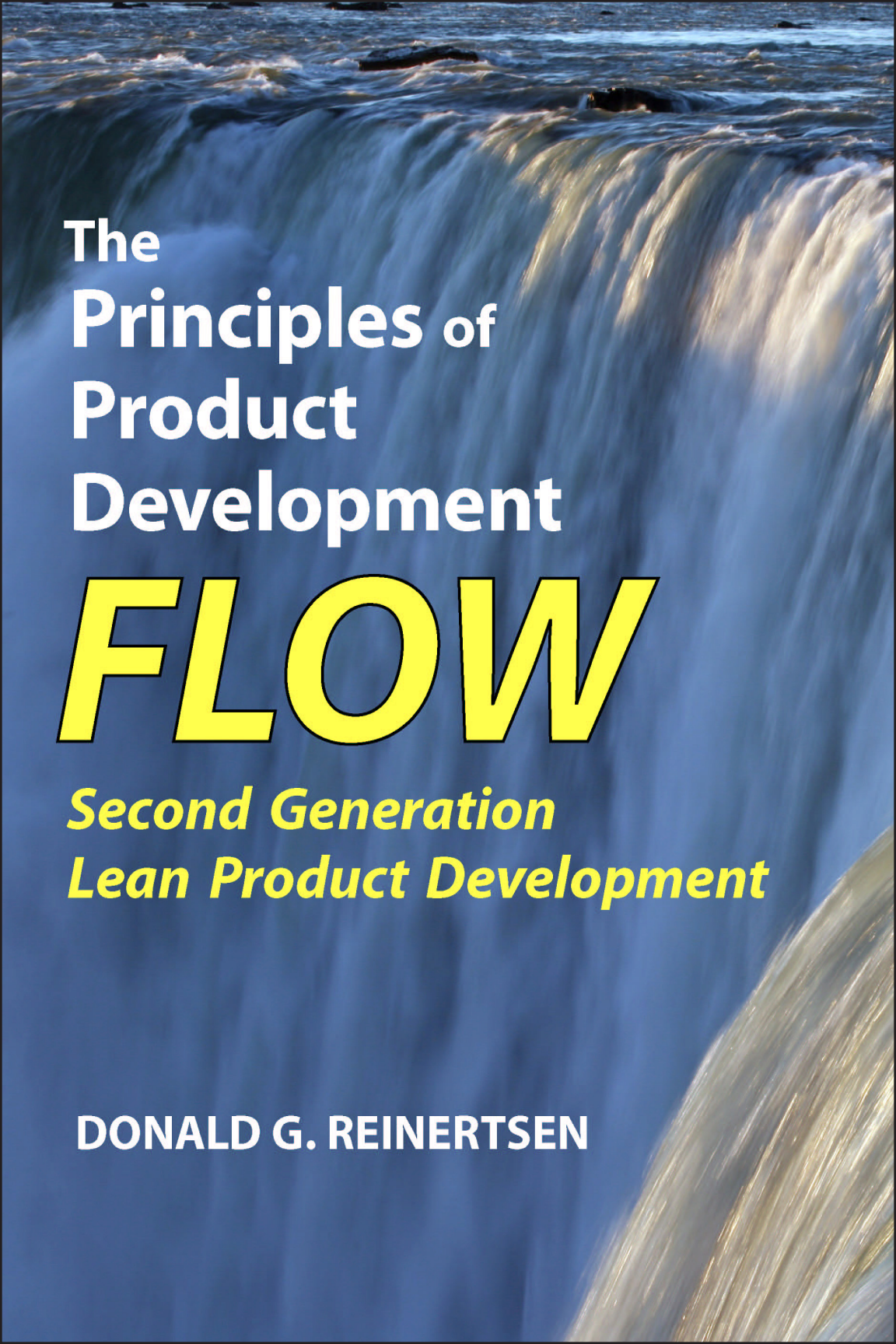
The
Principles of
Product
Development
FLOW
Second Generation
Lean Product Development
DONALD G. REINERTSEN

This PDF file contains the Table of Contents and Chapter 1 from the book, ***The Principles of Product Development Flow: Second Generation Lean Product Development***.

For further information you may contact us as:

info@CeleritasPublishing.com

Celeritas Publishing
Redondo Beach, CA 90277
www.CeleritasPublishing.com

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Library of Congress Control Number: 2008911906

Publisher's Cataloging-in-Publication Data

Reinertsen, Donald G., 1950–
The principles of product development flow: second generation lean product development/
Donald G. Reinertsen

p. cm.
Includes bibliographical references and index.
ISBN-13: 978-1-935401-00-1
ISBN-10: 1-935401-00-9
1. New products. 2. Product management. I. Title.

Contents

1

The Principles of Flow

It ain't what you don't know that gets you into trouble.
It's what you know for sure that just ain't so.
—Mark Twain

Why write this book? After about 30 years of advising product development organizations, I've been lucky enough to work with many smart and experienced developers. I've had a chance to see what they do, and what outcomes they get. At the same time, I've been exposed to all the standard advice about how product development should be managed. Curiously, what actually works is surprisingly different from the standard advice. Why?

I believe that the dominant paradigm for managing product development is fundamentally wrong. Not just a little wrong, but wrong to its very core. It is as wrong as we were in manufacturing, before the Japanese unlocked the secret of lean manufacturing. I believe that a new paradigm is emerging, one that challenges the current orthodoxy of product development. I want to help accelerate the adoption of this new approach. I believe I can do this by helping people understand it. That is the purpose of this book.

The approach I will describe has always been present in some form in development processes. After all, product developers repeat behaviors that work, even if, in theory, they are not supposed to work. But discovering and repeating successful behaviors is not enough to create large-scale use. Product developers need to extract the underlying mechanisms of action that make these methods work. It is only by generalizing these principles that we can expand their application from a handful of isolated successes into a general approach that transforms the entire development process.

Let's begin with an illustration. Officially, many product developers have phase-gate processes. Work is divided into mutually exclusive

phases separated by gates. One phase must be complete before the next one can begin. For example, such processes typically require that all product requirements be defined before beginning design activities. The team appears to do this, and delivers complete product requirements to management at the gate review. Then, they are given approval to exit the requirement definition phase and to begin the product design phase. On its surface, this procedure appears quite sensible, and it seems to work.

What really happens is quite different. When I survey managers in my product development courses, 95 percent will admit that they begin design before they know all requirements. In fact, the average product developer begins design when 50 percent of requirements are known. They simply do not publicize this to management. Instead, they engage in a time-honored ritual of asking for permission to proceed. There is a tacit "don't ask, don't tell" policy. Managers politely avoid asking if the next phase's activities have already begun, and developers discreetly avoid mentioning that they have already moved on. In practice, sensible behavior prevails, despite the presence of a dysfunctional formal procedure.

The underground overlap of design and specification activities is just one simple example of this alternative paradigm in action. Among other things, this paradigm emphasizes small batch transfers, rapid feedback, and limited work-in-process inventory (WIP). These three specific methods are actually broadly applicable throughout the development process. When we recognize why they work, we can adapt them to dozens of situations.

At its heart, this new paradigm emphasizes achieving flow. It bears many similarities to the methods of lean manufacturing and could be labeled lean product development (LPD). However, the methods of lean manufacturing have been optimized for a domain with very different characteristics than product development.

For example, manufacturing deals with predictable and repetitive tasks, homogeneous delay costs, and homogeneous task durations. As a result, manufacturing sequences work in a simple first-in-first-out (FIFO) order. FIFO prioritization is almost never economically optimal in product development, because product development deals with high variability, nonrepetitive, nonhomogeneous flows. Different projects almost always have different delay costs, and they present different loads on our resources.

As you shall see, these new challenges force us to go far beyond the ideas of lean manufacturing. If we are open to advanced ideas from other domains, we can do this. To distinguish this more advanced approach from the ideas of lean manufacturing, we call it Flow-Based Product Development.

What Is the Problem?

Today's orthodoxy has institutionalized a set of internally consistent but dysfunctional beliefs. This has created a tightly interlocking and self-reinforcing system, a system from which it is very difficult to break free. Even when we change one piece, the other pieces hold us back by blocking the benefits of our change. When our change fails to produce benefits, we revert to our old approaches.

Let's look at how our current beliefs reinforce each other. For example, if we combine the belief that efficiency is good, with a blindness to queues, we will load our development process to high levels of capacity utilization. After all, underutilized capacity appears to be waste. High capacity utilization may cause queues, but these queues have no apparent cost. Thus, these two beliefs combine to drive us to disastrous levels of capacity utilization. This, in turn, leads to large queues and long cycle times.

Let's take another example. If we combine the belief that variability is bad with the failure to correctly quantify economics, we will minimize variability by sacrificing other unquantified economic goals. We will create risk-averse development processes that strive to "do it right the first time." We will blindly embrace concepts like Six Sigma product development.

This risk aversion will drive innovation out of our development process. Eventually, we will wonder why our products have no differentiation and low profitability. Thus, our current belief system incorrectly causes us to choose variability reduction over profitability. Yet, it deludes us into thinking that we are increasing profits.

In this chapter, I want to challenge some of the core beliefs of the current orthodoxy and to highlight what is wrong with them. Of course, this is only a brief introduction to these controversial ideas. The orthodox beliefs are quite well-defended, so we will need to assault them with heavier weapons later in this book. For now, I will just fire a few

Problems with the Current Orthodoxy

1. **Failure to Correctly Quantify Economics**
2. **Blindness to Queues**
3. **Worship of Efficiency**
4. **Hostility to Variability**
5. **Worship of Conformance**
6. **Institutionalization of Large Batch Sizes**
7. **Underutilization of Cadence**
8. **Managing Timelines instead of Queues**
9. **Absence of WIP Constraints**
10. **Inflexibility**
11. **Noneconomic Flow Control**
12. **Centralized Control**

Figure 1-1 Twelve critical problems with the current product development orthodoxy.

tracer rounds so you can see where the targets lie. I only request that you be open to the possibility that the current orthodoxy is deeply dysfunctional. Let's pick twelve key problems with the current approach, as shown in Figure 1-1.

Failure to Correctly Quantify Economics

Perhaps the single most important weakness of the current orthodoxy is its failure to correctly quantify economics. The key word is *correctly*. When done correctly, an economic framework will shine a bright light into all the dark corners of product development. Today, we only light up part of the landscape. Then, we focus on these well-lit details. We are like the drunk under the lamppost looking for his keys where the light is best, instead of where he dropped them.

This criticism may surprise product developers since most are convinced they correctly quantify economics today. Yet, it is exactly this conviction that blocks them from making progress. For example, the only way to comprehend the economic importance of queues is to quantify the cost of delay for product development projects. Yet, today, only 15 percent of product developers know the cost of delay associated with their projects.

Today, developers focus on what we call proxy variables. A proxy variable is a quantified measure that substitutes for the real economic objective: life-cycle profits. For example, developers measure development cycle time and seek to make it shorter. However, when you ask them how much life-cycle profit will decrease due to a week of delay, they don't know.[1] Cycle time is only a proxy variable.

Developers measure the percent value-added time and seek to make it higher. But, if you ask them how much life-cycle profit will improve when they do this, they don't know. Percent value-added time is only a proxy variable. By focusing on proxy variables, product developers delude themselves into thinking they understand their economics. They do not.

The danger in focusing on proxy variables is that proxy variables interact with one another. Raising capacity utilization has the beneficial effect of improving efficiency and the negative effect of increasing cycle time. To understand the net impact, we must combine both effects. To do this, we must express changes in all proxy variables in the same unit of measure. This unit of measure should be life-cycle profit impact, which is the ultimate measure of product development success.

It is only when we understand the mapping between proxy variables and life-cycle profits that we can really see the economic consequences of our choices. I will show you how an economic framework helps us do this in Chapter 2.

Blindness to Queues

Few developers realize that queues are the single most important cause of poor product development performance. Queues cause our development process to have too much design-in-process inventory (DIP). Developers are unaware of DIP, they do not measure it, and they do not manage it. They do not even realize that DIP is a problem. For example, product developers will say that they want to be more innovative and never realize that high levels of DIP undermine this objective. When DIP is high, cycle times are long. When cycle times are long, innovation occurs so late that it becomes imitation.

---

[1] More precisely, companies will give answers to this question, but these answers have such high variance that they simply represent quantified ignorance. On average, at companies that do not calculate cost of delay, people working on the same project will give answers that vary by 50 to 1.

There are two important reasons why product developers are blind to DIP. First, inventory is financially invisible in product development. We do not carry partially completed designs as assets on our balance sheet; we expense R&D costs as they are incurred. If we ask the chief financial officer how much inventory we have in product development, the answer will be, "Zero."

Second, we are blind to product development inventory because it is usually physically invisible. DIP is information, not physical objects. We do not see piles of DIP when we walk through the engineering department. In product development, our inventory is bits on a disk drive, and we have very big disk drives in product development.

But if we are blind to DIP, then we are also blind to the danger in operating with high levels of DIP. As you shall see in Chapter 3, queues lie at the root of a large number of product development problems. They increase variability, risk, and cycle time. They decrease efficiency, quality, and motivation.

To understand the economic cost of queues, product developers must be able to answer two questions. First, how big are our queues? Today, only 2 percent of product developers measure queues. Second, what is the cost of these queues? To answer this second question, we must determine how queue size translates into delay cost, which requires knowing the cost of delay. Today, only 15 percent of product developers know their cost of delay. Since such few companies can answer both questions, it should be no surprise that queues are managed poorly today.

I must stress that the ultimate root cause of this queue blindness lies in the failure to correctly quantify our economics. We can learn to see whatever we consider to be important. Once we quantify the cost of delay, we become aware of the cost of queues. Once we recognize the cost of queues, we are motivated to measure and manage them. Without a cost of delay, queues appear to be free and therefore, unworthy of attention.

Worship of Efficiency

But, what do product developers pay attention to? Today's developers incorrectly try to maximize efficiency. This is a natural consequence of queue blindness and the failure to correctly quantify economics.

Any subprocess within product development can be viewed in economic terms. The total cost of the subprocess is composed of its cost of capacity and the delay cost associated with its cycle time.[2] If we are blind to queues, we won't know the delay cost, and we will only be aware of the cost of capacity. Then, if we seek to minimize total cost, we will only focus on the portion we can see, the efficient use of capacity.

This explains why today's product developers assume that efficiency is desirable, and that inefficiency is an undesirable form of waste. This leads them to load their processes to dangerously high levels of utilization. How high? Executives coming to my product development classes report operating at 98.5 percent utilization in the precourse surveys. What will this do? Chapter 3 will explain why large queues form when processes with variability are operated at high levels of capacity utilization. In reality, the misguided pursuit of efficiency creates enormous costs in the unmeasured, invisible portion of the product development process, its queues.

What is the root cause of this problem? Once again, it is the failure to correctly quantify economics. Although efficiency does have economic impact, it is only another proxy variable. We must make decisions based on overall economics. Since high capacity utilization simultaneously raises efficiency and increases delay cost, we need to look at the combined impact of these two factors. We can only do so if we express both factors in the same unit of measure, life-cycle profits. If we do this, we will always conclude that operating a product development process near full utilization is an economic disaster.

Hostility to Variability

The absence of clear economic thinking also leads to a profound misunderstanding of the role of variability in product development. Today, most product developers assume that variability is bad, and they try to eliminate this variability. This emphasis on variability reduction is deeply flawed for three important reasons.

First, without variability, we cannot innovate. Product development produces the recipes for products, not the products themselves.

---

[2] For introductory purposes, we will omit a third cost which is associated with the performance of the subprocess. Later in this book, we will use a more complete economic framework.

If a design does not change, there can be no value-added. But, when we change a design, we introduce uncertainty and variability in outcomes. We cannot eliminate all variability without eliminating all value-added.

Second, variability is only a proxy variable. We are actually interested in influencing the economic cost of this variability. As you shall see in Chapter 4, this economic cost depends on how variability is transformed by an economic payoff-function. It is critical to pay attention to both the amount of variability and the nature of this payoff-function. When we start viewing variability this way, we usually discover it is much easier to alter the payoff-function than it is to reduce the underlying amount of variability.

Third, with a little help from option pricing theory, we will discover that we can actually design development processes such that increases in variability will *improve*, rather than *worsen*, our economic performance.

Unfortunately, today's product development processes are permeated with methods and behaviors that uncritically accept the idea that variability is bad. Developers build buffers into their schedules to reduce variability. They adopt Six Sigma programs to reduce variability. They seek to create repeatable processes to reduce variability.

My surveys of product developers show that 65 percent of product developers consider it desirable to eliminate *as much variability as possible* in product development. Variability is viewed as the enemy. This view is completely disconnected from any deep understanding of product development economics. Minimizing the economic impact of variability is a profoundly different goal than minimizing variability.

Worship of Conformance

In addition to deeply misunderstanding variability, today's product developers have deep-rooted misconceptions on how to react to this variability. They believe that they should always strive to make actual performance conform to the original plan. They assume that the benefit of correcting a deviation from the plan will always exceed the cost of doing so. This places completely unwarranted trust in the original plan, and it blocks companies from exploiting emergent opportunities. Such behavior makes no economic sense.

We live in an uncertain world. We must recognize that our original plan was based on noisy data, viewed from a long time-horizon. For example, we may have started development believing a feature would take 1 week of effort and it would be valued by 50 percent of our customers. As we progressed through development, we may have discovered that this feature will require 10 weeks of effort and it will only be valued by 5 percent of our customers. This is a factor of 100 change in its cost-to-benefit ratio. This emergent information completely changes the economics of our original choice. In such cases, blindly insisting on conformance to the original plan destroys economic value.

To manage product development effectively, we must recognize that valuable new information is constantly arriving throughout the development cycle. Rather than remaining frozen in time, locked to our original plan, we must learn to make good economic choices using this emerging information.

Conformance to the original plan has become another obstacle blocking our ability to make good economic choices. Once again, we have a case of a proxy variable, conformance, obscuring the real issue, which is making good economic decisions.

Institutionalization of Large Batch Sizes

Our blindness toward queues and our focus on efficiency lead us to institutionalize large batch sizes. Large batches seem attractive because they appear to have scale economies that increase efficiency. Furthermore, large batches appear to reduce variability because they pool many activities together. However, this efficiency gain and variability reduction is only an illusion. As you shall see later, in Chapter 5, there can be strong diseconomies associated with large batches. Furthermore, the modest reduction in variability due to the pooling of variances will be completely overwhelmed by the geometric increase in uncertainty caused by the longer planning horizons associated with large batches.

Today's developers are blind to the issue of batch size. They do not measure batch size or try to reduce it. They fail to recognize both the critical relation between batch size and cycle time, and the critical relation between batch size and feedback speed. My surveys of product developers show that only 3 percent of them are trying to reduce batch sizes in their development processes.

Today's orthodoxy encourages maximum possible batch sizes. For example, the majority of product developers use phase-gate processes, which transfer 100 percent of the work product in one large batch to the next phase. We cannot make batch size any higher than 100 percent.

We will devote Chapter 5 to understanding the surprising benefits of smaller batch sizes and the methods we can use to achieve them. In particular, we will examine how we can enable small batch processes by systematically lowering transaction costs.

It should come as no surprise that since the current orthodoxy ignores the effects of large batch size, it also places no emphasis on lowering transaction costs. In fact, only 3 percent of developers have a formal program to reduce transaction costs.

Underutilization of Cadence

Today's development processes typically deliver information asynchronously in large batches. A flow-based process delivers information on a regular cadence in small batches. In fact, cadence helps lower transaction costs and makes small batches more economically feasible.

Let's look at a simple example. In many companies, manufacturing wants to receive a complete design from engineering. They tell engineering that they don't want to see any engineering drawings until engineering has stopped changing the design. Once all drawings are ready, they are reviewed in one large batch. All 200 drawings may be reviewed on a single day. Companies assume this makes reviewers more efficient because they can see the entire design at once.

But these 200 drawings were not completed in a single day. They may have been completed over a 10-week period. The first 20 drawings to be completed may have waited 9 weeks to be reviewed.

What happened during these 9 weeks? If an engineer makes a bad assumption in one of the first 20 drawings, then that bad assumption will go unchallenged until the final review. Without feedback from a review, this bad assumption can be incorporated into 180 more drawings. By breaking drawing review into small batches, we improve quality, efficiency, and cycle time.

But how do we prevent all these small review meetings from driving up overhead? We conduct these review meetings on a regular time-based cadence. Every Wednesday afternoon at 1:00 PM, we review all the drawings completed in the last week. There is no need for a meeting

announcement and no need to coordinate schedules. Meetings that are synchronized to a regular and predictable cadence have very low set-up costs. They contribute very little excess overhead.

In Chapter 6, we will see why cadence and synchronization are surprisingly powerful tools for product development. We will discover how cadence helps to control the progressive accumulation of variability in development processes. Today's orthodoxy constrains the scope of work and drives variability into timing. The new paradigm constrains timing and drives variability into the scope of work. These two approaches are fundamentally different.

Managing Timelines instead of Queues

We have already pointed out that companies do not manage product development queues. What do they manage? Timelines. Today's orthodoxy emphasizes managing timelines, rather than managing process queues. We train our project managers how to create these timelines and how to manage them. In fact, this was the same mistake we made in manufacturing before the arrival of the Toyota Production System (TPS). We created detailed timelines using our manufacturing planning systems. The more detailed we made our plans, the longer our cycle times became.

We favor highly granular planning because we don't understand the statistics of variability. Misunderstanding variability is dangerous in the repetitive world of manufacturing, but is it even more dangerous in product development where variability is much higher.

Most product developers do not understand the statistics of granular schedules. A granular timeline subdivides time intervals into very small buckets. When we do this, the coefficient of variation for each of these buckets becomes very high. This makes variance very high and conformance unlikely. Even worse, if we incentivize conformance, people will insert contingency reserves to prevent their tasks from missing the schedule. The more granular the schedule, the larger the schedule reserves. And these reserves aggregate into even longer timelines. The more we increase planning detail and the harder we try to incentivize performance, the worse our problem becomes.

In contrast, when we emphasize flow, we focus on queues rather than timelines. Queues are a far better control variable than cycle time because, as you shall see, queues are leading indicators of future

cycle-time problems. By controlling queue size, we automatically achieve control over timelines.

Our blindness to queues and our misunderstanding of variability combine to hide this opportunity. The current orthodoxy focuses on planning and managing timelines, instead of the more powerful approach of managing queues.

Absence of WIP Constraints

One of the most powerful ways to manage queues is to use WIP constraints. This technique is virtually absent in today's development processes. Only 3 percent of developers use WIP constraints. In contrast, this method is widespread in modern manufacturing. One of the most prominent examples is the kanban system used by Toyota.

In Chapter 7, with the help of some queueing theory, we will examine the power of WIP constraints and their effect on cycle time. WIP constraints are even more important in product development than manufacturing, because product development has higher inherent variability.

If we limit ourselves to the relatively simple WIP constraints used in lean manufacturing, we will underexploit the power of WIP constraints. Instead, we will explore some of the more advanced ideas used in the world of telecommunications.

If we showed the Toyota Production System to an Internet protocol engineer, it is likely that he would remark, "That is a tail-dropping FIFO queue. That is what we started with on the Internet 30 years ago. We are now four generations beyond that method." I mention this to encourage you to look beyond the manufacturing domain for approaches to control flow.

As mentioned earlier, manufacturing has an advanced view on how to achieve flow when four conditions are present: predictable and repetitive tasks, homogeneous delay costs, and homogeneous task durations. In product development, these four conditions are virtually never present.

In Chapter 7, I will expose you to the more advanced ideas that come from other domains. What is critically important is that these other domains do not achieve flow by eliminating variability; they have developed methods that achieve flow in the presence of high variability. As a result, these methods are extremely relevant to product developers.

Inflexibility

In pursuit of efficiency, product developers use specialized resources loaded to high levels of utilization. Our current orthodoxy accepts inflexibility in return for efficiency. But what happens when this inflexibility encounters variability? We get delays. So what do we do? The current orthodoxy suggests that we focus on the variability. We can reduce this variability directly, or we can build buffers and reserves to mask the variability.

I have already explained why variability reduction is not the answer. Later in this book, you will see that buffers and reserves are also a dangerous approach. They pay for variability reduction with cycle time, which is a very expensive parameter to trade for variability reduction.

Flow-based Product Development suggests that our development processes can be both efficient and responsive in the presence of variability. To do this, we must make resources, people, and processes flexible. We can learn some lessons on how to do this from our factories, and even more lessons from telecommunications networks. In our factories, we create flexibility by paying more to workers who can work at more stations on a production line. We value flexibility, and we pay for it. In contrast, most product development organizations exclusively reward specialization.

In telecommunications networks, we use adaptive approaches that enable us to adjust to emerging congestion in less than a second. In product development, we have not yet recognized the tremendous value of flexibility as an explicit strategy for exploiting the economic benefits of variability. Instead, we still try to eliminate or hide the variability.

Noneconomic Flow Control

There are many systems used to control flow in today's development processes, but sadly, none of them is based on economics. Some companies follow the approach of manufacturers, where jobs are handled in a FIFO sequence. This makes sense in manufacturing, because jobs are typically homogeneous. They all place the same time demands on resources, and they have the same cost of delay. These two conditions are virtually never present in product development.

Other companies prioritize on the basis of project profitability measures like return on investment (ROI). On the surface, this appears

to be an economic approach, but this is just an illusion. By prioritizing, we choose to service one project before another. In general, it is best to delay the project with a low cost of delay. This suggests that we should not prioritize on the basis of project profitability, but rather on how this profitability is affected by delay. Of course, this can only be done when we know the cost of delay, information that 85 percent of developers do not have.

Still, other companies have adopted more complex methods, like Goldratt's Critical Chain concept. This approach provides buffers to each project and gives priority to the project with the smallest remaining buffer. In classic scheduling theory, this method is called minimum slack time first. It can encourage you to apply resources to a low cost-of-delay job with a depleted buffer in preference to a high cost-of-delay job that is ahead of schedule. This is the opposite of what economics would suggest.

Today, there is no current orthodoxy for flow control, unless we consider ignoring economics a method. Again, we can trace this problem back to lack of economic understanding. A good economic framework resolves many scheduling dilemmas.

Centralized Control

Finally, today's product development orthodoxy tends to use centralized control. We create project management offices. We create centralized information systems. We centralize buffers to take advantage of the efficiency gains of this approach. Our focus on centralization arises because we value efficiency more than response time. We shall see that this is a mistake in Chapter 8, when we examine the role of fast feedback.

Product developers often avoid decentralized control because they are afraid decentralization will lead to chaos. I hope to offer a different view of decentralization. One of the most interesting examples of decentralizing control without losing alignment is the way the military deals with the uncertainty of warfare. We will examine their approach in Chapter 9 and consider how it can be applied in product development.

By this point, I hope I have convinced you to look at the current orthodoxy more critically. It has serious problems, and there are logical ways we can resolve them. The first step, as in any change, is to admit that things must be changed.

Major Themes

1. **Economics**
2. **Queues**
3. **Variability**
4. **Batch Size**
5. **WIP Constraints**
6. **Cadence, Synchronization, and Flow Control**
7. **Fast Feedback**
8. **Decentralized Control**

Figure 1-2 The eight major themes of this book.

A Possible Solution

The major chapters of this book will focus on how we can overcome the limitations of the current orthodoxy. We will explore eight major themes, shown in Figure 1-2. These themes are mutually supportive, and they lie at the core of Flow-based Product Development.

Economics

The first key theme is economically-based decision making. It is the ultimate foundation of this book. Our central premise is that we do product development to make money. This economic goal permits us to use economic thinking and allows us to see many issues with a fresh point of view. It illuminates the grave problems with the current orthodoxy.

The current orthodoxy does not focus on understanding deeper economic relationships. Instead, it is, at best, based on observing correlations between pairs of proxy variables. For example, it observes that late design changes have higher costs than early design changes and prescribes front-loading problem solving. This ignores the fact that late changes can also create enormous economic value. The economic effect of a late change can only be evaluated by considering its complete economic impact.

The current orthodoxy uses observation, a powerful tool, but it ignores the bias inherent in all methods based on observation. When we focus on observable phenomena, we can be biased by the fact that some characteristics are more observable than others. This increased visibility biases our findings.

For example, when we benchmark other companies, certain aspects of the development process are very visible. Observers quickly conclude that these visible characteristics are the root cause of superior performance. It is easy to see that the development team wears red hats. It is hard to see the control rule they use for managing DIP. As a result, we tend to focus on the easily visible characteristics that are often the least important drivers of development process performance.

Let me make a simple analogy. Suppose you want to learn how to design a radio. You find a well-designed radio and are allowed to carefully inspect it. You would see certain components, such as resistors, capacitors, transistors, and integrated circuits. If you duplicated exactly what you saw, you could create a functioning radio, but you would still know virtually nothing about radio design.

The essence of radio design is the logical process by which the components were selected. In fact, what you can observe visually tells you little about the way a radio functions. For this, you must understand the signals flowing through the system. Unfortunately, these signals are completely invisible. You may be quite convinced that you have carefully inspected the radio, but the way it works and why it works will remain a mystery. This is also true in product development. Critical process choices with great influence on economics, such as inventory levels and feedback loop design, are virtually invisible.

Queues

The second major theme is the importance of queues. Even a basic understanding of queueing theory will dramatically change your perspective on product development. Fortunately, queues have been studied by very smart people for almost a century. Unfortunately, product developers have a poor quantitative intuition regarding the behavior of queues. They are also unaware of the many problems caused by queues. We must get a deeper sense of how queues affect development processes and what can be done about them.

Our approach in this book is to give you industrial-strength queueing theory, rather than a more simplified version. This differs from the more popular versions such as Goldratt's Theory of Constraints (TOC) in two important ways. First, we are interested in the intersection of economics and queueing theory. TOC suggests that bottlenecks should not run out of work. A careful economic analysis reveals that this is not true. Second, we will treat queueing as the statistical science that it is, which also leads us to some different conclusions. For example, TOC suggests that we cannot reduce cycle time without increasing the capacity of bottlenecks. Industrial-strength queueing theory reaches a different conclusion.

TOC deserves respect because it has been an extraordinarily useful tool for making people aware of queues. However, the time has come to go a bit deeper. The issue is not queues, it is the economics of queues. Rather than saying queues are universally bad, we must treat them as having a quantifiable cost. This allows us to compare both the benefits and the costs of an intervention to reduce queue size.

Variability

The third major theme is variability. I want to present a radically different perspective on the role of variability and how it can be managed. In this case, we are interested in the intersection of economics and the science of statistics. In particular, we will focus on how variability is transformed into economic consequences by an economic payoff-function. Once you realize the critical role of this payoff-function, you will recognize it can be a powerful tool to improve economic performance.

This section of the book will challenge many of your existing beliefs about variability. It will show you methods for reducing variability and for reducing the cost of variability. Some of these methods, like variability pooling, are based on statistical theory. Other methods, such as variability substitution, are based on economic theory. We need to go far beyond the current belief that variability is always bad and that it should always be reduced.

Batch Size

The fourth key theme is the importance of reducing batch size. By recognizing that queues are the problem, we are led to examine methods for reducing queues. We will discover that reducing batch size is usually

the single most cost-effective way to reduce queues. Smaller batches do this by reducing unnecessary variability in flow. Once again, we are interested in the intersection between batch size and economics. In this case, batch sizes have their own economic tradeoffs, which lead to U-curve optimization problems.

Batch size reduction is a powerful technique that works for both predictable and unpredictable flows. In manufacturing, where flows are quite predictable, we strive for one-piece flow, which is minimum theoretical batch size. In packet-switching networks, where flows can be completely unpredictable, we manage highly variable flows by breaking them into small packets. However, such networks virtually never use minimum theoretical batch size because there are technical and economic tradeoffs that make this suboptimal. We can learn a lot from both domains.

WIP Constraints

The fifth theme is the power of WIP constraints. WIP constraints are a powerful way to gain control over cycle time in the presence of variability. This is particularly important for systems where variability accumulates, such as in product development. WIP constraints exploit the direct relationship between cycle time and inventory, which is known as Little's Formula.

We will discuss two common methods of constraining WIP: the kanban system and Goldratt's Theory of Constraints (TOC). These methods are relatively static. We will also examine how telecommunications networks use WIP constraints in a much more dynamic way. Once again, telecommunications networks are interesting to us as product developers, because they deal successfully with inherently high variability.

Cadence, Synchronization, and Flow Control

Our sixth theme is the use of cadence, synchronization, and dynamic flow control. Cadence involves processing work at regular time intervals. Synchronization involves making multiple tasks start or finish at the same time. We can synchronize with or without cadence. We can use cadence with or without synchronization. It is best when we combine both.

By flow control, we refer to the sequence in which we process jobs in our product development process. We are interested in finding

economically optimum sequences for tasks. Current practices use fairly crude approaches to sequencing.

For example, it is suggested that if subsystem B depends on subsystem A, it would be better to sequence the design of A first. This logic optimizes efficiency as a proxy variable. When we consider overall economics, as we do in this book, we often reach different conclusions. For example, it may be better to develop both A and B simultaneously, despite the risk of inefficient rework, because parallel development can save cycle time.

In this book, our model for flow control will not be manufacturing systems, since these systems primarily deal with predictable and homogeneous flows. Instead, we will look at lessons that can be learned from telecommunications networks and computer operating systems. Both of these domains have decades of experience dealing with nonhomogeneous and highly variable flows.

Fast Feedback

Our seventh theme is the importance of fast feedback loops. The speed of feedback is at least two orders of magnitude more important to product developers than manufacturers. Developers rely on feedback to influence subsequent choices. Or, at least, they should. Unfortunately, our current orthodoxy views feedback as an element of an undesirable rework loop. It asserts that we should prevent the need for rework by having engineers design things right the first time.

We will present a radically different view, suggesting that feedback is what permits us to operate our product development process effectively in a very noisy environment. Feedback allows us to efficiently adapt to unpredictability.

Feedback gives us new information, information that can be used to make better economic choices. Let's take a simple example, shown in Figure 1-3. I will offer you two ways to invest in a two-digit lottery ticket that pays out $100 for a winning number. Using the first method, you can buy a random two-digit number for $1. In such a case, you would have a 99 percent chance of losing $1 and a 1 percent chance of making $99. This bet produces no net gain.

Using the second method, you can buy the first digit for 50 cents, receive feedback as to whether this digit is correct, and then, if it is correct, you can choose to buy the second digit for 50 cents. The economics

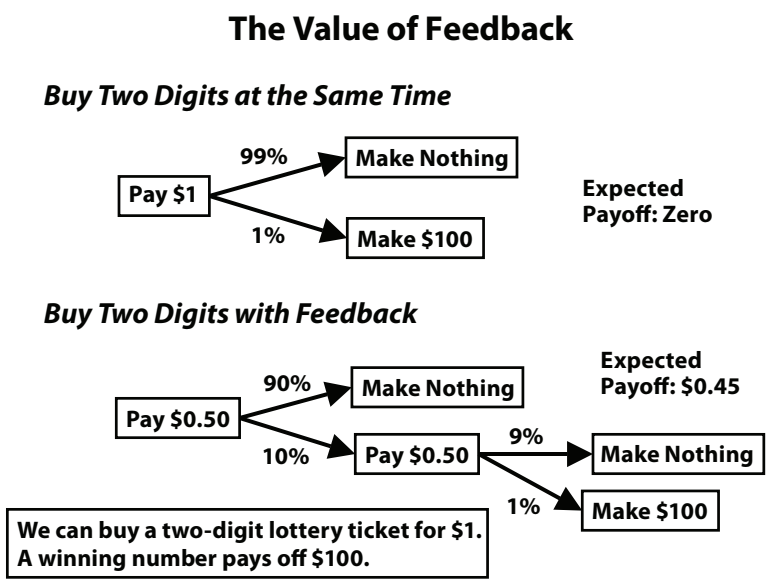

Figure 1-3 In this lottery ticket example, the expected payoff is higher if we buy one digit and obtain feedback, than if we buy both digits at the same time. The feedback stops us from buying the second digit 90 percent of the time, since we already know we cannot win.

of the second method is clearly better. You have a 90 percent chance of losing 50 cents on the first digit. You will only go on to invest in the second digit if you picked the first digit correctly. This occurs only 10 percent of the time. Ninety percent of the time you will save the cost of buying the second digit. When you buy the second digit, you have a 9 percent chance of losing $1 and a 1 percent chance of making $99.

By breaking the betting process into two steps, the overall bet produces a net gain of 45 cents. This added value occurs because 90 percent of the time, you will not purchase the second digit, because you already know you will not win.

When we start to view feedback through the perspective of economics and probability, we will discover its power to improve product development economics.

We will also look at feedback from a control engineering perspective. This allows us to consider problems of dynamic response and stability that are not considered in many traditional approaches.

Finally, this theme of fast feedback will include a discussion of metrics. The current orthodoxy uses a set of metrics that is based on a specific belief system about what causes economic success. The new paradigm has different beliefs about what causes economic success, so it tries to control different things.

Decentralized Control

Our final theme is decentralization of control. Decentralization is a logical response to unpredictable situations where response time is important. This occurs in product development, where problems can expand quickly and opportunities can be perishable.

Counterbalancing the response-time benefits of decentralization are the efficiency benefits of centralized control. Centralized control creates scale economies and more efficient alignment of our efforts.

Rather than arguing that we must eliminate all centralized control and create completely decentralized control, we will take a more balanced view. We will use the military as a role model. The military confronts fleeting opportunities and unexpected obstacles in warfare, a domain of high uncertainty. It has experimented for centuries with both centralization and decentralization. In its most advanced form, today's military combines decentralized control with alignment. We will explore how they do this and what lessons can be learned by product developers.

The Relevant Idea Sources

As I explore the major themes of this book, it should be clear that many fields have advanced ideas on the problem of managing flow in the presence of variability. I believe that many of these ideas have the power to transform product development. The science of flow does not reside tidily in a single location. Important insights come from the world of engineering, manufacturing, mathematics, economics, and military science. I have tried to consolidate those ideas that I find most useful. Let's look at where the key contributions come from:

Lean Manufacturing

Lean manufacturing provides us with a mature set of ideas for managing certain types of flow. We owe Toyota a great debt showing us the importance of using small batches in production processes. Many

of these ideas can be transferred into design processes. In doing so, we must recognize that manufacturing is a domain with fundamentally different economic characteristics. Its ideas must be significantly extended to make them useful in product development.

Economics

Almost all of the problems of the current orthodoxy can be traced back to the failure to apply economic frameworks to product development. An economic framework enables us to quantify the effects of multiple interacting variables. This is the key to making good economic choices. With an understanding of economics, you will learn that popular ideas, like the customer is the ultimate judge of value-added, actually make little economic sense.

Economics will also help us think in a more sophisticated way about variability. Economists would find the notion that minimizing variability maximizes economic value to be hopelessly simplistic.

Queueing Theory

Queueing theory is a field of applied statistics that studies the formation of waiting lines. It is a century old and has benefited from some very careful thinking by applied mathematicians and engineers. Queueing theory permits us to quantify the relationships between waiting times and capacity utilization, even when arrivals and service times are highly variable. We use queueing theory in the design of telecommunications networks, which are themselves a sophisticated and fertile source of ideas.

Statistics

The field of statistics is a rich source of insight into processes with variability. Yet, the ideas of statistics have only been applied in a very limited way in product development. We will use ideas that go beyond those taught in Six Sigma training.

First, we are extremely interested in the intersection of statistics and economics. This intersection enables us to understand the economics of variability.

Second, Six Sigma teaches us about random variables. Random variables help us to understand the problems of repetitive manufacturing. In product development, we are interested in a more advanced

application of statistics called random processes. These are time sequences of random variables. Random processes are much more relevant to product developers, and, as you shall see, often counterintuitive.

The Internet

The Internet is a highly successful application of ideas for managing flow in the presence of variability. Manufacturing networks emphasize variability reduction as the path to improving flow. In contrast, telecommunications networks assume variability is inherent, and adopt design approaches that are robust in the presence of variability. We can learn a great deal about flow control by looking at methods that have evolved in telecommunications networks over the last 100 years. Manufacturing networks only use a small subset of these flow control concepts.

Computer Operating System Design

Computer operating systems have evolved over the last 60 years. They are another rich source of ideas for two reasons. First, they deal with mixed-priority workflows. Different jobs have different priorities. Second, computer operating systems deal with tasks that have unpredictable arrival times and unpredictable durations. As a result, operating systems have developed much more advanced sequencing methods than the simple FIFO sequences used in manufacturing processes. Since product development must also deal with unpredictable and mixed-priority workflows, we can obtain useful approaches from computer operating systems.

Control Engineering

The feedback loops that most readers have been exposed to are the relatively simple feedback loops of the plan-do-check-act (PDCA) cycle. In the field of control engineering, the science of feedback is bit more sophisticated. In particular, this field gives us insights about the importance of dynamic response and the problem of instability. Such issues rarely show up in manufacturing, because manufacturing flow times are measured in hours or days. These issues become much more important in product development where our flow times are measured in months or years. We will draw upon these ideas when we examine the use of feedback loops in product development systems.

Maneuver Warfare

Finally, we will draw upon ideas from the military. Most readers will not be very familiar with military doctrine. What they know will be shaped by ideas received from Hollywood and television. This is unfortunate, because military thinkers actually have very advanced ideas on how to combine decentralized control and alignment of effort.

The organization I have chosen as a model of this approach is the U.S. Marine Corps. Marines believe that they cannot remove uncertainty from warfare, and that this uncertainty does not inherently favor either side. Instead, the side that can thrive in the presence of uncertainty is most likely to win.

Marines view warfare as a domain of unexpected obstacles and fleeting opportunities. They use decentralized control and local initiative to quickly exploit opportunities, but they also recognize that decentralized control can lead to chaos. Their methods are both practical and highly relevant to product developers.

The Design of This Book

Perhaps the most obvious structural feature of this book is that it is organized into 175 principles. It is not a business novel, telling the story of an intrepid manager on his or her journey to success. It is not a collection of company stories reflecting on their successes. It is not a set of rules to be rigidly followed. It is a set of principles. These are patterns and causal relationships that occur often enough to make them useful guidelines. They will help you think more deeply about your choices, but they will not eliminate the need for thinking.

I have chosen to emphasize principles for three reasons. First, I believe that a powerful set of underlying principles can be used to solve an incredibly wide range of problems. I have always admired the field of physics for its ability to distill many phenomena down to a much smaller number of governing principles.

Second, this principle-based approach allows me to increase the density of information in this book. None of us have time to read everything we would like to read. Increasing density helps reduce this problem. I aspire to write books with more content than fluff, and hope this book will achieve that goal.

Third, I hope this principle-based design will make it easier for a wide range of readers to use this book. Experienced developers can skip over ideas they already use, beginners can examine everything. Organizing around principles should also make it easier to retrieve specific information. It offers an alternative to indexes, which are sometimes a bit too detailed, and tables of contents, which are often a bit too general.

I've assumed a moderate degree of technical literacy on the part of the reader. Experience shows that most readers of my previous books have an engineering or scientific background. I have decided to take advantage of this. When working with previous publishers, I heeded their warnings that equations would frighten away readers. Today, I am willing to take the risk of inserting a few equations when they help explain key ideas. People who dislike equations should still find plenty of content in this book. Others, who appreciate the power and compactness of mathematics, may value its presence.

It Will Be a Long Journey, so Start Now

I used to think that sensible and compelling new ideas would be adopted quickly. Today, I believe this view is hopelessly naive. After all, it took 40 years before we recognized the power of the Toyota Production System.

Based on my personal experience, it takes a bit of time for new ideas to obtain traction. I began writing about the importance of calculating cost of delay in 1983. After 26 years, only 15 percent of product developers know the cost of delay on their projects. I began writing about the importance of product development queues in 1991. After 18 years, about 2 percent of product developers measure the size of queues in their development process. The ideas in this book challenge many existing practices. I do not expect they will be adopted overnight.

There are two common but bad reasons to delay adoption. First, some companies will wait until there are role-model companies demonstrating large-scale and mature implementations of these ideas. Once this has occurred, they will be ready to adopt all the ideas in one large batch.

I would suggest that there is a better approach. The methods in this book can be implemented using two general paths. They can be

implemented from the top down in one large batch, or from the bottom up in small batches. Everything we will discuss about the advantages of small batch size applies to implementation. Small batches have lower risk, are less expensive, and they produce faster results. They accelerate learning. In fact, companies that are already implementing these ideas in small batches have achieved 5–10 times improvements in portions of their development process.

Furthermore, I believe you will miss a huge economic opportunity if you wait until the full-scale implementations become visible. Did you ever consider why American industry adopted lean manufacturing 30 to 40 years after these methods began to produce benefits in Japan? Was it because the Japanese hid this approach from Western eyes? No. American industry waited until the full-scale implementations of these ideas at Toyota could no longer be ignored. As a result, we missed out on 30 years of economic benefits.

A second bad reason to delay adoption is to wait until this new approach has been carefully studied by academia. You might reflect on how long it took for academia to replace the orthodoxy of traditional manufacturing with the new ideas of lean manufacturing. The new methods of lean manufacturing began emerging in the late 1940s in Japan. They originated from practitioners like Toyota, not academia.

It took approximately 30 to 40 years for these ideas to be acknowledged and accepted by the academic community in the United States.[3] Again, I would stress that you can make a lot of money between the time when useful new ideas appear, and when they are widely accepted in academia.

So, start small, and start quickly. Pay attention to feedback. Above all, think about what you are doing. Good luck!

---

[3] It would be an oversimplification to attribute all this delay to inertia. Responsible academics reach conclusions based on factual evidence and rigorous analysis. It takes time to do solid research.

About the Author

For 30 years, Don Reinertsen has been a thought leader in the field of product development. In 1983, while a consultant at McKinsey & Co., he wrote the landmark article in *Electronic Business* magazine that first quantified the financial value of development speed. This article is believed by some observers to have triggered the movement to shorten development cycles in American industry. It is frequently cited as the McKinsey study that reported "six months' delay can be worth 33 percent of life cycle profits."

In 1985, he coined the term *Fuzzy Front End* to describe the earliest stage of product development. In 1991, he wrote the first article showing how queueing theory could be practically applied to product development.

His 1991 book, *Developing Products in Half the Time,* coauthored with Preston Smith, has become a product development classic. It has been translated into seven languages. His 1997 book, *Managing the Design Factory,* was the first book to describe how the principles of Just-in-Time manufacturing could be applied to product development. In the 12 years since this book appeared, this approach has become known as lean product development.

For 15 years, Don taught executive courses at California Institute of Technology, and he currently teaches public seminars throughout the world. He has a B.S. in electrical engineering from Cornell, and an M.B.A. with distinction from Harvard Business School.

His company, Reinertsen & Associates, has helped leading companies throughout the world improve their product development processes for over 20 years.