Abstract—A timely introduction of a new product has become invaluable to the firm since the competitors are capable of introducing new or similar products once the driving technology becomes available. In order to generate a high profit from a new product, managers in other departments, such as marketing and production, have to plan ahead of time so that a seamless series of operations can be executed from product development to mass production. Needless to say, a competitive edge is given to the firm with better knowledge on product development process. Such knowledge, nonetheless, is not easy to acquire since a typical product development process is a complex network of many relationships among activities, which we call patterns. In addition to its complex topology, the product development process is often uncertain, iterative, and evolving over time; therefore, even studying individual islands of relationships (patterns) is challenging. Although there were some existing models that shed lights on some of these patterns, very little has been done to systematically analyze the product development process as a whole. In this paper, we develop analytical models that capture essential properties, including uncertainty, iteration, and evolution, and estimate the cycle time of each pattern. With our proposed models, the cycle time of a set of patterns (or the whole product development process) can be effectively estimated. As demonstrated in a case study, our model provides valuable insights on how product development process progresses over time, while the corresponding time estimate can help managers to set appropriate manufacturing and marketing strategies.

Index Terms—Cycle time estimation, iteration, overlapping, process pattern, product development (PD) process.

I. INTRODUCTION

A TIMELY introduction of a new product is an invaluable asset to any firm as a product life cycle gets shorter and many competitors are able to quickly come up with new or similar products. It is clear that a firm with clear knowledge of its product development (PD) process has a considerable competitive edge over other firms. Such knowledge enables a firm to estimate the PD cycle time which is a prerequisite for efficient resource allocation. In fact, a good estimate of the development cycle time of the PD process will help other departments (e.g., marketing, manufacturing, etc.) make appropriate plans to allocate resources for a new product. In spite of these benefits, very few models are available for estimating the development cycle time because of the complexity of the PD process. As in Fig. 1, which shows the development process for a power seat motor of automobiles, a typical PD process consists of many activities where each activity is related to others by logical and chronological constraints.

Unlike manufacturing process, the PD process involves a number of decision-making activities that demand craftsmanship, aesthetic talent, creativity, and scientific mindset; therefore, it is quite common that each activity is performed by a group rather than a single individual (or department). In addition, the PD process has a few unique features not found in manufacturing process. First, the outputs of PD activities, materialized as engineering drawings, specifications, and technical calculations, are often unstable and inaccurate [1]–[3] since any design change may create a chain reaction of changes of data used in other activities. Second, most activities in the process are carried out together by a task force from multiple departments forming a complex relationship among activities. Finally, a portion of the PD process or the whole process may be repeated until the original design concept is materialized into acceptable outputs. This makes the PD process inevitably iterative.

These factors cause the duration of the PD process to be highly variable. As a result, estimating even the completion time of a single activity becomes difficult. Furthermore, when the order under which a set of activities will be performed is not determined a priori [4], the complexity of the PD process is beyond practical imagination. Nonetheless, it is possible to obtain a practical bound of the development cycle time of the PD process by carefully decomposing interrelated activities into a set of tractable relations, called patterns, and deriving expressions for the cycle time of each pattern. Following a taxonomy in Jun and Suh [5], in this paper, we identify seven commonly observed patterns and develop their analytical models while capturing essential features such as uncertainty, iteration, and learning effect. Using these models, we estimate the cycle time of each pattern as well as the entire PD process.

The rest of the paper is organized as follows. In Section II, we discuss the previous research. Section III introduces the seven-pattern classification by Jun and Suh [5]. In Section IV, we introduce notation and assumptions used in our models. We then present our models in two sections. Models for dependent patterns are presented in Section V, while models for interdependent patterns are presented in Section VI. For each pattern, we give an expression for the cycle time. To show how these models are used, we present a case study in Section VII and demonstrate that our analytic model can estimate the development cycle time.
II. Previous Research

Although very few studies analyze the entire PD process exist, there has been some research that analyzes small subsets of the PD process (see Table I). Smith and Eppinger [7] estimated the development cycle time in iterative PD activities using the expected reward of a Markov chain. Belhe and Kusiak [8] analyzed the risk of due date violation in a design activity network that has alternative routings and probabilistic branching. Browning and Eppinger [9] analyzed the effects of key characteristics, such as cost or duration uncertainty, rework probability, and improvement curves on the performance of the PD process.

Recently, several papers [6], [10]–[16] considered the overlap pattern(s) to estimate the makespan or to determine the optimal policy that minimizes the makespan. Krishnan et al. [11], [12] proposed a mathematical model to compute the makespan of a model with two characteristics of overlapping: evolution and sensitivity. Loch and Terwiesch [13] considered a decision model in which they determined the optimal communication policy that minimizes the makespan and analyzed the overlap rate and expected communication under the optimal policy. Roemer et al. [14] considered the tradeoff between time and
cost of overlapping and introduced an algorithm to determine a good overlapping strategy. Joglekar et al. [15] introduced the performance generation model (PGM) to develop insights on optimal concurrency strategies between coupled PD activities under a deadline. A similar work has been done in Chakravarty [16], in which he used an optimization model to analyze optimal time-cost tradeoff and the effects of key parameter values on overlapping decisions.

While these works and others have clearly shed light on the analysis of PD process, they are not free of limitations. First, most of these models and others focus on a small subset of patterns. Often, many of these work focus on just a few patterns (as shown in Table I), while leaving several other relations of design activities uncovered. Second, some key characteristics of a pattern are not explicitly captured in many existing models; therefore, it becomes questionable whether the cycle time estimates from these models are valid. For example, when analyzing branching and merging patterns, many existing models (e.g., [6], [10]–[13], and [17]–[19]) do not explicitly consider alternative routings, probabilistic branching, transfer time between activities, and synchronization. Third, most previous papers also do not consider the learning effect of an iterated activity. Finally, many models on overlap pattern [6], [10]–[13], [16] did not explicitly consider the downstream sensitivity which can be a significant factor when determining the cycle time. Although there have been some attempts to include sensitivity in analytical models (e.g., Krishnan et al. [11], [12] and Loch and Terwiesch [13]), these attempts remain conceptual definition and cannot be used to estimate a functional form with actual data.

To relax some of these limitations, we propose the analytical expressions that can estimate the cycle times of PD process patterns. Our work differs from previous work on a few key aspects. While others focused only on a selected subset of patterns, we derive analytical expressions for all seven patterns classified in the taxonomy of Jun and Suh [5]. To do that, we present analytical expressions for cycle, communication, and several types of branching and merging patterns, which are new to the literature. Furthermore, we improve analytical models for branching and merging patterns with different alternative routing types and synchronization types between activities. For overlap pattern, we develop a model with a sensitivity function which represents the rate at which rework occurs with respect to progress in an upstream activity, while keeping other assumptions consistent with existing models (e.g., [6], [11]–[14], [16], and [20]). We also consider the learning effect in an iterative pattern. With these models, we estimate the cycle time of each of seven patterns, as well as the overall PD process.

III. CLASSIFICATION OF PATTERNS

Although there exist several classifications of design activity patterns ([1], [6], [10], [11]), most only give a partial classification based on some selected criteria, such as, data, time, and semantics. Instead, we adopt the taxonomy developed by Jun and Suh [5], which classified the relationship between two or more activities in the PD process using the following six criteria.

A. Information Dependency

Depending on the extent of dependency between inputs and outputs of interconnected activities, the relation between PD activities can be categorized into three types: independent, dependent, and interdependent [6]. In an independent relation, each activity does not require the outputs from the other activity to complete. In a dependent relation, one activity requires outputs from the other activity to complete. In an interdependent relation, each activity requires the results from the other activity in order to complete.

B. Relation Cardinality

Based on the number of activities directly related to one particular activity, a single relation refers to the case when only one other activity is directly related to it and a multiple relation refers to the case when two or more other activities are related to it.

C. Degree of Overlapping

A single dependent relation between an upstream activity A and a downstream activity B is called a no-overlap pattern if activity B cannot start until A is finished. In contrast, an overlap pattern represents the case when activity B can begin with the draft design from the upstream activity A and proceed with the activity while accommodating changes in the upstream activity through reworks. We use the degree of overlapping to represent how early activity B can start before completion of activity A. Obviously, a no-overlap pattern has zero degree of overlapping.

D. Type of Collaboration

There are several different types of collaboration among multiple activities when solving problems and making decisions: feedback, interaction, cycle, communication, and branching and merging. We divide multiple interdependent relationships into feedback, interaction, cycle, and communication patterns based on how information is collaborated among activities. An interaction pattern occurs when two activities exchange design information through negotiations until they agree on a solution. A feedback pattern occurs when reworks are performed in all sequentially connected activities when the outputs fail to pass at the final activity. A cycle pattern occurs when three or more activities engage in sequential negotiations (information exchanges). In contrast, a communication pattern occurs when three or more activities engage in simultaneous negotiations. Interaction, cycle, and feedback patterns involve sequentially iterative tasks. But the communication pattern involves parallel tasks that repeat their works simultaneously until they agree on a solution.

In addition, we divide multiple dependent relationships into two patterns: Branching and Merging. Branching represents the behavior which one activity is separated into one or more according to logical and chronological constraints. On the other hand, merging represents the behavior when the results of several activities are combined to form one set of outputs.
branching always pairs with merging. To model branching and merging patterns, we need to further divide them based on the type of routing and the type of synchronization.

E. Type of Routing

We divide branching (or merging) patterns into AND, selective OR (SOR), or eXclusive OR (XOR) types based on how many activities are selected in a particular pattern. When all parallel activities are selected, we call it an AND routing type. When only one alternative activity is selected, we say that a pattern has an XOR routing type. When multiple (but not all) activities are selected, it is of a SOR type.

F. Type of Synchronization

Each routing type can be further decomposed to synchronous and asynchronous types depending on when each activity can start (complete) in a branching (merging) pattern.

These criteria differentiate the relations of PD activities and classify them into seven patterns: no-overlap, overlap, branching and merging, interaction, feedback, cycle, and communication, as the shaded areas show in Fig. 2.

IV. NOTATION AND ASSUMPTIONS

In order to develop a framework to express each of seven patterns in an analytic form, we make a few assumptions. First, we assume that the nominal duration of each activity is deterministic. Second, no self-loop is allowed in the PD process. Third, we assume that there exists a learning curve when the same design activity is repeated in an iteration pattern (c.f., [7]) as engineers accumulate experience, know-how, and skills. We also assume that a similar effect exists in negotiation and feedback patterns. Finally, we assume that all parameters are known or can be estimated using historical data or the knowledge of experienced engineers, or both.

To maintain brevity and consistency, we define the following notation:

- \( i \) activity index \( i \in \{ A_1, A_2, \ldots, A_n; B_1, B_2, \ldots, B_n; C \} \);
- \( S_i \) start time of activity \( i \);
- \( E_i \) completion time of activity \( i \);
- \( D_i \) duration of activity \( i \);
- \( I_i \) inputs to start activity \( i \);
- \( O_i \) outputs of activity \( i \);
- \( T_{i,j} \) elapsed time between the completion of activity \( i \) and the start of activity \( j \);
- \( N \) expected number of iterations;
- \( N_i \) expected number of iterations of activity \( i \);
- \( \Gamma_i \) probability that activity \( i \) is repeated, \( 0 \leq \Gamma_i < 1 \);
- \( \phi_i(x) \) learning effect function of activity \( y \) in the \( x \)th iteration;
- \( \phi_k \) parameter for the learning effect of activity \( i \), \( 0 < \phi_k \leq 1 \) (\( \phi_k = 1 \) indicates that there is no learning effect);
- \( \omega \) cycle time of a pattern.

In addition to these, we define the following notation for overlap pattern:

- \( \hat{N} \) random variable representing the number of reworks occurred during overlap;
- \( \bar{n} \) expected number of reworks during overlap;
- \( H_0 \) time at which the first information transfer occurs at an upstream activity to a downstream activity occurs;
- \( H_k \) time at which the \( k \)th change that triggers rework occurs during overlapping, \( k = 1, \ldots, \hat{N} \);
- \( \chi \) degree of evolution;
- \( \delta \) degree of sensitivity;
- \( \varphi \) degree of concurrency;
In the next two sections, we introduce the analytical models of seven patterns. We present the results in two sections: models and results for dependent patterns in Section V and models and results for interdependent patterns in Section VI.

V. MODELING DEPENDENT PATTERNS

In dependent patterns activities are performed only once and will not be repeated. Instead, some completion of some activities can be delayed by taking immature information from the upstream in advance. Such patterns include no-overlap, overlap, branching, and merging patterns.

A. No-Overlap

A no-overlap pattern [Fig. 3(a)] refers to a sequential relation between two activities, where a downstream activity can start only after the completion of an upstream activity. For example, as shown in Fig. 1(c), releasing preliminary specifications can start only after all reviews on the preliminary design are completed. We say that activities \( A \) and \( B \) form a no-overlap pattern if \( I_B = O_A \) and \( E_A \leq S_B \). This pattern is also called a serial
s-pattern, or sequential pattern [1], [10], [11], [24], [25]. The cycle time of a no-overlap pattern is simply

$$\omega = D_A + T_{A,B} + D_B.$$  \hfill (1)

B. Overlap

An overlap pattern is a relation between two activities, where the downstream activity can start with preliminary outputs from the upstream activity prior to the completion of the upstream activity. For example, as soon as some information from the activity “final design confirmation and release” becomes available, a firm can order (or produce) parts necessary for the end product [e.g., activities $P_{17}$ and $P_{18}$ in Fig. 1(e)]. We say that activity $A$ overlaps with activity $B$ if $I_B = O_A$ and $S_A \leq S_B \leq E_A \leq E_B$. To describe how much two activities are overlapped, we use the coefficient of concurrency ($\varphi$): If $0 < \varphi < 1$, it is said to be partial overlap [25], and if $\varphi = 1$, it is called concurrent [6], complete overlap [26], or simultaneous [26].

An overlap can accelerate the progress of a downstream activity by starting some tasks in advance before an upstream activity is completed (e.g., scheduling tentative production planning, estimating manufacturing manpower requirements, building or procuring special processing equipment and testing equipment [27]). On the other hand, an overlap may delay the development time if the preliminary information turns out to be false and misleading, and any partial progress based on this is scrapped and reworked. Therefore, starting the downstream activity early might reduce the duration, but it also increases the chance that a portion of the activity must be reworked when the information at the upstream activity changes suddenly. We model both effects in our estimated cycle time of an overlap pattern. Throughout this section, we assume the following:

- During the overlapped period, downstream activity $B$ continuously receives prereleased (incomplete and partial) information from upstream activity $A$.
- A downstream activity continuously monitors the progress of an upstream activity and occasionally receives a new piece of information which initiates rework at a downstream activity. We model the occurrences of such events with a nonhomogeneous Poisson process with an intensity function, $\lambda(t)$.
- Whenever the downstream activity undergoes rework, its completion time is delayed by the time spent on the rework.

Fig. 3(b) shows a typical example of overlap pattern between activities $A$ and $B$. During the overlapped period ($\Delta t$), a random number of events that will initiate reworks occur. We define $\bar{N}$ to be a random variable representing the number of reworks occurred during overlap and $H_i; i = 1, \ldots, \bar{N}$ to represent the time at which the ith rework starts. Whenever a rework occurs, the completion time of an activity $B$ is delayed by the time spent on a rework $B_i; i = 1, \ldots, \bar{N}$. If we ignore the time spent on rework, the duration of an overlap pattern is simply $(D_A + D_B - \Delta t)$, which we call a nominal duration. In order to accurately estimate the cycle time, we must account for additional time spent on reworks (denoted as $G_{1}$) and much of our effort in the rest of this subsection is to model this precisely.

We first note that the number of events that lead to reworks during overlapping period ($\bar{N}$) and the arrival patterns of such events affect $G_{1}$. To capture the idiosyncrasy of each activity and the synergy between two activities in an overlap pattern, we use a nonhomogeneous Poisson process to model the events that cause reworks and let $\lambda(t)$ be an instantaneous rate of an event at time $t$ (intensity function). In order to determine $\lambda(t)$, we examine how sensitive the downstream activity is with respect to changes in an upstream activity and take this into account in an intensity function.

In what follows, we describe how we define a sensitivity function from the field data, as well as how we use this to determine $\lambda(t)$. We then explain our procedure to estimate the time spent on reworks. A case study demonstrates how this can be done in practice. Finally, we present the results from numerical study to discuss the factors affecting the cycle time of an overlap pattern.

Typically, a downstream activity is highly susceptible to the changes in an upstream activity at the beginning of overlap, that is, when an upstream activity is not fully mature. To capture this, we use the following function to determine the sensitivity of a downstream activity:

$$\delta(t) = 1 - a - be^{-\left(\frac{\varphi \delta}{\lambda_A}\right)}, 0 \leq t \leq D_A, \delta \neq 0,$$

We note that $\delta(t)$ is a monotone decreasing function. In addition, we scale two coefficients, $a$ and $b$, so that $\delta(t)$ satisfies two boundary equations, $\delta(0) = 1$ and $\delta(D_A) = 0$. Solving for $a$ and $b$ in terms of $\delta$, we have

$$a = \frac{1}{1 - e^{-\delta}} \quad \text{and} \quad b = \frac{1}{e^{-\delta} - 1}.$$

Then, the downstream sensitivity at time $t$ is defined as follows:

$$\delta(t) = 1 - \frac{1}{1 - e^{-\delta}} - \frac{1}{e^{-\delta} - 1} e^{-\left(\frac{\varphi \delta}{\lambda_A}\right)}, 0 \leq t \leq D_A, \delta \neq 0.$$

In fact, $\delta$ is a shape parameter of this function. When $\delta > 0$, $\delta(t), t \in [0, D_A]$ is a convex and decreasing function which represents the case where the downstream activity is highly sensitive at the beginning of overlap, but quickly becomes stable and robust. When $\delta < 0$, the sensitivity function is concave, and represents the case where the downstream activity slowly becomes robust to the changes during the overlap.

In order to get a good sensitivity function, it is important to translate the subjective answers from the managers and engineers on the evolution of the activity in an absolute time scale. Engineers and managers often remember the events associated with the project in a relative time scale (i.e., milestone accomplishments) rather than in an absolute time scale. However, how these milestone events occur varies in the idiosyncrasy of an activity; therefore, we need to define a function that represents how the upstream activity evolves over time. We do this by constructing an evolution function. Following similar steps, we determine the evolution function $\chi(t) = (1/(1-e^{-\varphi})) + (1/(e^{-\varphi} - 1))e^{-(\alpha t/D_A)}, 0 \leq t \leq D_A, u \neq 0$ from the interview and regression analysis.

Using the evolution function, we can calibrate subjective interview results on the sensitivity of downstream activity in an
absolute time scale and obtain the sensitivity function. We believe that estimating both functions is necessary to obtain calibrated answers on how the upstream activity affects the downstream activity from subjective beliefs and opinions.

1) Arrival Pattern of Reworks: During an overlap, several changes in the upstream activity randomly occur and force the downstream activity to revise its outputs (i.e., rework). As we described earlier, the rate at which these events occur depends on both the maturity of an upstream process and the sensitivity of a downstream process. In order to represent the rate, we use a nonhomogeneous Poisson process, which is a reasonable approach to model the stochastic process with varying rate. As we will show shortly, we estimate this as a function of the time that the last rework during the overlap occurs and the average time between reworks. The approximated values of both quantities are immediately determined once we can estimate the intensity function \( \lambda(t) \) using the sensitivity function and the field data.

To begin, let \( \{ \hat{N}(\hat{t}), \hat{t} \geq 0 \} \) be a nonhomogeneous Poisson process which represents the number of reworks occurred by \( \hat{t} \) time units after the preliminary information was made available to the downstream activity for the first time (i.e., \( t^* \)). Then

\[
P\left( \hat{N}(\hat{t}) = x \right) = \frac{\lambda(\hat{t})^x}{x!} e^{-\lambda(\hat{t})}, \quad (x = 0, 1, 2, \ldots)
\]

where \( \Lambda(\hat{t}) = \int_0^\hat{t} \lambda(t)dt, \hat{t} > 0 \) is the average number of reworks during the first \( \hat{t} \) time units.

From the independent increment property, the number of reworks that occur between \( \hat{t} \geq 0 \) and \( \hat{t} + \Delta \hat{t}, \Delta \hat{t} \geq 0 \) follows also a nonhomogeneous Poisson process. We denote \( \Delta \lambda(\hat{t} + \Delta \hat{t}) \) as the expected number of reworks on this interval

\[
\Delta \lambda(\hat{t} + \Delta \hat{t}) = E\left( \hat{N}(\hat{t} + \Delta \hat{t}) \right) - E\left( \hat{N}(\hat{t}) \right) = \int_{\hat{t}}^{\hat{t} + \Delta \hat{t}} \lambda(t)dt.
\]

We assume that the intensity function \( \lambda \) is proportional to the downstream sensitivity function. Therefore, the rate at which a rework occurs at time \( t + t^* \) is

\[
\lambda(t) = k \cdot \delta(t + t^*), \quad 0 \leq t \leq \Delta t = D_A - t^*.
\]

A scaling constant \( k \) is determined to equate the expected number of reworks during the overlap to the historical average number of reworks for similar projects (i.e., \( \lambda_{\text{avg}}(D_A) = \bar{n} \)).

Using this function, we approximate the expected time at which the last rework occurs with \( E(H_\bar{n}) \). Using the property that the distribution of the \( \text{th} \) event of a nonhomogeneous Poisson process with rate \( \lambda(t) \) conditioned on \( \hat{N}(\Delta t) = \bar{n} \) follows a distribution of the \( \text{th} \) order statistic of \( \bar{n} \) independent identically distributed (i.i.d.) uniform random variables on \([0, \Delta t]\), we have

\[
E(H_\bar{n}) = \int_0^{\Delta t} \bar{n} \cdot H_\bar{n} [F(H_\bar{n})]^{(\bar{n} - 1)} f(H_\bar{n})dH_\bar{n}
\]

where \( f(u) = \lambda(u)/\Lambda(\Delta t), 0 \leq u \leq \Delta t \).

2) Estimating \( G_1 \): We now estimate the expected time spent on rework using the estimates computed above. We assume that a portion of work done prior to a new rework request can be saved. The extent of work that can be saved depends on the average progress of the downstream activity (\( \rho_B \)) and the coefficient which represents the extent of disruption of work caused by the frequency of reworks (\( \theta \)). While \( \rho_B \) is determined from the historical data and interviews, we determine \( \theta \) by measuring the relative spread between the expected time of the last rework (\( E(H_\bar{n}) \)) and the average time between reworks (\( E(\Omega) \)) over the overlap. That is

\[
\theta = \frac{E(H_\bar{n}) - E(\Omega)}{\Delta t}.
\]

We estimate the time spent on rework during overlap by subtracting the amount of progress made during overlap from the time last rework request occurs

\[
G_1 = E(H_\bar{n}) - \rho_B \cdot E(H_\bar{n}) \cdot (1 - \theta) = E(H_\bar{n}) \left( 1 - \rho_B \cdot (1 - \theta) \right).
\]

Combining this with the nominal duration of the cycle time, the estimated cycle time for an overlap pattern is \( \omega = D_A + D_B - \Delta t + G_1 \).

3) Example: We illustrate our method using an example in Fig. 4(a), which shows an overlap pattern between activities \( P_{17} \) and \( P_{38} \). In order to estimate its cycle time, we obtain the sensitivity function, the average progress rate during reworks (\( \rho_{P_{38}} \)), and the average number of reworks during overlap (\( \bar{n} \)) from the
field data and interviews. In particular, for the sensitivity function, we asked engineers to estimate how the upstream activity would evolve over time and conducted a regression analysis to determine the evolution function. We then asked them to estimate the downstream sensitivity at various points in time during overlap while making sure that their answers on how the upstream process evolves are consistent with the evolution function. We then performed a regression analysis to determine the sensitivity function. The results are summarized in Fig. 4.

Fig. 4(b) shows the time series data of evolution function and sensitivity function. From this, we obtained the following sensitivity function for activity 18:

\[ \delta_{P_{18}}(t) = 1 - \frac{1}{1 - e^{-\frac{2t}{30}}} - 2 \cdot \frac{e^{-\frac{2t}{30}}}{e^{-\frac{2t}{30}} - 1}, \quad 0 \leq t \leq 30. \tag{4} \]

We found \( \Delta t = 22 \) days after noting that the duration of activity 17 was 30 days \((D_A = 30 \text{ days})\) and the first time that the information was released to activity 18 is eight days after activity 17 began \((t^* = 8 \text{ days})\). We also found the average rate of progress on activity 18 during rework was around 0.5. From the fact that five reworks occurred on average in similar projects (i.e., \( n = 5 \)), we determine \( \lambda(t) \) as follows:

\[ \lambda(t) = 0.00331 + 0.76329e^{-0.24733t}, \quad 0 \leq t \leq 22. \]

From this, we get \( E(H_5) = 8.37 \text{ days} \), \( E(\Theta) = (1/5)E(H_5) = 1.67 \text{ days} \), and \( \theta = (8.37 - 1.67)/22 \approx 0.3 \). Replacing the corresponding terms with these numbers, we have

\[ \omega = D_P + D_R - (D_{P_R} + \Theta) + E(H_5) \cdot (1 - \rho_{P_{18}} \cdot (1 - \Theta)) = 30 + 180 - (30 - 8) + 8.37 \cdot (1 - 0.5 \cdot 0.7) = 193.44 \text{ days}. \]

The actual cycle time of the design process which we model as an overlap pattern was 195 days, which is not far from our estimate.

4) Computational Experiment: In order to gain insights on how the sensitivity and the expected number of rework affect the cycle time, we have conducted a numerical study. We consider three levels of sensitivity (low, medium, and high) and four levels of the expected number of reworks. All other parameters remain constant in examples. We present the estimated cycle time and \( E(H_5) \) (the expected time at which the final rework request will occur) in Table II. We find the cycle time tends to increase as a downstream activity becomes more sensitive or the expected number of reworks increases. We also find that when the expected number of reworks is the same, both the cycle time and \( E(H_5) \) decrease as the sensitivity decreases. That is because rework requests are concentrated at the beginning of the project when the sensitivity is low. This suggests that either reducing the number of chances that the upstream activity can prerelease the information or making the design process stable early (low sensitivity) will reduce the cycle time.

<table>
<thead>
<tr>
<th>No. of reworks</th>
<th>Degree of sensitivity</th>
<th>Low (5-7.42)</th>
<th>Medium (6-8)</th>
<th>High (8-16.25)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n = 1 )</td>
<td></td>
<td>188.76 (3.78)</td>
<td>189.55 (7.76)</td>
<td>189.88 (9.42)</td>
</tr>
<tr>
<td>( n = 3 )</td>
<td></td>
<td>190.55 (6.94)</td>
<td>194.26 (12.47)</td>
<td>195.93 (14.43)</td>
</tr>
<tr>
<td>( n = 5 )</td>
<td></td>
<td>191.71 (8.37)</td>
<td>196.89 (14.38)</td>
<td>198.93 (16.25)</td>
</tr>
<tr>
<td>( n = 10 )</td>
<td></td>
<td>193.72 (10.52)</td>
<td>200.82 (16.97)</td>
<td>202.35 (18.11)</td>
</tr>
</tbody>
</table>

* Estimated cycle time
† Estimated value of \( E(H_5) \)

Fig. 5. Branching and merging patterns. (a) Branching. (b) Merging. (c) Simple branching and merging pattern.

C. Branching and Merging

a) Branching: Branching [Fig. 5(a)] occurs when the output of an upstream activity \( A \) is an input for multiple downstream activities \( B_1, B_2, \ldots, B_n \). In a typical PD process, branching occurs quite often since one big upstream activity can split into a number of specific tasks (i.e., constituent elements). Formally, there exists a branching pattern if \( O_{B_i} \not\subset I_{B_j}, O_{B_j} \not\subset I_{B_i} \), and \( O_A = \bigcup_{i=1}^{n} I_{B_i} \) for any \( i, j \in \{1,2,\ldots,n\} \). This pattern is also called \( B \)-pattern [1], \( \text{Fan-out} \) [32], or \( \text{Fork} \) [33]. There are five different type of branching depending on routing and synchronization.

1) \( \text{AND} \) and synchronous type: After the completion of activity \( A \), all activities \( B_i \)’s start simultaneously as soon as they are ready to start.
2) \( \text{AND} \) and asynchronous type: After the completion of activity \( A \), any activities \( B_i \)’s that are ready can start.
3) \( \text{XOR} \) type: After the completion of activity \( A \), only one activity \( B_i \) (among \( B_i \)’s that are ready to start) is selected and starts.
4) \( \text{SOR} \) and synchronous type: After the completion of activity \( A \), all activities in a predefined subset of \( B_i \)’s start simultaneously as soon as they are ready to start.
5) \( \text{SOR} \) and asynchronous type: After the completion of activity \( A \), any activities in a predefined subset of \( B_i \)’s that are ready can start.

b) Merging: Merging [Fig. 5(b)] occurs when a downstream activity \( C \) waits for the outputs (finished or partial) from upstream activities \( B_1, B_2, \ldots, B_n \) before its start. Like branching, merging occurs quite often because the PD process
TABLE III
ANALYTICAL MODELS OF SIMPLE BRANCHING AND MERGING PATTERNS

<table>
<thead>
<tr>
<th>Type</th>
<th>Expressions</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND(Syn.)-AND(Syn.)</td>
<td>$D_A + \max(T_{A,B_1}, \ldots, T_{A,B_n}) + \max(D_{B_1}, T_{B_1}, \ldots, D_{B_n}, T_{B_n}, C) + DC$</td>
</tr>
<tr>
<td>AND(Syn.)-AND(Async.)</td>
<td>$D_A + \max(T_{A,B_1}, \ldots, T_{A,B_n}) + \min(D_{B_1}, T_{B_1}, \ldots, D_{B_n}, T_{B_n}, C) + DC$</td>
</tr>
<tr>
<td>AND(Async.)-AND(Async.)</td>
<td>$D_A + \max(T_{A,B_1}, D_{B_1}, T_{B_1}, \ldots, T_{A,B_n}, D_{B_n}, T_{B_n}, C) + DC$</td>
</tr>
<tr>
<td>AND(Async.)-AND(Syn.)</td>
<td>$D_A + \min(T_{A,B_1} + D_{B_1} + T_{B_1}, \ldots, T_{A,B_n} + D_{B_n} + T_{B_n}, C) + DC$</td>
</tr>
<tr>
<td>AND(Async.)-AND(Async.)</td>
<td>$D_A + \min(T_{A,B_1} + D_{B_1} + T_{B_1}, \ldots, T_{A,B_n} + D_{B_n} + T_{B_n}, C) + DC$</td>
</tr>
<tr>
<td>AND(Syn.)-SOR(Syn.)</td>
<td>$D_A + \max(T_{A,B_1}, \ldots, T_{A,B_n}, D_{B_1}, T_{B_1}, \ldots, D_{B_n}, T_{B_n}, C) + DC$</td>
</tr>
<tr>
<td>AND(Async.)-SOR(Syn.)</td>
<td>$D_A + \sum_{i=1}^{n} C_i \sum_{j=1}^{k} \min_{m=1}^{\text{max}} T_{A,B_i} + \sum_{i=1}^{n} C_i p_{r_j}(\max_{j=1}^{k} D_{r_j} + T_{r_j}, C) + DC$</td>
</tr>
<tr>
<td>SOR(Syn.)-SOR(Syn.)</td>
<td>$D_A + \sum_{i=1}^{n} C_i p_{r_j}(\max_{j=1}^{k} T_{A,B_i} + \max_{j=1}^{k} D_{r_j} + T_{r_j}, C) + DC$</td>
</tr>
<tr>
<td>SOR(Async.)-SOR(Async.)</td>
<td>$D_A + \sum_{i=1}^{n} C_i p_{r_j}(\max_{j=1}^{k} T_{A,B_i} + \max_{j=1}^{k} D_{r_j} + T_{r_j}, C) + DC$</td>
</tr>
<tr>
<td>SOR(Async.)-SOR(Syn.)</td>
<td>$D_A + \sum_{i=1}^{n} C_i p_{r_j}(\max_{j=1}^{k} T_{A,B_i} + \max_{j=1}^{k} D_{r_j} + T_{r_j}, C) + DC$</td>
</tr>
<tr>
<td>SOR(Async.)-AND(Async.)</td>
<td>$D_A + \sum_{i=1}^{n} C_i p_{r_j}(\max_{j=1}^{k} T_{A,B_i} + \max_{j=1}^{k} D_{r_j} + T_{r_j}, C) + DC$</td>
</tr>
<tr>
<td>SOR(Async.)-AND(Syn.)</td>
<td>$D_A + \sum_{i=1}^{n} C_i p_{r_j}(\max_{j=1}^{k} T_{A,B_i} + \max_{j=1}^{k} D_{r_j} + T_{r_j}, C) + DC$</td>
</tr>
<tr>
<td>AND(Syn.)-XOR</td>
<td>$D_A + \max(T_{A,B_1}, \ldots, T_{A,B_n}) + \sum_{i=1}^{n} p_i(D_{B_1} + T_{B_1}, C) + DC$</td>
</tr>
<tr>
<td>AND(Async.)-XOR</td>
<td>$D_A + \sum_{i=1}^{n} p_i(T_{A,B_i} + D_{B_i} + T_{B_i}, C) + DC$</td>
</tr>
<tr>
<td>XOR</td>
<td>$D_A + \sum_{i=1}^{n} p_i(T_{A,B_i} + D_{B_i} + T_{B_i}, C) + DC$</td>
</tr>
</tbody>
</table>

has many activities that integrate or combine several outputs of subactivities into one when designing, analyzing, or testing. We say that a merging pattern occurs if \( S \), \( C \), and \( T \) for any \( T \). Merging is also called \( \text{Fan-in} \) [32], and \( \text{Join} \) [33]. Similarly to branching, we divide merging into five different types.

1) \( \text{AND} \) and \( \text{synchronous} \) type: If all activities \( B_i \)'s are completed, then activity \( C \) can start.
2) \( \text{AND} \) and \( \text{asynchronous} \) type: If at least one of activities \( B_i \)'s is completed, then activity \( C \) starts.
3) \( \text{XOR} \) type: If one of selected activities in \( B_i \)'s is finished, then activity \( C \) can start.
4) \( \text{SOR} \) and \( \text{synchronous} \) type: If all activities in a subset of \( B_i \)'s are completed, then activity \( C \) starts.
5) \( \text{SOR} \) and \( \text{asynchronous} \) type: If at least one activity in a subset of \( B_i \)'s is completed, then activity \( C \) starts.

c) Branching and Merging: We note that the branching and merging always exist in pairs [see Fig. 1(b)] since branching or merging cannot be used alone in the general project network that has a sink node and a source node. Therefore, we estimate the cycle time for each combination of branching and merging and present results in Table III. We rule out some combinations that do not have any logical meaning. For each combination, we merge intermediate activities between branching node and merging node and convert it into an equivalent \( \text{simple branching and merging} \) pair with only one phase of intermediate activities, as in Fig. 5(c).

VI. MODELING INTERDEPENDENT PATTERNS

Based on our taxonomy we have so far characterized dependent patterns and presented an estimate of the cycle time for each pattern. In this section, we model interdependent patterns in which one or more activities can be repeated as the results of data exchange, communication, or negotiations.

A. Interaction

An interaction (also called \( \text{I-pattern} \) [1]) occurs when engineers of two activities negotiate with each other in order to produce the required outcome, as shown in Fig. 1(d). An interaction pattern exists between two activities \( A \) and \( B \) if \( I_B = O_A, I_A = O_B, \text{and } E_A \leq S_B \) [see Fig. 6(a)]. Two activities \( A \) and \( B \) are performed at least once; after the first iteration, the outputs will be reviewed and activities will be repeated until either the output of activity \( A \) agrees on the outputs of activity \( B \) or the output of activity \( B \) agrees on the outputs of activity \( A \). Let \( 1 - \Gamma_A \) be the probability that activity \( A \) agrees with activity \( B \).
in a single iteration, and let $1 - \Gamma_B$ be the one that activity $B$ agrees with activity $A$ in a single iteration. Then, the expected number of iterations of activity $A$ (denoted by $N_A$) and that of activity $B$ (denoted by $N_B$) are, respectively,

$$N_A = 1 \cdot (1 - \Gamma_B) + \sum_{n=2}^{\infty} n(\Gamma_A \Gamma_B)^{n-2} \cdot \Gamma_B(1 - \Gamma_A \Gamma_B)$$

$$N_B = \sum_{n=1}^{\infty} n(\Gamma_A \Gamma_B)^{n-1} \cdot (1 - \Gamma_A \Gamma_B)$$

$$= \frac{1}{(1 - \Gamma_A \Gamma_B)}, \ 0 < \Gamma_A, \Gamma_B < 1$$

where each term of the summation in (5) and (6) is the probability that there are exactly $n$ iterations multiplied by $n$.

When the same activity is repeated, the time needed to complete the task decreases as engineers and designers learn more about the task (i.e., a learning effect). We model this reduction of time with a learning curve, consistent with models in learning literature (e.g., Yelle [34]). Let $x \rightarrow 1$ be the number of iterations performed and $\varepsilon_1$ be the learning index for activity $i$. The two learning curves ($I_A(x)$ for activity $A$ and $I_B(x)$ for activity $B$) are then modeled as follows: $I_A(x) = e^{-\varepsilon_1}$, $I_B(x) = e^{-\varepsilon_1 \phi_B}$, and $\varepsilon_2 = -\log_2 \phi_B$ for $x > 1$ and $\phi_A, \phi_B \leq 1$.

Applying the learning curve to the duration of each activity, we have the following estimate for the cycle time of a feedback pattern

$$\omega = \sum_{x=1}^{N_A} (D_A \cdot I_A(x) + T_{B,A}) - T_{B,A}$$

$$+ \sum_{x=1}^{N_B} (D_B \cdot I_B(x) + T_{A,B})$$

$$(N_A - \lfloor N_A \rfloor) \{D_A \cdot I_A(\lfloor N_A \rfloor) + T_{B,A}\}$$

$$+(N_B - \lfloor N_B \rfloor) \{D_B \cdot I_B(\lfloor N_B \rfloor) + T_{A,B}\}. \quad (7)$$

In (7), $\lfloor y \rfloor$ is the largest integer that does not exceed $y$, and $N_A = N_A + 0.5$ and $N_B = N_B + 0.5$ are integers rounded-off of $N_A$ and $N_B$, respectively. The estimated cycle time is the sum of the total time spent in activities $A$ and $B$ with $\lfloor N_A \rfloor$ and $\lfloor N_B \rfloor$ iterations (first three terms) and corrective terms to compensate for the round-off error of iterations.

### B. Feedback

When activity $A_i$’s are repeated after reviewing its outputs at some downstream activity $A_j$, a feedback pattern occurs (Fig. 6(b)). Feedback patterns are quite common in the PD process since many design activities involve reviewing the output of precedent activities. When the feedback occurs, all activities in a feedback loop will be repeated until activity $A_j$ finally releases the output and the PD process moves forward. Let $1 - \Gamma_{A_j}$ be the probability that activity $A_j$ approves the output. Then, the expected number of iterations in a feedback pattern follows a geometric distribution, where parameter $1 - \Gamma_{A_j}$ is simply $N = 1/(1 - \Gamma_{A_j})$. Similar to the interaction pattern, learning applies when the same activities are repeated. The estimated cycle time is

$$\omega = \sum_{x=1}^{N_A} \sum_{k=1}^{J} (D_{A_k} \cdot I_{A_k}(x) + T_{A_k,A_{k+1}}) - T_{A_j,A_i}$$

$$+(N - \lfloor N \rfloor) \sum_{k=i}^{J} (D_{A_k} \cdot I_{A_k}([N*]) + T_{A_k,A_{k+1}})$$

where $T_{A_j,A_{j+1}} = T_{A_j,A_i}$ and $N* = N + 0.5$. The first and the second terms denote the duration with $\lfloor N \rfloor$ iterations. The second term is used to compensate for the overestimated value of the elapsed time. Others denote the duration with $(N - \lfloor N \rfloor)$ iterations.

### C. Cycle

In many instances the design outputs of one activity cannot be determined independently from the outputs of other activities. Instead, activities $A_1, A_2, \ldots, A_n$ form a cyclic and sequential
relationship, which we call a cycle pattern (also known as C-pattern [1] or coupled [35]). We say that a cycle pattern exists if \( I_{A_2} = O_{A_1}, I_{A_3} = O_{A_2}, \ldots, I_{A_n} = O_{A_{n-1}} \). In a cycle pattern, all activities are performed once, then the outputs are evaluated. If all outputs meet the specifications, then the PD process moves forward and exits a cycle pattern. Otherwise, each activity negotiates with the next activity until all activities agree on a set of outputs. Therefore, the expected number of iterations for activity \( i \) is for \( k = 1, 2, \ldots, n \)

\[
N_{A_k} = 1 + \sum_{i=1}^{\infty} i \cdot (1 - \Gamma_1 \cdots \Gamma_n)^{i-1} \cdot \prod_{j=1}^{n} \Gamma_{I_{A_k}^{(j)}} \cdot (1 - \sigma_{i-k-1})^{(j)}
\]

\[
= \frac{1}{(1 - \Gamma_1 \cdots \Gamma_n)^2} \sum_{j=1}^{n} \prod_{j=1}^{n} \Gamma_{I_{A_k}^{(j)}} \cdot (1 - \sigma_{i-j-1})^{(j)}
\]

(8)

where \( \prod_{j=1}^{n} \Gamma_{I_{A_k}^{(j)}} = 1 \) and \( \prod_{j=1}^{n} \Gamma_{I_{A_k}^{(j)}} = 1 \) for any \( j \). Here, \( \Gamma_k \) indicates \( \Gamma_{A_k} \) and \( \sigma \) is a bijective function: \( \{1, 2, \ldots, n\} \rightarrow \{1, 2, \ldots, n\} \) such that \( \sigma(i) = i + 1, i = 1, 2, \ldots, n - 1 \), and \( \sigma(n) = 1 \). In addition, \( \sigma^{(k)}(i) \) is a kth recursive relation such that \( \sigma^{(k)}(i) = i + k \). Accounting for the savings from learning when the same activities are repeated, we estimate the cycle time as follows:

\[
\omega = \sum_{k=1}^{n} \sum_{x=1}^{N_{A_k}} (D_{A_k} \cdot I_{A_k}(x) + T_{A_k + 1, A_k}) - T_{A_k, A_1}
\]

\[
+ \sum_{k=1}^{n} (N_{A_k} - [N_{A_k}]) (D_{A_k} \cdot I_{A_k}([N_{A_k}^*]) + T_{A_k + 1, A_k})
\]

(9)

where \( N_{A_k}^* = N_{A_k} + 0.5 \) and \( T_{A_k + 1, A_k} = T_{A_k, A_1} \). The second term is used to compensate for the overestimated elapsed time.

D. Communication

When the PD process moves forward only if two or more activities simultaneously agree on a set of design outputs, we say that a communication pattern exists. Specifically, we say that a communication pattern exists among activities \( A_1, A_2, \ldots, A_n \), if \( I_{A_i} = \bigcup_{j=1}^{n} O_{A_j} \) [Fig. 6(d)]. Each activity should be simultaneously repeated until all engineers agree on their outputs. Therefore, the expected number of iterations in a communication pattern is

\[
N = \sum_{k=1}^{\infty} k (1 - \Gamma_{I_{A_k}^{(1)}})^{k-1} \cdot \prod_{j=1}^{n} (1 - \Gamma_{A_k})
\]

\[
= \frac{1}{\prod_{j=1}^{n} (1 - \Gamma_{A_k})}.
\]

With probability \( (1 - \prod_{j=1}^{n} (1 - \Gamma_{A_k})) \), the pattern fails to disseminate the final output in a given iteration and must repeat the activities. Then, the estimated cycle time of a communication pattern is computed by adding up of each activity until the pattern terminates, while taking into account learning effect

\[
\omega = \sum_{x=1}^{\lfloor N \rfloor} \max_{i=1, \ldots, n} D_{A_i} \cdot I_{A_i}(x) + (N - \lfloor N \rfloor)
\]

\[
\cdot \max_{i=1, \ldots, n} D_{A_i} \cdot I_{A_i}([N^*]) + N \cdot \sigma
\]

where \( N^* = N + 0.5 \) and \( \sigma \) is the average communication time per iteration.

VII. CASE STUDY

We now present a case study to show that our model can decompose a complex design process into patterns and give a good estimate of the cycle time of each pattern as well as the entire PD process. To demonstrate how this approach works, we use the example depicted in Fig. 7, which is a part of PD process of a motor used in an automobile (Fig. 1). Although we can certainly show our work for the entire PD process, we present
the result for Fig. 7, which is a large subset of Fig. 1 since the cycle times for remaining patterns can be computed in a similar manner. Note that this example consists of several patterns described in Sections V and VI, such as feedback, cycle, interaction, and branching and merging.

While collecting the data for our case study, we found that it was quite challenging to obtain a large pool of reliable data in practice. One main reason is that the company has not yet realized how past PD data could be used in managing the future PD projects. In order to estimate the parameters for our model, we collected available data and conducted extensive interviews with senior engineers and department managers. For example, using historical data and interviews, we also determined average processing time \( D_i \) and the probability that each activity is repeated (\( \Gamma_i \)).

From the field data, we found that some activities showed a strong learning effect mostly at the beginning of the process. We model this behavior with a learning effect function. We also found that considerable delays had occurred when transferring information from one activity to another. Considering the size of the company (a mediumsized part supplier for an automobile), this was quite surprising. However, a closer look at the company’s information, which revealed the company has not been using the most sophisticated information system, explained such delays. Again, our model could account for delays during information transfer.

We first estimate the cycle time of the cycle pattern embedded in activity \( P_b \). Applying (8), we obtain \( N_{P_{b a}} = 2.012 \), \( N_{P_{b b}} = 1.305 \), and \( N_{P_{b c}} = 1.002 \), respectively. Substituting these into (9), which is an expression for a cycle pattern, we have

\[
D_{P_{b}} = (D_{P_{b a}} \cdot I_{P_{b a}}(1) + T_{P_{b c}, P_{b a}}) + (D_{P_{b b}} \cdot I_{P_{b b}}(2) + T_{P_{b c}, P_{b b}}) + (D_{P_{b c}} \cdot I_{P_{b c}}(1) + T_{P_{b c}, P_{b b}}) + (N_{P_{b a}} - [N_{P_{b a}}]) \times (D_{P_{b a}} \cdot I_{P_{b a}}(\left[ N_{P_{b a}}^{\text{a}} \right]) + T_{P_{b c}, P_{b a}}) + (N_{P_{b b}} - [N_{P_{b b}}]) \times (D_{P_{b b}} \cdot I_{P_{b b}}(\left[ N_{P_{b b}}^{\text{b}} \right]) + T_{P_{b c}, P_{b b}}) + (N_{P_{b c}} - [N_{P_{b c}}]) \times (D_{P_{b c}} \cdot I_{P_{b c}}(\left[ N_{P_{b c}}^{\text{c}} \right]) + T_{P_{b c}, P_{b c}}) = (1 + 0.5) + (0.2 + 0.5) + (1 + 0.5) + (1 + 0.5) - 0.5 + (2.012 - 2) \times (0.3 + 0.5) + (1.305 - 1) \times (1 + 0.5) + (1.002 - 1) \times (1 + 0.5) = 5.36 days.
\]

We then use this to estimate the cycle time of a branching and merging process. Since both branching and merging follow a AND/synchronous routing rule, the first iteration of the branching and merging process will take

\[
\omega_1 = D_{P_{a}} + \max_{i=2,...,6} T_{P_{a}, P_{i}} + \max_{i=2,...,6} (D_{P_{1} + T_{P_{i}, P_{r}}}) + D_{P_{r}}.
\]

We note that any change in activity \( P_b \) initiates the corresponding changes in activities (\( P_1 \sim P_7 \)) because it is a part of a feedback pattern. On the other hand, the tasks will be completed more efficiently in the second or third iteration of the same activities. Taking both a feedback pattern and a learning effect into account, we get the following estimate:

\[
\omega'_1 = D_{P_{a}} \cdot I_{P_{a}}(1) + \max_{i=2,...,6} T_{P_{a}, P_{i}} + \max_{i=2,...,6} (D_{P_{1} \cdot I_{P_{1}}(1) + T_{P_{r}, P_{i}}}) + D_{P_{r}} \cdot I_{P_{r}}(1) + T_{P_{r}, P_{b}} + D_{P_{b}} \cdot I_{P_{b}}(1) + T_{P_{b}, P_{r}} - T_{P_{b}, P_{r}} + 0.43 \times \left[ D_{P_{a}} \cdot I_{P_{a}}(1) + \max_{i=2,...,6} T_{P_{a}, P_{i}} + \max_{i=2,...,6} (D_{P_{1} \cdot I_{P_{1}}(1) + T_{P_{r}, P_{i}}}) + D_{P_{r}} \cdot I_{P_{r}}(1) + T_{P_{r}, P_{b}} + D_{P_{b}} \cdot I_{P_{b}}(1) + T_{P_{b}, P_{r}} \right] = 127.7 days.
\]

Similarly, we compute the cycle time of the interaction pattern (\( P_b \) and \( P_{10} \)). We first estimate \( N_{P_{b a}} = 1.79 \) and \( N_{P_{b b}} = 1.32 \), then infer the learning effect function (\( I_{P_{b}}(x) = I_{P_{b a}}(x) = x^{-6} e^{-5} \)) from the data, then give an estimated cycle time of 126.8 days from (7). Based on these data, we finally calculate the cycle time of a network of patterns from activity \( P_a \) to activity \( P_{10} \), which includes branching and merging, feedback, interaction, and cycle patterns

\[
\omega = \omega'_1 + T_{P_{b}, P_{b}} + \omega_2 = 127.7 + 7 + 126.8 = 261.5 \text{ days}.
\]

We report that our estimate is very close to the actual development time of 270 days. Considering the complexity of a PD process, our method is an efficient and simple way to estimate the PD cycle time. Although we only present the approximation method for the development cycle time, our method also can be applied to estimate the variance of a particular pattern or an entire PD process with simulation. In addition, our model provides a useful tool to test the sensitivity of the cycle time with respect to changes or errors of estimated key parameters. All these can be quite helpful to managers and engineers participating in the PD process.

Although our method is a good starting point, our model has some limitations. First, we assume there is no resource allocation constraint (other than logical and sequential constraints imposed by a PD process itself). However, integrating resource constraints into a model will give a realistic estimate of the development time for firms with limited resource. Second, we only consider a convex or concave sensitivity function in an overlap pattern. Although we believe this represents many practical examples, our model may not give an accurate estimate when an estimated sensitivity function is neither convex nor concave, such as a S curve.

Furthermore, there are other alternative models available for some patterns, which are different from the ones we consider. For example, we model rework occurrences with a nonstationary Poisson process. Although this is certainly a reasonable model, one may follow a different approach as in Koushik [36].
In addition, we do not explicitly model the magnitude of each information change as a random variable and its impact on the cycle time. Although we implicitly assume the magnitude of each information change is evenly distributed via continuous information dissemination from an upstream activity, a different estimate can be obtained if we allow that some change incurs more reworks than others. In addition, we do not consider an interdependent overlap [16], which deals with the case where any of two activities affect the other, or a multiple overlap [16], which represents the overlapping relation involving three or more activities.

Third, in spite of our best effort, we were able to find only a small number of companies willing to share their PD processes. We find it quite common that manufacturing firms are reluctant to discuss their experiences in the PD process. Even for these companies who have participated in our study, no systematic database is available for their past PDs. Beside obvious security reasons, many companies have not collected the data related to their PD process; therefore, they have not yet realized that a systematic analysis of the past data can be useful for managing future projects. A majority of our data was collected through extensive interviews with engineers and managers, and the process of collecting data was time consuming.

Finally, we note that our framework is not exhaustive; therefore, there might be some relationships for which our seven patterns cannot model. We are currently working to extend our models to a more general setting.

VIII. CONCLUSION

This study is motivated by the difficulty of estimating the cycle time of a complex PD process. Our primary goal is to provide an analytical framework that can be used to approximate the expected duration of the PD project. We have proposed a simple but effective analytical model of seven PD patterns, while capturing key characteristics of each pattern. Using a divide-and-conquer approach, we present an approach to estimate the cycle time of each pattern, as well as a whole process. As demonstrated in the case study, our approach gives a simple but good estimate for the cycle time of a complex PD process. We note that our approach can be automated if a company has a systematic database that records necessary data from the PD process. For the companies that are contemplating an upgrade to the information system, our framework provides one more reason to do so. Although a further study must follow, our work provides a much-needed first step in estimating the cycle time of a PD process.

This study can be extended in multiple directions. We are currently working to extend our model to include additional patterns, which may appear in a more complex PD process. One can also extend the overlap model by considering several situations such as interdependent overlap and multiple overlap. One can also define other type of sensitivity function or use a different model to describe how information changes at an upstream activity affect the rework occurrences at a downstream activity. Furthermore, one can develop a decision-making model that determines the optimal resource allocation in a PD process with limited resource. In addition, one can explore an underlying stochastic model associated with the PD process and use it to provide a more accurate estimate of the cycle time.

REFERENCES


Hong-Bae Jun received the B.A. degree in applied statistics from Yonsei University, Seoul, Korea, in 1995, and the M.S. and Ph.D. degrees in industrial engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, in 1997 and 2003, respectively.

He is now working as an Assistant Researcher in the Laboratory for Computer-Aided Design and Production, Ecole Polytechnique Federale de Lausanne (EPFL), Lausanne, Switzerland. At present, he is involved in the product lifecycle management and information tracking using smart embedded systems (PROMISE) project (EU and IMS project). Before joining EPFL in 2004, he worked for the Manufacturing Consulting Division, Samsung SDS for one year (2003). His current research interests are product lifecycle management (PLM), lifecycle information modeling, product recovery/EOL management, and ID/RFID technologies.

Hyungmoo Ahn received the B.S.E. degree in industrial engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, and the M.S.E. and Ph.D. degrees from the Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor.

He is an Assistant Professor at the Department of Operations and Management Science, Ross School of Business, University of Michigan. His publications and research interests are in the areas of supply chain management, stochastic control, workforce flexibility, and queueing.

Hyun-Soo Ahn received the B.S.E. degree in industrial engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, and the M.S.E. and Ph.D. degrees from the Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor.

He is an Assistant Professor at the Department of Operations and Management Science, Ross School of Business, University of Michigan. His publications and research interests are in the areas of supply chain management, stochastic control, workforce flexibility, and queueing.

Hyo-Won Suh received the B.S. degree in mechanical engineering from Yonsei University, Seoul, Korea, the M.S. degree in mechanical engineering from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, Korea, and the Ph.D. degree in industrial engineering from West Virginia University, Morgantown.

He is an Associate Professor at the Department of Industrial Engineering, KAIST. His research interests include collaborative engineering, business process management, product data management, and ID/RFID technologies.