 3).

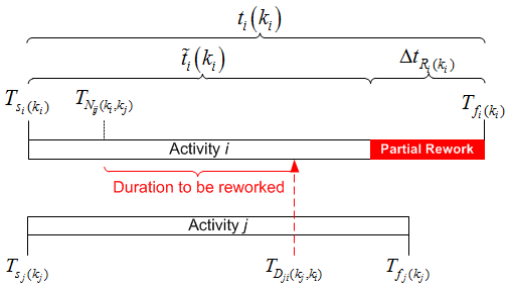


Figure 3: Illustration of singular partial rework for upstream activity i.

As a consequence of partial rework, the cost and duration of an activity i are modified by $\Delta c_{R_i(k_i)}$ and $\Delta t_{R_i(k_i)}$ as shown in equations (3)-(4). If activity i is partially reworked for the z^{th} time, caused by an activity j, then the change in duration due to partial rework is given by:

$$\Delta t_{R_i(k_i),z} = l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right) \tag{19}$$

The calculation of the cost change depends on the crashing payment mode. For modes A and B:

$$\Delta c_{R_i(k_i),z} = \frac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{S_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} \right) \tag{20}$$

and for mode C:

$$\Delta c_{R_i(k_i),z} = \frac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{S_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} + \Delta c_{C_i(k_i)} \right) \tag{21}$$

Generally, an activity i might be partially reworked $n_{R,i}$ times until it is completely reworked—i.e., until k_i changes. If $z > 1$ then we must change the temporal reference system. Once an activity has been partially reworked, its finish time must be updated. We introduce a point in time for the information input that provoked the latest partial rework, $T_{S_{R,i}(k_i)}$. Hence, the overall changes in cost and duration due to all partial reworks of activity i are given by the following equations (for crashing modes A and B):

$$\Delta t_{R_i(k_i)} = \sum_{z=1}^{n_{R,i}} \begin{cases} l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right) & \text{, if } z = 1 \\ l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{S_{R,i}(k_i)} \right) & \text{, if } z > 1 \end{cases} \tag{22}$$

$$\Delta c_{R_i(k_i)} = \sum_{z=1}^{n_{R,i}} \begin{cases} \dfrac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{S_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} \right) & \text{, if } z = 1 \\ \dfrac{l_i(k_i) m_{2,ij} \left(T_{D_{ji}(k_j,k_i)} - T_{N_{ij}(k_i,k_j)} \right)}{T_{f_i(k_i)} - T_{S_i(k_i)}} \left(\tilde{c}_i(k_i) + \Delta c_{O_i(k_i)} + \sum_{i=1}^{z-1} \Delta c_{R_i(k_i),z} \right) & \text{, if } z > 1 \end{cases} \tag{23}$$

4. Analysis Methods

This section describes the techniques by which we analyzed the rich model presented in the previous section to explore the effects of work policies on project outcomes.

4.1 Simulation Analysis

Only the most basic models of crashing and overlapping lend themselves to closed-form analytical solutions by making a number of restrictive assumptions. The complexity of a realistic model that accounts for the *crashing*, *overlapping*, and *stochastic rework* of *many distinct activities* requires a simulation-based approach. The PD literature contains several simulation models [e.g., 8, 53, 54] which mainly seek to estimate process completion time (and sometimes cost) for a single process architecture [15]. Some have compared architectures [17] and attempted architecture optimization [55]. However, as described in §2, none of these models have explored crashing, overlapping, and rework simultaneously. Doing so requires a departure from analytical models and the utilization of carefully designed simulations.

To ensure correctness in the simulation, we adopt a modified form of Browning and Eppinger's [17] well-established algorithm for a discrete-event, Monte Carlo simulation that transverses a project network. Summarized briefly, the procedure works as follows. First, using work policy rules, the algorithm determines the subset of activities eligible for work. It then adds the time and cost of working activities until reaching a time event, T_E, such as one of those shown in Figure 4. Next, it checks (probabilistically) for rework (iteration) that any completed activities may have caused, as well as any newly eligible activities. When all activities and rework are finished, a single run is complete. By conducting many runs, the simulation allows exploration of the frequency distribution (and thus the mean and other parameters) of cost and duration outcomes for any combination of process architecture and work policy. We verified that performing 2,000 runs for each test problem would yield a conservatively stable output distribution.

Although many PD process models reported in the literature do not note verification or validation efforts, we used several techniques recommended by Sargent [56] for this purpose. The model's conceptual validity hinged on acceptance of the theoretical motivations described in §2 and the basic constructs described in §3. The bulk of these have been substantiated by previous models, as has the simulation itself. We verified the computerization of the model by testing the correctness of the programming and implementation. The input data, described in the next subsection, were also scrutinized for realism against two industrial projects. We compared multiple batches of runs to ensure internal validity and consistency, along with other event, face, and internal validation techniques [56, 57]. Overall validity compares favorably with that of other published PD process models [58].

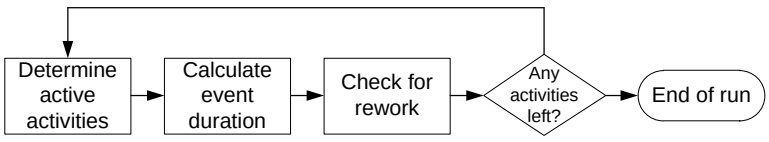


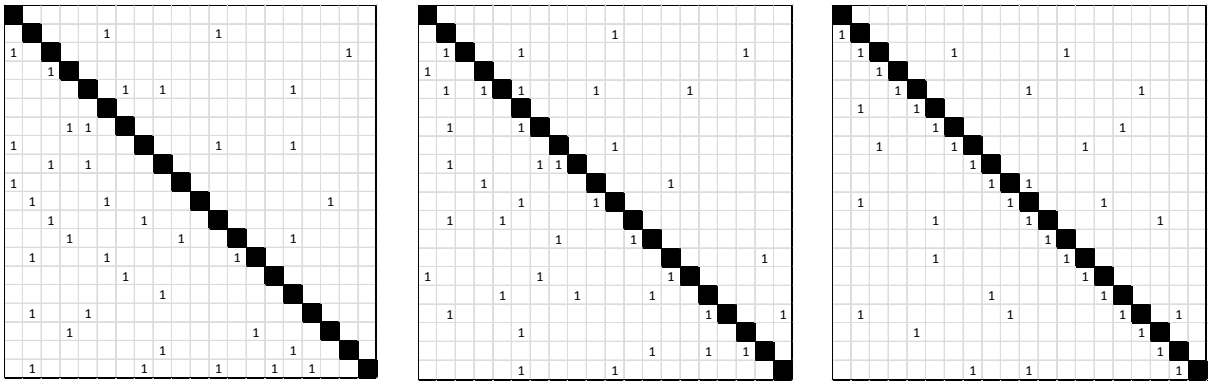
Figure 4: Event graph for a single simulation run.

4.2 Test Problems with Varied Process Architectures

We gathered data from two real projects, a drone aircraft design project and an automobile hood design project, to verify the model's behaviors [65]. We used these and other PD projects from our own experiences and the literature, as well as discussions with experts, to confirm the model's constructs and realistic ranges of input data. However, this relatively small number of data points would not enable a full

exploration of the model's characteristics and the derivation of more general insights. For this, it is appropriate to use a set of test problems designed to cover the ranges of the characteristics of interest. We developed a bank of 480,000 problem instances. Each test process contains 20 activities (n_A = 20), which is sufficient to obtain results that distinguish the effects of process architecture and work policy while maintaining a reasonable problem size.[4] Each process contains a varied number of feed-forward and feedback dependencies (n_{ff} and n_{fb}, respectively) among the activities. n_{ff} is set at four levels {30,60,90,120}, which corresponds to 16-63% of the 190 possible feed-forward relationships among 20 activities ((n_A^2 - n_A) / 2). n_{fb} is set at {0,20,40,60}. The case with zero feedbacks allows us to compare results with acyclic processes. With four settings each for n_{ff} and n_{fb}, we obtain 16 combinations of relationship densities among the activities, comprising a 4x4 experimental design.

For each of these 16 combinations, we constructed 10,000 random processes as follows. First, we randomly allocated the n_{ff} feed-forward relationships among the 20 activities 100 times. Then, for each of these 100 processes, we randomly allocated the n_{fb} feedbacks 100 times. This procedure generated 10,000 random processes for each n_{ff} and n_{fb} combination, 160,000 total. Figure 5(a) shows an example result from this procedure for the case of n_{ff} = 30 and n_{fb} = 10. In addition to the 160,000 *purely random* process architectures, we constructed two sets of *sequential* and *mixed* processes also with 160,000 processes each. Sequential processes ensure the assignment of at least one feed-forward relationship from each activity to the next (Figure 5(c)); mixed processes encourage a mix of sequential and parallel activities by connecting every second activity with its adjacent successor in $\boldsymbol{S}$ (Figure 5(b)). Because of their relative lack of constraints, the completely random processes feature the most natural concurrency among activities, followed by the mixed processes; the sequential processes exhibit the least parallelism. So, like Rivkin and Siggelkow [62], we vary the process architecture in deliberate ways, although instead of tens of DSMs we use 480,000 test processes (simulated 2,000 times each). Although we refer to only one of our three sets as "random," all are actually random except for the specific constraints pertaining to the sequential and mixed sets. Collectively, the sets allow us to explore an immense variety of process architectures. Comparisons with data from the two industry projects confirmed the realistic nature of these test problems.



(a) Purely Random (b) Mixed (c) Sequential

Figure 5: An example of M_1 for each type of test process, shown as a binary DSM with n_{ff} = 30 and n_{fb} = 10.

[4] In the project scheduling literature, project size has not been found to have a significant effect [59-61].

4.3 Test Settings

Table 3 shows the settings for the results presented in this paper, based on the rationale presented in §3. To focus these results, we fix several of the parameters at constant settings. We model learning as a simple step function where activity i requires l_i% of its original duration and cost in all iterations. Activity durations and costs are sampled from a uniform distribution in [9,11] so that each activity would have a fairly comparable duration and cost (i.e., no activities would be much shorter, longer, or more or less expensive than other activities) without being identical; this prohibits mere activity stochasticity from contributing too much variation. We set the product of rework impact and learning curve effects for each activity to 0.5, meaning that any full repetition of an activity will take only half of its original time and cost. We limit the maximum number of iterations for each activity to five. We adopt the linear overlapping function in equation (10) and the continuous crashing function in equation (18). By setting $\alpha = 0.5$, $T_D = 0.8$, and $T_R = 0.2$ for all overlaps, we let the combined cost of any two overlapped activities increase by 10% while reducing their combined duration by 10%. We conducted sensitivity analyses to confirm that the results presented in this paper are sufficiently robust with respect to all of these settings. Further results with other settings are available elsewhere, as noted below.

5. Results

Here we present several results and distill theoretical propositions [63] on the management of cyclical PD projects in terms of the effects of work policies, crashing, and overlapping on project duration and cost. In the interest of brevity, we omit some merely confirmatory results and tests of model and simulation fidelity and sensitivity; those interested in such background may find additional details in [65].

Table 3: Model settings.

Parameter	Description	Setting(s)
$\boldsymbol{M}$	Process architecture	160,000 each of random, sequential and mixed processes
n_A	Number of activities in the process	20 (constant)
n_{ff}	Number of feed-forward dependencies	30, 60, 90, or 120
n_{fb}	Number of feedback dependencies	0, 20, 40, or 60
p_{fb}	Probability of feedback for any dependency	0.5 (constant); note: $p_{fb} = m_{1,ij}\ \forall\ (i,j)$ in $\boldsymbol{M}$
$l_i(k_i)m_{2,ij}$	Product of learning and rework impact	0.5 (constant) $\forall\ (i,j)$ in $\boldsymbol{M}$ and $\forall\ k_i$
c_i	Activity cost	Random real number in [9,11] (e.g., 9.37 units)
t_i	Activity duration	Random real number in [9,11] (e.g., 10.64 units)
Max(k_i)	Iteration limit	$k_i \leq 5$
Parameters for Crashing Model		
$r_i(k_i)$	Crashing intensity	0.2 (constant)
γ	Return on crashing investment	13.644 (constant)
Parameters for Overlapping Model		
α	Rework scaling factor	0.5 (constant) $\forall\ (i,j)$ in $\boldsymbol{M}$
β	Cumulative rework factor	0.5 (constant) $\forall\ j$
M_3	Minimum information available *without* penalty	1 (constant) $\forall\ (i,j)$ in $\boldsymbol{M}$
M_4	Maximum information needed *without* penalty	0 (constant) $\forall\ (i,j)$ in $\boldsymbol{M}$
M_5	Minimum information available *with* penalty	0.8 (constant) $\forall\ (i,j)$ in $\boldsymbol{M}$
M_6	Maximum information needed *with* penalty	0.2 (constant) $\forall\ (i,j)$ in $\boldsymbol{M}$
T_D	Point in time when information is delivered	0.8 (constant) $\forall\ i$
T_R	Point in time when information is received	0.2 (constant) $\forall\ j$

5.1 Basic Time-Cost Scale-up Behaviors in Iterative Processes

A major decision for PD project managers is whether or not to perform activity iterations. Although

these tend to increase product quality, they require time and money. Therefore, a manager needs to understand the implications of iteration on project duration and cost. While prior research has found that feedbacks generally increase process duration and cost, the specific nature and scale-up of this relationship has not been explored. Moreover, a manager's choice of work policy, which governs the processing of activities, affects process duration and cost in the presence of iteration.

To dig deeper into the scale-up issue, we explored the effects of varying n_{ff} and n_{fb} on project cost and duration. Figure 6 displays results from the *random* test processes with four work policies (all but P_3, which is identical to P_2 except for overlapping and crashing and which we will address later).[5] As expected, c_{tot} and t_{tot} increase with n_{ff} and n_{fb} for all work policies. Although iterations increase the cost of all processes, the rework impact and learning curve effects absorb some of these costs, thereby preventing an exponential boost. Unlike cost, duration is rarely a mere summation of the individual activities' durations because of concurrency: some work and rework might occur off the critical path, mitigating the effect on t_{tot}. These behaviors remained as p_{fb} varied over 0.1 to 0.5. Polynomial regression analyses in Figure 6 show linear and non-linear trends in the upper and lower panels, respectively.[6] Although the results are not shown in Figure 6, the *sequential* and *mixed* processes exhibited similar scale-up behaviors.

Next, to examine the random processes with all combinations of n_{fb} and n_{ff}, we defined d_{fb} and d_{ff} as the dependency densities normalized over [0,1]. $d_{fb} = 2n_{fb} / (n^2 - n)$, and $(n^2 - n) / 2 = 190$ when $n = 20$. Polynomial regression models with three terms—d_{fb}, d_{ff}, and $d_{fb}d_{ff}$ (to test for interaction effects)—provided the best fit (adjusted R^2 values of 0.89 to 0.95 and statistical significance for all terms at 95% confidence; see equations in Table 4). d_{fb} has a greater influence on cost than d_{ff}, and, except for P_1 in the case of cost, $d_{fb}d_{ff}$ has more influence than either d_{fb} or d_{ff} alone. As d_{ff} increases, an increasing proportion of these feed-forwards are redundant arcs, so their influence is limited in the absence of iteration. The influence of $d_{fb}d_{ff}$ is greater because a single feedback loop, depending on its position in a network, can cause a cascade of rework for a large number of dependent activities, and increasing d_{ff} increases the likelihood that more activities will be involved in the cycle. Although redundant arcs can be ignored when scheduling acyclical processes, they have a significant effect in cyclical processes because they increase the number of higher-order rework paths. The evidence from our results suggests that:

Proposition 1: *The cost and duration of iterative processes scale up differently in n_{ff} and n_{fb} (d_{fb} and d_{ff}) depending on work policy, regardless of p_{fb}.*

Thus, an increasing number of dependencies among the activities in a project network increases the constraints on that network and thus its cost and duration. Both feed-forward and feedback dependencies have this effect, although the marginal effect of additional feedbacks is greater than that of feed-forwards, because it only takes one feedback loop to cause process cyclicality. Increasing both feedbacks and feed-forwards compounds the effect. Whereas design iterations are attractive to PD project managers from a quality standpoint, they require sacrifices of time and cost. However, this higher quality need not

[5] For ease of exposition, Figure 6 shows results for varying n_{ff} only where $n_{fb} = 60$ and varying n_{fb} only where $n_{ff} = 120$. The other settings produced similar results.

[6] The equations shown in Figure 6 represent the best R^2 value obtained from linear and polynomial regression models.

necessarily require an exponential increase in cost or duration. Understanding the scale-up behavior of process cost and duration as a function of the amount of information flow and other dependencies, and especially the feedbacks that cause iterations, can help managers navigate time-cost-quality tradeoffs.

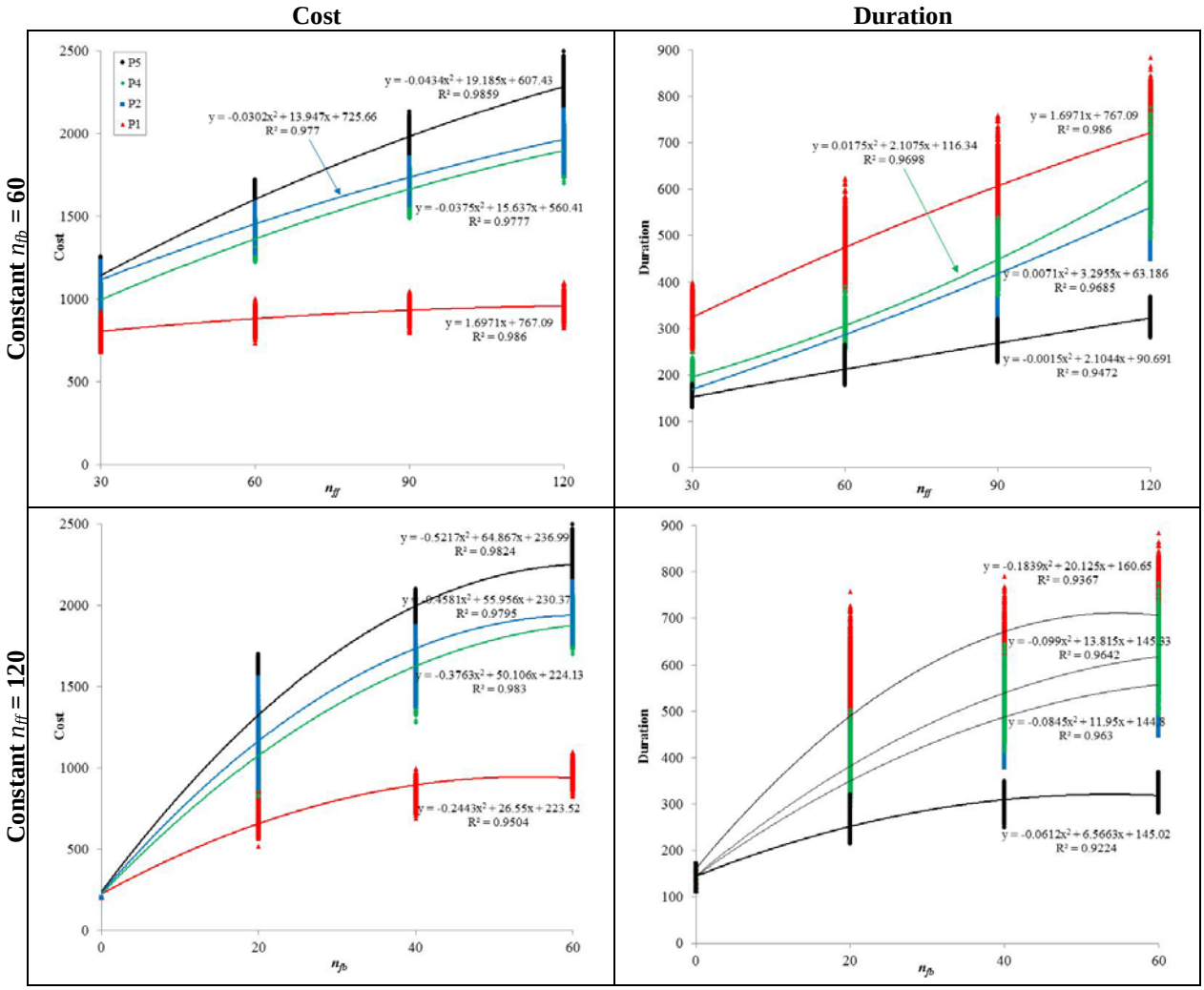


Figure 6: Comparison of cost (left) and duration (right) outcomes of random processes with four work policies and varied n_{ff} (top) and n_{fb} (bottom). (All means significantly different with 95% confidence where $n_{fb} > 0$.)

Table 4: Regression models in terms of d_{fb}, d_{ff}, and $d_{fb}d_{ff}$ for each of four work policies.

	P_1	P_2	P_4	P_5
Cost (c_{tot})	$189+1858d_{fb}+228d_{ff}+660d_{fb}d_{ff}$	$122+2226d_{fb}+481d_{ff}+5123d_{fb}d_{ff}$	$147+1784d_{fb}+380d_{ff}+5574d_{fb}d_{ff}$	$105+2062d_{fb}+554d_{ff}+6985d_{fb}d_{ff}$
Duration (t_{tot})	$33+519d_{fb}+324d_{ff}+1936d_{fb}d_{ff}$	$29+d_{fb}+238d_{ff}+2023d_{fb}d_{ff}$	$28+26d_{fb}+251d_{ff}+2234d_{fb}d_{ff}$	$36+209d_{fb}+215d_{ff}+534d_{fb}d_{ff}$

5.2 Basic Time-Cost Implications of Work Policy Decisions in Iterative Processes

What are the implications of a manager's work policy decision? Figure 7 shows results from the random processes for each of four work policies with n_{ff} = 120 and varying n_{fb}. (We obtained similar behaviors from the sequential and mixed process architectures.) The identical results when n_{fb} = 0 (in the lower-left corner of Figure 7) confirm that these work policies do not affect acyclic processes. As n_{fb} increases, project duration and cost increase and the choice of work policy becomes meaningful with respect to both mean and variation. For instance, P_1 has more variation with respect to duration, whereas

P_5 tends to be relatively robust in duration but much more variable in cost.

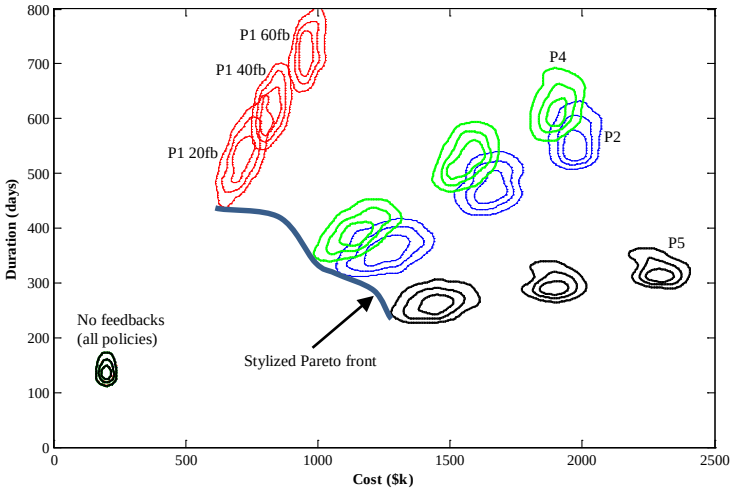


Figure 7: Project time-cost frequency distributions for four work policies (random processes, n_{ff} = 120, varying n_{fb}).

The emergence of a Pareto frontier for each level of n_{fb} (stylized in Figure 7) evinces the distinctness of the work policies. The Pareto-front's discontinuities with increasing n_{fb}, mainly with respect to cost, is a counterintuitive phenomenon, because we would expect increasing process dynamics to provoke more varied and diverse outcome spaces, leading to greater overlapping of the outcome spaces and a continuous Pareto-front. (Hybrid work policies might fill in some of the gaps between the outcomes.) Sensitivity analysis confirmed that these results were not due to the setting of Max(k_i). Rather, rework impact and learning curve effects, which set iteration cost and duration at half of their original values, curtail variation as the number of iterations increases. These results align with reports from the field on the experiences of PD project participants.

Analysis of variance (ANOVA) results (Table 5) confirm the significant impact of work policy choice. All three factors and their interactions are highly significant at $p < 0.001$ for both cost and duration. The number of feedbacks has by far the greatest influence on cost, whereas its effect on duration is still the greatest but more comparable to the effect of the number of feed-forward marks. The reason for this is that a project with more (feed-forward) precedence constraints will take longer, although its cost need not necessarily differ. For both duration and cost, work policy also makes a significant difference, although different policies dominate in each case. P_1 performs best for cost, followed by P_4, P_2, and then P_5, at all levels of n_{ff} and n_{fb}, whereas for duration this order of dominance is reversed. Increasing amounts of n_{ff} and n_{fb} magnify the policy differences.

With increasing n_{fb}, and thus additional rework, the conservative work policy P_1 becomes more dominant in the cost dimension, whereas the more aggressive policy P_5 pays for speed. When work is done more sequentially, as P_1 requires, the result is slower but cheaper. At the other extreme, P_5 allows activities to start before much of their input is finalized, causing a much higher amount of concurrency. It yields faster but much more expensive results. Between these two extremes, P_2 and P_4 feature specific rules that drive intermediate amounts of parallelism and thus cost and duration results between P_1 and P_5. P_2's allowance of non-adjacent activities in $\boldsymbol{S}$ to work concurrently increases the amount of parallelism and

rework, yielding processes that are faster than but more expensive than P_1 or P_4. Delaying the execution of rework, as prescribed by P_1, reduces the amount of parallelism and rework, making processes cheaper but slower compared to P_2 and P_5. No single "super work policy" dominates. On one hand, the time advantages are typically realized by increasing artificial concurrency, albeit at the added expense of rework (e.g., P_5 fosters iterative overlapping). On the other hand, inexpensive processes are characterized by work policies that curtail rework with a more sequential process and less artificial concurrency. Existing theory has not formalized these types of time-cost tradeoffs among work policies. We therefore propose that:

Proposition 2: *In iterative processes, the choice of work policy (which influences the degree of concurrency and rework) causes significant time-cost tradeoffs.*

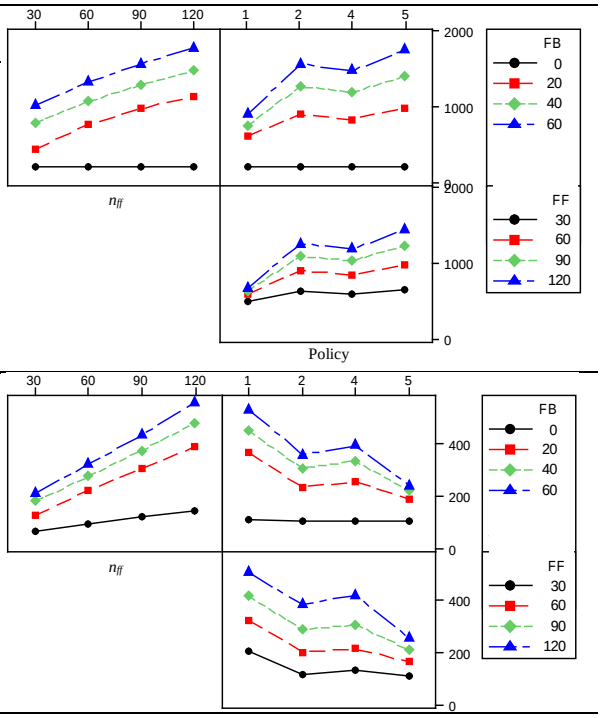
Proposition 2a: *Work policy dominance in terms of expected c_{tot} is:* $c_{P_1} < c_{P_4} < c_{P_2} < c_{P_5}$.

Proposition 2b: *Work policy dominance in terms of expected t_{tot} is:* $t_{P_5} < t_{P_2} < t_{P_4} < t_{P_1}$.

These propositions prompt further research into the implications of work policies in the management of PD projects and provide guidance on the expected results. They also provide insights into the time and cost implications of managers' decisions about when to begin activities in iterative processes. As the amount of process iteration increases, the choice of work policy increasingly matters, and managers must have a clear understanding of their project's relative priorities regarding cost and duration to navigate these tradeoffs successfully.

Table 5: ANOVA in n_{ff}, n_{fb}, and work policy.

	Factor	DoF	Sum of Squares	Mean Square Error	% of Variation Explained	P-value
Cost R^2 = 99.24%	n_{ff}	3	6519012871	2173004290	13%	0.000
	n_{fb}	3	33302714483	11100904828	65%	0.000
	Policy	3	4977361982	1659120661	10%	0.000
	n_{ff}*n_{fb}	9	2194424103	243824900	4%	0.000
	n_{ff}*Policy	9	1198070853	133118984	2%	0.000
	n_{fb}*Policy	9	2320380180	257820020	5%	0.000
	n_{ff}*n_{fb}*Policy	27	453707043	16803965	1%	0.000
	Error	159936	392719367	2455	1%	
	Total	159999	51358390882		100%	
Duration R^2 = 97.82%	n_{ff}	3	1362721402	454240467	31%	0.000
	n_{fb}	3	1731566517	577188839	39%	0.000
	Policy	3	642499333	214166444	14%	0.000
	n_{ff}*n_{fb}	9	229765277	25529475	5%	0.000
	n_{ff}*Policy	9	91740230	10193359	2%	0.000
	n_{fb}*Policy	9	233425548	25936172	5%	0.000
	n_{ff}*n_{fb}*Policy	27	44945397	1664644	1%	0.000
	Error	159936	96676427	604	2%	
	Total	159999	4433340132		100%	



To compare work policies in terms of a single time-cost objective, we define the index λ:

$$\lambda = \left(\frac{c_{tot}}{\text{Min}(c_{tot})}\right)^{\omega_c} \left(\frac{t_{tot}}{\text{Min}(t_{tot})}\right)^{\omega_t} \tag{26}$$

where the minima refer to the best average values obtained with any combination of n_{ff} and n_{fb} and the ω

terms are weighting factors such that $\omega_c + \omega_t = 1$. Table 6 displays the average results for random processes using $\omega_c = \omega_t = 0.5$ (weighting cost and duration equally) and the minimum values of λ normalized to 1.00. The best (lowest) average values for each combination of n_{ff} and n_{fb} are shaded green; the worst are shaded red. When duration and cost are weighted equally, P_5 dominates for cyclical processes while P_1, P_2, and P_4 each perform worst in different situations. These results occur because the time savings generated by P_5 outweigh its cost increases to a greater extent than with the other work policies. However, if cost had a greater weighting than time then P_5 would not always be best. Also note that, once again, when we compare the scale-up of λ with respect to n_{ff} in an acyclic process ($n_{fb} = 0$) to one with $n_{fb} = 20$, the behavior changes. Even a small $n_{fb} > 0$ alters the scale-up behavior of n_{ff} by introducing the dynamics of feedback.

Table 6: Normalized values of λ for varying n_{ff} and n_{fb} for work policies P_1, P_2, P_4, and P_5 (left to right).

n_{fb} \ n_{ff}	30	60	90	120
0	1.00	1.25	1.41	1.56
20	2.44	4.04	5.00	5.77
40	3.72	5.24	6.82	7.90
60	4.53	6.27	7.64	8.58

n_{fb} \ n_{ff}	30	60	90	120
0	1.00	1.25	1.41	1.56
20	2.06	3.95	5.42	6.234
40	3.21	5.32	6.59	8.21
60	3.91	6.20	7.86	9.74

n_{fb} \ n_{ff}	30	60	90	120
0	1.00	1.25	1.41	1.56
20	2.13	3.73	4.97	6.229
40	3.34	4.96	6.62	8.39
60	4.09	6.04	7.95	10.07

n_{fb} \ n_{ff}	30	60	90	120
0	1.00	1.25	1.41	1.56
20	2.05	3.61	4.69	5.70
40	3.19	4.60	5.83	6.95
60	3.87	5.42	6.74	7.96

5.3 Effects of Crashing in Iterative Processes

Crashing an activity has a predictable cost increase and time savings in an acyclic process. But how efficient is crashing in the presence of activity iteration? Should activities be expedited when they may have to be reworked anyway? Do process architecture and work policy make a difference? This subsection presents results and develops theory regarding these questions.

5.3.1 Effects of Activity Concurrency, Increasing Iteration, and Payment Mode

Process architecture and iteration can have a significant effect on the economic efficiency of crashing, ϕ (equation 18). While we expect $\phi = -1$ (a unit of time savings for each unit of cost increase) for acyclic processes, the results change for iterative processes. Averaging ϕ across all 160,000 of each type of test process (each simulated 2,000 times) and varying $p_{fb} \in \{0.1, 0.3, 0.5\}$ yields the graphs shown in Figure 8 for work policy P_{3a} (i.e., P_3 without overlapping). Crashing mode C (see §3.5.2) maintains $\phi = -1$ in all cases, so we do not show these results, but a separate graph is shown for the set-up (A) and one-time (B) modes. ϕ improves (decreases) with p_{fb} (i.e., as processes become more iterative) in both of these modes because the additional cost of crashing is paid only once or once per full iteration, regardless of the activity's duration. ϕ improves linearly with p_{fb} in mode A (when crashing costs must be paid once per full iteration) and logarithmically in mode B (when crashing costs occur only once per activity, because here the increasing number of iterations are sped up without additional crashing cost). Overall, crashing mode has a major effect of efficiency:

Proposition 3: *In increasingly iterative processes (p_{fb}), the economic efficiency of crashing (ϕ) improves (decreases) for payment modes A and B but does not change for C.*

Proposition 3a: *If the cost of crashing must be paid once per activity iteration, then* $\phi_A \propto \boldsymbol{O}(p_{fb})$.

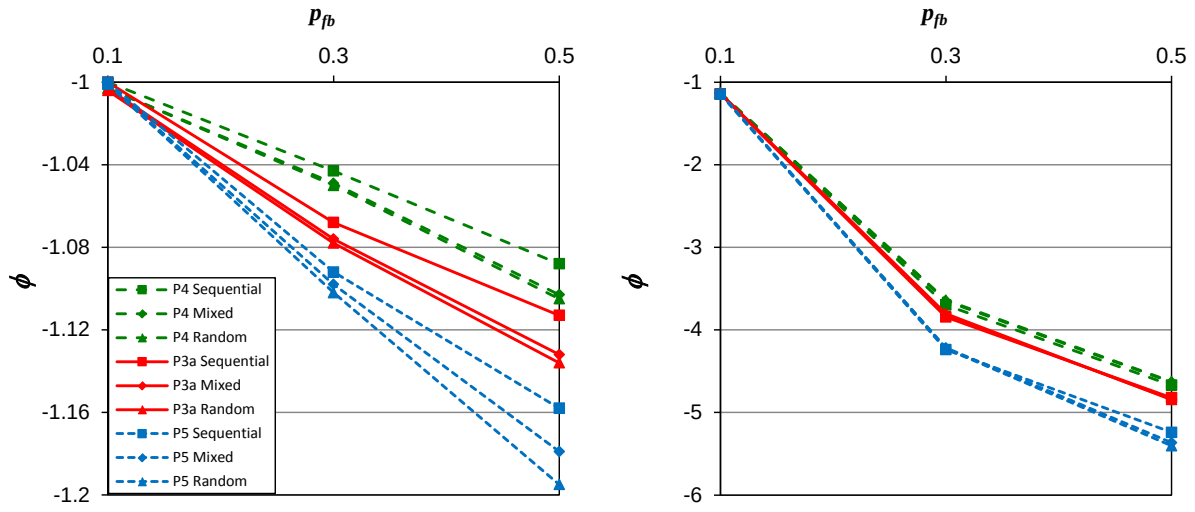
Proposition 3b: *If the cost of crashing must be paid only once per activity, then* $\phi_B \propto \boldsymbol{O}(\log(p_{fb}))$.

Proposition 3c: *If the cost of crashing must be paid proportionally to activity duration (regular work or rework), then* $\phi_C = -1$.

Proposition 4: *With crashing payment modes A and B, the average rate of improvement in* ϕ *will decrease with increasing iteration* (p_{fb}).

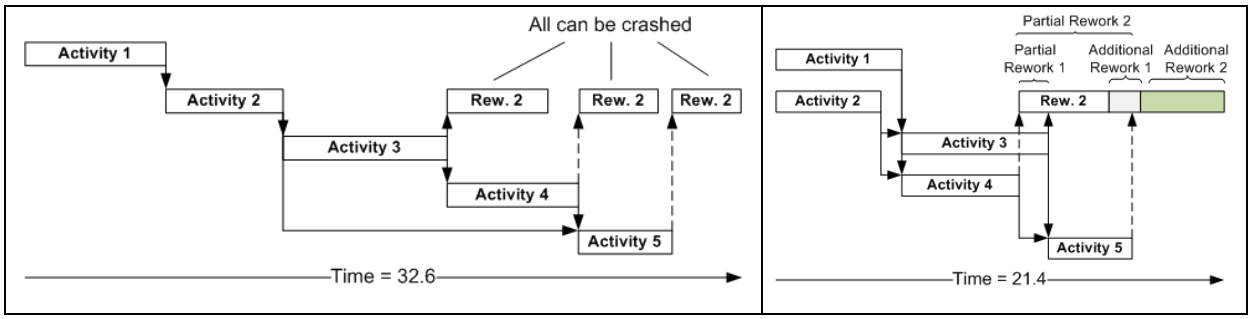
In Figure 8(b) the three traces are almost identical for P_{3a}, meaning that process architecture does not have a strong effect with crashing mode B (regardless of p_{fb}). However, in Figure 8(a) crashing becomes more favorable with increasing activity concurrency (*random* test processes performed best on average, followed by *mixed* and *sequential*) when it is paid once per iteration. This is a surprising finding, because we would expect crashing to have the greatest effect on more sequential processes (with more activities on the critical path). However, this intuition is misleading in cyclical processes. Because of less natural concurrency among activities, the sequential processes contain a much larger number of activities that must be completely reworked than in the other two types, causing its higher cost and ϕ. That is, when an activity's output becomes an input for another activity, it is more likely that this latter activity is not executed simultaneously and therefore will have to be completely reworked (and crashed) again (partial reworks do not matter in mode A). Meanwhile, in more concurrent processes, it is more likely that this latter activity would be executed in parallel, with only partial rework, which is cheaper than completely iterating and crashing it again. Figure 9 illustrates this situation by comparing a simple sequential and mixed process. In this example the sequential process structure produces three full iterations, whereas the mixed process structure provokes only one full iteration and two partial reworks (which do not have to pay a "re-crashing" cost). This evidence further suggests that:

Proposition 5: *In iterative processes with crashing payment mode A,* ϕ *improves (decreases) with increasing process concurrency:* $\phi_{Random} < \phi_{Mixed} < \phi_{Sequential}$.



(a) Crashing paid once per full version of activity (mode A) (b) Crashing paid only once per activity (mode B)

Figure 8: Economic efficiencies of crashing with respect to p_{fb} for work policies P_3-P_5 in three types of iterative process architectures (averaged across all n_{ff} and $n_{fb} > 0$ levels; note different scales of *y*-axes).



(a) Sequential process type (b) Mixed process type

Figure 9: Comparison of activity concurrency examples.

Therefore, PD project managers should be willing to crash activities even when their process is highly iterative, regardless of process architecture. Furthermore, if managers are aware of the process architecture, even if only in general terms of the relative amounts of concurrency and iteration, then they should be increasingly willing to choose crashing as activity concurrency or iteration increases. Again, further results with other crashing modes are available in [65].

5.3.2 Effects of Work Policy

Figure 8 also shows the behavior of ϕ with respect to P_4 and P_5 (the other two work policies that allow crashing). For payment modes A and B, P_5 performed best and P_4 worst. Hence, in the context of crashing with payment modes A or B, P_5 is superior to P_3 and P_4: despite its steeper increase in cost (see Figure 6, upper-left), it more than compensates for this with an even steeper decrease in duration, giving P_5 the best (lowest) values of ϕ. Thus:

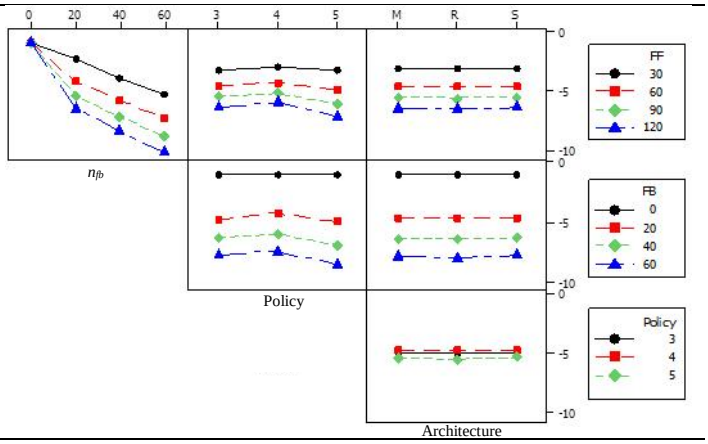
Proposition 6: *In iterative processes with crashing payment modes A and B (where crashing costs are* not *paid proportionally to activity duration), work policy influences ϕ such that* $\phi_{P_5} < \phi_{P_3} < \phi_{P_4}$ *(lower is better).*

This result implies that *managers should use the aggressive work policy P_5 to receive the greatest benefit from crashing in increasingly iterative processes.* Since P_5 fosters artificial concurrency, Proposition 6 meshes with Proposition 5, which states that crashing becomes more efficient as activity concurrency increases (albeit with decreasing returns, according to Proposition 4).

ANOVA (Table 7) provides further insight into the relative significance of the factors with respect to ϕ (in crashing mode B only, for 400 test problems per setting, with $p_{fb} = 0.5$). The vast majority of the variation in this case is explained by n_{fb}, n_{ff}, and their interaction. As shown in Figure 8(b), process architecture makes relatively little difference (although it and other factors are still significant with $p < 0.001$). In particular, P3 differs dramatically with changes in n_{fb}. Thus, for highly iterative processes, these results support Propositions 5 and 6.

Table 7: ANOVA in ϕ in crashing mode B with p_{fb} = 0.5 (R^2 = 93.17%).

Factor	DoF	Sum of Squares	Mean Square Error	% of Variation Explained	P-value
n_{ff}	3	89875.9	29958.6	16%	0.000
n_{fb}	3	384304.6	128101.5	70%	0.000
Policy	2	4573.7	2286.8	1%	0.000
Arch	2	29.7	14.9	0%	0.000
n_{ff}*n_{fb}	9	30638.9	3404.3	6%	0.000
n_{ff}*Policy	6	1204.2	200.7	0%	0.000
n_{ff}*Arch	6	62.2	10.4	0%	0.000
n_{fb}*Policy	6	1936	322.7	0%	0.000
n_{fb}*Arch	6	137.4	22.9	0%	0.000
Policy*Arch	4	65	16.2	0%	0.000
n_{ff}*n_{fb}*Policy	18	674.4	37.5	0%	0.000
n_{ff}*n_{fb}*Arch	18	41	2.3	0%	0.000
Error	57516	37659	0.7	7%	
Total	57599	551202		100%	



5.4 Effects of Overlapping in Iterative Processes

In principle, overlapping and crashing affect process cost and duration similarly in that both are supposed to save time by spending money. Nevertheless, substantial differences exist. First, the costs of overlapping only accrue to sequential or coupled activities; independent activities may be done concurrently without any cost penalty. Hence, we would expect process architecture to play a more pivotal role in overlapping than in crashing. Second, overlapping requires the consideration of at least two activities, one or more upstream and one or more downstream. The potential expenses for overlapping (i.e., rework cost) usually affect the downstream activities, whereas the cost and duration of upstream activities remain unchanged. In contrast, crashing focuses on an individual activity (although the location of that activity on the critical path makes a difference in crashing efficiency).

5.4.1 Effects of Process Architecture and Increasing Iteration

The middle three traces in Figure 10 summarize the average outcomes for the basic overlapping strategy, work policy P_{3b} (i.e., P_3 without allowing for crashing). Each point plotted in Figure 10 represents the average of 160,000 test processes (each simulated 2,000 times). As the probability of feedback increases, P_{3b} becomes more economical, and ψ (averaged across all n_{ff} and n_{fb} combinations and process architecture types) decreases (improves) logarithmically.

Proposition 7: *The economic efficiency of overlapping improves (decreases) in increasingly iterative processes*: $\psi \propto \boldsymbol{O}(log(p_{fb}))$.

Figure 10 also indicates the effects of process architecture on ψ. Overlapping is more economical for random (more concurrent) processes than for more sequential ones. While they contain less natural parallelism, sequential processes exhibit greater amounts of rework and thus have more overlapping events, which is the reason for their lower overall time savings compared with mixed or random processes. When p_{fb} = 0.1, the overall cost increase of the process is governed mainly by the rework cost of overlapping events, not by the number of partial reworks. This makes sequential processes more expensive than the other two process types, which have fewer new overlaps with reworking activities. As p_{fb} increases, the partial reworks grow exponentially thus increase the cost due to overlapping. Since sequential processes exhibited the smallest increase in partial reworks, the difference between the cost savings of the varied

process architectures declines with increasing p_{fb}, and the average values of ψ for sequential processes approach those for random and mixed processes. Hence:

Proposition 8: *In iterative processes, process architecture affects the economic efficiency of overlapping such that:* $\psi_{Random} < \psi_{Mixed} < \psi_{Sequential}$ *(where lower* ψ *is better).*

As with crashing, overlapping becomes an increasingly favorable lever for managers in the context of iterative processes, especially as the amount of natural concurrency increases. However, these returns decrease for $p_{fb} > 0.3$.

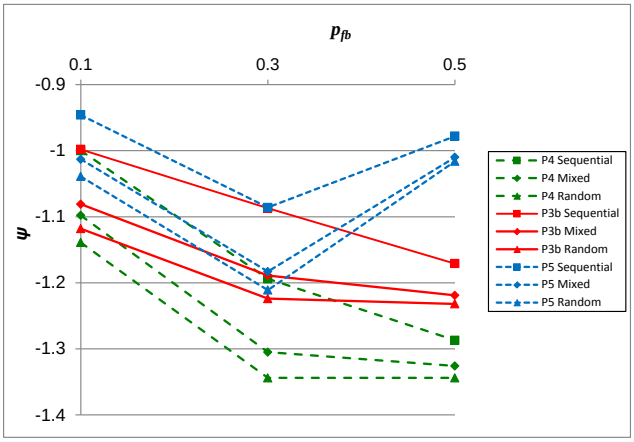


Figure 10: Economic efficiencies (ψ) of overlapping with respect to p_{fb} in three types of iterative process architectures with varying n_{ff} and n_{fb}.

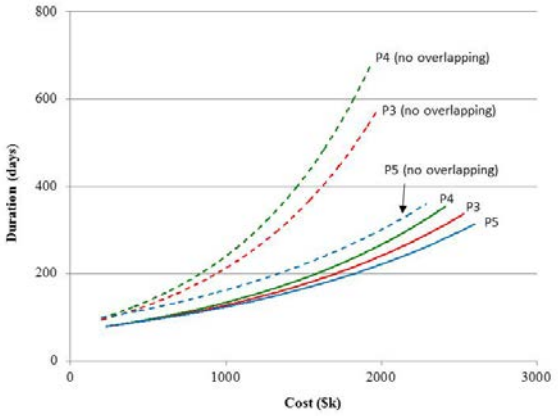


Figure 11: Time-cost trend curves, with and without overlapping, for the 160,000 random processes with varying n_{ff} and n_{fb} ($p_{fb} = 0.5$).

5.4.2 Effects of Work Policy

We compared the results for P_{3b} with those for P_4 and P_5. These are also shown in Figure 10, clearly demonstrating that work policy has a significant impact on the efficiency of overlapping. Unlike with crashing, we found that P_4 is most favorable for overlapping, while P_5 turned out to be the least economical, especially as p_{fb} increases. Figure 11 confirms this result with the trend curves for the random test processes, with and without overlapping ($p_{fb} = 0.5$ for all n_{ff} and n_{fb} combinations). Regarding P_5, it is also interesting to note that ψ does not continue to improve with increasing p_{fb} but actually worsens for $p_{fb} > 0.3$. We explain these two observations with the fact that, out of these three work policies, P_4 most fosters the occurrence of sequential information flows. As sequentially linked activities are overlapped, overlapping can have a greater impact on processes applying P_4 instead of P_5 or P_{3b}, which generate a more parallel information flow between reworked activities. This evidence leads to the following proposition:

Proposition 9: *Work policy influences* ψ *such that:* $\psi_{P_4} < \psi_{P_3} < \psi_{P_5}$ *(where lower* ψ *is better).*

This result implies that managers should use P_4 to receive the greatest benefit from overlapping in increasingly iterative processes. The most aggressive policy, P_5, can actually be counter-productive, with $\psi > -1$ for sequential processes for $p_{fb} = 0.1$ and 0.5 (for all n_{ff} and n_{fb} combinations).

ANOVA (Table 8) provides further insight into the relative significance of the factors with respect to ψ (with $p_{fb} = 0.5$). Although n_{ff} is more influential and the error term much larger than with ϕ, fully half

of the variance is explained by work policy, n_{fb}, and their interaction. Architecture has a relatively small (albeit statistically significant) effect overall, although it is much more influential at lower levels of n_{ff} (where the results support Proposition 8). (Increasing n_{ff} diminishes the differences between architectures.) The interaction between n_{ff} and n_{fb} is also interesting in that just a small amount of n_{fb} has a much larger impact when n_{ff} is small, whereas, when both are large, their compounding effects are more proportional. However, while a manager may have limited ability to influence the physics of the project network (n_{ff}, n_{fb}, and architecture), he is free to set work policy, which actually has the most significant influence. The dominance of P_4 increases in n_{ff} and n_{fb}, and P_5 actually worsens with increasing n_{fb} (fully supporting Proposition 9).

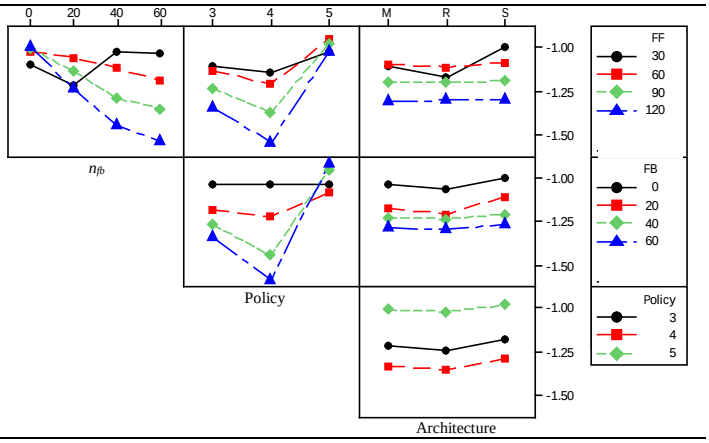
How does overlapping compare with crashing? Comparing the results shown in Figures 8 and 10 (and noting the scales of their respective *y*-axes) and Tables 7 and 8, we see that this depends on the crashing mode employed. Our study finds the relative effects of these two managerial levers to be as follows:

Proposition 10: *In increasingly iterative processes, crashing mode B is more economical than overlapping, which is more economical than crashing mode A:* $\phi_B < \psi < \phi_A$ *(where lower* ψ *and* ϕ *are better).*

Thus, all else being equal, as the probability of having to rework activities increases, managers will get highest returns where they can make one-time investments in technologies to shorten activities. Where this is not possible, and shortening an activity requires a repeated investment in each iteration, then a project is more likely to benefit from an emphasis on overlapping. Although these results depend on specific (albeit reasonable) assumptions about crashing and overlapping intensities, our sensitivity analyses confirmed that these results were not too sensitive to the values we used. Nevertheless, further research is needed to clarify the strength of the proposed relationships.

Table 8: ANOVA in ψ with $p_{fb} = 0.5$ ($R^2 = 82.85\%$).

Factor	DoF	Sum of Squares	Mean Square Error	% of Variation Explained	P-value
n_{ff}	3	2673.46	891.15	10%	0.000
n_{fb}	3	3045.34	1015.11	11%	0.000
Policy	2	6394.69	3197.35	23%	0.000
Arch	2	187.81	93.91	1%	0.000
n_{ff}*n_{fb}	9	3370.62	374.51	12%	0.000
n_{ff}*Policy	6	1418.37	236.39	5%	0.000
n_{ff}*Arch	6	326.25	54.37	1%	0.000
n_{fb}*Policy	6	4296.61	716.1	16%	0.000
n_{fb}*Arch	6	56.07	9.34	0%	0.000
Policy*Arch	4	8.69	2.17	0%	0.000
n_{ff}*n_{fb}*Policy	18	880.44	48.91	3%	0.000
n_{ff}*n_{fb}*Arch	18	190.72	10.6	1%	0.000
Error	359916	4728.64	0.01	17%	
Total	359999	27577.71		100%	



6. Discussion and Conclusion

Managerial choices about when to start work and rework (work policies) have a significant effect on the time and cost of PD projects, and these choices are easy to get wrong in situations with opportunities for crashing, overlapping, and iteration. Project management theory on managing *iterative* PD projects

does not yet address the effects of work policies and process architectures in conjunction with crashing and overlapping, even though these managerial choices drastically affect a project's prospects. This paper begins to close this important gap with a new model and several initial results that change our thinking about managing PD projects. Although exceptions have been noted [e.g., 8, 9], the quality of a product design tends to increase with additional iterations, as issues and changes get resolved and the efficacy of the design path is further confirmed [36, 64]. However, we find that this benefit does not necessarily require an exponential or even a linear increase in cost or duration; it can be achieved at a more moderate rate of increase with appropriate policy choices. In addition, managers should consider the number of feed-forward *and* feedback dependencies in a process, because the product $n_{fb}n_{ff}$ can drive time-cost scale-up behavior by making redundant, feed-forward arcs matter as potential paths for rework propagation.

The salient finding from this study is that *the choice of work policy causes time-cost tradeoffs in iterative processes*. Although this basic point is quite intuitive, the implications of varied work policies have not been much discussed in the literature on iterative PD processes (and the work policies we study do not apply to acyclical processes). Furthermore, to contribute to the project management literature, we provide insight on the implications of five particular policies: P_1, a fairly conservative policy with respect to rework that maintains a relatively sequential execution of activities by preferring to wait for any delinquent inputs before starting an activity; P_2, a policy allowing slightly more concurrency than P_1; P_3, a policy like P_2 that allows crashing and overlapping; P_4, which is identical to P_3 except that it defers rework; and P_5, an aggressive policy of greater concurrency of work, charging ahead, despite delinquent inputs, by making assumptions that may have to be corrected later via rework. We find that increasing the amount of iteration amplifies the time-cost differences between work policies: the Pareto-front expands as n_{fb} increases. Our results also illuminate differences in cost and duration volatilities among work policies. For instance, whereas P_1 is more volatile with respect to duration, P_5 tends to be relatively robust concerning duration but much more volatile with respect to cost. Although a particular work policy may be superior for particular process architectures and parameters, there is no dominant "super work policy" for all situations (Table 9). Schedule-related advantages of a work policy are typically realized through the promotion of increased concurrency within the process, albeit at the cost of additional rework (e.g., P_5 fosters iterative overlapping but generates more costly processes). Conversely, less expensive processes are characterized by work policies that limit the occurrence of rework through more sequential execution of activities. Hence, a "super work policy" would have to foster concurrency without increasing cost. Such a scenario might be possible for a process with low rework impact and high learning rates, but not as a result of work policy alone. Elsewhere, we confirm these results with analysis of data from two industrial projects [65].

Table 9: Relative performance of work policies in terms of four metrics ("1st" means lowest/best).

	P_1	**P_2**	**P_3**	**P_4**	**P_5**
Cost (c_{tot})	1st	3rd	-	2nd	4th
Duration (t_{tot})	4th	2nd	-	3rd	1st
Crashing Efficiency (ϕ)	-	-	2nd	3rd	1st
Overlapping Efficiency (ψ)	-	-	2nd	1st	3rd

We also provide significant results regarding crashing and overlapping, which have not yet been addressed by project management theory in iterative projects with more than two activities. Table 10 summarizes these findings. Regarding crashing, its effects in iterative processes depend on the payment mode, and its economic efficiency improves with increasing iteration in payment modes A and B (but does not change in payment mode C). This guidance is robust across a variety of process architectures, although it becomes even stronger with increasing activity concurrency—a surprising finding given the conventional wisdom on crashing in acyclic processes, which expects crashing to be more efficacious for more sequential processes. In other words, whereas crashing is more beneficial in sequential architectures for acyclical processes, it becomes more efficacious in concurrent process architectures for cyclical processes. The iterative nature of PD projects changes the game. Therefore, a PD manager could get into trouble following the conventional project management advice on crashing. It does make sense to crash activities even when it is likely that they will have to be redone. Aggressive work policy P_5 provides the greatest benefits from crashing.

Table 10: Summary of main findings regarding crashing and overlapping.

	Crashing (in modes A and B)	**Overlapping**
Amount of Iteration	Becomes more economically efficient as the amount of iteration increases (linearly for mode A, logarithmically for mode B)	Becomes more economically efficient (logarithmically) as the amount of iteration increases
Process Architecture	Becomes more economically efficient with increasing process concurrency such that Random ≻ Mixed ≻ Sequential	Becomes more economically efficient with increasing process concurrency such that Random ≻ Mixed ≻ Sequential
Work Policy	Becomes more economically efficient with $P_5 \succ P_3 \succ P_4$	Becomes more economically efficient with $P_4 \succ P_3 \succ P_5$

Although our results emerged from a rich model and simulation of many process architectures, implementation of these findings does not require such an effort from managers. With even a general knowledge of the dominant payment modes, the approximate process architecture, and the expected amount of iteration, a manger now has a heuristic to ascertain whether or not to emphasize crashing—*even without a formal activity network model of his or her project.*

The economic efficiency of overlapping also improves with increasing amounts of iteration. Moreover, on average, managers can expect even greater benefits from overlapping as the natural concurrency in the process architecture increases. That is, projects with more independently parallel activities should also increase the overlapping of their sequential activities. Managers should use P_4 to gain the greatest benefits from overlapping—i.e., they should be willing to avoid some rework by waiting on more predecessors to finish. Finally, a comparison of our results on crashing and overlapping in iterative process environments suggests that crashing (in mode B) is a more economical managerial lever than overlapping. This interesting result should prompt further research into crashing in the context of iterative PD projects; research in this context to date has focused on overlapping. All of these findings provide additional heuristics for practicing managers of PD projects.

The model presented in this paper is rich enough to enable many further studies, including an investigation of more specific characteristics of process architectures (besides the random, mixed, and sequential process architectures we study) and the question of which specific activities to crash and overlap

to develop a project's time-cost tradeoff curve. Although the propositions resulting from this study serve to extend project management theory, they deserve further empirical testing in various contexts. Other important research questions include the following: What are the combined effects of crashing and overlapping? What is the optimal amount of each and of both together? Which work policy works best for both at once? What additional, hybrid work policies could be identified? Should a manager switch work policies at some point during a project?

7. References

[1] Krishnan, V. and K.T. Ulrich, "Product Development Decisions: A Review of the Literature," *Management Science*, vol. 47, pp. 1-21, 2001.

[2] Ulrich, K.T. and S.D. Eppinger, *Product Design and Development*, 4th ed. New York: McGraw-Hill, 2008.

[3] Herroelen, W.S., "Project Scheduling - Theory and Practice," *Production & Operations Management*, vol. 14, pp. 413-432, 2005.

[4] Kline, S.J., "Innovation is not a Linear Process," *Research Management*, vol. 28, pp. 36-45, 1985.

[5] Smith, R.P. and S.D. Eppinger, "Identifying Controlling Features of Engineering Design Iteration," *Management Science*, vol. 43, pp. 276-293, 1997.

[6] Braha, D. and Y. Bar-Yam, "The Statistical Mechanics of Complex Product Development: Empirical and Analytical Results," *Management Science*, vol. 53, pp. 1127-1145, 2007.

[7] Braha, D. and O. Maimon, "The Design Process: Properties, Paradigms, and Structure," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 27, pp. 146-166, 1997.

[8] Mihm, J., C. Loch, and A. Huchzermeier, "Problem-Solving Oscillations in Complex Engineering Projects," *Management Science*, vol. 49, pp. 733-750, 2003.

[9] Yassine, A.A., N. Joglekar, D. Braha, S. Eppinger, and D. Whitney, "Information Hiding in Product Development: The Design Churn Effect," *Research in Engineering Design*, vol. 14, pp. 145-161, 2003.

[10] Meredith, J.R. and S.J. Mantel, *Project Management*, 8th ed. New York: Wiley, 2012.

[11] PMI, *A Guide to the Project Management Body of Knowledge*, 5th ed. Newtown Square, PA: Project Management Institute, 2013.

[12] Joglekar, N.R., A.A. Yassine, S.D. Eppinger, and D.E. Whitney, "Performance of Coupled Product Development Activities with a Deadline," *Management Science*, vol. 47, pp. 1605-1620, 2001.

[13] Krishnan, V., S.D. Eppinger, and D.E. Whitney, "A Model-Based Framework to Overlap Product Development Activities," *Management Science*, vol. 43, pp. 437-451, 1997.

[14] Loch, C.H. and C. Terwiesch, "Rush and Be Wrong or Wait and Be Late? A Model of Information in Collaborative Processes," *Production & Operations Management*, vol. 14, pp. 331-343, 2005.

[15] Browning, T.R. and R.V. Ramasesh, "A Survey of Activity Network-Based Process Models for Managing Product Development Projects," *Production & Operations Management*, vol. 16, pp. 217-240, 2007.

[16] Roemer, T.A. and R. Ahmadi, "Concurrent Crashing and Overlapping in Product Development," *Operations Research*, vol. 52, pp. 606-622, 2004.

[17] Browning, T.R. and S.D. Eppinger, "Modeling Impacts of Process Architecture on Cost and Schedule Risk in Product Development," *IEEE Transactions on Engineering Management*, vol. 49, pp. 428-442, Nov. 2002.

[18] Brooks, F.P., Jr., *The Mythical Man-Month*, Anniversary ed. Reading, MA: Addison-Wesley, 1995.

[19] Doerner, K.F., W.J. Gutjahr, R.F. Hartl, C. Strauss, and C. Stummer, "Nature-inspired Metaheuristics for Multiobjective Activity Crashing," *Omega*, vol. 36, pp. 1019-1037, 2008.

[20] Kelley, J., James E. and M.R. Walker, "Critical-Path Planning and Scheduling," Eastern Joint IRE-AIEE-ACM Computer Conference, Boston, MA, Dec 1-3, 1959.

[21] Kelley, J., James E., "Critical-Path Planning and Scheduling: Mathematical Basis," *Operations Research*, vol. 9, pp. 296-320, 1961.

[22] Fulkerson, D.R., "A Network Flow Computation for Project Cost Curves," *Management Science*, vol. 7, pp. 167-178, 1961.

[23] Berman, E.B., "Resource Allocation in a PERT Network under Continuous Activity Time-Cost Functions," *Management Science*, vol. 10, pp. 734-745, 1964.

[24] Falk, J.E. and J.L. Horowitz, "Critical Path Problems with Concave Cost-Time Curves," *Management Science*, vol. 19, pp. 446-455, 1972.

[25] Kanda, A. and U.R.K. Rao, "A Network Flow Procedure for Project Crashing with Penalty Nodes," *European Journal of Operational Research*, vol. 16, pp. 174-182, 1984.

[26] Mitchell, G. and T. Klastorin, "An Effective Methodology for the Stochastic Project Compression Problem," *IIE Transactions*, vol. 39, pp. 957-969, 2007.

[27] Goh, J. and N.G. Hall, "Total Cost Control in Project Management via Satisficing," *Management Science*, vol. 59, pp. 1354-1372, 2013.

[28] Funk, J.L., "Concurrent Engineering and the Underlying Structure of the Design Problem," *IEEE Transactions on Engineering Management*, vol. 44, pp. 305-315, 1997.
[29] Smith, R.P., "The Historical Roots of Concurrent Engineering Fundamentals," *IEEE Transactions on Engineering Management*, vol. 44, pp. 67-78, 1997.
[30] Terwiesch, C., C.H. Loch, and A.D. Meyer, "Exchanging Preliminary Information in Concurrent Engineering: Alternative Coordination Strategies," *Organization Science*, vol. 13, pp. 402-419, 2002.
[31] Lin, J., K.H. Chai, Y.S. Wong, and A.C. Brombacher, "A Dynamic Model for Managing Overlapped Iterative Product Development," *European Journal of Operational Research*, vol. 185, pp. 378-392, 2008.
[32] Thomke, S.H., "Managing Experimentation in the Design of New Products," *Management Science*, vol. 44, pp. 743-762, 1998.
[33] Eppinger, S.D., "Innovation at the Speed of Information," *Harvard Business Review*, vol. 79, pp. 149-158, Jan. 2001.
[34] Simon, H.A., *The Sciences of the Artificial*, 3rd ed. Cambridge, MA: MIT Press, 1996.
[35] Jarratt, T.A.W., C.M. Eckert, N.H.M. Caldwell, and P.J. Clarkson, "Engineering Change: An Overview and Perspective on the Literature," *Research in Engineering Design*, vol. 22, pp. 103-124, 2011.
[36] Lévárdy, V. and T.R. Browning, "An Adaptive Process Model to Support Product Development Project Management," *IEEE Transactions on Engineering Management*, vol. 56, pp. 600-620, 2009.
[37] Cooper, K.G., "The Rework Cycle: Benchmarks for the Project Manager," *Project Management Journal*, vol. 24, pp. 17-21, 1993.
[38] Osborne, S.M., "Product Development Cycle Time Characterization Through Modeling of Process Iteration," Master's, Mgmt./Eng., MIT, Cambridge, MA, 1993.
[39] Meier, C., A.A. Yassine, and T.R. Browning, "Design Process Sequencing with Competent Genetic Algorithms," *Journal of Mechanical Design*, vol. 129, pp. 566-585, 2007.
[40] Yassine, A. and D. Braha, "Complex Concurrent Engineering and the Design Structure Matrix Method," *Concurrent Engineering: Research & Applications*, vol. 11, pp. 165-176, 2003.
[41] Yassine, A.A., D.E. Whitney, J. Lavine, and T. Zambito, "Do-It-Right-First-Time (DRFT) Approach to DSM Restructuring," ASME International Design Engineering Technical Conferences (Design Theory & Methodology Conference), Baltimore, MD, Sep. 10-13, 2000.
[42] Thomke, S. and T. Fujimoto, "The Effect of 'Front-Loading' Problem-Solving on Product Development Performance," *Journal of Product Innovation Management*, vol. 17, pp. 128-142, 2000.
[43] Browning, T.R., E. Fricke, and H. Negele, "Key Concepts in Modeling Product Development Processes," *Systems Engineering*, vol. 9, pp. 104-128, 2006.
[44] Eppinger, S.D. and T.R. Browning, *Design Structure Matrix Methods and Applications*. Cambridge, MA: MIT Press, 2012.
[45] Calantone, R.J. and C.A. Di Benedetto, "Performance and Time to Market: Accelerating Cycle Time with Overlapping Stages," *IEEE Transactions on Engineering Management*, vol. 47, pp. 232-244, 2000.
[46] Ahmadi, R.H., T.A. Roemer, and R.H. Wang, "Structuring Product Development Processes," *European Journal of Operational Research*, vol. 130, pp. 539-558, 2001.
[47] Brown, S.L. and K.M. Eisenhardt, "Product Development: Past Research, Present Findings, and Future Directions," *Academy of Management Review*, vol. 20, pp. 343-378, 1995.
[48] Stalk, G., Jr. and T.M. Hout, *Competing Against Time*. New York, NY: The Free Press, 1990.
[49] Roemer, T.A., R. Ahmadi, and R.H. Wang, "Time-Cost Trade-Offs in Overlapped Product Development," *Operations Research*, vol. 48, pp. 858-865, Nov.-Dec. 2000.
[50] Gerk, J.E.V. and R.Y. Qassim, "Project Acceleration via Activity Crashing, Overlapping, and Substitution," *IEEE Transactions on Engineering Management*, vol. 55, pp. 590-601, 2008.
[51] Kalyanaram, G. and V. Krishnan, "Deliberate Product Definition: Customizing the Product Definition Process," *Journal of Marketing Research*, vol. 34, pp. 276-285, 1997.
[52] Smith, P.G. and D.G. Reinertsen, *Developing Products in Half the Time*, 3rd ed. New York: Wiley, 1998.
[53] Adler, P.S., A. Mandelbaum, V. Nguyen, and E. Schwerer, "From Project to Process Management: An Empirically-based Framework for Analyzing Product Development Time," *Management Science*, vol. 41, pp. 458-484, 1995.
[54] Bhuiyan, N., D. Gerwin, and V. Thomson, "Simulation of the New Product Development Process for Performance Improvement," *Management Science*, vol. 50, pp. 1690-1703, 2004.
[55] Abdelsalam, H.M.E. and H.P. Bao, "A Simulation-based Optimization Framework for Product Development Cycle Time Reduction," *IEEE Transactions on Engineering Management*, vol. 53, pp. 69-85, 2006.
[56] Sargent, R.G., "Validation and Verification of Simulation Models," Winter Simulation Conference, Phoenix, AZ, Dec 5-8, 1999.
[57] Johnson, J., "The 'Can You Trust It?' Problem of Simulation Science in the Design of Socio-Technical Systems," *Complexity*, vol. 6, pp. 34-40, 2001.
[58] Smith, R.P. and J.A. Morrow, "Product Development Process Modeling," *Design Studies*, vol. 20, pp. 237-261, 1999.

[59] Hartmann, S. and R. Kolisch, "Experimental Evaluation of State-of-the-Art Heuristics for the Resource-Constrained Project Scheduling Problem," *European Journal of Operational Research*, vol. 127, pp. 394-407, 2000.
[60] Kurtulus, I. and E.W. Davis, "Multi-Project Scheduling: Categorization of Heuristic Rules Performance," *Management Science*, vol. 28, pp. 161-172, 1982.
[61] Lova, A. and P. Tormos, "Analysis of Scheduling Schemes and Heuristic Rules Performance in Resource-Constrained Multi-Project Scheduling," *Annals of Operations Research*, vol. 102, pp. 263-286, 2001.
[62] Rivkin, J.W. and N. Siggelkow, "Patterned Interactions in Complex Systems: Implications for Exploration," *Management Science*, vol. 53, pp. 1068-1085, 2007.
[63] Handfield, R.B. and S.A. Melnyk, "The Scientific Theory-Building Process: A Primer using the Case of TQM," *Journal of Operations Management*, vol. 16, pp. 321-339, 1998.
[64] Calcagno, V., E. Demoinet, K. Gollner, L. Guidi, D. Ruths, and C. de Mazancourt, "Flows of Research Manuscripts Among Scientific Journals Reveal Hidden Submission Patterns," *Science*, vol. 338, pp. 1065-1069, 2012.
[65] C. Meier, *Time-Cost Tradeoffs in Product Development Processes*, Doktor-Ingenieurs (Dr.-Ing.) thesis, Technische Universität München, Munich, Germany, 2011.