

*The Addison-Wesley Signature Series*



KENT BECK SIGNATURE  
BOOK

# LEADING LEAN SOFTWARE DEVELOPMENT

RESULTS ARE NOT THE POINT

MARY AND TOM  
POPPENDIECK



*Foreword by Dottie Acton,  
Senior Fellow, Lockheed Martin*

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and the publisher was aware of a trademark claim, the designations have been printed with initial capital letters or in all capitals.

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both: Websphere, Websphere Service Registry and Repository, IBM, and the IBM logo.

Capability Maturity Model Integration and SCAMPI are service marks of Carnegie Mellon University.

Post-it is a registered trademark of 3M Company in the United States, other countries, or both.

The authors and publisher have taken care in the preparation of this book, but make no expressed or implied warranty of any kind and assume no responsibility for errors or omissions. No liability is assumed for incidental or consequential damages in connection with or arising out of the use of the information or programs contained herein.

The publisher offers excellent discounts on this book when ordered in quantity for bulk purchases or special sales, which may include electronic versions and/or custom covers and content particular to your business, training goals, marketing focus, and branding interests. For more information, please contact:

U.S. Corporate and Government Sales  
(800) 382-3419  
corpsales@pearsontechgroup.com

For sales outside the United States please contact:

International Sales  
international@pearson.com

Visit us on the Web: [informit.com/aw](http://informit.com/aw)

*Library of Congress Cataloging-in-Publication Data*

Poppendieck, Mary.

Leading lean software development : results are not the point / Mary and Tom Poppendieck.

p. cm.

Includes bibliographical references and index.

ISBN 0-321-62070-4 (pbk. : alk. paper)

1. Management information systems. 2. Computer software—Development. 3. Organizational effectiveness. I. Poppendieck, Tom. II. Title.

HD30.213.P65 2010

658.4'038011—dc22

2009035319

Copyright © 2010 Poppendieck LLC

All rights reserved. Printed in the United States of America. This publication is protected by copyright, and permission must be obtained from the publisher prior to any prohibited reproduction, storage in a retrieval system, or transmission in any form or by any means, electronic, mechanical, photocopying, recording, or likewise. For information regarding permissions, write to:

Pearson Education, Inc  
Rights and Contracts Department  
501 Boylston Street, Suite 900  
Boston, MA 02116  
Fax (617) 671-3447

ISBN-13: 978-0-321-62070-4

ISBN-10: 0-321-62070-5

Text printed in the United States on recycled paper at RR Donnelly in Crawfordsville, Indiana.  
First printing, October 2009.

# Foreword

I was a relative newbie to both agile and lean when I first met Mary and Tom Poppendieck at an Agile Conference in Salt Lake City in 2003, so of course I started our conversation with the question that had been bothering me: “How do you reconcile the lean view that tests are waste with the need for tests in software development?” Mary’s immediate response: “Unit tests are what let you stop the line.”

I’ve learned a lot about both agile and lean since then, and the more I learn, the more I appreciate the profound understanding that led to that simple answer so clearly tying multiple agile practices to fundamental lean principles.

So, when Mary asked me to review her new book, I jumped at the chance, and I wasn’t disappointed. The approach of framing various views into lean leadership provided fresh perspectives on many of the issues that we must all address if we are to be effective in developing software in the twenty-first century. I could rave on about the book, but that would just keep you from reading it, so I decided to limit myself to sharing my three favorite parts of the book (in chronological order)—not an easy task when the whole book is packed with valuable information.

Since I’m a geek at heart, my first favorite part of the book was Chapter 2, Technical Excellence, with its fresh look at agile practices as a continuation and extension of the fundamentals expressed in the early days of computer science by giants such as Dijkstra, Parnas, and Mills. Seeing agile practices as an evolution of the best of the past, rather than as a revolution that rejects the past, is an important step in enabling adoption of the best that agile has to offer.

Since I’m an idealist, my second favorite part of the book was in Chapter 4, Relentless Improvement, with the discussion of the need for a shared vision of what success will look like for your organization. When you couple this with the material in the same chapter about the importance of managers as mentors in developing problem solvers, you start to see the potential for an organization where everyone works every day to remove the impediments to future success.

Finally, since I’m a realist, my third favorite part of the book was Chapter 5, Great People. “Success comes from people. Results are not the point. Developing

people so that they can achieve successful results is the point.” Statistically speaking, no large company has all above-average employees, suppliers, and customers. However, by following the principles and advice in this book, any company, large or small, has the potential of getting above-average results from its average employees, suppliers, and customers. That’s really what business success is all about.

If I read the book again, would I pick the same three favorite parts? I don’t know, but I’ll find out. As soon as the book is published, I’ll be starting book clubs across our company to share these ideas and see where they lead us. I encourage all of you to do the same.

Dottie Acton  
Senior Fellow  
Lockheed Martin

# Introduction: Framing

---

## Weathering the Perfect Storm<sup>1</sup>

*Sweden's Handelsbanken has placed a large sum of money in an account at the Riksbank in a bid to help the central bank safeguard the financial system.*

—Stockholm, June 17, 2009 (Reuters)

Svenska Handelsbanken is one of the top 25 banks in Europe. It was not just the only bank that survived the Swedish banking crisis in the 1990s without asking for government support—it has also done very well in the 2008/2009 crisis. Handelsbanken did not have to raise capital or ask for government support and its shares have been the best performing European bank stock by a wide margin. Despite its large size Handelsbanken has, in many ways, acted as a shock absorber, not a shock amplifier, to the financial system.<sup>2</sup>

In 1970, Svenska Handelsbanken was 100 years old and deeply troubled. It was trying mightily to become the largest bank in Sweden, and costs had gotten out of control. To make a bad situation worse, a small provincial bank in northern Sweden was eating into its market share. When the Handelsbanken management team abruptly resigned, its board decided to recruit the head of the upstart competitor as Handelsbanken's new managing director. Dr. Jan Wallander was a professional economist turned banker who had his own ideas about how to run a bank.

The first thing Wallander did was to make it clear that growth is not the point; profitability is what matters. He insisted that everyone in the bank stop trying to bring in as much revenue as possible and start focusing on generating

- 
1. Information in this section is from Wallander, "Budgeting—An Unnecessary Evil," 1999.
  2. Jacket text for Kroner. *Svenska Handelsbanken: A Blueprint for Better Banking*, 2009.

profitable revenue instead. To accomplish this goal, he had information systems publish a few key metrics for each branch immediately after the end of each month, numbers such as the cost-to-income ratio (the inverse of profitability) and income per employee (productivity). Then he gave branch managers the freedom to manage their affairs locally. They could decide which products to sell, how much money to spend, how many employees to hire, and so on. Wallander felt that central control got in the way of the people closest to customers, slowed down their reaction time, and stifled their creativity. So decision-making authority devolved to branch managers; staff groups were dramatically reduced in number and size, and those that remained had to sell their services to the branches.

Each month branch managers could see how they stood relative to their peers on the key measures. Wallander believed that this kind of competition among branches provided a continuing challenge that drove each branch to constantly improve the things that really mattered. To keep this competition friendly, there were no bonuses based on the relative internal standings. Instead, Wallander established one of the earliest profit-sharing programs in Sweden, with payouts to a pension fund based on the overall profitability of the company relative to its external competitors.

The bank's goal was to be more profitable than comparable banks. From 1972 onward, Handelsbanken has been more profitable than the mean of all of its competitors, and generally it has been the most profitable bank in Sweden, although occasionally it was in second or third place. Shortly after Wallander took over, two Swedish banks merged to form a bank much larger than Handelsbanken; it took 23 years for Handelsbanken to grow back into the largest bank in Sweden—this time without explicitly trying.

The results of Wallander's approach were both immediate and long-lasting. Svenska Handelsbanken is among the most cost-efficient banks *in the world*. Moody ranks its financial strength among the top ten European banks, something that comes in handy during the periodic financial crises that plague the banking industry. It has continued to grow, expanding to other Nordic countries as well as the UK. Frequently ranked as a top place to work, Handelsbanken has the most satisfied customers, the lowest employee turnover, and the highest investor return of any Swedish bank.

Jan Wallander understood that the most effective, responsive organization is one where small unit leaders make local decisions. He devised an organizational structure, governance approach, and culture that reliably engaged the creativity and dedication of knowledge workers at hundreds of local branches. Four decades later, his vision is credited with helping Handelsbanken weather one of the worst financial storms in a century.

## The Leadership Frame of Great Companies

What does Handelsbanken have in common with Nucor Steel, SAS Institute, W. L. Gore, Southwest Airlines, Semco, and Toyota? Each of these companies has developed a *culture of high involvement*, each thrives in *an industry of high change*, and each has sustained *best-in-industry performance* over time. And each company credits its unique culture for its success. Interestingly, the cultures of these companies are not all that unique; in fact, they are remarkably similar. Consider these descriptions of company culture, direct from each company's Web site:

### *Nucor Steel*<sup>3</sup>

#### *Our Culture:*

Safety First  
Eliminating Hierarchy  
Granting Trust and Freedom  
Giving All Workers a Stake in the Company  
Turning Everyone into a Decision Maker  
Inspiring a Work Ethic

### *SAS Institute*<sup>4</sup>

If you treat employees as if they make a difference to the company, they will make a difference to the company. That has been the employee-focused philosophy behind SAS' corporate culture since our founding in 1976. At the heart of this unique business model is a simple idea: satisfied employees create satisfied customers.

"We've worked hard to create a corporate culture that is based on trust between our employees and the company," explains SAS President and CEO Jim Goodnight, "a culture that rewards innovation, encourages employees to try new things and yet doesn't penalize them for taking chances, and a culture that cares about employees' personal and professional growth."

### *W. L. Gore & Associates*<sup>5</sup>

How we work at Gore sets us apart. Since Bill Gore founded the company in 1958, Gore has been a team-based, flat lattice organization that fosters personal initiative. There are no traditional organizational charts, no chains of command, nor predetermined channels of communication.

Instead, we communicate directly with each other and are accountable to fellow members of our multi-disciplined teams. We encourage hands-on innovation, involving those closest to a project in decision making. Teams organize around opportunities and leaders emerge.

---

3. [www.nucor.com/story/chapter3/](http://www.nucor.com/story/chapter3/).

4. [www.sas.com/jobs/corporate/index.html](http://www.sas.com/jobs/corporate/index.html).

5. [www.gore.com/en\\_xx/aboutus/culture/index.html](http://www.gore.com/en_xx/aboutus/culture/index.html).

### *Southwest Airlines*<sup>6</sup>

The mission of Southwest Airlines is dedication to the highest quality of Customer Service delivered with a sense of warmth, friendliness, individual pride, and Company Spirit.

We are committed to provide our Employees a stable work environment with equal opportunity for learning and personal growth. Creativity and innovation are encouraged for improving the effectiveness of Southwest Airlines. Above all, Employees will be provided the same concern, respect, and caring attitude within the organization that they are expected to share externally with every Southwest Customer.

Southwest Airlines is famous for its remarkable management philosophy: employees first, customers second, shareholders a distant third. When we look at other truly successful companies, we notice that they have a similar philosophy. In these companies, front-line people are highly valued, are expected to make local decisions, and are effectively engaged in delivering superior customer outcomes. As a result, the companies gain two significant advantages: (1) Workers routinely dedicate their intelligence and creativity to help the company be successful, and (2) the company is adaptive; it can detect and quickly respond to changing market conditions and opportunities.

The purpose of this book is to explore how we might adapt the way these great companies frame the role of leadership to organizations involved in developing software-intensive systems.

---

## Frames

*Deep frames pervade TPS that fundamentally alter how the system is understood and therefore how to proceed with implementation. If managers and program leaders fail to understand the frameworks underlying TPS, they miss the point and therefore fail to achieve the expected results.*<sup>7</sup>

When coauthor Tom isn't thinking about software development, he pursues his passion, which is photography. He captures stunning scenic views and creates dynamic photo journals of conferences. When you look at his photographs, you see the world through his eyes; he has carefully framed each photograph to guide your attention to the subject and purpose of the image, whether it is an

---

6. [www.southwest.com/about\\_swa/mission.html](http://www.southwest.com/about_swa/mission.html).

7. We thank Michael Ballé, Godefroy Beauvallet, Art Smalley, and Durward Sobek for their paper "The Thinking Production System," 2006, which inspired the theme of this book. This quote is from that paper. TPS is an abbreviation for the Toyota Production System, and also the Thinking People System.



engaged conversation, a sweeping view of a spectacular sunrise, or a tightly framed view of an incongruous detail. The first thing Tom does when he creates a photograph is to decide on the perspective and framing that will lead to an effective composition. Sometimes he creates a frame with his hands to help him concentrate both on how the elements inside the photograph flow together and on which things to leave out of the frame because they would detract from the subject. Only after Tom has framed a picture does he consider the details of focus, depth of field, exposure, improving the lighting, and so on. In the end, each of Tom's photographs tells a story about the subject from his chosen perspective.

According to cognitive scientists, we all interpret our surroundings through frames—mental constructs that shape our perspective of the world. Frames are sets of beliefs about what elements to pay attention to and how these elements interact with each other. Frames place significant limits on our perspective; we can see only what our frames tell us is meaningful, and we usually ignore what lies outside the boundaries. Most of us are unaware of the way our background and experience shape the way we frame our decisions and actions; only a few of us consciously adjust our frames as if we were photographers. In fact, we seldom even think about the direction in which we are pointing our cameras.

Everyone shapes his or her view of the world through framing, and people with different backgrounds are likely to see their surroundings through vastly different frames. By themselves, frames are not inherently good or bad; they just are. However, evidence has shown that certain frames are more likely than others to lead to long-term business success. For example, as we will see later, Southwest Airlines frames its business in a strikingly different manner from most other airlines. And over its almost 40-year life span, Southwest has been more successful than airlines employing different frames, including airlines that have copied many of Southwest's practices. Similarly, Toyota frames its business differently from the Big Three automakers in Detroit, and although most automakers have adopted lean practices, their thinking frames have not really changed. Like Southwest, Toyota is a major player in the automotive business, and not coincidentally, Toyota, Southwest Airlines, and Svenska Handelsbanken all see the world through similar frames.

Whatever frame you use limits the questions you think to ask, the decision alternatives you consider, and the consequences you anticipate from those decisions. Sometimes you may be surprised and perhaps disappointed when things don't work out as expected. It is possible that the execution of the decision was at fault, or that events conspired to change the results. But it is more likely that your frame of reference, your thinking system, is the culprit. If you are not seeing the results you expect from your current direction, consider moving to a different place, re-aim your camera, and look at the problem through a different frame.

## Framing the System Development Process

This book is divided into six chapters. Each chapter contains four frames, so over the course of the book we will look through 24 different frames to get a good picture of how a lean development process should work. Taken as a whole, the 24 frames present a coherent leadership framework for system development. We conclude each chapter with a portrait of the leader who is responsible for creating focus through its four frames. (See Figure I-1.)

Chapter 1, *Systems Thinking*, starts, appropriately enough, by focusing on customers. It examines the nature of customer demand, distinguishing between value demand and failure demand. It investigates how our work systems can predictably and effectively anticipate and deal with demand, and how our policies often interfere and create waste. The chapter concludes with a portrait of a leader who creates a vision of how to delight customers.

Chapter 2, *Technical Excellence*, covers the basics of excellent software development—low-dependency architecture, a test-driven development process, an evolutionary development approach, and the importance of deep expertise among developers. The highlighted leader in this chapter is the competency leader.

Chapter 3, *Reliable Delivery*, starts by finding the biggest constraint in the system and learning how to manage the risk posed by that constraint. It discusses workflow and schedule, and the essential contribution of feedback. The highlighted leader is the product champion—the same leader we met in Chapter 1, but this time in the role of leading implementation.

Chapter 4, *Relentless Improvement*, discusses the essential characteristic of any lean organization: constant, ongoing, never-be-satisfied improvement. It shows how improvement works, suggests some tools, and features the manager as mentor of improvement.

Chapter 5, *Great People*, starts with the assumption that great results come from great people, so the essential question is how to create great people. It starts with the basic tenet that people treat others the way they are treated, then walks through the four Ps: Purpose, Passion, Persistence, and Pride. The chapter highlights the front-line leader, who has the most influence on the way people think about their work.

Chapter 6, *Aligned Leaders*, discusses and gives examples of developing alignment on the leadership team. It has several lists of ideas for your leadership team to ponder as you turn theory into practice.



Figure I-1 *The big picture*

# Chapter 1

---

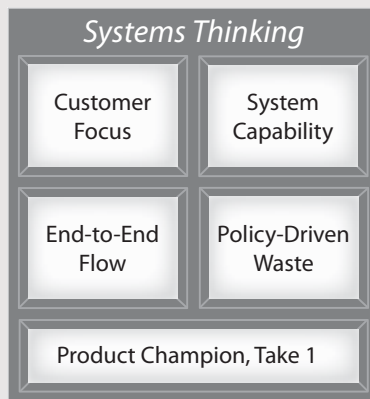
## Systems Thinking

### Snapshot

Southwest Airlines is a company that has defied the odds against success in the rough world of the airline industry and become a model for others to follow—a model that has been enormously difficult to copy. Why? Because would-be imitators copy the parts of the system that fit their mental models, but they don't copy the way these parts work together as a system. They fail to appreciate the fundamentally different way of thinking behind the Southwest Airlines system: *systems thinking*.

For decades, John Seddon has used systems thinking to dramatically improve service organizations. “To take a systems view is to think about the organization from the outside-in, to understand customer demand and to design a system that meets it,” Seddon says. “To enable control in this high variety environment, it is necessary to integrate decision-making with work (so the workers control the work) and use measures derived from the work. . . . If workers are controlling the work, they need managers to be working on the things beyond the control of the workers which affect the system conditions: the way work works. The result is an adaptive, customer-centric system.”<sup>1</sup>

Seddon's approach to systems thinking begins by asking an organization to answer five questions:<sup>2</sup> (1) **Purpose:** What is the purpose of this organization? (2) **Demand:** What is the nature of customer demand? (3) **Capability:** What is the system predictably achieving? (4) **Flow:** How does the work work? (5) **System conditions:** What are the causes of waste in the system? We discuss these five questions in this chapter. Then we introduce the leader whose job it is to bring systems thinking to the creation of a product idea, the *product champion*. We will meet the product champion again in Chapter 3, leading the product development effort.



---

1. Seddon, *Systems Thinking in the Public Sector*, 2008, p. 70.  
2. Seddon, *Freedom from Command and Control: Rethinking Management for Lean Service*, 2005, pp. 101–10.

---

## A Different Way to Run an Airline

*Are empty airplane seats waste?* If you were an airline executive, what would your answer be?

Every month the Airline Transport Association publishes the load factors (revenue passenger miles divided by available seat miles—a utilization measure) for U.S. airlines. Business reporters are quick to praise airlines that increased load factors over the previous year. Financial analysts maintain that once an airline load factor exceeds its break-even point, more and more revenue will hit the bottom line. So it's pretty clear that empty seats are waste—unless you are an airline passenger. Or Southwest Airlines.

Completely full airplanes aren't very comfortable. It's hard to find space for carry-on luggage, and people carry on a lot more these days, since most U.S. airlines are charging to check bags. But the real problem comes during the holidays when bad weather wreaks havoc with airline schedules, and there is no spare capacity. Far too many airline passengers have spent Thanksgiving in Denver or Christmas in Chicago instead of getting all the way home for the holidays.

It costs airlines a lot to untangle the mess of a winter storm at a hub, especially with load factors approaching 100%. To soften the financial blow, U.S. airlines long ago stopped providing lodging for passengers who were stranded because of a weather-caused delay, but they still have to find a way to get those passengers to their destination. Inconvenienced passengers are not happy, but most airlines don't seem to worry about the cost of annoyed customers. After all, the bad weather wasn't their fault.

Over the years, Southwest Airlines has reported load factors that were approximately 10% lower than those of other airlines of similar size. This might lead you to suspect that Southwest has been struggling financially—but you would be wrong. Donald Converse summed up the amazing track record of Southwest Airlines in *Fast Company* magazine in June 2008:<sup>3</sup>

Founded in 1971, Southwest Airlines began to establish a consistent pattern of deviating from convention. In 1978 the airline industry was deregulated and 120 plus airlines have gone bankrupt since. Why, in this difficult environment, has SWA continued to grow and thrive? Notably, SWA is the only airline to continuously show a profit every year since 1973. How has SWA managed to increase its traffic by as much as 139%? Here are some facts that might help to understand how SWA has achieved this incredible record:

- The company consistently leads the industry in low fares and dominates the short haul market with an average of 60% market share.

---

3. Converse, "Thank You Herb!," 2008.

- The company serves over 2400 customers per employee annually—making SWA employees by far the most productive workforce in the airline industry.
- Employee turnover averages 6.4%—again one of the best records in the industry.
- SWA is consistently ranked in the top 100 of the best U.S. companies to work for.
- They have never been forced to lay off employees regardless of external market factors such as recession or high fuel prices.
- They have the best record for baggage handling in the industry.
- They have the best on-time performance record.
- Fewest customer complaints.
- Youngest fleet of airplanes, and the best safety record!

Southwest's stated purpose is to make flying possible for those who would not otherwise be able to afford it. Thus Southwest sees its main competitor as the car, not other airlines.<sup>4</sup> Southwest started out in Texas serving three cities—Houston, Dallas, and San Antonio—that form a triangle, about a four-hour drive, or an hour flight, apart. About a year after it was founded, Southwest had four planes on these routes, but it was running out of money, so it sold one of the planes. However, executives decided that canceling a quarter of its flights would be a disaster—this would not lead to higher load factors; it would lead to many fewer customers.

This was the start of Southwest's famous ability to rapidly turn around aircraft at the gate. The ground operations manager decided that planes could be emptied of passengers and filled up for the next flight in ten minutes, allowing three planes to fly the schedule designed for four.<sup>5</sup> What Southwest realized is that load factors don't take into account the time that airplanes spend on the ground. By operating with fewer planes, Southwest saved significantly on capital investment and labor costs. Maintaining the same number of flights provided plenty of options for passengers and empty seats for growing traffic and absorbing variation—at a small marginal cost. James Parker, former Southwest CEO, notes:<sup>6</sup>

- 
4. In Europe, Southwest would compete with trains, but in the United States there are few equivalent train systems.
  5. Southwest planes used to leave the gate before everyone was seated, encouraging people to find a seat. This is no longer allowed, and turnaround times are a bit longer these days.
  6. Parker, *Do the Right Thing: How Dedicated Employees Create Loyal Customers and Large Profits*, 2008, p. 57. Italics in the original.

With Southwest's predominantly short and medium haul route structure, an increase in turnaround times of 20 minutes per flight would reduce the available flying time for each aircraft by about two hours every day. Spreading this effect over the entire fleet of more than 450 airplanes would mean the loss of about 900 hours of flying every day. Remember, an aircraft doesn't make any money sitting on the ground. So Southwest would have to buy at least 80 more airplanes. With a list price of around \$40 million per 737, this comes to a tidy sum of well over \$3 billion. *This is money Southwest doesn't have to spend because of its efficient turnaround.*

Southwest realizes that increasing its revenue potential is more profitable than trying to make as much money as possible on every flight, and customers get more of what they want at the same time. Southwest's airplanes may be less full than those of its competitors, but the planes spend a lot more time in the air, which more than compensates for the lower load factors. By one account, the typical Southwest 737 is used 11.5 hours a day, compared with an average of 8.6 hours for other carriers.<sup>7</sup> Rapid aircraft turnaround leading to high capital equipment utilization bears a resemblance to the use of short setup times in manufacturing, pioneered by Toyota. Both companies discovered new, counter-intuitive ways to make more productive use of both capital equipment and people while maintaining increased flexibility in the face of variation in demand. In both cases, their approach trumped the apparent economies of scale enjoyed by their competitors.

Since Southwest can turn a plane around much faster than other airlines, the people involved can be assigned to a new plane more quickly, and thus they are more productive. Short turnarounds have been difficult for other airlines to copy; Southwest's times average 15 to 20 minutes faster than those of their best competitors.<sup>8</sup> One of the reasons short turnarounds are so difficult to emulate is that many different departments must be coordinated in a short span of time: pilots, flight attendants, caterers, cabin cleaners, gate agents, operations agents, ramp agents, ticket agents, baggage agents, freight agents, fuelers, mechanics. Southwest nurtures a strong culture of cooperation across these departments; people from different areas routinely help each other out. At many other airlines, work rules and measurement systems discourage such cooperation.<sup>9</sup>

---

7. Freiberg and Freiberg, *Nuts: Southwest Airlines' Crazy Recipe for Business and Personal Success*, 1998, p. 51.

8. Parker, *Do the Right Thing: How Dedicated Employees Create Loyal Customers and Large Profits*, 2008, p. 57.

9. See Grittell, *The Southwest Way*, 2003, Chapter 3.

---

## **Any Pilot Can Fly Any Plane, Any Plane Can Fly Any Route**

We were flying from Los Angeles to Denver on Southwest. We were pleasantly surprised by the waiting area: a double row of comfortable seats with small tables and power outlets in between, right at the gate, and free! It was just what we road warriors look for in an airline club room, especially when the flight is delayed. And our flight was late, due to fog in San Francisco. When it arrived, we experienced the famous rapid turnaround with a well-managed boarding queue. But alas, we were not going to depart on that plane; the pilot announced that it had a mechanical problem.

We were asked to go to another airplane and please take the same seat; we didn't have boarding passes, after all. The plane was about 70% full—typical for Southwest—so most of the center seats were empty and no one was too worried about seating. A quick exit from the plane and a short walk brought us to a gate where the plane we were now going to take was just pulling up. It was empty in short order, and two agents stood at the gate checking each of us off on a paper list as we boarded. Soon we were all in the same seats with the same crew and the same luggage in the hold. Less than a half hour after the mechanical problem was discovered, we were leaving in a different plane.

Not long after that trip we found ourselves in Atlanta, flying home to Minneapolis on a late-evening flight, this time on a different airline. Again, fog took its toll. As the evening wore on, our flight was delayed, first to 11:00 P.M. Then 11:30 P.M. Then a quarter past midnight. I checked the arrival time of the plane coming in to our departing gate to be sure we would have a plane. An arrival was scheduled at 11:05 P.M.—over an hour before our flight was scheduled to depart. Surely this could not be our plane. How could a simple turnaround take 80 minutes? We were soon to find out.

We went to the gate and found our future cabin crew already there; they'd been waiting for the plane for a couple of hours, just like us. In this traditional system, there was no concept that a flight and its crew could simply be assigned the next available plane; everyone had to wait for the appropriate aircraft to arrive. We could see why Southwest achieves higher productivity through its emphasis on simplicity: Any flight and any crew can use any available plane.

When the plane we were to take finally arrived, it seemed to take forever for people to get off. Of course it was completely full, but even worse, it was overflowing with carry-on luggage. The airline had a \$15 luggage fee, which encouraged passengers to carry on everything they could. As we boarded, the same thing happened—the plane was completely full and almost everyone was overburdened with luggage. It took forever to get all the luggage stowed or checked. The impact of luggage fees on turnaround times seems to be lost on most airlines. One wonders if the fees cover the decreased aircraft utilization.

Mary and Tom Poppendieck

---



Jim Parker, Southwest's former CEO, claims that Southwest Airlines isn't really in the airline business at all. The company is in the customer service business; it just happens to fly airplanes.<sup>10</sup> He believes that the secret to success in the customer service business is to take good care of your employees, because then they will take good care of your customers, and satisfied customers lead to a successful business. So Southwest focuses on three things: Create a great place to work, provide customers with what they really want, and make sure that the airline always makes a profit so it can stay in business for the long term.

---

## Frame 1: Customer Focus

If you think of your organization as a system,<sup>11</sup> then a clear, customer-focused purpose that can be used to drive holistic decisions is the starting point for systems thinking. Start by asking yourself, "Who are my customers, and what do they really want from my organization?" This is not as easy as it sounds, because it may not be clear who your customers really are. So let's start by defining whom we mean by customers.

### Who Are Your Customers?

A customer is anyone who *pays for, uses, supports, or derives value from* the systems you create. Thus you probably have a lot of different customers. But wait; there's more. If you are working on a subsystem of a larger system, you have customers of your subsystem and customers of the overall system. So arriving at clarity about who your customers are can be challenging.

We recommend that you simplify the issue of identifying your customers by considering the whole system you are involved in creating, not just the software. If you are creating a subsystem of a larger system, think of those creating the other parts of the system as partners and focus on understanding the customers of the overall system. For example, if you are developing the software for a medical device programmer, the primary customers are those whose lives are improved by the medical device, followed closely by the medical personnel who select, deliver, and program the medical device. If you are developing soft-

---

10. Parker, *Do the Right Thing: How Dedicated Employees Create Loyal Customers and Large Profits*, 2008, p. 65.

11. We use the word *system* in two ways: First, there is your organization's work system, and second, there are the systems you deliver to customers. The meaning should be clear from context. Systems thinking should be applied to both types of systems.

ware to automate a business process, the purpose of the software is to support the most effective business process possible, and the most appropriate measurements of success would be the business results generated by the improved business process.

### *Customers Who Pay for the System*

Often the first customers who come to mind are the sponsors who pay for your systems. Ask yourself, “Why do these customers spend good money for my systems? What purpose are they trying to achieve? What are their key constraints?” Sponsors usually have a clear purpose: They know the overall results they expect from an investment. Cost is frequently constrained by a limit beyond which the investment would not make sense, and sometimes there is a deadline beyond which the system would not be useful.

Sponsors usually don’t care about the details of the system, as long as their purpose is served. Thus it is very important to be clear about what this purpose is and to focus on it separately from the details of implementation. The objective is to be sure that the sponsor’s purpose is achieved.

### *Customers Who Use the System*

Customers who use your system may not be the same as those who pay for the system, and sometimes these users have a different purpose. They have a job to do, and they want the system to help them do their job more effectively, more comfortably, and more intuitively. As Carl Kessler and John Sweitzer say in *Outside-in Software Development*, “If your product makes end-users feel smart, effective, and in control of their work, you have a winning product.”<sup>12</sup>

### *Customers Who Support the System*

When you release a system, you may breathe a sigh of relief now that your work is done; but the work is just beginning for those who support the system. Do you know how *consumable* your systems are—that is, how rapidly and easily your customers can begin realizing value from your system? What does it take to install a system or a release? How difficult is it to use? How much training is required?

You should also understand how *robust* your systems are. How often do they fail in normal use? How long does it take to recover? How easy is it to find and eliminate the cause? What does a failure cost your customers?

---

12. Kessler and Sweitzer, *Outside-in Software Development: A Practical Approach to Building Successful Stakeholder-Based Products*, 2008, p. 25.

### *Customers Who Derive Value from the System*

One of the most effective ways to create a competitive advantage is to help your customers be successful. So ask yourself whether you know how long it takes your customers to start deriving value from your systems. Is there anything you can do to help them?

Many companies expand their business by understanding their customers' value chain and helping their *customers' customers* be successful. For example, if a company helps its customers develop a truly effective call center, or sell software tools that increase the effectiveness of financial analysts, *their* customers will be delighted.

---

### **Product Owners Aren't Customers**

I was giving a talk at a company with maybe 70 people in the room. "How many here are developers?" my host asked. About 50 people raised their hands. "And how many of you developers understand the purpose of the code you work on?" Only two hands stayed up!

We found a similar sentiment in the developers who attended our class over the next two days; few of them knew why they were working on their assigned stories. I challenged them to take responsibility for understanding and achieving the purpose of their work; they should spend time with the production workers in the manufacturing plant and find out how well the system was working for them. As it turned out, many had already done that, but the issues the production workers thought were important never made it to the developers' "to do" list.

"We are a central IT department," the department manager told me. "When we implemented Scrum, we decided that the IT people in the business units should be the product owners. But the problem is they are actually project managers, so they usually tell the development teams exactly what to do without telling them why. They are measured on how many new features get into the system, so they don't put priority on maintenance work, even though our system annoys the production workers and often slows down factory production."

Product managers and product owners are not customers. Their job should be to connect the development teams with customers, but that clearly wasn't happening here. Instead, the product owners represented a handover between the development teams and their real customers, keeping developers from engaging in the purpose of their work.

Later in our class we did a problem-solving exercise, and two groups independently decided to work on the problem of not being allowed to do the maintenance work they knew should be done. Both groups came up with the

same solution to the problem: They decided they would reserve 20% of the velocity of each iteration to work on tasks of their own choosing, which would give them time to fix the system so they could be proud of it.

Mary Poppendieck



## What Is Your Purpose?

A simple statement of purpose *from a customer's perspective* can do wonders to clarify what is important and what is not. For example, everyone at Southwest knows that the company's purpose is to make flying affordable and enjoyable for everyone. This means that costs have to be low and hassles minimized. This purpose guides both strategic initiatives and day-to-day decisions at all levels of the company.

Once you have identified your customers, it is time to come to a clear understanding of your organization's purpose from your customers' perspective. Create a brief, crisp statement of what is important for your organization to succeed. This is what your work system must deliver. Note: Your purpose is probably *not* to develop software. It is a rare customer who wants software; customers want their problems solved. If customers could solve their problems without software, they would be delighted.

As an example, let's look at how we would describe the purpose of the development group in the sidebar "Product Owners Aren't Customers." In this case, the primary customers were the users, the production operators; another important customer was the sponsor; and a third set of customers were the operations people who supported the system. The purpose of the development group was to provide a production information system that made production operators' jobs easier while improving quality and productivity in the manufacturing plant.

Let's look at another example. Werner Vogels, CTO of Amazon.com, defines the purpose of his IT organization thus: Provide a highly available, highly scalable technology platform for Amazon.com and its business partners.<sup>13</sup> By "available," Vogels means that shoppers can always browse a site and put things in their shopping basket, even if other parts of the system aren't working. By "scalable," Vogels means that there should be positive, not negative, economies of scale, and scaling should be available on demand. Over the past several years, pursuing this purpose has turned Amazon.com into an impressive technology provider as well as a retailer, while Amazon's IT department has become a true profit center.

---

13. Vogels, "A Conversation with Werner Vogels," 2006; see also the videos Vogels, "Availability & Consistency," 2007, and Vogels, Keynote, 2008.

## What Is the Nature of Customer Demand?

The next step in establishing a systems view of your organization is to understand the patterns of customer demand for the products and services you provide. Customer demand comes in two forms. First there is demand for products and services that provide value. Second, when a product or service doesn't meet the customers' expectations, there is demand for failure remediation—that is, demand to fix something that appears to be broken, inadequate, or difficult to use, configure, or modify.

### *Failure Demand*

Failure demand is the demand on the resources of an organization caused by its own failures. Support calls, for example, are almost always failure demand. They may be caused by some aspect of the system that is unclear to a user, or they may be caused by an outright failure of the system to perform.

Change requests may also be failure demand; for example, a change request may come from a failure on your part to understand your customers or a premature decision on your part about what customers actually want. Even if you deliver exactly what your customers asked for, once they see it they may realize that it doesn't solve their problem; from *your customer's* perspective change requests in this case are failure demand.

An insidious form of failure demand is the demand on your resources created by technical debt: things such as defects you have chosen to ignore, messy coding practices, duplication, lack of effective test automation, a tightly coupled architecture, multiple code branches—anything that makes it difficult to respond to a request for a change. All these things increase the demand on your capacity when customers ask for changes, even if the changes themselves are value demand.

Demand from support organizations that have to deal with your systems is usually failure demand. When your product is difficult to integrate with other systems, database migrations between versions are a nightmare, or intermittent lockups bring down the data center, you have serious failure demand. Even if your software performs exactly the way it was designed to operate, if it gives operations and support organizations problems, both you and they are wasting valuable time.

---

### **What Is Failure Demand?**

We were explaining failure demand to a management team. I asked, "Is the support manager here?" Yes, he was.

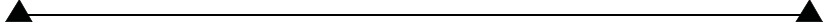
“So how much of the demand on your organization do you think is failure demand?”

“About 95%,” he was quick to reply.

Later that day I was giving a talk to a larger group and a question came from the audience: “You say that we should be able to rapidly update our code. But how can we do that when so many of our customers have customized versions of our software?”

Suddenly the magnitude of the support problem dawned on me. “It sounds to me like you have a huge branching problem,” I replied. And they did. The branching and support policies had created ever-growing failure demand on the organization.

Mary Poppendieck



The purpose of looking for failure demand is to *identify as much failure demand as possible*, because failure demand is waste that you can do something about. In the beginning you should expect to find a *lot* of failure demand. Welcome it; do *not* penalize anyone for it. Failure demand is created by the way your work is done; exposing it gives you the opportunity for significant improvement. Your primary objective is to find and eliminate failure demand so that you have more time to accommodate additional value demand.

Once you have identified failure demand, calculate what percentage of the demand on your organization is failure demand. It is likely to be a big number; we have seen estimates between 30% and 70%. Think about it: If one-third of your demand is failure demand, you would increase the capacity of your organization by 50% if you could eliminate that failure demand. Eliminating failure demand provides a huge opportunity for increased productivity. In Chapter 4, Relentless Improvement, you will find ideas for ways to uncover the causes and reduce the amount of failure demand.

Of course, failure demand is not going to go away overnight, and while it exists, your secondary objective is to remediate each problem as rapidly as possible. John Seddon says, “As a rule of thumb, end-to-end time from the customer’s point of view is almost always an essential measure in any ‘break-fix’ system.”<sup>14</sup> A break-fix system starts with a customer in distress (something is broken) and ends when the problem is resolved (fixed). For all failure demand that you choose to fix, the essential customer-focused measurement is the request-to-resolution time. Clearly, your first priority should be to eliminate the

---

14. Seddon, *Freedom from Command and Control: Rethinking Management for Lean Service*, 2005, p. 106.

failure demand, but as long as it's there, minimize your customers' pain by fixing their problem as rapidly as possible.

### *Value Demand*

The primary demand on your organization should be value demand. This demand can be in the form of requests for work that will add value from a customer's perspective, or it can be in the form of unmet needs that customers don't know they have, but that you discover and satisfy through your products and services. Value demand, when satisfied, usually generates revenue for your organization.

Almost every software development organization we know of has more demand for its services than it can accommodate, so a good question to ask is "What portion of incoming value demand does my organization have the capacity to satisfy?" If value demand exceeds your capacity to deliver, there are two steps to take: (1) Focus on eliminating failure demand so you have more time to work on value demand, and (2) evaluate how you filter value demand to decide which work you will accept and which you will turn down.

Once you have a good idea about the volume of value demand your organization is experiencing, how you qualify it, and what percentage you can accept, the next step is to establish a customer-focused view of the value you deliver. What do your customers really value? Once you understand what customers value, you can establish customer-centric measures to focus everyone on improving customer outcomes, rather than meeting internal targets. If you provide visibility into what it means to deliver customer value, development teams can act independently and creatively to give customers more of what they really want.

Examples of good customer-centric measurements might be the following:

1. Time-to-market for product development (for the whole product)
2. End-to-end response time for customer requests (request-to-resolution time)
3. Success of a product in the marketplace (profitability, market share)
4. Business benefits attributable to a new system (measureable business improvement)
5. Customer time-to-value after delivery (consumability)
6. Impact of escaped (post-release) defects (customer downtime, financial impact)

---

## Frame 2: System Capability

Once you understand your customers and what it means to deliver value to those customers, the next step is to gain a clear understanding of your capability to deliver that value. What capabilities do you need to be successful in the marketplace, both now and in the future? It is important to have the capability to deliver what customers want today, but it is also necessary to look into the future and make sure that today's efforts are moving your organization toward tomorrow's success. Realistic long-term planning and risk assessment, along with understanding the current and future competitive environment, are essential ingredients of sustainable success. If people and teams are going to use their creative energy to decide what is important to do today, they need to understand both the immediate customer needs and the challenges of the future.

### What Is Your System Predictably Achieving?

The starting point for improving your capability to satisfy customers is to understand how your current work methods actually work, what they are currently capable of delivering. You need to understand how your capability compares with the needs of your customers and the capability of your competitors, both now and in the future.

#### *Understanding Capability*

If you want to know how well your organization is performing, don't just look at a few data points; look at a sequence of data over time. All systems exhibit variation, and looking at an occasional data point does not tell you much about how much variation there is in your system. In fact, random data points give a distorted view of your current capability.

A good way to visualize capability is to create a time series chart. Let's say that you want to look at your capability to quickly respond to a particular type of demand, for example, small urgent requests from customers. When a request arrives, give it a date stamp; when the request is closed out, subtract the arrival date from the current date to get the end-to-end response time, then plot these times. A time series plot of response times for small urgent requests might look something like Figure 1-1.

Your immediate reaction to a chart like this might be to look for the cause of the long delays for several of the requests. You don't have to look very far—your *work methods* are the cause. This is a chart showing what your work system is currently capable of delivering. You are not going to change things by



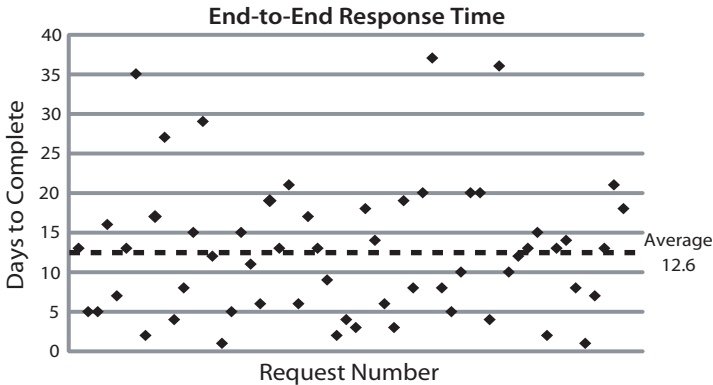


Figure 1-1 Time series chart of end-to-end response time for small urgent requests

looking for scapegoats or setting more aggressive targets. Your current way of working produces the results in the chart; if you don't like the results, you have to change the way the work is done.

W. Edwards Deming worked to spread an understanding of variation to industry leaders, but we think that his message is often distorted. We frequently see efforts aimed at removing variation from every process, without recognizing that such efforts usually make things worse. When you try to remove normal (common-cause) variation from a process, the process actually gets worse, not better. Deming pointed out that almost all variation (perhaps 95%) is common-cause variation—it is inherent in the system—and the only way to remove it is to change the way work is done. He concluded that most variation is a management problem. So if you are looking for scapegoats, start by looking in the mirror.

### Positive or Negative Reinforcement?

A well-known approach to training is to give positive reinforcement when someone does something especially well, because this is supposed to lead to more of the same behavior.

We heard of a pilot training center that decided to test this theory. Careful records were kept of the results of positive and negative reinforcement of pilots in training. Much to the surprise of the researchers, they found that whenever a trainee pilot was given positive reinforcement, performance got worse, and whenever negative reinforcement was given, performance always improved! The training center concluded that giving negative feedback for poor performance was an effective training technique, and that positive feedback for good performance should be avoided.



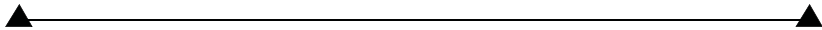
Figure 1-2 *Performance of a pilot in training*

What's wrong with this research? Consider a time series of the performance of a pilot in Figure 1-2.

This time series shows expected variation in performance—variation due to random causes largely unrelated to pilot skill. Assume that the pilot is praised at the high points (circled). Of course, performance will appear to be lower the next time! If the pilot is reprimanded at the low points (boxed), performance would appear to improve.

The evidence shows very little about positive or negative reinforcement, but it clearly indicates that those gathering the data did not understand variation.

Tom Poppendieck



## What Does Your System Need to Achieve?

The first step in changing the way work is done is to develop an understanding of what capabilities you need for both short- and long-term success. Do you have competitors who are better at delivering value to your customers than you are, or is there a threat that such competitors might emerge in the future? Could you attain a meaningful competitive advantage from significant improvements? Will your performance deteriorate over time if you proceed down your current path? Will your architecture support sustainable growth? Are you building up technical debt that will slow you down in the future?

Once you understand where you need to be in order to be successful, both now and in the future, you can create a long-term challenge that will make it clear to everyone what the organization needs to happen in order to be successful both now and over the long run. This will enable people and teams to balance short-term or narrowly focused decisions against long-term and system-wide imperatives.

### *Don't Set Targets*

You should *not* set targets. Your current system is what it is; targets are not going to change its capability. You *measure* system capability; you do not prescribe

it. As Deming once said, “If you have a stable system, then there is no use to specify a goal. You will get whatever the system will deliver. A goal beyond the capability of the system will not be reached. If you have not a stable system, then there is no point in setting a goal. There is no way to know what the system will produce: it has no capability.”<sup>15</sup>

Think of it this way. Let’s say you set a target that is beyond the capability of your system, a goal that your current work processes cannot achieve. Since the target is a strong motivator (no argument there), people have three choices: (1) redesign the work, (2) distort the system (for instance, by ignoring defects), or (3) cheat (game the system) to hit the target.<sup>16</sup> If people do not have the know-how to redesign the system, they are left with two options: distort or game the system.

Suppose the people do have the know-how to redesign the work. In our experience, redesigned work is likely to produce far better results than an arbitrary target—but where is the motivation to surpass the target? Similarly, if you set a target below the current system capability, you are likely to get less than the system is capable of delivering, because there is little motivation to surpass the target.

---

### Goals Gone Wild

Managers have no doubt heard that they should set challenging goals in order to motivate the best performance. But in the paper “Goals Gone Wild,”<sup>17</sup> researchers from four top business schools question this practice. They agree that goals are powerful motivators, so powerful that the goals often have severe side effects, ranging from dangerous products to unethical behavior to seriously underperforming systems to reduced learning.

There may be too many goals, causing people to choose the ones they like. Or they can be too specific, causing sub-optimizing behavior. Or they can be too challenging, causing people to meet them at all costs—inducing unnecessary risk taking, cheating, or gaming. The authors issue this warning: “Goals may cause systematic problems in organizations due to narrowed focus, unethical behavior, increased risk taking, decreased cooperation, and

---

15. Deming, *Out of the Crisis*, 2000, p. 76.

16. From Seddon, *Systems Thinking in the Public Sector*, 2008, p. 97.

17. Ordóñez et al., “Goals Gone Wild: The Systematic Side Effects of Over-Prescribing Goal Setting,” 2009. Italics in the original. See also Bazerman, “When Goal Setting Goes Bad,” 2009.

decreased intrinsic motivation.” And they postulate, “*Aggressive goal setting within an organization will foster an organizational climate ripe for unethical behavior.*”

There is no question that goals influence behavior; it's just that they are a blunt instrument. So be careful what you ask for; you will probably get it.

Mary Poppendieck

---

A target is a predetermined level of performance that people are expected to achieve; if there is significant variance from the target (or plan), explanations are usually required. Targets are generally associated with incentives; people are rewarded or punished based on their variance from the target.

Incentive systems based on performance targets make the assumption that if people would only try harder, they could achieve better results—hence targets communicate an implicit assumption that people are not currently contributing their best efforts. Wouldn't it be better to challenge people and teams to excel at delivering exceptional customer outcomes and communicate trust that they will do their best?

### *Use Relative Goals with Caution*

Not all goals are based on predetermined targets; they can be based on relative measures. For example, goals can be based on performance relative to competitors, performance relative to peers, or performance compared to past performance. Relative performance goals do not attempt to set a fixed level of performance for the future; they look backward to see how actual performance compares against the performance of others. In most sports, winning or losing depends upon relative performance: A swimmer only has to swim faster than seven other swimmers in order to win; a baseball team has to score more runs than the other team.

Relative goals can be very motivating, and they have some advantages over targets or plans. Relative goals are not based on a prediction of the future, nor do they encourage people to distort or game the system; and finally, they don't act as a floor on performance. On the other hand, relative goals leave competitors with little motivation to help each other out—quite the contrary: Relative goals can motivate competitors to sabotage each other's performance. Thus ranking performance relative to peers can be damaging inside a company, especially if reward systems are based on this ranking. If, however, the competition is friendly and performance rewards are shared equally among all competitors, the destructive side effects of tracking performance against peers can be mitigated.

### *Challenge: Pull from the Future*

Perhaps the best way to engage people in making the organization better is to create a long-term vision of where you want to be—where you need to be to survive—and challenge everyone to help move the organization from where it is now to where it needs to be. Challenge people and teams to improve the way work is done by redesigning their work methods. Challenge them to develop systems that provide clear business value. Challenge them to devise an architecture that will meet future growth projections. Challenge them to create new products that will be successful in the marketplace.

Let's say that you would like to decrease the “hardening” time at the end of a release cycle. A target might be “Cut the verification time in half!” Given enough emphasis on this target, the verification time will probably be cut in half, but often at the expense of more defects escaping to production.

A far better approach is to challenge teams to redesign the way development works so that defects are discovered and fixed as soon as possible after they are injected into the code. With this challenge, a team would quit focusing on how many features are delivered and start thinking about how to make sure that every delivered feature is defect-free. The team would introduce automated tests and continuous integration and adopt the practice of stopping to fix every defect as soon as it is detected. When done well, this approach has a track record of dramatically decreasing back-end testing time—by far more than half—at the same time as it increases quality and productivity. But these results are achieved only when a team works to improve the capability of its work methods to produce defect-free code. They are rarely achieved by a decree that system verification time must be cut in half.

Challenges are different from fixed performance targets:

1. Challenges are not necessarily SMART;<sup>18</sup> they are open-ended, customer-centric, and designed to elicit passion and pride.
2. Challenges communicate confidence that people and teams are intelligent, innovative, capable of thinking for themselves, and trusted to do their best to further the purpose of the organization.
3. Challenges flow from a long-term vision of what is necessary to be successful over time and contain enough information that people and teams can act independently and with confidence that their work will contribute to achieving the vision.
4. Thus challenges are a pull from the future rather than a forecast of the future.

---

18. SMART goals are specific, measurable, attainable, realistic, timely.

---

## Frame 3: End-to-End Flow

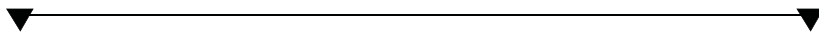
Far too often, providing customer value is thought of as a series of separate input-process-output steps, rather than an integrated flow of work through an organization. But if no one takes an end-to-end view of a customer problem or a product as it moves through a work system, customer problems fall into the cracks between departments, hard-earned knowledge evaporates at handovers, and the things that customers will really value get lost along the way. Developing an understanding of the end-to-end flow of work through a work system is fundamental to systems thinking.

One way to evaluate the end-to-end workflow through your system is to draw a process flow map—also known as a value stream map—of the end-to-end flow of a product or a customer problem as it makes its way through your organization. There are two reasons for drawing these maps. They help you

1. Discover the reasons for failure demand, so it can be eliminated
2. Find waste in the workflow, so it can be removed

### Eliminate Failure Demand

Failure demand is waste. So the best questions to ask are “Why is the process producing failure demand? What can be changed to prevent failure demand?” In general, your bias should be against creating a process map for failure demand, because every activity on that map would be waste. But there are times when a process map for failure demand can help you discover how to reduce and eventually eliminate the waste.



### Failure Demand Process Map

We were at a Web portal company drawing a process map of the critical defect resolution process. The group wanted to start their map when a critical defect report reached the development team. But critical defect resolution is a break-fix problem, so the process should always be mapped starting when the customer discovers that something is broken (see Figure 1-3).

“I’m one of your customers,” I said to get the mapping process started in the right place. “So let’s say that I discover a critical defect. What do I do?”

“Call Level 1 customer support.”

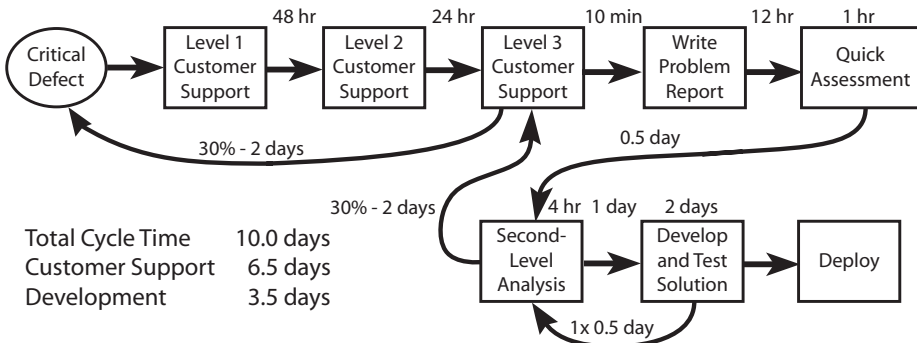


Figure 1-3 Critical defect process map

“Right. I know how that goes: They tell me I must be doing something wrong; nothing can be wrong with your system. So how long does it take me to convince them that it’s your problem, not mine?”

“Maybe a couple of days” came the sheepish answer. Next my problem moves to Level 2 customer support; a day later my problem makes it to Level 3 customer support. Often there is not enough information at this point, so they need to get back to me about 30% of the time. This might take two days.

Just over a day later my problem reaches the development team, which looks at the problem report immediately. Thirty percent of the time they have to get back to Level 3 customer support, which takes another two days. But often (30% of the time) Level 3 customer support needs to get back to me to answer the development team’s question. It takes two days to contact me and another day to get the answer back to the development team.

Once it’s clear what the problem is, it takes four hours to develop a solution and one day to wait for testing, which takes another two days. An additional half day, on the average, is needed to fix problems discovered in testing. Finally the fix is deployed.

All told it takes an average of six and a half days for the customer support process and another three and a half days for development. So it takes an average of ten days to solve my problem.

“What if the first six and a half days disappeared?” I asked.

“You mean have customers call developers directly?” The development VP was appalled. It didn’t seem like a productive path to take.

But then someone spoke up from the back of the room. “We did that,” she said. “A few years ago when I was working in another company, we were going crazy, we needed to try something. So we got the CEO to let us bypass customer support. It worked well.”

“Who answered the phones?” the development VP challenged.

“The developers did, in rotation,” she responded.

“How much of their time did *that* take?” He was clearly skeptical.

“Well, we did a lot of experiments, and we quickly settled on two rules: (1) Immediately after a release, the responsible team staffed the phones. (2) Developers were required to create a knowledge base report accessible to customers for every call, before they could close it out.

“With those two rules, we were astonished at the results. We had 800 developers. Before we changed the process, 60% of development time was spent on critical defects. After we changed the process, 20% of development time was spent on critical defects.”

By connecting developers directly with customers, this company effectively gained 320 experienced developers—for free.

Mary Poppendieck

---

Remember, all time spent handling failure demand is waste. So it is usually better to create a map of the process that *creates* failure, rather than mapping the process that *handles* failure. Your objective is not to handle failure demand more efficiently; it is to eliminate failure altogether. Before mapping the flow of failure demand, ask yourself, “What is causing the development process to produce failure demand in the first place? What can be changed in the process to prevent (or dramatically reduce) the failure demand?”

In particular, we recommend that you think hard before creating a process map for handling a change approval process. Customers don’t usually see change approval systems as a value; they see the *changes they need* as a value, but asking for permission to make the change is annoying. If customers see your change request approval process as a nuisance, that process is handling failure demand. Instead of improving your change request approval process, consider why you should have one in the first place. A change request approval process for a system under development is usually a sign that the development process can’t absorb new ideas or handle typical variation in your customers’ situation. Perhaps you are making decisions too soon, or waiting too long to ask for feedback, or not leaving enough space in your plan to accommodate that feedback. Perhaps your customer engagement and collaboration practices are ineffective. Ask yourself, “How can we reorganize our process so it can discover changes more rapidly and accommodate changes more easily?”

## Map Value Demand

A value stream map is a diagram of the end-to-end flow of value-creating work through your process; its purpose is to help you understand the workflow for



value demand so you can improve your process. Usually a value stream map starts when a customer has a need and ends when the need is met; however, a time-to-market map may start when a product concept is approved and end when customers start realizing value from the product. Before you start a value stream map, use a time series chart, as we discussed in Frame 2: System Capability, to visualize the end-to-end flow time for an average item moving through your system. This is your current capability. The value stream map will show you how much of that time is spent actually adding value.<sup>19</sup>

---

## Value Stream Map

Henrik Kniberg showed a value stream map of the development process at a video game development company.<sup>20</sup> The company was experiencing very long development times, which led to missed market windows and high overhead costs, not to mention the toll that lack of success took on the people developing the products.

Henrik described the value stream like this:

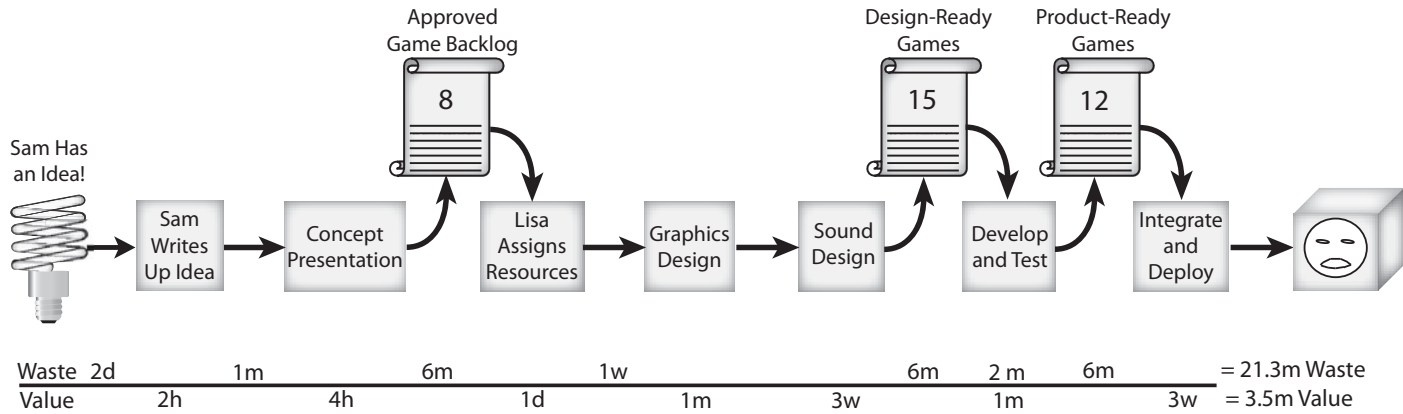
When someone—call him Sam—came up with an idea for a new game, he took a couple of hours to prepare a concept presentation. After about a month, the presentation was made to the idea review committee, and since this was a good idea, it went into the game backlog. This was not the only game in the queue, however; there were several others, so it took six months for graphics and sound designers to become available (see Figure 1-4).

Once the designers were free, Lisa assigned the new game to them, and two months later the design was done. Only a week was wasted during those two months, but when the game was designed, it was put in the design-ready queue, where it languished for six months. When the development team finally got to the game, it was working on two other games at the same time, so it took three months to complete development, even though the team spent only one month of that time actually working on Sam's game. When the team was done, the game sat in the product-ready queue for six months, waiting for final integration and deployment. Finally, over two years after Sam initially had his idea, the game was released. During those two years, only three and a half months—or 14% of the time—were spent actually adding value to the game.

---

19. More examples of value stream maps can be found in Poppendieck, *Implementing Lean Software Development: From Concept to Cash*, 2006, pp. 83–92.

20. Henrik Kniberg of Crisp, a Stockholm company, drew this value stream map at the Deep Lean Conference in Stockholm, September 2008. Mattias Skarin of Crisp helped us understand the case in more detail. Used with permission.



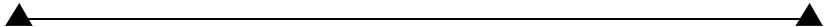
**Process Cycle Efficiency =  $3.5 / (21.3 + 3.5) = 14\%$**

Figure 1-4 Time-to-market value stream map

**Diagnosis:**

There is obviously too much in-process inventory in this system, too many handovers, and a lot of multitasking. To address all of these issues at the same time, the company created cross-functional teams, each of which developed one game at a time, end-to-end, with no handovers. The first team demonstrated that it could repeatedly develop new games in less than four months (six times faster than before).

Mary and Tom Poppendieck



## Find the Biggest Opportunity

The purpose of a value stream map is to help spot opportunities to improve your process capability. When you compare the value-added time to the total process time, you get a feel for how much better things could be. Ask yourself, “Why do we have so much non-value-adding time? Is it really necessary? What’s causing it?”

Generally the biggest delays or loopbacks in a value stream map provide the biggest opportunity for improving the process capability. We recommend that you pick the most likely opportunity and tackle it. Forget the value stream map while you work on improving your process; you can draw a new one once you have changed the process.

Don’t take the ratio of value-added time to total process cycle time (the process cycle efficiency) too literally. We find that after the process is improved, a new value stream map might show that the process cycle efficiency has *decreased!* Why? Because there was a lot of waste in the process that was not recognized when the first value stream map was drawn; once you get used to seeing and removing waste, you tend to find a lot more of it.

---

## Frame 4: Policy-Driven Waste

Waste is anything that depletes resources of *time, effort, space, or money* without adding customer value. Most people do not frame their view of a system in terms of waste, so waste tends to fall outside their field of view. You can remove only the waste you see, so it is important to adjust the way you look at work so that the always-present waste becomes clearly visible.

You would be amazed at how much waste in a system is caused by the system itself, that is, by the way the work in the system is done. Too often waste is disguised under the cloak of habit or conventional wisdom—and more often than not, these sources of waste are embedded in the policies and standard procedures of the organization. Unless and until these policies change, the waste is not going to go away.

## How Can Policies Cause Waste?

There are many ways that policies can cause waste. For example, leaders in many companies believe that developers should not talk to customers because this is a waste of valuable developer time. In Frame 3: End-to-End Flow, we discussed a critical defect process in which three levels of customer support insulated developers from customers (see Figure 1-3). By simply removing the three levels of customer support and having developers talk directly to customers, *40% of the total time of 800 developers was freed up*. This is not an isolated case. We will see in a case study in Chapter 6 that direct developer-customer interaction delivered more of the right content, increased sales, and dramatically reduced support calls.

Of course, simply providing direct customer-developer interaction does not necessarily eliminate the biggest cause of waste; other policies can overwhelm the advantages of good customer interaction.

---

### “We Felt Like Pawns in a Game”

“I was part of a team where we had very strong pull incentives, from a customer point of view. We could add features and reprioritize every two weeks if we wanted to. We had in fact worked very hard to turn a waterfall company more flexible. And succeeded in a way. We listened to the customers, the ones really using the product. And we could build a release rather quickly.

“What we, the team, didn’t realize was how hard the product was to install. This dawned on me when I went to a site using the product to do an upgrade. The end user asked me why he couldn’t do the upgrade himself. He pointed out that he was a very seasoned end user. He had lots and lots of experience with computers and programming. I told him I didn’t doubt his capabilities and explained how things were for me. I was told to travel tens of thousands of kilometers through Europe to do upgrades of systems that could have been designed to be more easily upgraded if one would have thought of it earlier. I kept on telling him about how I felt it should have been done and we felt more and more like pawns in a game. I guess he did more than I.

“What I realized at a later point was this: The upgrades and responsibilities for the data the application generated were regulated with contracts. The customer wasn’t allowed to do upgrades; this was controlled by contracts.

“So, I guess, whatever your intentions are during the development of a product, they can easily be destroyed by early thoughts (or lack thereof) on how to handle your customers.”

Ola Ellnestam, CEO, Owner, and Agile Coach at Agical AB, Sweden

---

Another example of policy-driven waste was depicted in the time-to-market value stream map in Frame 3: End-to-End Flow (see Figure 1-4). A quick glance showed that three long queues were the cause of most of the delay in getting products to market, yet the organization was blind to them. This was probably because the queues were not the responsibility of any department manager; their real purpose was to buffer departments from the variation of neighboring departments. Failure to see the impact of queues is often caused by focusing exclusively on department-level performance or by trying to achieve high utilization of scarce talent. These policies leave most companies oblivious to the tremendous drag local optimization has on time-to-market and end-to-end flow and hence on cost, revenue, and yes, even utilization.

## The Five Biggest Causes of Policy-Driven Waste

In our experience, the most common causes of policy-driven waste in software development are

1. Complexity
2. Economies of scale
3. Separating decision making from work
4. Wishful thinking
5. Technical debt

### *Complexity*

In “No Silver Bullet” Fred Brooks wrote, “Software entities are more complex for their size than perhaps any other human construct. . . . Many of the classic problems of developing software products derive from this essential complexity and its nonlinear increases with size.”<sup>21</sup> We know this. And yet . . .

1. Our software systems contain far more features than are ever going to be used.<sup>22</sup> Those extra features increase the complexity of the code, driving up costs nonlinearly. If even half of our code is unnecessary—a conservative estimate—the cost of the system with that extra code is not just double; it’s perhaps ten times more expensive than it needs to be. Our best opportunity to improve software development productivity is this: Stop putting features into our systems that aren’t absolutely necessary.

---

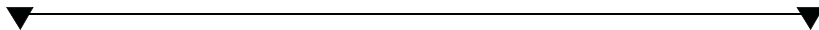
21. Brooks, “No Silver Bullet: Essence and Accidents of Software Engineering,” 1986.

22. More detail on this is provided in Poppendieck, *Implementing Lean Software Development: From Concept to Cash*, 2006, p. 24.

2. Many of our policies are predicated on the assumption that scope is non-negotiable. This leaves us no way to stop adding those unnecessary features. We need a process that lets us develop the first 20% of a system, get it in production, get feedback, and add features incrementally as time and money permit. We need policies that say: If something has to be compromised—cost, schedule, or scope—the default choice should *routinely* be scope.
3. Even our measurements give subtle messages that we should squeeze as much code into a system as possible. We measure productivity based on lines of code or function points, as if these things were good. They're not; they're bad. Function points might provide interesting *relative data*, but they should never be used as performance metrics.
4. We need to keep our code bases simple. This means we shouldn't add features until they are needed. Forget just in case; develop just in time. We need architectures that foster incremental development. We need policies that make refactoring—removing complexity introduced when changing the code—a normal and expected part of adding new features.
5. Our customers often want to use software to automate their complex processes. This is not a good idea. Business processes should be simplified first and automated later. How often do we help our customers simplify their processes before automating them?

The lean frame of reference focuses on simplicity. Lean thinkers know that complexity clogs up the flow of work and inevitably slows things down. The cost of complexity is hidden; it has a second-order effect on cost, so we just don't see it in our financial systems. This makes complexity all the more pernicious—it's hard to cost-justify spending money to keep things simple.

At the end of this chapter we introduce the concept of *ideation*, the process of coming up with a design so fitting to the problem that it seems inevitable. Solutions that “just fit” are necessarily simple; great designs always make us wonder how something so obvious could have escaped us for so long.



### **Cut Scope, Meet the Deadline**

“We were doing very well,” they said. “The code base was solid, our velocity was stable, we were proud of the way our system was coming along. But we weren't getting done fast enough for the management team. So one day our boss took the senior people aside and offered us a very large bonus if we could meet the deadline. We mean, it was *really* big.”

I was surprised at their candor, since they were presenting at a public conference.

“We agreed to go for it. We worked day and night and weekends. We shoveled code out as fast as we could. We stopped testing; it didn’t matter if it worked, it just had to be done. And we made it; we got the bonus.

“But the code is a mess. Nothing works right. The work-arounds are terrible. It’s going to take us years to clean it up.”

I wasn’t surprised. I could tell they were not proud of what they had done.

“What do you think you should have done?” came a question from the audience.

They were quick to answer: “We should have said *no*. We should have been late.”

I wasn’t so sure. I had been at their company a year before, and I admired their boss. He was a good manager in an impossible position, I suspected. I was pretty sure that missing the deadline was *not* an option.

Later at break I asked to the two speakers, “Wouldn’t it have been possible to cut scope and meet the deadline without abandoning your testing discipline?”

“Oh, yes, definitely!” came the immediate reply. “A lot of what we coded isn’t being used yet anyway. But we couldn’t. When you visited us a year ago, you told our management that they had to cut scope if they wanted to succeed. So a senior engineer went around to all the managers and got an agreement on moving a bunch of features to later releases. But once that was done, no one was willing to cut any more. We couldn’t even talk about it.”

I think that management was not close enough to the work to realize that delaying the implementation of major features was an easy and logical option. I expect they had no idea of the havoc they would create by setting impossible goals and leaving the team to figure out how to reach them. It was a very costly mistake.

In my experience, cutting scope and meeting the deadline is almost always the best approach.

Mary Poppendieck

---

### *Economies of Scale*

Many of our instincts, policies, and procedures are rooted in the economies of scale, which drove huge improvements in productivity as industrial production replaced craft production in the first half of the twentieth century. But during the second half of that century, it became apparent that in any system with high variety, the economies of flow outperform the economies of scale, even in manufacturing. Software groups develop one-of-a-kind systems—the essence of

variety. It should be obvious that we should base our policies and processes on the economies of flow. And yet . . .

1. It's difficult to abandon batch and queue mentality. We sort work into batches so we can assign each batch to the appropriate specialist, making maximum use of the specialist's time and skills. Full utilization of our most skilled workers is considered essential, and people are conditioned to focus on doing their part of the work without regard to its impact on the next step or the final customers. As a result, neither our workflow nor our workers are capable of absorbing variety.
2. It is so difficult to abandon batch and queue mentality that we fail to see queues that are staring us in the face. We can't figure out why it takes so long for things to move through our backlog-laden processes. We are blind to lists of customer requests that would take years to clear. We can't bring ourselves to shorten our queues because it would mean saying no to customers rather than letting their requests die a slow death.
3. Instead of designing a system that can absorb urgent requests, we pull workers off their current job to rush a yet-more-important job through our system. We ask people to work on three, five, ten, or more things at once. We are blind to the enormous amount of time wasted in context switching. It never occurs to us that if we did one thing at a time rather than three, everything would get done a lot faster and we would deliver value a lot sooner.
4. Sometimes we do let people work on one thing at a time, and then use computer systems to make sure that everyone is busy all of the time. We schedule projects and assign teams with an eye to full utilization. This scheme has little capability to absorb ever-present variation, so it is absorbed in the ramp-up time of newly formed teams. It would be much better to assign work to established teams than to reconstitute teams around projects.
5. We create annual budgets or long project plans that justify every person by committing to what they will deliver. Then we dump this big batch of work on our organization all at once. We must deliver everything that was promised, but as time goes on, reality intervenes. Customers want other things but aren't willing to pay more or give up what was promised. We know this system never works, but we don't see a way to escape the policy of making big batch promises.

Lean thinking uses economies of flow, rather than economies of scale, to frame the world we look at. Variety is an essential ingredient of software development,



and so we need processes that absorb the variety gracefully. We will discuss such flow processes in Chapter 3. The problem is, when the world is framed with economies of scale, these approaches seem counterintuitive.

### *Separating Decision Making from Work*

Any solution to a problem is necessarily simplified and abstracted when removed from its source, and it is for this reason that a design should not be separated from the concrete context in which it is implemented. We know that our deepest insights into our work are based on the tacit knowledge we get from being there: watching, experiencing, and getting our hands dirty. We know that great designs come from designers who are deeply engaged with solving the problem. We know that throwing things over the wall doesn't work. And yet . . .

1. There is widespread belief that it's not necessary for managers to understand work they manage. Yet without a technical background, managers are not in a position to provide guidance to technical workers. Some managers simply establish targets and leave it to workers to figure out how to meet them. Others put a team together and charter the members with figuring out how to do the right thing. From a lean perspective, the fundamental job of managers is to understand how the work they manage works, and then focus on how to make it better.<sup>23</sup> This is not to say that all leaders must know all the answers; the critical thing is that they *know what questions to ask*.
2. We have created organizational cultures where the only available career path is to leave the technical details behind. We do not honor or adequately reward the seasoned architect or brilliant user interaction designer. What future awaits a tech lead who can reinvent the testing process and put together a set of tools that runs every bit of code through every operating system and database every single night? When the only career path for these people is to leave their core expertise behind and become managers, we will never have top-notch technical leadership on the ground, where we need it.
3. In *Lean Product and Process Development*, Allen Ward writes that handoffs (handovers) are the biggest waste in product development. He says that a handoff occurs whenever we separate *responsibility* (what to do),

---

23. See Seddon, *Freedom from Command and Control: Rethinking Management for Lean Service*, 2005, Chapter 4.

*knowledge* (how to do it), *action* (doing the work), and *feedback* (learning from the results).<sup>24</sup> Our processes are full of these handovers, and we fail to see what's wrong with them. But in practice, many day-to-day decisions are based on tacit knowledge, which gets left behind in a handover. We must think that tacit knowledge transfers by magic, if we understand tacit knowledge in the first place.

4. Our language betrays us when we talk about “The Business.” With these words we separate development decisions from the work they are automating (see Figure 1-5).

Even agile software development methodologies make this mistake. They may recommend a “customer” or a “product owner” who is supposed to decide what all of our customers want and prioritize the order of development. But the most successful development occurs when developers talk directly to customers or are part of business teams. And those things called requirements? They are really candidate solutions; separating requirements from implementation is just another form of handover.

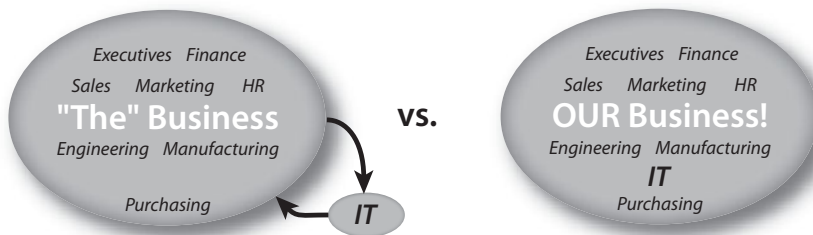


Figure 1-5 From “the” business to “our” business

5. Once our systems are deployed, there are policies and interpretations of laws (Sarbanes-Oxley, for example) that keep developers away from their code. So we walk away and leave the support team to deal with any problems that occur. The support team members are the ones who get the phone calls in the middle of the night, yet we wonder why they don't like frequent deployments. They are the victims of our risky design practices, but we are not interested in hearing about the causes of system failures. Perhaps our world would be a better place if all developers had to walk in the shoes of the operations and support team for a month every year.

24. Ward, *Lean Product and Process Development*, 2007, p. 43. We use the word *hand-over* instead of *handoff* because it seems to translate better into other languages.

Ever since Adam Smith wrote about division of labor in a pin factory,<sup>25</sup> it has been commonly accepted wisdom that division of labor increases productivity—the more specialization the better. Fortunately, this “fact” was lost on Toyota’s Taiichi Ohno, who devised a system where multiskilled workers and easy-to-reconfigure machines are more productive than specialists. There are two reasons for this: First, lean systems are designed to absorb variety, and second, they are designed to be relentlessly improved as the workers devise ever better ways to do the work. In the context of system development, Adam Smith was dead wrong.

## Developer on Site

What CIO wouldn’t love to read this kind of article?

### ***Agile Development and SOA at Standard Life***<sup>26</sup>

Thursday, 18th October 2007

*Investors Chronicle* is not famed for its hyperbole, but it certainly likes what it sees in Standard Life. . . . its rise has been “little short of meteoric”, the magazine gushes. Half-year financials show how huge leaps in efficiency led to a 71% jump in operating profit and a 31% rise in new business.

A great deal of the credit for that gilded resurgence is being placed at the door of Standard Life’s IT organization. . . . It has created one of the most advanced implementations of service-oriented architecture, put into practice lean processes and radically shaken up the structure of the development of its core applications, with IT staff “embedded” within the business units they deliver to.

We were pretty impressed, because we had been there two years earlier. We read on:

Keith Jones, CIO recalls: “The light bulb moment for me was when [Mary Poppendieck] said that something like 60% of all code that is delivered is never exercised. I thought that was astonishing, that there must be some terrible companies out there doing some really bad things. Then she said, ‘By the way, we have done a sample of Standard Life’s projects and it’s something like 64% for you guys’.”

The article goes on to say that the leadership group asked themselves why they had analysts and developers located in a different place from their customers. Why not have the people building the software located with the business team, listening to what customers are struggling with, talking with the people creating the business case? So Standard Life created one department, roughly half IS staff and half pensions people, all sitting together and working as a team.

25. Smith, *An Inquiry into the Nature and Causes of the Wealth of Nations*, 1776.

26. Swabey, “Agility Applied at Standard Life,” 2007.

Putting developers on the business team is a successful pattern. We once taught a class at a bank that scanned mortgage closing papers to archive them digitally. It had a large lean effort for the paper scanning process; the problem was the lean teams could not get anyone to make changes to the workflow software that governed their work. The changes seemed so minor that they couldn't get enough priority. The leader of the lean effort complained, "In the past 18 months, I have submitted 1536 requests for changes to the software, and not *one* of them has been implemented."

I knew the workflow software they were using and I was pretty sure that it was relatively easy to configure. So I suggested that they put a couple of developers on the operations teams for a while and just let them help their teammates out. We later heard that this approach was very successful, and much of the software development work is now done by developers who are embedded in cross-functional business teams.

Mary Poppendieck

### *Wishful Thinking*<sup>27</sup>

Frederick Winslow Taylor got one thing right. He insisted that work improvement should be based on the scientific method. Taiichi Ohno embraced this idea—but instead of having “experts” measure and improve the work of production workers, he trained production workers to measure and improve their own work. We know that making decisions based on data rather than opinion is the right approach. Being in software, we can create tools to gather any data we want any day of the week. And yet . . .

1. We chase the latest ideas in software development without bothering with the scientific method. We think it is a waste of time to understand the theory, create hypotheses, run experiments, gather data, and find out what really works in our environment. We fail to appreciate that “best practices” are somebody else’s solutions to their problems, not necessarily the right solutions to our problems. We adopt new development approaches with an unhealthy dose of wishful thinking, rather than determining the most appropriate practices for our environment—and then we are surprised at the disappointing results.
2. We manage by looking at single data points instead of a series of data in context. We set targets without understanding our process capability relative

---

27. The idea of wishful thinking as waste, as well as many other ideas in this section, are from Allen Ward (*Lean Product and Process Development*, 2007).

to the target. We don't appreciate the fact that trying to remove normal (common-cause) variation will make the situation worse, not better.

3. We don't like uncertainty, so we try to make decisions and get them out of the way. Our natural inclination is to look at one alternative for solving a problem, because we think that's cheaper and faster than looking at several alternatives. For tough problems, this is usually wrong; making early decisions when we are the most ignorant is the least likely way to get good results and the most likely way to force us to start over again.
4. Despite our prowess in handling information, we have very few techniques for preserving knowledge. One approach has been to collect massive, detailed documents. But who reads them? Even search engines fail us. Another approach has been whiteboards, coupled with a camera if we really need to save the sketches. Still another approach is to video a whiteboard talk on the fundamental architecture of an application. There is no doubt a middle ground, but we're still searching for it. We might learn a lesson from the open-source movement, where all communication is written and the focus is on making the communication system extremely simple, appropriately concise, quickly searchable, and never bypassed by verbal communication.
5. We feel a great sense of accomplishment when our code passes its tests and we celebrate because it meets the specification—as if the specification could contain everything we needed to think about. What about that security hole or the memory leak or the ungraceful exit from the database that occasionally causes a lockup? How easy will the system be to install, interface to, populate with data? Thinking we're done when the regression tests pass is wishful thinking.

When you think of learning as uncovering the shortcomings of plans, somehow plan-driven development loses its charm. Instead, creating useful knowledge becomes the essence of developing a new product. But there's more to learn about than the product being developed; we also learn about our process for developing products. Constant learning is the essence of improving both the product itself and the product development process.

### *Technical Debt*

We know that all successful software gets changed.<sup>28</sup> So if we think we're working on code that will be successful, we know we need to keep it easy to change.

---

28. See Brooks, "No Silver Bullet: Essence and Accidents of Software Engineering," 1986.

Anything that makes code difficult to change is technical debt.<sup>29</sup> We know that technical debt drives the total cost of software ownership relentlessly higher, and that eventually we will have to pay it off or the system will go bankrupt. And yet . . .

1. We tolerate obscure code, instead of making sure that all code reveals its intentions to the next person who comes along. Developers, especially apprentices, should be taught how to write “clean code”:<sup>30</sup> code that is simple and direct, with straightforward logic. Senior technical people need to ensure that messy code, even if it passes the tests, is never admitted into the code base.
2. Far too often we don’t take the time for refactoring: consolidating changes into existing code. Refactoring is essential for iterative development. Adding new features to existing code creates complexity, ambiguity, and duplication; refactoring pays down the debt.
3. We run regression tests on our systems before deployment. At first they are quick, but with each addition of code, regression tests take longer and longer and longer. As the regression deficit<sup>31</sup> grows, we increase the interval between releases. The only way to break this unending cycle of increasing release overhead is to decrease the regression deficit. If we had started with automated test harnesses back when the code base was small and added to and maintained them, we could make changes to our code almost as quickly today as when the code base was new.
4. We know that dependencies are one of the biggest generators of technical debt, and yet we are ambivalent about replacing obsolete systems with massive dependencies. We must develop, and migrate to, architectures that minimize dependencies. We have known for a long time how to do this: Focus on information hiding<sup>32</sup> and separation of concerns.<sup>33</sup>

---

29. The idea of debt as a metaphor was introduced by Ward Cunningham, “The WyCash Portfolio Management System,” 1992.

30. See Martin, *Clean Code: A Handbook of Agile Software Craftsmanship*, 2009. Definitions are from pp. 7–11.

31. We first heard the term *regression deficit* from Owen Rogers.

32. *Information hiding* means putting features that change together into a single module. See Parnas, “On the Criteria to Be Used in Decomposing Systems into Modules,” 1972.

33. *Separation of concerns* means keeping features that will change separately in different modules. See Dijkstra, “On the Role of Scientific Thought,” 1982.

5. We branch code for many reasons: to isolate new development, to focus on an individual application, to create parallel feature sets. And we know that the longer two branches of code are apart, the harder they will be to merge. And yet we wait for days to build our code, and worse, we delay system testing until the end of development. We don't realize that this isn't necessary anymore; the big bang is obsolete.

We need to expose technical debt for what it is: a costly burden to be avoided lest it lead us into bankruptcy. Chapter 2 will look at software development through a technical frame and discuss solid techniques for avoiding technical debt.

---

## Portrait: Product Champion, Take 1

We close out this chapter by taking a look at what is probably the most important part of the system development process—even though it happens before most people would think the development process has begun. Before you dive into developing a system, it is critical to define its purpose *in customer terms*, just as you defined the purpose of your organization in customer terms earlier in this chapter.

When Thomas Edison invented the light bulb in 1879, it was a rather useless novelty. So he invented an electric power distribution system, formed a company to distribute power, and built a steam power-generating plant—to make the light bulb broadly useful. Edison's genius lay in his ability to envision how people would want to use light bulbs and to imagine a fully developed marketplace. We saw this same genius in Steve Jobs, as the iPod and iPhone came to life as complete ecosystems. This is the essential challenge of development: imagining how people will want to use a product and envisioning a complete system and fully developed marketplace. We call this *ideation*.<sup>34</sup>

Just as Edison went on to found the companies that brought his light bulb to the masses, we expect that ideation leaders will remain at the helm as their concepts are implemented. After all, people develop a certain passion around their creative ideas and are eager to bring them to life. In honor of that passion and dedication, we call this leader a *product champion*.<sup>35</sup>

---

34. The term *ideation* is from Tim Brown, CEO of IDEO, as described in Brown, "Design Thinking," 2008.

35. The term *product champion* is borrowed from 3M, where Mary worked for 20 years. The term *chief engineer* is widely used for the same role and is interchangeable with *product champion*.

A product champion, much like an entrepreneur, has business responsibility for the success or failure of the product. This means that for a product with a profit-and-loss statement (P&L), they are responsible for the P&L of that product. This is why the product champion leads the ideation effort—if ideation is not done well, the product will not be successful.<sup>36</sup>

---

## The Story of a Chief Engineer

Mr. Nobuaki Katayama, former chief engineer of the Lexus/SC, IS, & Altezza, took the time to discuss the job of a chief engineer with a group of us on a study tour of Japan in April 2009.<sup>37</sup> He noted that new car development is led by a chief engineer who is responsible for the business success of the car. This person should have a passion for the car; so if the car is to be a sports car, the chief engineer should *love* to drive fast.

Toyota develops new cars in three phases: creation, development, and production.

**Creation:** The creation stage of a new car is the backbone of the development process, and also the most difficult part. During this stage planning and concept development take place, including market research, predevelopment, styling, and cost and profit targets. The chief engineer negotiates with section heads (heads of body styling, engine, transmission, etc.) concerning what kind of performance and key features the car will have, what it will take to develop the car, and so on. Creation is not timeboxed; work continues until the concept is ready and typically takes about a year. As handshake agreements are reached, the product concept is refined and reviewed and, ultimately, approved by the board.

**Development:** This is the easier part of creating a new car, and Toyota is good at it. A schedule of 20 to 24 months is established by the chief engineer, who leads the work, with support from section managers. There are clear milestones that can be expected to be met, with, of course, more negotiations and ongoing design decisions.

---

36. See Levine, *A Tale of Two Systems: Lean and Agile Software Development for Business Leaders*, 2009, for an in-depth example of the role of a chief engineer in software development.

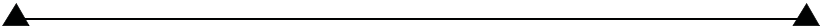
37. This meeting summary is used with permission. Some information in this sidebar is from a presentation by Kenji Hiranabe at Agile 2008, reporting on a presentation by Mr. Nobuaki Katayama to Developers Summit 2008, February 13, 2008, Gajoen, Tokyo (Hiranabe, “New Car Development at Toyota,” 2008). Used with permission.



Keeping development on schedule requires honest communication and a philosophy of “bad news first” so that problems can be addressed as early as possible. Key decisions during development are made by choosing among well-developed options so that the best combination of quality, cost, and delivery can be achieved.

**Production:** The chief engineer remains responsible for the business success of the car as it goes into production and over its lifetime.

Mary and Tom Poppendieck



For the sake of simplicity, we use the term *product* champion, although we recognize that you may not use the word *product* to refer to your systems. You may be developing

1. Software as a product
2. Software embedded in a product
3. Software enabling a process
4. Software under contract

In the first two cases, the product champion should be the person with business responsibility for the final product. With larger systems, the champion may need assistance with subsystem leadership. Even then, division into subsystems should not be along technology lines, but along subsystem lines, for example, an engine for a car, a medical device programmer, the human interface for an electronic device, and so on.

In the third case—software that enables a process—it is particularly important that the product champion (actually, in this case, the process champion) be responsible for the design and success of the overall process, not just the software.

The fourth case—software developed under contract—is the most problematic for a product champion, especially if the contracting party divorces software development from the rest of system development. In the end, software developed under contract serves a broader purpose, and it would be best to have a product champion responsible for the broader purpose guiding the learning and feedback necessary for system development. We recognize that this is not always possible, but in any case, the product champion(s) must keep the whole system in mind.

A product champion leads two critical activities: a customer-facing role and a technology-facing role. Very often these roles reside in one person, but they can also be successfully shared by two people working in close harmony. In either case, the product champion initiates development by leading a team through the ideation phase.

## Customer-Facing Ideation

IDEO is a design firm in California that has been extraordinarily successful at discovering unmet needs and matching them with technically feasible, commercially viable designs. These have led to a remarkable lineup of products that have been extremely successful in delighting customers. As design awards pile up year after year, IDEO has gone into the business of helping other companies copy its design process.<sup>38</sup>

IDEO's design approach is outlined by general manager Tom Kelley in *The Art of Innovation*:<sup>39</sup>

1. **Understand** the market, the client, the technology, and the perceived constraints on the problem. Later we often challenge those constraints, but it's important to understand current perceptions.
2. **Observe** real people in real life situations to find out what makes them tick; what confuses them, what they like, what they hate, where they have latent needs not addressed by current products and services.
3. **Visualize** new-to-the-world concepts and the customers who will use them. Some people think of this step as predicting the future, and it is probably the most brainstorming-intensive phase of the process.
4. **Evaluate** and refine the prototypes in a series of quick iterations. We try not to get too attached to the first few prototypes, because we know they'll change. No idea is so good that it can't be improved upon, and we plan on a *series* of improvements. . . . We watch for what works and what doesn't, what confuses people, what they seem to like, and we incrementally improve the product.
5. **Implement** the new concept for commercialization.

We can find no better summary of how to go about ideation. First frame the problem with its constraints. Then become an ethnographer and carefully observe people in that frame. This isn't about focus groups or market studies; go and watch the people who will use the product. Get inside their heads.

The next step is to visualize, model, and discuss what was observed. Add to the mix a forward-looking view of technology trends over the next few years. You want to skate to where the puck is going,<sup>40</sup> and our technology puck moves fast. Brainstorm, concoct scenarios, tell stories about customers, whip up some prototypes, keep ideas alive.

---

38. This section summarizes sections of Brown, "Design Thinking," 2008. See also Brown, "Strategy by Design," 2007.

39. From Kelley and Littman, *The Art of Innovation*, 2001, pp. 6–7.

40. Hockey star Wayne Gretzky gave the secret to his success: "I skate to where the puck will be, not to where it is."

Visualization leads to evaluation, a series of quick experiments that incrementally improve the product. “It doesn’t matter how clever you are, your first idea about something is never right,” says Tim Brown, CEO of IDEO. “So the great value of prototyping—and prototyping quickly and inexpensively—is that you learn about the idea and make it better.”<sup>41</sup>

---

### So *That’s* How They Do It!

I have always loved cooking, and for years my kitchen was equipped with the same gadgets that I had grown up with as a child. Then a company called OXO invented a new kind of measuring cup. It’s called an angled measuring cup, and it has an indentation so you can read the liquid level by looking *inside* the cup. I immediately loved it. No more bending over or holding the cup up high to read the liquid level on the outside of the cup. I could simply glance down inside the measuring cup and see the level. I bought several for my house and plenty for gifts.

Then I started seeing other OXO products: A peeler that felt a lot better in my hand. Utensils that were easy to store. Great cutting boards. Soon my kitchen was full of OXO gadgets. I wondered, “How can a company routinely come out with improvements on cooking tools that have been around for decades? How did they know that I would rather not bend over to measure liquids, that I prefer a bigger handle on a peeler? I didn’t even know it myself!”

Then I read about how IDEO does ethnography; they call it archaeology. They go into houses of people like me and watch me using a measuring cup. They look for times when I’m a bit uncomfortable—bending over or holding a full cup high in the air. They notice when I pick up the peeler that I use an uncomfortable grip. And they invent a gadget that gets rid of these annoyances that I didn’t even know were there.

So *that’s* how they do it! Simple enough, really. You’d think that if we used the same approach, we might get similar results—fantastic systems that solve problems our customers didn’t even know they had.

Mary Poppendieck

---

In due time, a product concept is ready for implementation. Ideation should not be a long, drawn-out affair, and the resulting concept should be at a high level to leave plenty of room for further learning.

---

41. Brown, “The Deans of Design,” 2006.

## Technology-Facing Ideation

Envisioning the product from a customer perspective is not enough in software development. Without an equally effective technology vision—let’s call it an architecture—you aren’t ready to proceed. The same design steps that worked to create a customer-centric view of the product can be used to develop the architectural vision:

1. **Understand:** Technology is forever changing. Start by understanding where it has been and, especially, where it is likely to go over the life of the product.
2. **Observe:** Take some time to observe the people struggling with the problem, from a technology point of view. The technical-facing concept and the customer-facing concept gain their integrity as a system when they are developed together and inform each other.
3. **Visualize:** Model, discuss, brainstorm, test ideas with spikes.<sup>42</sup> Inform these discussions with a keen awareness of where technology is heading and what will be possible over the life of the product.
4. **Evaluate:** Don’t get caught up in the first idea that comes to mind. Experiment. Select three or four options and use set-based design during implementation to make the final choice. Above all, remember that a good software architecture is one that facilitates change in the code over the short term and evolution of the architecture over the long term.
5. **Implement:** At this point the architecture is a technical vision that will grow and evolve as development proceeds and learning takes place. It’s time to start implementation.

The ideation phase will be done when it is done; it shouldn’t have a deadline. It’s not that ideation takes a lot of time; it usually doesn’t. But without a breakaway concept, there can be no breakaway product. So don’t short-change this important step. Develop a clear vision of how the product will meet market needs, how the architecture will support that vision, and how development will proceed. We call this a *product concept*. A product concept is not a detailed plan; it is a framework for proceeding with development.

What is the difference between a detailed plan and a framework for proceeding? Think of it this way: Plans are subject to change as learning occurs, whereas frameworks provide a space for learning to occur. So if you find that

---

42. A *spike* is a quick technical prototype to test out the viability of a technical approach.

you have to make substantial changes to the product concept after approval, it was too detailed and did not provide for learning.

How do you know when ideation is done? This differs from one context to another, so we don't have a canned answer. In companies with good ideation processes, the product champion knows when the concept is developed sufficiently to move on to implementation. If you aren't quite sure, you'll have to experiment and find out what works for you. We recommend that you start your experiments with a bias toward action. If you wonder whether you're ready to start implementation, the answer is probably yes.

---

## Your Shot

1. Answer John Seddon's five questions about your organization:
  - a. **Purpose:** What is the purpose of this organization?
  - b. **Demand:** What is the nature of customer demand?
  - c. **Capability:** What is the system predictably achieving?
  - d. **Flow:** How does the work work?
  - e. **System conditions:** What are the causes of waste in the system?
2. Choose two to five customer-centric measurements for your system. Some you might consider are
  - a. Time-to-market for product development (for the whole product)
  - b. End-to-end response time for customer requests (request-to-resolution time)
  - c. Success of the product in the marketplace (profitability, market share)
  - d. Business benefits attributable to a new system (measurable business improvement)
  - e. Customer time-to-value after delivery (consumability)
  - f. Impact of escaped (post-release) defects (customer downtime, financial impact)
3. Create time series charts for each of your customer-centric measures. (Some people call these charts the "voice of the process.") If the data doesn't exist, now would be a good time to start measuring.
  - a. What do the charts say to you?
  - b. Are your work processes stable?
  - c. Can you distinguish between common-cause and special-cause variation?
  - d. Is the way you do work delivering the results your customers expect?
  - e. If not, what should be done?
4. Analyze how you handle requests from your customers:
  - a. How much value demand do you receive in a month?

- b. How much failure demand do you receive in a month?
  - c. What is the ratio of failure demand to value demand?
  - d. What kind of approval process do you use to filter customer demand?
  - e. What criteria are used?
  - f. What percentage of the requests are accepted?
  - g. How long does the approval process take?
  - h. How long on the average does it take for customers to find out about rejected requests?
  - i. Do you measure after the fact to see that the projected customer outcomes are achieved?
  - j. How long does an average approved request take to complete?
  - k. What do your customers think about your approval process?
5. Gather a team that includes the person responsible for an end-to-end process, if that person exists. Sketch a value stream map of an actual end-to-end flow of a customer demand through your organization's various processes, through the customer's processes, and back to the original customer.
- a. What is the total cycle time?
  - b. How much of that time was spent adding value?
6. Have the team brainstorm the goals, policies, and beliefs that cause waste that might exist in your organization in each of the five categories:
- a. Complexity
  - b. Economies of scale
  - c. Separating decision making from work
  - d. Wishful thinking
  - e. Technical debt
7. How does ideation take place in your company?
- a. Does your organization have the role of product champion or chief engineer?
  - b. How do you move ideas from the fuzzy front end of product development into an approved product concept?
  - c. How does it work for you?

# Index

## A

A3 problem-solving report, 176–180

Abe, Namiko, 187

Abstraction. *See* Information hiding.

Acceptance tests, 73–74

Adaptive control, 145

Adaptive control Frame 12, 143–147

consumability, 145

customer feedback, 143–144

customer outcomes, 146–147

escaped defects, 146

frequent releases, 144–145

Adler, Nancy, 191

Agical AB, 25

Agile development

cross-functional teams, 69

future of, 61–63

in large systems with tightly  
coupled architecture, 65

Agile Manifesto, 192

Agile@IBM

changing the business process, 224

continuous integration, 226

creating a sense of urgency,  
229–230

an early experiment, 223

focus on customer outcomes, 228

governance, 230

key characteristics, 221

key problems, 224–225

lessons learned, 223–227

stakeholder feedback, 226

stakeholder involvement, 222

support call reduction, 223,  
228–229

technical debt, 226

test automation, 227

from theory to practice, 228–230

training teams, 225–226

transformation, 220–221

understanding “done,” 226

WebSphere Service Registry and  
Repository team, 223

Airline industry. *See* Southwest  
Airlines.

Alcoa safety record, 156, 159–161

Alignment Frame, 236–241

Amazon.com

experiencing the workplace, 172

IT organization, statement of  
purpose, 9

team size, 66–67

waste, identifying, 172

Ambiguity, 157–158, 166, 190

Arbitrating with value, 134

Architectural vision, ideation, 41–42

ARPANET, history of software  
development, 57–60

*The Art of Innovation*, 39

Articles. *See* Publications.

Assembly language, history of  
software development, 52

## B

B-17 bomber, case study, 154–155

Backus, John, 52

Bad news first, 38, 169–170

Balzer, Robert, 56

Barrier to rapid learning, 161

- Baseline, establishing
    - boundary-spanning problems, 163
    - overview, 163
    - workflow, designing
      - connections, 164–165
      - handovers, immediate failure detection, 165–166
      - immediate customers, 164
      - immediate suppliers, 164
      - methods, 164
      - output, 163, 164, 166–167
      - pathway, 164
      - process standards, 167–169
      - test-driven handovers, 165–166
  - Batch and queue mentality, 29, 239
  - Batch size, 54, 62, 127, 144, 158–159
  - Batching work, 72, 114–127, 132–133
  - BBRT (Beyond Budgeting Roundtable), 233–234. *See also* Budgeting problems.
    - leadership principles, 234
    - overview, 230–231
    - process principles, 234–235
    - productivity, 235–236
  - Beck, Kent, 71–72
  - Benchmarking, 161
  - Beyond Budgeting*, 233–234
  - Bezos, Jeff, 67, 172
  - Black box design, 64
  - Boehm, Barry
    - on assembly language, 52
    - controlling complexity, 64
    - software development processes, 54
  - Bogsnes, Bjarte, 231
  - Bonuses, Agile@IBM, 232
  - Booch, Grady, 81
  - Books. *See* Publications.
  - Boundary-spanning problems, 163
  - Boyd, John, 215
  - Brooks, Fred, 26, 63, 90
  - Brown, Tim, 40
  - Budgetary commitments, cause of waste, 29
  - Budgeting problems
    - bonuses, 232
    - cost management, 231–232
    - efficiency, 233
    - quality, 232
    - target-setting and evaluation, 232
    - timing, 232
    - trust, 231
  - Business process, 7, 27, 70, 146, 224
- ## C
- Cadence, kanban, 127–128
  - Capacity
    - arbitrating with value, 134
    - cost-benefit ratio of scheduled activities, 134
    - kanban, 128–129
  - Case studies. *See also* Agile@IBM; Southwest Airlines; Svenska Handelsbanken; Toyota.
    - Amazon.com, IT organization statement of purpose, 9
    - B-17 bomber, 154–155
    - continuous improvement. *See* Continuous improvement.
    - cross-cultural teams, 204–205
    - ethnography, 40, 85–86
    - expertise, developing, 91–92
    - I&CS (Instrumentation and Control Systems), 139–140
    - ideation, 39–40
    - IDEO, design process, 39–40
    - Intuit, 85–86
    - Orpheus Chamber Orchestra, 206
    - OXO, identifying customer needs, 40



- policy-driven waste, 25
- pride in workmanship, 210–211
- product champions, 37–38
- pull scheduling large systems, 139–140
- Quicken, 85–86
- ready-ready to be done-done, 118–120
- reliable delivery. *See* Empire State Building construction.
- self-organizing teams, 206
- Systematic, 118–120
- Tandberg, 210–211
- waste, causes, 32–33
- waste, cures, 32–33
- Cash flow thinking, 107, 240
- Cause and effect, 178, 236–242
- Cerf, Vinton, 65
- Champions. *See* Product champions.
- Change requests, customer focus, 10, 21
- Chasing the Rabbit*, 154, 156, 160–161, 181, 183
- Checklists, 154–156, 165, 169
- Chief engineer. *See* Product champions.
- Cho, Fujio, 153, 171, 246
- Christensen, Clayton, 236
- Code, social setting, 76
- Code branching, cause of waste, 10–11, 36
- Code clarity 93, 95–96
  - information hiding, 82
  - overview, 80–82
  - refactoring, 82
- Code reviews, developing expertise, 95–96
- Collaborative modeling, 84, 86–88
- Collective decisions, 206
- “The Coming Commoditization of Processes,” 167
- “The Coming Commoditization of Processes,” 167
- Commitment, 68–69, 108, 179, 201–202, 238
  - deferring, 193
  - kanban, 126–127
  - shape the details, 129–130, 139
- Company culture. *See also* Knowledge workers.
  - collective decisions, 206
  - cross-cultural teams, 204–206
  - customers, respect for, 203–204
  - diversity, value of, 205–206
  - groupthink, 205–206
  - internal customers, respect for, 203–204
  - managers, respect for, 204
  - mutual respect, 193, 203–209
  - need for consensus, 204–205
  - pride of workmanship, 209–211
  - purpose-passion-persistence-pride, 210
  - reciprocity, 200–203
  - remuneration, 201–202
  - self-organizing teams, 206–209
  - subordinates, respect for, 204
  - suppliers, respect for, 204
  - teammates, respect for, 204
  - Toyota Motor Corporation, 195–196
- Competency leaders, 41, 95–97, 182
- Complexity, 35, 41, 47, 51, 53, 62, 80
  - cause of waste, 26–28
  - constraint, 109
- Conway’s Law, 67–70
- cross-functional teams, 68–69
- divide and conquer
  - black box design, 63–64
  - Empire State Building construction, 112
  - error recovery, 64

Complexity (*continued*)

- gateways, 64
- Internet architecture, 64–65
- network connectivity, 64
- by process, 64
- by responsibility, 64
- routers, 64
- by structure, 64
- by value, 64
- effects of, 154
- in hospitals, 154–155
- inherent, 63
- Internet architecture, 64–65
- learning about, 161
- low-dependency architecture, 65–67
- schedules, reducing, 112–114, 137
- software development, separating
  - from larger system, 69–70
- work design, 163
- “Complexity Controlled by Hierarchical Order . . .”, 47
- Conformance to plan, question for leadership teams, 238–239
- Connections, workflow design, 164–165
- Consensus
  - need for, 153, 176, 193, 204–205
  - pull-based authority, 180
- Constraints, 7, 39, 108–109, 166
  - exposing risks, 109–110
  - scheduling, 129, 138
- “A Constructive Approach to . . . Program Correctness,” 47
- Consumability, 12, 145, 222, 229–230
- Context switching, cause of waste, 29
- Continuous improvement
  - exposing problems
    - bad news first, 169–170
    - experiencing the workplace, 172
    - test failures, analyzing, 169–170

## hospitals

- ambiguity, 157–158
- checklists for procedures, 154–156
- complexity, effects of, 154
- filling prescriptions, 157–158
- medical and medication errors, 154
- missing medications, 158–159
- organizing patient charts, 156–157
- quick experiments, 158–159
- work-arounds, 156
- learning to improve
  - A3 problem-solving report, 176–180
  - directive management, 184
  - disincentives for sharing knowledge, 182
  - fishbone diagrams, 179
  - five whys, 178–179
  - goal of, 173–174
  - managers as mentors, 183–184
  - problem owners, 180–181
  - problem solving, 174
  - problem/countermeasure board, 174–175. *See also* Kanban boards.
  - pull-based authority, 180–181
  - responsibility authority, 181
  - retrospectives, 173
  - root causes, identifying, 179–180
  - self-organizing teams, 184
  - sharing knowledge, 181–182
  - suggestion systems, 175–176
- Toyota Motor Corporation, 153, 195
- visualizing perfection
  - Alcoa’s safety record, 159–160
  - barrier to rapid learning, 161

- benchmarking, 161
- customer focus, 162
- high-velocity organizations, 161–162
- low-velocity organizations, 161
- theoretical limit, 160–161
- Toyota Motor Corporation, 160–161
- variance, as a learning opportunity, 162
- workflow across boundaries, 162
- Continuous improvement,
  - establishing a baseline. *See also* Workflow, designing.
  - boundary-spanning problems, 163
  - overview, 163
- Continuous integration
  - Agile@IBM, 226
  - code, social setting, 76
  - defect injection processes, 76
  - frequency of, 78–80
  - history of software development, 49
  - integration testing, time budget for, 77
  - step-wise integration, 76–77
  - stress testing, system level, 79–80
  - user acceptance tests, 80
- Contracts, 25, 38, 108, 204, 234
  - design by, 50, 72
  - development, 231
  - effect on flexibility, 88, 103, 110–111
- Converse, Donald, 2
- Convis, Gary, 162, 246
- Conway, Mel, 67–70
- Conway's Law, 67–70, 92
- Cost management
  - budgeting problems, Agile@IBM, 231–232
  - questions for leadership teams, 237–238
- Cost-benefit ratio of scheduled activities, 134
- Crescêncio, Samuel, 96
- Critical path schedules, 112–114, 140
- Cross-cultural commonalties, 188–190. *See also* Company culture.
- Cross-cultural factors, 193
- Cross-cultural psychology, 188–191. *See also* Company culture.
- Cross-cultural teams, 204–206. *See also* Company culture.
- Cross-functional teams, 24, 33, 68–70, 88, 132, 142, 163–164, 176
- Cultural assumptions, 188–191. *See also* Company culture.
- Cunningham, Ward, 58–59, 81
- Customer feedback, adaptive control, 143–144, 219, 228, 232
- Customer focus, Frame 1, 6–12
  - change requests, 10
  - customer needs, identifying, 40. *See also* Ideation.
  - customer-centric measurements, 12
  - customers, identifying, 6
  - failure demand, 10–12
  - product managers, 8–9
  - product owners, 8–9
  - product purpose, 9
  - respect for customers, 203–204
  - support calls, 10
  - Toyota Motor Manufacturing Kentucky, 162
  - types of, 7–8
  - value demand, 12
  - visualizing perfection, 162
  - whole-system view, 6
- Customer outcomes, 17, 112, 134, 142–143, 146–147, 200
  - adaptive control, 146–147

Customer outcomes (*continued*)  
 Agile@IBM, 228  
 BBRT principles, 234–236  
 Customer-centric measurements, 12  
 Customer-facing ideation, 39–40  
 Cycles, evolutionary development.  
*See* Evolutionary development,  
 Frame 7.  
 Cycles of discovery, 88

## D

Dahl, Ole-Johan, 51  
 Data hiding. *See* Information hiding.  
 Davenport, Thomas, 167  
 Decisions  
   cross-cultural factors, 194  
   premature, cause of waste, 34  
   separating from work, cause of  
     waste, 30–33  
 Decoupling activities, 106, 132  
 Deep Expertise, Frame 8. *See*  
 Expertise.  
 Defect avoidance, 71–76  
 Defect injection processes, 73, 76  
 Deliberate practice, 92–93, 95, 145  
 Deming, W. Edwards, 14, 16, 209  
 Deming approach, 199, 241  
 Denning, Peter, 83  
 Dependencies  
   cause of waste, 35, 60, 65, 81, 166  
   schedules, 112  
   between teams 113, 132–133  
 Design, separating from  
   implementation, 54–56  
 “Design and Code Inspections . . .”,  
 53, 62, 70  
 Design loopbacks, 110–114  
 Detailed command *versus* mission  
   command, 214–215

Development processes, history of, 54  
 Dijkstra, Edsger W.  
   “A Constructive Approach to . . .  
     Program Correctness,” 47  
   on COBOL, 53  
   “Complexity Controlled by  
     Hierarchical Order . . .”, 47  
   controlling complexity, 64  
   design, separating from  
     implementation, 54–56  
   on impact of high-level languages,  
     53  
   information hiding, 51  
 Directive management, 184, 214  
 Disincentives for sharing knowledge,  
 182  
 Disruptive technologies, history of, 59  
 Distribution, Internet architecture, 65  
 Diversity, value of, 205–206  
 Divide and conquer. *See* Complexity,  
 divide and conquer.  
 “Done-done,” understanding, 118,  
 120, 165, 226

## E

Economies of scale, 4, 9, 28–30  
 Efficiency, 66, 118, 224, 233, 239,  
 243. *See also* Process cycle  
 efficiency.  
*The Elegant Solution*, 87  
 Ellnestam, Ola, 25  
 E-mail, history of, 59  
 Empire State Building construction.  
*See also* Reliable delivery.  
 cash flow thinking, 107  
 complexity, reducing by  
   decomposition, 112  
 constraints exposing risks, 109  
 contracts, effect on flexibility, 111

- Empire State Building construction
  - (*continued*)
  - decoupling activities, 106
  - design loopbacks, 110–111
  - design mistakes, 111
  - designing effort to fit constraints, 109
  - four pacemakers, 104–106
  - implementation complexity, 112–114
  - logistics, 106–107
  - overview, 102–103
  - proven experience, 108–114
  - subcontracting, 112
  - system design, 109–111
  - team design, 103–104
  - workflow, 104
- Empire State Building construction, schedules
  - alternate approaches, 113–114
  - critical path, 113–114
  - dependencies, 112
  - per workflow, 104–106
  - pull scheduling, 129
  - push scheduling, 129–130
  - reducing complexity, 112–114
  - set-based design, 113–114
  - utilization, 113
- End-to-end Flow, Frame 3, 19–24,
  - process flow maps
  - evaluating, 19
  - failure demand, eliminating, 19–21
  - process flow maps
    - definition, 19
    - of failure demand, 19–21
    - improving process capability, 24
    - of value demand, 21–24
- Error recovery
  - complexity, 64
  - history of software development, 54
  - Internet architecture, 64
- Escaped defects, 146
- Essential Complexity, Frame 5, 63–70. *See also* Complexity.
- Establish a Baseline, Frame 14, 163–169. *See also* Baseline, establishing.
- Ethnography
  - case study, 40, 85–86
  - identifying customer needs, 40
  - phase of evolutionary
    - development, 84–86
- “Evolutionary Development,” 57
- Evolutionary development. Frame 7, 83–88
  - collaborative modeling, 84, 86–87
  - cycles of discovery, 88
  - ethnography, 84–86
  - history of, 57–61
  - iterative development, 87–88
  - Moore’s Law, 83
  - overview, 84
  - prototyping, 86–87
  - quick experimentation, 84, 87–88
  - set-based design, 87–88
  - types of, 83
- Experiencing the workplace, 172
- Experiment, 21, 23, 40–42, 87, 158, 225
- Expertise, 89–95
  - developing
    - case studies, 91–92
    - code reviews, 95
    - competency leaders, 95–97
    - deliberate practice, 92–93
    - maestros, 96–97
    - overview, 91–92
    - retention, 94
    - standards, 94–95
    - ten-year rule, 93
    - turnover, 94
  - importance of, 90–91

Exploratory testing, 75  
 Exposing problems Frame 15,  
 169–172. *See also* Problems,  
 exposing.

**F**

Fagan, Michael, 53  
 Failure, main cause, 86  
 Failure, testing to, 75–80  
 Failure demand  
   customer focus, 10–12  
   eliminating, 19–21  
   process flow maps, 19–21  
   support calls, 10  
   waste caused by, 21, 31  
 Feathers, Michael, 81  
 Fifer, Julian, 206  
 Filling prescriptions, 157–158  
 Finances, question for leadership  
   teams, 240  
 Fishbone diagrams, 179  
 Five whys, 178–179  
 Flight 1549, 91  
 FLOW-MATIC language, 52  
 Four pacemakers, 104–106  
 Framing  
   for business success, xviii–xix  
   mental constructs, xviii–xix  
   system development process,  
   xviii–xix  
 Fraser, Robin, 233  
 Freeman, Peter, 47  
 Frequency of, 78–80  
 Frequent releases, adaptive control,  
 144–145  
 From Theory to Practice, Frame 21,  
 228–230  
 Front-line leaders, 212–216

Full utilization of manpower, cause of  
 waste, 29, 239–40

## G

Gamma, Erich, 72  
 Gateways, 64  
 GE, leadership traits, 95  
 Gibson, Paul, 221, 223  
 Gilb, Tom  
   divide by value, 64  
   evolutionary development, 57, 84  
   on system design, 109–110  
 Goals. *See also* Targets.  
   history of software development,  
   61–63  
   of learning to improve, 173–174  
   motivation for unethical behavior,  
   17  
   pulling from the future, 18  
   relative measures, 17  
   setting targets, 15–17  
   for system capability, 15–18  
 “Goals Gone Wild,” 16  
 Governance, Frame 22, 230–236,  
   budgeting problems  
 Groupthink, 205–206

## H

Handoffs. *See* Handovers.  
 Handovers  
   cause of waste, 19, 24, 30–31, 111  
   immediate failure detection,  
   163–166  
   test-driven, 165–166  
 Hierarchical layers, history of, 48  
 Higashi, Kan, 246

- High-level languages, history of, 52–53
- High-velocity organizations,  
161–162, 167
- Hill, Ployer, P., 154–155
- History of software development  
agile development, future of, 61–63  
ARPANET, 57–60  
assembly language, 52  
consistent goals, 61–63  
continuous integration, 49  
design, separating from  
implementation, 54–56  
development processes, 54  
disruptive technologies, 59  
early error correction, 54  
e-mail, 59  
evolutionary development, 57–61  
FLOW-MATIC language, 52  
hierarchical layers, 48  
high-level languages, 52–53  
information cascades, 46–47  
Internet, 57–60  
milestones, 58  
object-oriented programming,  
50–52  
open source, 60–61  
personal computers, 57–60  
plank road analogy, 46–47  
project management *versus* system  
development, 62  
Simula language, 51  
Smalltalk language, 51  
software life cycle concept, 53–57  
spreadsheets, 58–59  
step-wise integration, 49  
structured programming, 47–50  
test-driven development, 48  
top-down programming, 48–49  
Y2K problem, 61
- Hooper, Grace, 52
- Hope, Jeremy, 233
- Hospitals, continuous improvement.  
*See* Continuous improvement,  
hospitals.
- Hughes, Chris, 147–149
- I**
- IBM  
agile development. *See* Agile@IBM.  
evolutionary development, 57–61  
personal computers, 57–60  
software life cycle concept, 53  
top-down programming, 48–49
- I&CS (3M Instrumentation and  
Control Systems), 139–140
- Ideation  
architectural vision, 41–42  
case studies, 39–40  
customer-facing, 39–40  
definition, 36  
ethnography, 40  
product champion, 37–38  
product concept, 41  
prototyping, 40  
technology-facing, 41–42
- IDEO, design process, 39–40
- Immediate customers, 164
- Immediate suppliers, 164
- Immelt, Jeff, 95
- Implementation  
complexity, 112–114  
*Implementing Beyond Budgeting*,  
231  
“On the Inevitable Intertwining of  
Specification and  
Implementation,” 56  
*versus* preparation, 194  
separating from design, 54–56

- Implementing Beyond Budgeting*, 231
- Improvement. *See* Continuous improvement.
- Individualism, cross-cultural, 189
- Information cascades, history of, 46–47
- Information hiding, 35, 82
- Inherent complexity, 63
- Inside Intuit*, 85–86
- Integration. *See* Continuous integration.
- Integration testing, time budget for, 77
- Internal customers, respect for, 203–204
- International Dimensions of Organizational Behavior*, 191
- Internet architecture
  - black box design, 64
  - complexity, 64–65
  - distribution, 65
  - error recovery, 64
  - gateways, 64
  - history of, 57–60
  - network connectivity, 64
  - routers, 64
  - TCP/IP protocol, 64
- Intuit, case study, 85–86
- Ishikawa, Kaoru, 169
- Iterations
  - capacity, 135
  - customer feedback, 143–144
  - decomposing features. *See* Stories.
  - experimentation, 87
  - versus* kanban technique, 126–129
  - making work ready, 118–122
  - overlapping steps, 120–122
  - overview, 117–122
  - ready-ready to be done-done, 118–122
  - small batches, 114
  - stories, 118
  - testing, 39, 79
  - velocity, 128
  - workflow chart, 117–122
- Iterative development phase, 87–88
- J**
- Jackson, Michael, 56–57
- Jishuken (voluntary self-study), 199
- Jobs, Steve, 36
- Johnson, H. Thomas, 199
- Jones, Bassett, 103
- Jones, Keith, 32–33
- Journals. *See* Publications.
- K**
- Kahn, Robert, 65
- Kaizen. *See* Continuous improvement.
- Kanban boards, 123–125
- Kanban cards, 122–123
- Kanban technique
  - batch size, 127
  - cadence, 127–128
  - capacity, 128–129
  - commitment, 126
  - definition, 122
  - versus* iterations, 126–129
  - kanban boards, 123–125
  - kanban cards, 122–123
  - process description, 122–126
  - teamwork, 127
  - throughput, 129
  - velocity, 128
- Katayama, Nobuaki, 37
- Kelley, Tom, 39, 85–86



- Kerr, Steven, 182  
 Kessler, Carl, 7, 222  
 Kniberg, Henrik, 22–24, 126  
 Knowledge. *See also* Ethnography.  
   domain, 131  
   loss at handovers, 30–31  
   mentoring, 184  
   preservation, cause of waste, 34  
   product development, 34  
   sharing, 181–182  
 Knowledge workers Frame 17. *See also*  
   People in lean development.  
   building on strengths, 197–198  
   competency leader, 95  
   definition, 196  
   Deming approach, 199  
   jishuken (voluntary self-study), 199  
   kaizen events, 199  
   *versus* manual workers, 197  
   problem-solving skills, importance  
     of, 200  
   productivity, 196–198  
   results, importance of, 198–200  
   sensei (mentor), 199  
   The Toyota Way, 199  
 Kroll, Per, 224
- L**
- Laws and policies, cause of waste, 31  
 Layoffs, 202–203  
*The Leader's Handbook*, 184  
 Leadership of great companies. *See also* Agile@IBM; Management;  
   Southwest Airlines; Svenska  
   Handelsbanken; Toyota;  
   specific leaders.  
   Agile@IBM, 244–246  
   Nucor Steel, xv  
   personal traits, 245  
   SAS Institute, xv  
   W.L. Gore & Associates, xv  
 Leadership principles, 234  
 Leadership teams, questions for  
   conformance to plan, 238–239  
   cost control, 237–238  
   finances, 240  
   maturity, 239–240  
   performance measurements, 241  
   utilization, 239  
   work standards, 239–240  
*Lean Product and Process  
 Development*, 30  
 Learning to improve, Frame 16,  
   173–182. *See also* Continuous  
   improvement.  
   A3 problem-solving report, 176–180  
   directive management, 184  
   disincentives for sharing  
     knowledge, 182  
   fishbone diagrams, 179  
   five whys, 178–179  
   goal of, 173–174  
   managers as mentors, 183–184  
   problem owners, 180–181  
   problem solving, 174  
   problem/countermeasure board,  
     174–175  
   pull-based authority, 180–181  
   responsibility authority, 181  
   retrospectives, 173  
   root causes, identifying, 179–180  
   self-organizing teams, 184  
   sharing knowledge, 181–182  
   suggestion systems, 175–176  
 Lennox, Tomo, 208–209  
 Level Workflow, Frame 10  
   iterations, overview, 117–122  
   iterations, *vs.* kanban, 126–129

Level Workflow (*continued*)  
 overview, 114  
 small batches, 114–117  
 staging, 115  
 “Life Cycle Concept Considered Harmful,” 56–57, 56–57  
 Literature. *See* Publications.  
 Logistics, Empire State Building  
   construction, 106–107  
 Long-term orientation, 190  
 Low-dependency architecture, 65–67  
 Low-velocity organizations, 161  
 Luggage fees, impact of, 5

## M

Maestros, developing expertise, 96–97  
 Making work ready, 118–122  
 Malinowski, Bronislaw, 200–201  
 Management. *See also* Capacity; Knowledge workers.  
   cross-cultural traits, 190–191  
   cultural heritage, 191–194  
   customer focus, 162  
   front-line leaders, 212–216  
   military organizations, 212–215  
   mission command *versus* detailed command, 214–215  
   mission tactics, 212–215  
   OODA (Observe-Orient-Decide-Act) loop, 215  
   policy driven waste, 24–34  
   quality of, 207–208  
   refactoring into agile, 208–209  
*Management Challenges for the 21st Century*, 196–197  
 Management theories, 191–192

Managers  
   lack of technical expertise, cause of waste, 30  
   as mentors, 95–97, 183–184  
   respect for, 204  
*Managing to Learn*, 180, 183  
 Manual workers *versus* knowledge workers, 197  
 Manuals. *See* Publications.  
 Medical and medication errors, 154  
 Mental constructs, xvii  
 Mentors  
   among knowledge workers, 199  
   managers as, 183–184  
   sensei, 199  
 Methods, workflow design, 164  
 Milestones in the history of software development, 58  
 Military organizations, management, 212–215  
 Mutual respect, Frame 19, 193, 203–209

## N

Network connectivity, 64  
 “Never” lists, 137  
 “No Silver Bullet,” 26, 63  
 The Norm of Reciprocity, Frame 18, 200–209  
 Nygaard, Kristen, 51

## O

Obama, Barack, 147–149  
 Object-oriented programming, history of, 50–52

Obscure code, 35  
 Obscure code, cause of waste, 35  
 Ohno, Taiichi  
   on process standards, 168–169  
   specialization, 32  
   work improvement through  
     scientific methods, 33–34  
   *Workplace Management*, 168  
 “On the Inevitable Intertwining of  
   Specification and  
   Implementation,” 56  
 OnCast Technologies, 96  
 O’Neill, Paul, 156–157, 159–160  
 On-site developers, waste prevention,  
   32–33  
 OODA (Observe-Orient-Decide-Act)  
   loop, 215  
 Open source, history of, 60–61  
 Organizing patient charts, 156–157  
 Output, workflow design, 163–164,  
   166–167  
*Outside-in Software Development*,  
   222  
 Overlapping steps, 120–122  
 OXO, identifying customer needs, 40

## P

Papers. *See* Publications.  
 Parker, James  
   customer service, 6  
   leadership, 244–245  
   Southwest Airlines, 3, 202, 215  
 Parnas, David  
   controlling complexity, 64–66  
   information hiding, 82  
 Pathway, workflow design, 164  
 Patient charts, organizing, 156–157

*People and Performance: The Best of  
 Peter Drucker . . .*, 197–198  
 People in lean development. *See also*  
   Management; specific people.  
 Agile Manifesto principles, 192  
 cross-cultural commonalities,  
   188–190  
 cross-cultural psychology, 188–191  
 cultural assumptions, 188–191  
 decision making, 194  
 individualism, 189  
 long-term orientation, 190  
 masculinity, 189  
 motivation theories, 190  
 power distance, 189  
 preparation *versus*  
   implementation, 194  
 principles of lean development,  
   193–194  
 short-term orientation, 194  
 uncertainty avoidance, 189–190  
 People in lean development, company  
   culture. *See also* Knowledge  
   workers.  
 collective decisions, 206  
 cross-cultural teams, 204–206  
 customers, respect for, 203–204  
 diversity, value of, 205–206  
 groupthink, 205–206  
 internal customers, respect for,  
   203–204  
 managers, respect for, 204  
 mutual respect, 203–209  
 need for consensus, 204–205  
 pride of workmanship, 209–211  
 purpose-passion-persistence-pride,  
   210  
 reciprocity, 200–203  
 remuneration, 201–202

- People in lean development (*continued*)  
 self-organizing teams, 206–209  
 subordinates, respect for, 204  
 suppliers, respect for, 204  
 teammates, respect for, 204  
 Toyota Motor Corporation,  
 195–196
- Perfection, visualizing. *See*  
 Visualizing perfection.
- Performance measurements, question  
 for leadership teams, 241
- Personal computers, history of, 57–60
- Plank road analogy, history of  
 software development, 46–47
- Policies and laws, 31
- Policy-driven waste, Frame 4, 24–36.  
*See also* Waste, policy-driven.
- Portfolio management, 141–142
- Positive *versus* negative  
 reinforcement, 14
- Power distance, cross-cultural, 189  
*The Practice of Management*, 196
- Premature decisions, cause of waste, 34
- Preparation *versus* implementation,  
 194
- Prescriptions  
 filling, 157–158  
 medical and medication errors, 154  
 missing medications, 158–159
- Presentation layer, testing, 75
- Pride of workmanship, Frame 20,  
 209–211
- Principles of lean development,  
 193–194  
*The Principles of Product  
 Development Flow*, 115
- Problem owners, 180–181
- Problem/countermeasure board,  
 174–175. *See also* Kanban  
 boards.
- Problems, exposing. *See also*  
 Continuous improvement.  
 bad news first, 169–170  
 experiencing the workplace, 172  
 test failures, analyzing, 169–170
- Problem-solving skills  
 importance of, 200  
 learning, 174
- Process cycle efficiency, 24
- Process description, 122–126
- Process flow maps. *See* Value stream  
 maps.
- Process principles, 234–235
- Process standards, workflow design,  
 167–169
- Processes, dividing projects by, 64
- Product champions  
 Barack Obama's Facebook page,  
 147–149  
 case studies, 37–38, 147–149  
 critical activities, 38  
 definition, 36  
 Hughes, Chris, 147–149  
 Leading Ideation, 39–41  
 MyBarackObama.com, 148  
 responsibility, 37–38  
 system design, 109
- Product concept, 41
- Product managers as customers, 8–9
- Product owners as customers, 8–9
- Product purpose, determining, 9
- Productivity  
 definition, 235–236  
 knowledge workers, 196–198  
 self measurement, 33–34
- Programmers, authors *versus*  
 translators, 89–90
- Project management  
 agile, 63  
 lifecycle, 56, 62

- portfolio management, 141–142
  - versus* system development, 62
  - versus* technical competence, 71
- Prototyping
  - meeting customer expectations, 40
  - phase of evolutionary development, 86–87
- Proven experience, Frame 9, 108–114
- Publications
  - The Art of Innovation*, 39
  - Beyond Budgeting*, 233–234
  - Chasing the Rabbit*, 161
  - “The Coming Commoditization of Processes,” 167
  - “Complexity Controlled by Hierarchical Order . . .”, 47
  - “A Constructive Approach to . . . Program Correctness,” 47
  - “Design and Code Inspections . . .”, 53
  - The Elegant Solution*, 87
  - “Evolutionary Development,” 57
  - “Goals Gone Wild,” 16
  - Implementing Beyond Budgeting*, 231
  - Inside Intuit*, 85–86
  - International Dimensions of Organizational Behavior*, 191
  - The Leader’s Handbook*, 184
  - Lean Product and Process Development*, 30
  - “Life Cycle Concept Considered Harmful,” 56–57
  - Management Challenges for the 21st Century*, 196–197
  - Managing to Learn*, 180, 183
  - “No Silver Bullet,” 26, 63
  - “On the Inevitable Intertwining of Specification and Implementation,” 56
  - Outside-in Software Development*, 222
  - People and Performance: The Best of Peter Drucker . . .*, 197–198
  - The Practice of Management*, 196
  - The Principles of Product Development Flow*, 115
  - Reward Systems*, 182
  - A Sense of Urgency*, 229–230
  - “Simple Smalltalk Testing,” 71
  - “Software Engineering,” 54
  - Structured Programming*, 51
  - What Is Total Quality Control? . . .*, 169
  - The Wisdom of Crowds*, 205
  - Workplace Management*, 168
- Pull scheduling Frame 11
  - arbitrating with value, 134
  - cost-benefit ratio of scheduled activities, 134
  - decoupling, 132
  - definition, 129
  - description, 131
  - feature teams, 132–133
  - larger systems, 137–140
  - limiting queues, 135–136
  - medium-sized projects, 131–132
  - MUFs (minimum useful feature sets), 132
  - “never” lists, 137
  - portfolio management, 141–142
  - small, frequent requests, 133–136
  - timeboxing *versus* scopeboxing, 138–139
- Pull-based authority, 180–181
- Pulling from the future, 18
- Purpose-passion-persistence-pride, 210
- Push scheduling, 129–130

## Q

- Quality, budgeting problems at
  - Agile@IBM, 232
- Quality by construction, Frame 6. *See also* Test-driven development.
  - code clarity
    - information hiding, 82
    - overview, 80–82
    - refactoring, 82
  - continuous integration
    - code, social setting, 76
    - defect injection processes, 76
    - frequency of, 78–80
    - integration testing, time budget
      - for, 77
    - step-wise integration, 76–77
    - stress testing, system level, 79–80
    - user acceptance tests, 80
  - sequential development
    - drawbacks, 70–71
- Queues, limiting, 135–136
- Quick experimentation phase, 84, 87–88
- Quick experiments in hospitals, 158–159
- Quicken, case study, 85–86

## R

- Ready-ready to be done-done, 118–122
- Reciprocity, as motivation, 200–203
- Refactoring code, 35, 82
- Regression deficits, cause of waste, 35
- Reinertsen, Donald, 115
- Relative goals, 17
- Reliable delivery. *See also* Empire State Building construction; Schedules.
  - adaptive control
    - consumability, 145
    - customer feedback, 143–144
    - customer outcomes, 146–147
    - escaped defects, 146
    - frequent releases, 144–145
  - iterations
    - versus* kanban technique, 126–129
    - making work ready, 118–122
    - overlapping steps, 120–122
    - ready-ready to be done-done, 118–122
    - stories, 118
    - workflow chart, 117
- kanban technique
  - batch size, 127
  - cadence, 127–128
  - capacity, 128–129
  - commitment, 126
  - definition, 122
  - versus* iterations, 126–129
  - kanban boards, 123–125
  - kanban cards, 122–123
  - process description, 122–126
  - teamwork, 127
  - throughput, 129
  - velocity, 128
- Reliable delivery, workflow leveling
  - iterations
    - versus* kanban, 126–129
    - overview, 117–122
  - overview, 114
  - small batches, 114–117
  - staging, 115
- Remuneration, as motivation, 201–202
- Resources. *See* Publications.
- Respect for people
  - customers, 203–204
  - internal customers, 203–204

- managers, 204
  - mutual respect, 193, 203–209
  - subordinates, 204
  - suppliers, 204
  - teammates, 204
  - Toyota Motor Corporation, 153, 195
  - Responsibility, dividing projects by, 64
  - Responsibility authority, 181
  - Results, importance of, 198–200
  - Retention, developing expertise, 94
  - Retrospectives, 173
  - Reward Systems*, 182
  - Rivera, Ted, 221
  - Root causes, identifying, 179–180
  - Routers, 64
- S**
- SAS Institute, corporate culture, xv
  - Schedules. *See also* Pull scheduling.
    - alternate approaches, 113–114
    - critical path, 113–114
    - dependencies, 112
    - per workflow, 104–106
    - push scheduling, 129–130
    - reducing complexity, 112–114
    - set-based design, 113–114
    - utilization, 113
  - Scholtes, Peter, 184
  - Scope of projects, cutting, 27–28
  - Scopeboxing *versus* timeboxing, 138–139
  - Seddon, John, 1, 11
  - Self-organizing teams, 184, 206–209
  - A Sense of Urgency*, 229–230
  - Sense of urgency, creating, 229–230
  - Sensei (mentor), results are not the point, 199
  - Separating responsibility, knowledge, action, and feedback, 31. *See also* Handovers.
  - Separation of concerns, 35. *See also* Information hiding.
  - Sequential development drawbacks, 70–71
  - Set-based design, 87–88, 113–114
  - Sharing knowledge, 181–182
  - Shook, John, 180, 183–184, 207, 242–243
  - Short-term orientation, 194
  - Shreve, Richmond, 102
  - “Simple Smalltalk Testing,” 71
  - Simula language, history of, 51
  - Skiles, Jeffrey, 92
  - Small batches, 114–117
  - Smalltalk language, history of
    - software development, 51
  - Smith, Adam, 32
  - Software development
    - history of. *See* History of software development.
    - separating from larger system, 69–70
  - “Software Engineering,” 54
  - Software life cycle concept, history of, 53–57
  - Southwest Airlines, xvi, corporate culture
    - history of, 2–3
    - layoffs, 203
    - luggage fees, impact of, 5
    - quick turnaround, 5
    - reciprocity, 202
    - secrets of success, 3–6, 215–216
    - systems thinking, 1
    - turnover rate, 202
  - Spear, Steven
    - on complexity, 163

- Spear, Steven (*continued*)  
 exposing problems, 169  
 high-velocity organizations, 161  
 sharing knowledge, 181–182
- Spreadsheets, history of, 58–59
- Staging work, 115
- Stakeholder feedback, 144, 222–223, 226, 228, 230
- Stakeholder involvement, 8, 156, 181, 221–225
- Standard Life, 32–33
- Standards, developing expertise, 94–95
- Starrett, Paul, 103
- Starrett, William, 103
- Step-wise integration, 49, 76–77
- Stories, 118
- Stress testing, 76, 79–80
- Stroustrup, Bjarne, 81
- Structure, dividing projects by, 64  
*Structured Programming*, 51
- Structured programming, history of, 47–50
- Subcontracting, Empire State Building, 112
- Subordinates, respect for, 204
- Suggestion systems, 175–176
- Sullenberger, Chesley B., 91
- Suppliers, respect for, 204
- Support calls  
 failure demand, 10  
 reducing, 223, 228–229
- Support organization, 32–33, 80, 111, 185  
 as customers, 6–7, 9  
 delegation to, 31, 67  
 help desk, 10  
 mental model discrepancies, 223  
 waste in, 19–25
- Sustainability, Frame 24, 215, 242–243. *See also* System capability.  
 developing leaders, 243  
 implementation, 194
- Svenska Handelsbanken, xiii–xiv, 233
- Swartout, William, 56
- Sweitzer, John, 7, 222
- System capability, Frame 2  
 current work methods, 13–15  
 goals, 15–18  
 measuring *versus* prescribing, 15–16  
 targets, setting, 15–18  
 time series charts, 13–14  
 variation, as a management problem, 14
- System design, Empire State Building, 109–111
- System development *versus* project management, 62
- Systematic, case study, 118–120
- ## T
- Targets, setting, 15–17. *See also* Goals.
- Target-setting and evaluation, 232
- Taylor, Frederick Winslow, 33
- TCP/IP protocol, 64
- Teachers. *See* Mentors.
- Team design, Empire State Building, 103–104
- Teammates, respect for, 204
- Teams  
 at Amazon.com, 66–67  
 and architectural complexity, 68–69  
 cross functional, 68–69



- cross-cultural, 204–206
  - decoupling, 133
  - feature, 132–133
  - groupthink, 205–206
  - ideal size, 66–67
  - self-organizing
    - cross-cultural factors, 193
    - versus* directive management, 184
    - mutual respect, 193, 206–209
  - teammates, respect for, 204
  - training, Agile@IBM, 225–226
  - two-pizza, 67
  - WebSphere Service Registry and Repository team, 223
- Teamwork, kanban, 127
- Technical debt, 15, 34–36, 66, 80, 82
  - Agile@IBM, 226
  - cause of waste, 34–36
  - failure demand, 10
- Technical excellence. *See specific topics.*
- Technology-facing ideation, 41–42
- Ten-year rule for developing expertise, 93
- Test failures, analyzing, 169–170
- Test-driven development
  - acceptance tests, 73–74
  - automation, 10, 74–75, 79, 227
  - defect avoidance, 71–76
  - exploratory, 75
  - failure analysis, 169–170
  - history of software development, 48
  - integration testing, time budget for, 77
  - presentation layer, 75
  - stress, 76
  - stress testing, description, 76
  - stress testing, system level, 79–80
  - testing to failure, 75–76
  - through the user interface, 75
  - timing of, 70–71
  - unit tests, 72
  - usability, 76
  - user acceptance tests, 80
  - writing tests before code, 73–74
  - xUnit frameworks, 71–72
- Test-driven handovers, 165–166
- Theoretical limits, 160–161
- Thomas, Dave, 81
- Throughput, 129
- Throughput, kanban, 129
- Time series charts, 13–14
- Timeboxing *versus* scopeboxing, 138–139
- Timing, budgeting problems at Agile@IBM, 232
- Top-down programming, history of, 48–49
- Toyota Motor Corporation
  - car development phases, 37–38
  - continuous improvement, 153, 195
  - core tenets, 153, 195
  - layoffs, 203
  - respect for people, 153, 195
  - secret of success, 243
  - suggestion implementations, 175–176
  - TPS (Toyota Production System), 162
  - visualizing perfection, 160–161
  - working with American employees, 170–172, 195
- Toyota Motor Manufacturing Kentucky, 162, 246
- The Toyota Way
  - company culture, 195
  - creation of, 153
  - meeting targets, 199
  - self-organizing teams, 207

TPS (Toyota Production System), 162  
 Training, 97, 221, 230  
   Agile@IBM, 222–227  
   costs, 116  
   military, 200  
   reciprocity, 202  
   reinforcement, 14  
   retention, 94  
   testing, 75–76  
 Trust, budgeting problems at  
   Agile@IBM, 231  
 Turnover, developing expertise, 94  
 Two-pizza teams, 67

## U

Uncertainty avoidance, cross-cultural, 189–190  
 Unethical behavior, motivated by unrealistic goals, 17  
 Unit tests, 72  
 Urgency, 115, 229–230  
 Usability testing, 76  
 User acceptance tests, 80  
 User interface, 75–76  
 Utilization  
   of manpower, cause of waste, 29  
   questions for leadership teams, 239  
   scheduling, 113

## V

Value, dividing projects by, 64  
 Value demand, 12, 21–24  
 Value stream maps  
   definition, 19  
   of failure demand, 19–21

  improving process capability, 24  
   of value demand, 21–24  
 Variance  
   as a learning opportunity, 162  
   as a management problem, 14–15  
   *versus* targets, 17, 232–334  
 Velocity, kanban, 128  
 Visualizing perfection Frame 13,  
   160–163. *See also* Continuous improvement.  
   Alcoa's safety record, 159–160  
   barrier to rapid learning, 161  
   benchmarking, 161  
   customer focus, 162  
   high-velocity organizations,  
     161–162  
   low-velocity organizations, 161  
   theoretical limit, 160–161  
   Toyota Motor Corporation,  
     160–161  
   variance, as a learning  
     opportunity, 162  
   workflow across boundaries, 162  
 Vogels, Werner, 9, 67  
 Voluntary self-study (jishuken), 199

## W

Wallander, Jan, xiii-xiv, 233, 236  
 Ward, Allen, 30  
 Waste  
   handling failure demand, 11, 19,  
     21  
   in process-flow maps, 19  
 Waste, policy-driven  
   case study, 25  
   causes  
     batch and queue mentality, 29

- budgetary commitments, 29
- case study, 32–33
- code branching, 36
- complexity, 26–28
- context switching, 29
- dependencies, 35
- economies of scale, 28–30
- full utilization of manpower, 29
- handoffs, 30–31
- inadequate knowledge
  - preservation, 34
- insufficient refactoring, 35
- managers' lack of technical expertise, 30
- obscure code, 35
- overview, 25–26
- policies and laws, 31
- premature decisions, 34
- regression deficits, 35
- separating decision making from work, 30–33
- separating responsibility, knowledge, action, and feedback, 31
- technical debt, 34–36
- wishful thinking, 33–34
- cures
  - case study, 32–33
  - information hiding, 35
  - on-site developers, 32–33
  - separation of concerns, 35
  - work improvement through scientific methods, 33–34
  - worker self measurement, 33–34
- definition, 24
- Watanabe, Katsuaki, 195
- WebSphere Service Registry and Repository team, 223
- What Is Total Quality Control? . . .*, 169
- The Wisdom of Crowds*, 205
- Wiseman, James, 171
- Wishful thinking, cause of waste, 33–34
- Work, improvement through scientific methods, 33–34
- Work standards, question for leadership teams, 239–240
- Work-arounds in hospitals, 156
- Worker productivity, self measurement, 33–34
- Worker self measurement, 33–34
- Workflow
  - across boundaries, 162
  - designing
    - connections, 164–165
    - handovers, immediate failure detection, 165–166
    - immediate customers, 164
    - immediate suppliers, 164
    - methods, 164
    - output, 163, 164, 166–167
    - pathway, 164
    - process standards, 167–169
    - test-driven handovers, 165–166
  - Empire State Building
    - construction, 104
  - scheduling. *See* Schedules.
- Workflow, leveling
  - iterations
    - versus* kanban, 126–129
    - overview, 117–122
  - overview, 114
  - small batches, 114–117
  - staging, 115
- Workmanship, pride in, 209–211
- Workplace, firsthand experience, 172
- Workplace Management*, 168
- Writing *versus* programming, 89–90

**X**

xUnit frameworks, 71–72

**Y**

Y2K problem, history of, 61

Yamaguchi, Masayuki, 194