Principles of Lean Software Development

Eliminate Waste
The three biggest wastes in software development are:

Extra Features
We need a process that allows us to develop just those 20 percent of the features that give 80 percent of the value.

Churn
If you have requirements churn, you are specifying too early.
If you have test and fix cycles, you are testing too late.

Crossing Boundaries
Organizational boundaries can increase costs by 25 percent or more. They create buffers that slow down response time and interfere with communication.

Build Quality In
If you routinely find defects in your verification process, your process is defective.

Mistake-Proof Code with Test-Driven Development
Write executable specifications instead of requirements.

Stop Building Legacy Code
Legacy code is code that lacks automated unit and acceptance tests.

The Big Bang Is Obsolete
Use continuous integration and nested synchronization.

Create Knowledge
Planning is useful. Learning is essential.

Use the Scientific Method
Teach teams to establish hypotheses, conduct many rapid experiments, create concise documentation, and implement the best alternative.

Standards Exist to Be Challenged and Improved
Embody the current best known practices in standards that are always followed while actively encouraging everyone to challenge and change the standards.

Predictable Performance Is Driven by Feedback
A predictable organization does not guess about the future and call it a plan; it develops the capacity to rapidly respond to the future as it unfolds.
Defer Commitment
Abolish the idea that it is a good idea to start development with a complete specification.

Break Dependencies
System architecture should support the addition of any feature at any time.

Maintain Options
Think of code as an experiment—make it change-tolerant.

Schedule Irreversible Decisions at the Last Responsible Moment
Learn as much as possible before making irreversible decisions.

Deliver Fast
Lists and queues are buffers between organizations that slow things down.

Rapid Delivery, High Quality, and Low Cost Are Fully Compatible
Companies that compete on the basis of speed have a significant cost advantage, deliver superior quality, and are more attuned to their customers’ needs.

Queuing Theory Applies to Development, Not Just Servers
Focusing on utilization creates traffic jams that actually reduce utilization.
Drive down cycle time with small batches and fewer things-in-process.

Limit Work to Capacity
Establish a reliable, repeatable velocity with iterative development.
Aggressively limit the size of lists and queues to your capacity to deliver.

Respect People
Engaged, thinking people provide the most sustainable competitive advantage.

Teams Thrive on Pride, Commitment, Trust, and Applause
What makes a team? Members are mutually committed to achieve a common goal.

Provide Effective Leadership
Effective teams have effective leaders who bring out the best in the team.

Respect Partners
Allegiance to the joint venture must never create a conflict of interest.

Optimize the Whole
Brilliant products emerge from a unique combination of opportunity and technology.

Focus on the Entire Value Stream
—from concept to cash.
—from customer request to deployed software.

Deliver a Complete Product
Develop a complete product, not just software.
Complete products are built by complete teams.

Measure UP
Measure process capability with cycle time.
Measure team performance with delivered business value.
Measure customer satisfaction with a net promoter score.
Praise for Implementing Lean Software Development

“This book offers a wealth of advice for any organization that wishes to succeed at the software development game. It will help you to realize the value of adopting a product mindset to software development to recognize the inherent wastage and risk in traditional software development practices. Mary has hit another one out of the park.”
—Scott Ambler, practice leader, Agile modeling

“This remarkable book combines practical advice, ready-to-use techniques, and a deep understanding of why this is the right way to develop software. I have seen software teams transformed by the ideas in this book.”
—Mike Cohn, author of Agile Estimating and Planning

“As a lean practitioner myself, I have loved and used their first book for years. When this second book came out, I was delighted that it was even better. If you are interested in how lean principles can be useful for software development organizations, this is the book you are looking for. The Poppendiecks offer a beautiful blend of history, theory, and practice.”
—Alan Shalloway, coauthor of Design Patterns Explained

“I’ve enjoyed reading the book very much. I feel it might even be better than the first lean book by Tom and Mary, while that one was already exceptionally good! Mary especially has a lot of knowledge related to lean techniques in product development and manufacturing. It’s rare that these techniques are actually translated to software. This is something no other book does well (except their first book).”
—Bas Vodde

“The new book by Mary and Tom Poppendieck provides a well-written and comprehensive introduction to lean principles and selected practices for software managers and engineers. It illustrates the application of the values and practices with well-suited success stories. I enjoyed reading it.”
—Roman Pichler

“In Implementing Lean Software Development, the Poppendiecks explore more deeply the themes they introduced in Lean Software Development. They begin with a compelling history of lean thinking, then move to key areas such as value, waste, and people. Each chapter includes exercises to help you apply key points. If you want a better understanding of how lean ideas can work with software, this book is for you.”
—Bill Wake, independent consultant
Implementing Lean Software Development
The Addison-Wesley Signature Series
Kent Beck, Mike Cohn, and Martin Fowler, Consulting Editors

Visit informit.com/awss for a complete list of available products.

The Addison-Wesley Signature Series provides readers with practical and authoritative information on the latest trends in modern technology for computer professionals. The series is based on one simple premise: Great books come from great authors. Books in the series are personally chosen by expert advisors, world-class authors in their own right. These experts are proud to put their signatures on the covers, and their signatures ensure that these thought leaders have worked closely with authors to define topic coverage, book scope, critical content, and overall uniqueness. The expert signatures also symbolize a promise to our readers: You are reading a future classic.
Implementing Lean Software Development

From Concept to Cash

Mary and Tom Poppendieck
To our parents:
John and Marge Brust
and
Elmer and Ruth Poppendieck
This page intentionally left blank
## Contents

Foreword by Jeff Sutherland ............................ xvii  
Foreword by Kent Beck ................................. xx  
Preface ................................................... xxiii  

### Chapter 1: History ................................. 1  
- Interchangeable Parts ............................... 1  
- Interchangeable People ............................... 2  
- The Toyodas ......................................... 3  
- The Toyota Production System ...................... 4  
  - Taiichi Ohno ....................................... 5  
  - *Just-in-Time Flow* .............................. 5  
  - *Autonomation (Jidoka)* ...................... 5  
  - Shigeo Shingo ................................... 6  
  - *Nonstock Production* ......................... 6  
  - *Zero Inspection* ............................... 6  
- Just-in-Time ......................................... 7  
- Lean .................................................. 11  
  - Lean Manufacturing / Lean Operations .......... 11  
  - Lean Supply Chain .................................. 12  
  - Lean Product Development ....................... 13  
  - Lean Software Development ..................... 17  
  - Try This ......................................... 17  

### Chapter 2: Principles ............................... 19  
- Principles and Practices ........................... 19
Design for Operations ............................................ 58
Custom Development ............................................. 60
From Projects to Products ........................................ 60
IT—Business Collaboration ....................................... 62
Accountability ...................................................... 64
Try This ............................................................ 65

Chapter 4: Waste .................................................. 67

Write Less Code .................................................... 67
Zara ................................................................. 67
Complexity .......................................................... 69
Justify Every Feature .............................................. 70
Minimum Useful Feature Sets ................................. 71
Don’t Automate Complexity .................................... 72

The Seven Wastes .................................................. 73
Partially Done Work ............................................... 74
Extra Features ...................................................... 75
Relearning ........................................................... 76
Handoffs ............................................................. 77
Task Switching ...................................................... 78
Delays ................................................................. 80
Defects ................................................................. 81

Mapping the Value Stream ..................................... 83

Preparation .......................................................... 83
Choose a Value Stream .......................................... 83
Choose When to Start and Stop the Timeline ............... 84
Identify the Value Stream Owner ............................. 84
Keep It Simple ....................................................... 85

Examples ............................................................ 85
Example 1 ........................................................... 86
Example 2 ........................................................... 86
Example 3 ........................................................... 88
Example 4 ........................................................... 89

Diagnosis ............................................................. 91

Future Value Stream Maps ...................................... 92

Try This ............................................................ 92
Guideline No. 2: De-emphasize Annual Raises ........... 144
Guideline No. 3: Reward Based on Span of Influence,
Not Span of Control ......................... 144
Guideline No. 4: Find Better Motivators Than Money .... 145
Try This ................................................. 147

Chapter 7: Knowledge .................................. 149
Creating Knowledge .................................. 149
Rally ................................................. 149
What, Exactly, Is Your Problem? ...................... 152
A Scientific Way of Thinking ......................... 154
Keeping Track of What You Know .................... 155
The A3 report ........................................ 157
The Internet Age ..................................... 159

Just-in-Time Commitment .............................. 159
Set-Based Design .................................... 160
Example 1: Medical Device Interface Design .......... 162
Example 2: Red-Eye Reduction ....................... 162
Example 3: Pluggable Interfaces ..................... 163
Why Isn’t This Waste? ................................ 164
Refactoring .......................................... 164
Legacy Systems ..................................... 166

Problem Solving ..................................... 168
A Disciplined Approach ............................... 169
1. Define the Problem ................................ 169
2. Analyze the Situation .............................. 169
3. Create a Hypothesis .............................. 171
4. Perform Experiments ............................ 171
5. Verify Results ................................... 172
6. Follow Up/Standardize ........................... 172
Kaizen Events ...................................... 173
Large Group Improvement Events .................... 173
Try This ................................................. 175

Chapter 8: Quality ..................................... 177
Feedback ............................................. 177
The Polaris Program ................................ 177
Release Planning .................................... 179
## Contents

Chapter 10: Journey .............................................. 223
  Where Do You Want to Go? ................................. 223
  A Computer on Wheels ...................................... 224
  A Long-Term Perspective ................................. 225
  Centered on People ........................................ 227
What Have We Learned? .................................. 229
  Six Sigma .................................................. 229
  *Process Leaders—Natural Work Team Leaders* .... 229
  *Tools—Results* ........................................ 229
  Theory of Constraints ................................... 230
  *Critical Chain* .......................................... 232
  *Accommodations* ....................................... 233
Hypothesis ...................................................... 234
  Training ..................................................... 234
  Thinking .................................................... 236
  Measurement .............................................. 237
  *Cycle Time* ............................................. 238
  *Financial Return* ....................................... 240
  *Customer Satisfaction* ................................ 241
Roadmap ......................................................... 242
  Try This .................................................... 243
  Optimize the Whole .................................... 243
  Respect People .......................................... 243
  Deliver Fast .............................................. 244
  Defer Commitment .................................... 244
  Create Knowledge ...................................... 245
  Build Quality In ....................................... 245
  Eliminate Waste ........................................ 246
Bibliography .................................................... 247
Index ........................................................... 257
This page intentionally left blank
Foreword

Jeff Sutherland

I created the first Scrum in 1993 at Easel Corporation in Burlington, Massachusetts, in cooperation with our CEO, who bet his company on the team that made the first Scrum work. In 1995, I began working with Ken Schwaber, who formalized the process for worldwide deployment. In four companies since Easel, I have used the self-organizing nature of Scrum to move from a Type A Scrum (winning teams) to a Type B Scrum (winning product portfolios) to a Type C Scrum (winning companies).

In 2000, I introduced Scrum to PatientKeeper, and this has proven to be an excellent vehicle for lean development. Over the years we have shortened cycle times to exactly what customers need: one week for critical items, one month for minor enhancements, and three months for significant new products. The three-month releases are accumulations of one-month Sprints, and every Sprint is a release.

At PatientKeeper the entire company runs as a Scrum and inspects, adapts, self-organizes, and changes every Monday when we have our MetaScrum meeting. The Product Owner runs this meeting, and all company stakeholders are present, including the CEO. Lean concepts are carefully examined here, and Sprints are started, stopped, or changed only in this meeting. The whole company, including affected customers, can be reset in one afternoon with decisions made during the weekly MetaScrum.

We have a chief Product Owner with a team that gets the Product Backlog “ready.” We have a chief ScrumMaster, who runs a 15-minute daily Scrum of Scrums. What did teams do yesterday, what will they do today, and what are the impediments to delivery? We manage 45 releases of software per year into large, mission-critical enterprises using these short meetings and using an automated system that tracks the state of concurrent Sprint backlogs. Dates are committed to customers before the start of a Sprint, and thousands of Web users and hundreds of physicians with PDAs go live at the end of every Sprint. For PatientKeeper, concept (Product Backlog) to cash (product in production) occurs in one-month intervals. We have had to eliminate waste everywhere—before, during, and after the implementation process.

In 1993, lean product development was not well understood in any industry. Scrum was the first concrete implementation of lean thinking to software development that allowed organizations of all types and sizes to start up lean teams in a couple of days using a standard pattern that was easily understood. What was hard was explaining why and how to implement the pattern to generate continuous quality and productivity improvement.

Today, the writing and courses from Mary and Tom Poppendieck provide a proven set of principles that organizations can use to adapt tools, techniques, and methods to their own specific unique contexts and capabilities. Now we can explain how to use Scrum to lean out software development. In addition to my company, PatientKeeper, I use the procedures and processes outlined in this book to teach practitioners worldwide how to optimize Scrum.

Lean software development views all agile methods as valid, proven applications of lean thinking to software. It also goes beyond agile, providing a broader perspective that enables agile methods to thrive. First, it looks along the whole value chain, from concept to cash and tries to address all the waste and delays that happen before and after the coders contribute their part. Second, it establishes a management context to extend, nourish, and leverage agile software practices. Third, it provides a proven set of principles that each organization can use to adapt tools, techniques, and methods to their own specific unique contexts and capabilities.

Every chapter in this book illustrates a set of principles that can be implemented to build more productive teams. If you want to be like Toyota, where productivity is consistently four times that of its competitors and quality is twelve times better, these practices are essential. If you execute well on the principles in this book, resistance by your competition is futile, and winning in your market is certain. The return on investment in practices outlined in this book can be very high.
To create the lean “secret sauce” for your company, we have found you must systematically implement this accumulated lean wisdom. In short, you must deeply understand the Japanese concepts of Muri, Mura, and Muda. Mary and Tom unfold them as the seven lean principles and the seven wastes of software development to help you easily understand how they work and know what to do.

Muri relates to properly loading a system and Mura relates to never stressing a person, system, or process. Yet many managers want to load developers at 110 percent. They desperately want to create a greater sense of “urgency” so developers will “work harder.” They want to micromanage teams, which stifles self-organization. These ill-conceived notions often introduce wait time, churn, death marches, burnout, and failed projects.

When I ask technical managers whether they load the CPU on their laptop to 110 percent they laugh and say, “Of course not. My computer would stop running!” Yet by overloading teams, projects are often late, software is brittle and hard to maintain, and things gradually get worse, not better. One needs to understand that Toyota and Scrum use a pull system that avoids stress (Mura) and eliminates bottlenecks (Muri). The developers take what they need off the Product Backlog just in time. They choose only Product Backlog that is ready to take into a Sprint, and they never take more than they can get done in a Sprint. They go faster by surfacing impediments and working with management to eliminate waste (Muda). By removing waste, they have less work to do. Productivity goes up, and they have time to deliver quality software and focus on exactly what the customers need.

So by understanding loading (Muri) and avoiding stress (Mura) you shorten cycle time. This leans out the environment causing impediments that create waste (Muda) to be highly visible. When you eliminate these impediments, your teams move faster, do more with less, increase quality, and move the product right into the sweet spot for the customer.

I recommend you keep the Poppendiecks’ books on your desk and use them regularly to help with systematic and continuous implementation of lean principles. Practice hard, and you will rapidly double productivity by using lean development to do less by avoiding waste, and then double it again by working smarter by eliminating impediments. By going four times faster in the right way, your quality will improve by a factor of twelve like Toyota.

—Jeff Sutherland, Ph.D.
Chief Technology Officer, PatientKeeper
Certified ScrumMaster Trainer
Inventor of the Scrum Development Process
July 2006
Foreword

Kent Beck

Software development is a chain with many links, and improving overall effectiveness requires looking at the whole chain. This book is for people involved in software development—not just programmers, but also managers, sponsors, customers, testers, and designers. The principles presented here eventually affect everyone connected to the development of software. This book speaks to those of you who are ready to look at the big picture of development.

For ideas to have impact, they must be grounded in both theory and experience. By translating well-proven ideas from lean manufacturing, this book presents theory and practice and applies them to software development. Many of the fundamental problems with manufacturing are also problems with software development: dealing with uncertainty and change, continuously improving processes, and delivering value to customers.

What does this book offer you? First, a wide variety of theories that provide alternative ways of thinking about how software development can be improved. Second, a set of stories about the application of these theories to real projects. Third, provocative questions to help you apply the theories and the lessons in the stories to your unique situation.

Mary is uniquely qualified to present this material. She experienced lean manufacturing firsthand, helping rescue a plant from being shut down by transforming its operations. She’s also experienced software development as a programmer and a manager. I once shared a seminar with Mary, and it was a treat to see the force and confidence with which she presents her material. I can hear that same direct voice in her writing here.

If you’ve read the Poppendiecks’ precursor book, Lean Software Development, this book offers new guidance towards its implementation. It reiterates the theory in the previous volume, but always with an eye towards applying the ideas in real situations. The result is a readable and practical guide to ideas that have the potential to help you transform your development.
This is a book for doers, doers who want their actions aligned with where they can do the most good. If you are such a person, I recommend this book to you as a source of ideas and energy for positive change.

—Kent Beck
Three Rivers Institute
July 2006
Preface

The Sequel

Lean was an idea borrowed from the 1990s when we wrote the book Lean Software Development: An Agile Toolkit in 2003. We had observed that breakthrough ideas from manufacturing and logistics often take a decade or two before they are adapted to provide suitable guidance for development efforts. So we decided it was not too late to use well-proven lean concepts from the 1980s and 1990s to help us explain why agile methods are a very effective approach to software development.

The strategy worked. The book Lean Software Development presents a set of thinking tools based on lean thinking that leaders continue to find useful for understanding agile software development. The book has been purchased by many a developer who gave it to his or her manager to read, and many managers have distributed multiple copies of the book to colleagues in support of a transition to lean/agile software development.

Meanwhile, something unexpected happened to lean. In the last couple of years lean initiatives have experienced a resurgence in popularity. The word lean was originally popularized in the early 1990s to characterize the Japanese approach to automobile manufacturing.¹ In recent years, Honda and Toyota have been doing increasingly well in the North American auto market, while Detroit automakers are restructuring. For example, Toyota’s profits rose from more than $8 billion in the fiscal year ending March 31, 2003, to more than $10 billion in 2004, $11 billion in 2005, and $12 billion in 2006. Many com-

panies have taken a second look at lean to try to understand what's behind such steady and sustained success.

Lean initiatives seldom start in the software development or product development area of a company, but over time, successful lean initiatives make their way from manufacturing or logistics to development departments. However, lean practices from manufacturing and other operational areas do not adapt easily to a development environment, so lean initiatives have a tendency to stall when they reach software development. While the underlying lean principles remain valid, it is usually inappropriate to apply operational practices and measurements to a development environment. When lean initiatives stall in software development areas, many companies have discovered that the book *Lean Software Development* gives them a good foundation for thinking about how to modify their approach and adapt lean ideas to a development organization.

The benefits of lean and agile software development have become widely known and appreciated in the last few years, and many organizations are changing the way they develop software. We have traveled around the world visiting organizations as they implement these new approaches, and we have learned a lot from our interaction with people working hard to change the way they develop software. As our knowledge has grown, so has the demand for more information on implementing lean software development. We realized that a new book would allow us to share what we’ve learned with many more people than we can contact personally. Therefore we have summarized our experiences in this book, *Implementing Lean Software Development: From Concept to Cash*.

This book is not a cookbook for implementing lean software development. Like our last book, it is a set of thinking tools about how to go about adapting lean principles to your world. We start this book where the last book left off and go deeper into the issues and problems that people encounter when trying to implement lean and agile software development. You might consider this book a sequel to *Lean Software Development*. Instead of repeating what is in that book, we take a different perspective. We assume the reader is convinced that lean software development is a good idea, and focus on the essential elements of a successful implementation. We look at key aspects of implementation and discuss what is important, what isn’t, and why. Our objective is to help organizations get started down the path toward more effective software development.

The first chapter of this book reviews the history of lean, and the second chapter reviews the seven principles of lean software development presented in *Lean Software Development*. These are followed by chapters on value, waste, speed, people, knowledge, quality, partners, and the journey ahead. Each of
these eight chapters begins with a story that illustrates how one organization dealt with the issue at hand. This is followed by a discussion of key topics we have found to be important, along with short stories that illustrate the topic, and answers to typical questions we often hear. Each chapter ends with a set of exercises that helps you explore the topics more deeply.

Acknowledgments

We would like to thank everyone who has attended our talks and classes and invited us into their companies, especially those who have shared experiences and asked penetrating questions. In this book we share the knowledge that we have gained from all of you.

Many thanks to Jeff Sutherland and Kent Beck for their kind forewords, and thanks to our editor, Greg Doench, project editor, Tyrrell Albaugh, and especially our copy editor, Nancy Hendryx. Particular thanks goes to those who have contributed ideas and stories to the book: Jill Aden, Brad Appleton, Ralph Bohnet, Mike Cohn, Bent Jensen, Brian Marick, Clare Crawford-Mason, Ryan Martens, Gerard Meszaros, Lynn Miller, Kent Schnaith, Ian Shimmings, Joel Spolsky, Jeff Sutherland, Nancy Van Schooenderwoert, Bill Wake, Werner Wild, and our friend and operations manager, who prefers to remain anonymous.

We thank everyone who read the first draft and contributed suggestions, particularly Glen Alleman, Brad Appleton, Daniel Brolund, Bob Corrick, Allan Kelly, Kent Schnaith, Dave Simpson, Alan Shalloway, and Willem van den Ende. We thank reviewers Yi Ly, Roman Pichler, Bill Wake, and especially Bas Vodde for his extensive and insightful comments. A particularly heartfelt thanks goes to Mike Cohn, who reviewed both drafts of the book, keeping up with us under a tight deadline while he was very busy. His comments and suggestions have truly helped make this a better book.

—Mary and Tom Poppendieck
July 2006
Chapter 1

History

Interchangeable Parts

Paris, France, July 1785. It was 18 months after the end of the Revolutionary War in America, and four years before the start of the French Revolution. The need for weapons was on everyone’s mind when Honoré Blanc invited high-ranking military men and diplomats to his gunsmith shop in Paris. He had taken apart 50 firing mechanisms (called “locks”) and placed the pieces in boxes. The astonished visitors took random parts from the bins, assembled them into locks, and added them to muskets. They found that the parts fit together perfectly. For the first time it seemed possible to make guns out of interchangeable parts.

Thomas Jefferson, a diplomat in Paris at the time, was at the demonstration. The future United States president saw a way to address a big problem in his fledgling country. The United States was facing a shortage of weapons to defend itself and expand its boundaries. If interchangeable parts could be easily produced, then relatively unskilled workers could assemble a lot of guns at low cost, a real boon to the start-up country that had neither the money to buy guns nor the craftsmen to make them.

The challenge of creating a manufacturing process precise enough to make interchangeable parts for guns was taken up by Eli Whitney, who had recently patented the cotton gin. In 1798 Whitney was awarded a government contract to make 10,000 guns in two years. Ten years and several cost overruns later he finally delivered the guns, and even then the parts were not fully interchangeable. Nevertheless, Whitney is considered a central figure in developing the “American system of manufacture,” a manufacturing system in which semiskilled workers use machine tools and precise jigs to make standardized parts that are then assembled into products.

During the 1800s the United States grew dramatically as an industrial power, with much of the credit given to the new manufacturing system. Meanwhile in Europe there was strong resistance to replacing craft production. In France,
Honoré Blanc’s work was terminated by a government that feared losing its control over manufacturing if unregulated workers could assemble a musket. In England, the inventors of machines that automated both spinning and weaving were attacked by angry crowds who feared losing their jobs. But in America, labor was scarce and there were few craft traditions, so the new industrial model of interchangeable parts took root and flourished.

**Interchangeable People**

Detroit, USA, January 1914. Henry Ford raised wages of workers from $2.40 for a nine-hour day to $5 for an eight-hour day as he began assembly line production of the Model T. The press suggested that he was crazy, but it was a shrewd move. Ford had taken more than 85 percent of the labor out of a car, so he could well afford to double wages. He had already dropped the price of the car dramatically. Now he drove up wages and shortened work hours to help create a middle class with the time and money to buy automobiles.

It used to take more than 12 hours to assemble an automobile; now it took about 90 minutes. What happened to all of the time? Ford managers applied the ideas of efficiency expert Frederick Winslow Taylor as they designed the production line jobs. Taylor believed that most fixed wage workers spent their time trying to figure out how to work slowly, since being efficient brought no extra pay and could threaten jobs. His approach was to divide the assembly line work into very small steps, and time the workers to uncover the “one best way” to do each step.

Work on the assembly line was boring, repetitive, and tightly controlled. The workers were shown exactly how to do their job and told how much time they had to complete it. They could be trained in ten minutes, and they could be replaced in ten minutes. Like the interchangeable parts of a century earlier, interchangeable workers were at the center of a new industrial model: mass production.

High wages were supposed to make up for the lack of variety and autonomy, and for a while they did. And for a while, things went very well for Ford. Sales soared, and Ford owned the market. But after a while the Model T grew old and an increasingly prosperous middle class wanted to trade in their old cars for more stylish sedans. Ford was slow to respond, because his production system was most efficient when making only one kind of car. Meanwhile at General Motors, Alfred P. Sloan had created an organization structured to produce multiple models aimed at segmented markets. As the demand for variety and complexity grew, Ford’s production system grew unwieldy.
Also as time passed workers began to feel trapped in untenable working conditions. They had become accustomed to a high standard of living and were unable to find comparable salaries elsewhere. The widespread labor unrest in the United States in the 1930s is often attributed to a system which held little respect for workers and regarded them as interchangeable.

The Toyodas

Kariya, Japan, February 1927. Toyoda Automatic Loom Works held a workshop for textile engineers to showcase the company’s new loom. First the visitors saw how Toyoda looms were manufactured with high precision tools, and then they were taken on a tour of the experimental spinning and weaving facility where 520 of the Toyoda looms were in operation. The looms were a wonder to behold; they ran at a blazingly fast 240 picks per minute and were operated by only 20 weavers. Anticipating a law abolishing nighttime labor, the machines were fully automatic and could run unattended all night. When a shuttle flying across the loom was just about out of thread, a new shuttle replaced it in a smooth, reliable exchange. If even one of the hundreds of warp threads broke or the weft thread ran out, the loom immediately stopped and signaled a weaver to fix the problem.

If you want to understand the Toyota1 Production System, it is important to appreciate just how difficult it was to develop and manufacture the “perfect loom.” Sakichi Toyoda built his first power loom in 1896 and invented an automated shuttle changing device in 1903. A test was set up to compare 50 Toyoda shuttle changing looms with a similar number of simple power looms from Europe. The results were disappointing. These early Toyoda looms were complex, low precision machines that were balky and difficult to maintain.

Sakichi Toyoda recruited technically competent employees and hired an American engineer, Charles A. Francis, to bring the American system of manufacture to his company. Francis redesigned the manufacturing equipment and built a machine tool shop to produce it. He developed standard specifications, produced standardized gauges and jigs, and reorganized the manufacturing line. At the same time, Sakichi Toyoda designed wider all-iron looms, and by 1909 he had patented a superior automated shuttle-change mechanism. Over the next decade, as war distracted Europe and America, looms designed by Sakichi Toyoda sold very well.

1. The “d” in the Toyoda family name was changed to a “t” when the Toyota Motor Company was established. The Japanese characters are similar, but Toyota takes two less brush strokes than Toyoda.
Although Sakichi Toyoda readily adopted high precision interchangeable parts, the loom manufacturing business had no room for interchangeable people. Automatic looms are complex, high precision machines, very sensitive to changes in materials and a challenge to keep running smoothly. Thus, highly skilled weavers were needed to set up and keep 25 or 30 machines running at once. If running a loom required skill, the design and manufacture of automated looms was even more demanding. Sakichi Toyoda had a reputation for hiring some of the most capable engineers being trained at Japanese universities. He kept his development team intact even as he started new companies, and he depended on them to carry on research in loom design and manufacture.

In 1921 Sakichi Toyoda’s son Kiichiro joined his father’s company and focused on advancing loom automation. In 1924 they jointly filed a patent for an improved automatic shuttle-change mechanism. The research team also developed methods to detect problems and stop the loom, so that looms could run unattended at night. Kiichiro Toyoda oversaw the building and start-up of a factory to produce the new looms, and set up 520 of them in the Toyoda experimental weaving factory. After he proudly showed off these “perfect looms,” orders for the automated looms poured in. Kiichiro used the profits to start up an automotive business. He toured Detroit and spent years learning how to build engines. Toyota’s first production car hit the market in 1936, but manufacturing was soon interrupted by war.

The Toyota Production System

Koromo, Japan, October 1949. Passenger car production restrictions were lifted in post-war Japan. In 1945, Kiichiro Toyoda had challenged his company to “catch up with America,” but it was clear that Toyota could not catch up by adopting America’s mass production model. Mass production meant making thousands of identical parts to gain economies of scale, but materials were scarce, orders were spotty, and variety was in demand. Economies of scale were simply not available.

Kiichiro Toyoda’s vision was that all parts for assembly should arrive at the assembly line “Just-in-Time” for their use. This was not to be accomplished by warehousing parts; parts should be made just before they are needed. It took time to make this vision a reality, but in 1962, a decade after Kiichiro Toyoda’s death, his company adopted the Toyota Production System companywide.
Taiichi Ohno

Taiichi Ohno was a machine shop manager who responded to Kiichiro Toyoda’s challenge and vision by developing what came to be known as the Toyota Production System. He studied Ford’s production system and gained insight from the way American supermarkets handled inventory. To this he added his knowledge of spinning and weaving and the insights of the workers he supervised. It took years of experimentation to gradually develop the Toyota Production System, a process that Ohno considered never-ending. He spread the ideas across the company as he was given increasingly broad areas of responsibility.

In his book, *Toyota Production System*, Ohno calls the Toyota Production System “a system for the absolute elimination of waste.” He explains that the system rests on two pillars: Just-in-Time flow and autonomation (also called Jidoka).

**Just-in-Time Flow**

It is important to note that Just-in-Time flow went completely against all conventional wisdom of the time. Resistance to Ohno’s efforts was tremendous, and he succeeded because he was backed by Eiji Toyoda, who held various senior management positions in the company after his cousin Kiichiro left in 1950. Both Toyodas had brilliantly perceived that the game to be played was not economies of scale, but conquering complexity. Economies of scale will reduce costs about 15 percent to 25 percent per unit when volume doubles. But costs go up by 20 percent to 35 percent every time variety doubles. Just-in-Time flow drives out major contributors to the cost of variety. In fact, it is the only industrial model we have that effectively manages complexity.

**Autonomation (Jidoka)**

Toyoda automated looms could operate without weavers present because the looms detected when anything went wrong and shut down automatically. Autonomation, or its Japanese name Jidoka, means that work is organized so that the slightest abnormality is immediately detected, work stops, and the cause of the problem is remedied before work resumes. Another name for this critical concept, and one that is perhaps easier to remember, is “stop-the-line.”

---

2. This section is based on Taiichi Ohno’s book, *Toyota Production System: Beyond Large-Scale Production*, Productivity Press, written in Japanese in 1978 and translated into English in 1988. It is an excellent book, very readable and highly recommended even today.


Ohno called autonomation “automation with a human touch.” He pointed out how the related word “autonomic” brings to mind another way to look at this concept. Our bodies have an autonomic nervous system that governs reflexes such as breathing, heartbeat, and digestion. If we touch something hot, our autonomic nerves cause us to withdraw our hand without waiting for the brain to send a message. Autonomation means the organization has reflexes in place that will respond instantly and correctly to events without having to go to the brain for instructions.5

Shigeo Shingo

Shigeo Shingo was a consultant who helped Ohno implement the Toyota Production System at Toyota, and later helped companies around the world understand and implement the system. Those of us who implemented Just-in-Time manufacturing in the early ’80s fondly remember the “Green Book,” 6 the first book on Just-in-Time published in English. It was not a good translation, and the material is heavy and technical, but it is a stunningly insightful book.

Shingo covers two major themes in the book: nonstock production and zero inspection. A careful look shows that these are actually the engineering equivalent of Ohno’s pillars of the Toyota Production System.

Nonstock Production

Just-in-Time flow means eliminating the stockpiles of in-process inventory that used to be made in the name of economies of scale. The focus is on making everything in small batches, and in order to do this, it is necessary to be able to changeover a machine from making one part to making a different part very quickly. In software development, one way to look at set-up time is to consider the time it takes to deploy software. Some organizations take weeks and months to deploy new software, and because of this they put as many features into a release as possible. This gives them a large batch of testing, training, and integration work to do for each release. On the other hand, I expect the anti-virus software on my computer to be updated with a well-tested release within hours after a new threat is discovered. The change will be small, so integration and training are generally not a concern.

Zero Inspection

The idea behind autonomation is that a system must be designed to be mistake-proof. There should not be someone looking for a machine to break or testing

---

5. Taiichi Ohno, Ibid., p. 46.
product to see if it is good. A properly mistake-proofed system will not need inspection. My video cable is an example of mistake-proofing. I can’t plug a monitor cable into a computer or video projector upside down because the cable and plug are keyed. So I don’t need someone to inspect that I plugged the cable in correctly, because it’s impossible to get it wrong. Mistake-proofing assumes that any mistake that can be made will eventually be made, so take the time at the start to make the mistake impossible.

Just-in-Time

The Toyota Production System was largely ignored, even in Japan, until the oil crisis of 1973, because companies were growing quickly and they could sell everything they made. But the economic slowdown triggered by the oil crisis sorted out excellent companies from mediocre ones, and Toyota emerged from the crisis quickly. The Toyota Production System was studied by other Japanese companies and many of its features were adopted. Within a decade America and Europe began to feel serious competition from Japan. For example, I (Mary) was working in a video cassette plant in the early ’80s when Japanese competitors entered the market with dramatically low pricing. Investigation showed that the Japanese companies were using a new approach called Just-in-Time, so my plant studied and adopted Just-in-Time to remain competitive.

The picture that we used at our plant to depict Just-in-Time manufacturing is shown in Figure 1.1.

Figure 1.1  Lower inventory to surface problems
Inventory is the water level in a stream, and when the water level is high, a lot of big rocks lurking under the water are hidden. If you lower the water level, the big rocks begin to surface. At that point, you have to clear the rock out of the way, or your boat will crash into them. As the big rocks are removed, you can lower inventory level some more, find more rocks, clear them out of the stream, and keep on going until there are just pebbles left.

Why not just keep the inventory high and ignore the rocks? Well, the rocks are things like defects that creep into the product without being detected, processes that drift out of control, finished goods that people aren’t going to buy before the shelf life expires, an inventory tracking system that keeps on losing track of inventory—things like that. The rocks are hidden waste that is costing you a lot of money—you just don’t know it unless you lower the inventory level.

A key lesson from our Just-in-Time initiative was that we had to stop trying to maximize local efficiencies. We had a lot of expensive machines, so we thought we should run them each at maximum productivity. But that only increased our inventory, because a pile of inventory built up at the input to each machine to keep it running, and at the output from each machine as it merrily produced product that had nowhere to go. When we implemented Just-in-Time, the piles of inventory disappeared, and we were surprised to discover that the overall performance of the plant actually increased when we did not try to run our machines at maximum utilization (see Figure 1.2).

Figure 1.2  Stop trying to maximize local efficiencies.
Stop-the-Line and Safety Consciousness

One Just-in-Time practice that was easy to adopt was a stop-the-line culture. Our video tape plant made tape out of some rather volatile materials, so we had an aggressive safety program in place. Through our safety program we already knew that it was important to investigate even the smallest accident, because small accidents will eventually turn into big accidents if they are ignored.

The book Managing the Unexpected by Weick and Sutcliffe shows that organizations like our plant create an environment where people pay attention to safety by maintaining a state of mindfulness. According to the authors, mindfulness has five characteristics:

1. **Preoccupation with Failure**
   We spent a lot of time thinking about what could go wrong and being prepared.

2. **Reluctance to Simplify**
   We had a large, complex plant, so safety was a large, complex issue.

3. **Sensitivity to Operations**
   Every manager in the plant was expected to spend time working on the line.

4. **Commitment to Learn from Mistakes**
   Even the smallest incident was investigated to determine how to prevent it from ever happening again.

5. **Deference to Expertise**
   Every manager knew that the people doing the work were the ones who really understood how the plant worked.

It was a small step to turn our safety culture into a stop-the-line culture. We added to our preoccupation with accidents a preoccupation with defects. Every step of every operation was mistake-proofed as we focused on eliminating the need for after-the-fact inspection. Whenever a defect occurred, the work team stopped producing product and looked for the root cause of the problem. If defective material made it through a process undetected, we studied the process to find out how to keep that from happening again. When I say "we" I refer to our production workers, because they were the ones who designed the process in the first place.

—Mary Poppendieck

---

When we decided to move our plant to Just-in-Time, there were few consultants around to tell us what to do, so we had to figure it out ourselves. We created a simulation by covering a huge conference table with a big sheet of brown paper, then drawing the plant processes on the paper. We made “kanban cards” by writing various inventory types on strips cut from index cards. We put an inventory strip into a coffee cup and—viola!—that cup became a cart full of the indicated inventory. (See Figure 1.3.) Then we printed a week’s packing orders and simulated a pull system by attempting to fill the orders, using the cups and the big sheet of paper like a game board. When a cup of inventory was packed, the inventory strip (kanban card) was moved to the previous process, which used it as a signal to make more of that material.

With this manual simulation we showed the concept of a pull system to the production managers, then the general supervisors, then the shift supervisors. Finally, the shift supervisors ran through the simulation with every worker in their area. Each work area was asked to figure out the details of how to make this new pull system work in their environment. It took some months of detailed preparation, but finally everything was ready. We held our collective breath as we changed the whole plant over to a pull system in one weekend. Computerized scheduling was turned off, its place taken by manual scheduling.

---

8. This scheduling approach is called Kanban, and the token showing what each process should make is called a kanban card.
Lean

In 1990 the book *The Machine That Changed the World* gave a new name to what had previously been called Just-in-Time or the Toyota Production System. From then on, Toyota’s approach to manufacturing would become known as Lean Production. During the next few years, many companies attempted to adopt Lean Production, but it proved remarkably difficult. Like all new industrial models, resistance from those invested in the old model was fierce.

Many people found Lean counterintuitive and lacked a deep motivation to change long established habits. Quite often companies implemented only part of the system, perhaps trying Just-in-Time without its partner, stop-the-line. They missed the point that, “The truly lean plant...transfers the maximum number of tasks and responsibilities to those workers actually adding value to the car on the line, and it has in place a system for detecting defects that quickly traces every problem, once discovered, to its ultimate source.”

Despite the challenges faced when implementing a counterintuitive new paradigm, many lean initiatives have been immensely successful, creating truly lean businesses, which have invariably flourished. Lean thinking has moved from manufacturing to other operational areas as diverse as order processing, retail sales, and aircraft maintenance. Lean principles have also been extended to the supply chain, to product development, and to software development. See Figure 1.4.

Lean Manufacturing/Lean Operations

Today lean manufacturing sets the standard for discipline, efficiency, and effectiveness. In fact, using lean principles in manufacturing often creates a significant competitive advantage that can be surprisingly difficult to copy. For example, Dell Computer’s make-to-order system routinely delivers a “custom-built” computer in a few days, a feat which is not easily copied by competitors unwilling to give up their distribution systems. Lean has moved into nonmanufacturing operations as well. Southwest Airlines focuses on transporting custom-
ers directly from point A to point B in relatively small planes, while competitors can’t easily abandon their large-batch oriented hub-and-spoke systems. A few industries, such as rapid package delivery, have been structured based on lean principles, and in those industries, only companies with lean operations can survive.

**Lean Supply Chain**

When lean production practices reach the plant walls, they have to be extended to suppliers, because mass production and lean manufacturing do not work well together. Toyota realized this early, and helped its suppliers adopt the Toyota Production System. Peter Drucker estimated that Toyota’s supplier network, which Drucker calls a Keiretsu, gives it a 25 percent to 30 percent cost advantage relative to its peers.\(^{11}\) When Toyota moved to the United States in the late 1980s, it established a similar supplier network. Remarkably, US automotive suppliers often have lean sections of their plants dedicated to supplying Toyota, while the rest of the plant has to be run the “traditional” way because other automotive companies cannot deal with a lean supplier.\(^ {12}\) A lean supply chain is

---

also essential to Dell, since it assembles parts designed and manufactured by other companies. Through “virtual integration,” Dell treats its partners as if they are inside the company, exchanging information freely so that the entire supply chain can remain lean.

In lean supply chains, companies have learned how to work across company boundaries in a seamless manner, and individual companies understand that their best interests are aligned with the best interests of the entire supply chain. For organizations involved in developing software across company boundaries, supply chain management provides a well-tested model of how separate companies might formulate and administer lean contractual relationships.

**Lean Product Development**

“The real differential between Toyota and other vehicle manufacturers is not the Toyota Production System. It’s the Toyota Product Development System,” says Kosaku Yamada, chief engineer for the Lexus ES 300. Product development is quite different than operations, and techniques that are successful in operations are often inappropriate for development work. Yet the landmark book *Product Development Performance* by Clark and Fujimoto shows that effective product development has much in common with lean manufacturing. Table 1.1 summarizes the similarities described by Clark and Fujimoto.

If any company can extract the essence of the Toyota Production System and properly apply it to product development, Toyota would be the top candidate. So there was no surprise when it became apparent in the late 1990s that Toyota has a unique and highly successful approach to product development. Toyota’s approach is both counterintuitive and insightful. There is little attempt to use the manufacturing-specific practices of the Toyota Production System in product development, but the underlying principles clearly come from the same heritage.

The product coming out of a development process can be brilliant or mundane. It might have an elegant design and hit the market exactly right, or it might fall short of both customer and revenue expectations. Toyota products tend to routinely fall in the first category. Observers attribute this to the leadership of a chief engineer, responsible for the business success of the product, who has both a keen grasp of what the market will value and the technical capability

---


Table 1.1  Similarities between Lean Manufacturing and Effective Product Development

<table>
<thead>
<tr>
<th>Lean Manufacturing</th>
<th>Lean Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frequent set-up changes</td>
<td>Frequent product changes (software releases)</td>
</tr>
<tr>
<td>Short manufacturing throughput time</td>
<td>Short development time</td>
</tr>
<tr>
<td>Reduced work-in-process inventory between manufacturing steps</td>
<td>Reduced information inventory between development steps</td>
</tr>
<tr>
<td>Frequent transfer of small batches of parts between manufacturing steps</td>
<td>Frequent transfer of preliminary information between development steps</td>
</tr>
<tr>
<td>Reduced inventory requires slack resources and more information flow between steps</td>
<td>Reduced development time requires slack resources and information flow between stages</td>
</tr>
<tr>
<td>Adaptability to changes in volume, product mix, and product design</td>
<td>Adaptability to changes in product design, schedule, and cost targets</td>
</tr>
<tr>
<td>Broad task assignments for production workers gives higher productivity</td>
<td>Broad task assignments for engineers (developers) gives higher productivity</td>
</tr>
<tr>
<td>Focus on quick problem solving and continuous process improvement</td>
<td>Focus on frequent incremental innovation and continuous product and process improvement</td>
</tr>
<tr>
<td>Simultaneous improvement in quality, delivery time, and manufacturing productivity</td>
<td>Simultaneous improvement in quality, development time, and development productivity</td>
</tr>
</tbody>
</table>

to oversee the systems design. In the book *The Toyota Way*, Jeffrey Liker recounts the stories of the development of the Lexus and the Prius, emphasizing how these breakthrough designs were brought to market in record time under the leadership of two brilliant chief engineers.

Product development is a knowledge creation process. Toyota’s Product Development System creates knowledge through broad exploration of design spaces, hands-on experimentation with multiple prototypes, and regular inte-

Integration meetings at which the emerging design is evaluated and decisions are made based on as much detailed information as possible. The tacit knowledge gained during both development and production is condensed into concise and useful one-page summaries that effectively make the knowledge explicit. Generating and preserving knowledge for future use is the hallmark of the Toyota Product Development System.

The National Center for Manufacturing Sciences (NCMS) conducted a multi-year study of the Toyota Product Development System, and the findings are summarized by Michael Kennedy in the book *Product Development for the Lean Enterprise*. In this book Kennedy identifies four cornerstone elements of the Toyota Product Development System (see Figure 1.5).

![Knowledge-Based Engineering](The Lean Development System)

**Figure 1.5** Cornerstone elements of the Toyota Product Development System

---


18. This figure is from Michael Kennedy, Ibid., p. 120. Used with permission.
The Toyota Product Development System

The Toyota Product Development System has four cornerstone elements:

1. **System Design by an Entrepreneurial Leader**
   The chief engineer at Toyota owns responsibility for the business success of the product. He is a very experienced engineer, fully capable of creating the system-level design of the vehicle. But he is also responsible for developing a deep understanding of the target market and creating a vehicle that will delight the customers. The chief engineer creates a vision of the new product which he transmits to the development team and refreshes frequently by talking to the engineers making day-to-day decisions. He defends the vision when necessary and arbitrates tradeoffs if disagreements arise. He sets the schedule and modifies the process so everything is pulled together on time.

2. **Expert Engineering Workforce**
   From the days of Sakichi Toyoda, the Toyoda and Toyota companies have always had top notch technical people designing their technically sophisticated products. It takes years for an engineer to really become an expert in a particular area, and at Toyota, engineers are not moved around or motivated to move into management before they truly master their field. Managers are teachers who have become masters in the area they supervise; they train new engineers and move them from apprentice to journeyman to master engineer.

3. **Responsibility-Based Planning and Control**
   The chief engineer sets the vehicle development schedule, which consists of key synchronization points about two or three months apart. Engineers know what is expected at the next synchronization point, and they deliver the expected results without being tracked. If engineers need information to do their job, they are expected to “pull” it from its source. Recently, Toyota chief engineers have pioneered the practice of an “Oobeya” or large room where team members may work, and the whole team meets regularly. The Oobeya contains big visible charts to show issues and status.

4. **Set-Based Concurrent Engineering**
   Set-based engineering means exploring multiple design spaces and converging on an optimal solution by gradually narrowing options. What does this mean in practice? It means being very careful not to make decisions until they absolutely must be made and working hard to maintain options so that decisions can be made as late as possible with the most amount of information possible. The paradox of set-based design is that this approach to creating knowledge builds redundancy into the development approach, which might appear to be a waste. However, when looking at the whole system, set-based design allows the development team to arrive at a more optimal solution much faster than an approach that closes off options quickly for the sake of being “decisive.”

---

19. For more on set-based engineering, see Chapter 7.
Lean Software Development

Software development is a form of product development. In fact, much of the software you use was probably purchased as a product. Software that is not developed as a standalone product may be embedded in hardware, or it may be the essence of a game or a search capability. Some software, including much custom software, is developed as part of a business process. Customers don’t buy the software we develop. They buy games or word processors or search capability or a hardware device or a business process. In this sense, most useful software is embedded in something larger than its code base.

It is the product, the activity, or the process in which software is embedded that is the real product under development. The software development is just a subset of the overall product development process. So in a very real sense, we can call software development a subset of product development. And thus, if we want to understand lean software development, we would do well to discover what constitutes excellent product development.

The Toyota Production System and the Toyota Product Development System stem from the same underlying principles. The first step in implementing lean software development is to understand these underlying principles, which will be discussed in the next chapter.

Try This

1. Go to the Toyota Web site, and view the videos on Jidoka (www.toyota.co.jp/en/vision/production_system/video.html). The videos on Just-in-Time and the Toyota Production System are also worth viewing.

2. Do you have a tendency to work in batches? If you had to mail 100 letters, how would you go about folding the letters, stuffing the envelopes, adding address labels and stamps? Would you process one envelope at a time, or would you perform each step in a batch? Why? Try timing both ways and see which is faster. If you have children, ask them how they would approach the problem.

---

20. This was a newly published Web site as of April, 2006. The page can also be reached by going to www.toyota.co.jp/en/ and following this sequence: Top Page > Company > Vision & Philosophy > Toyota Production System > Video Introducing the Toyota Production System.
3. Table 1.1 lists similarities between manufacturing and product development. Discuss this table with your team, one line at a time. Does it make sense in your world to think of partially done work as inventory? Do the other analogies make sense? Analogies are a double-edged sword. Where might the analogies between manufacturing and product development lead you astray?

4. Work-arounds: You have an organization of intelligent people. Do these people make it their job to work around problems, or are problems considered a trigger to stop-the-line and find the root cause? Make a list of the Top 10 problems that occurred in your group in the last week. List after each problem the way it was resolved. Rank each problem on a scale of 0–5. The rank of 5 means that you are confident that the cause of the problem has been identified and eliminated and it is unlikely to occur again. The rank of 0 means that there is no doubt the problem will crop up again. What is your total score?

5. If people in your organization instinctively work around problems, they have the wrong reflexes! Brainstorm what it will take to develop a culture that does not tolerate abnormalities, whether it is a broken build or a miscommunication, a failed installation or code that is not robust enough to hold up in production. Have a “stop-the-line” committee investigate the ideas and choose the best candidate to get started. In the one chosen area, switch from a work-around culture to a stop-the-line culture. Be sure reflexive stop-the-line habits are developed! Repeat.
## Index

3D modeling, 118  
3M  
   example, 213–214  
   feasibility stage, 47  
   focus on the job, 52  
   handspreads, 47  
   samples of new products, 47  
7 principles. See seven principles.  
7 wastes. See seven wastes.  
14 points of Deming, 122–123  
80/20 rule, 25–26

A  
A3 reports, 157–158  
acceptance tests, 150, 186. See also story tests.  
accommodations, 233  
accountability, 64–65  
Aden, Jill, 195  
adopting new technologies, 230–231  
agile software development, tools for. See Rally.  
Airline Information Management System, 119  
airport check-in desk example, 110  
airport delays, example, 136–137  
Aisin fire, 208–209, 211  
AJAX, 150  
Alias, 55  
alignment, 69  
allegiance, 214–216  
Allen, Charles, 234  
American auto industry, 2–3  
American System of Manufacture, 1  
analyzing the situation, 169  
andon, 139–140  
annual performance rating. See performance evaluations.  
applause, 210  
Appleton, Brad, 202  
approval process, 84, 103  
architecture, software definition, 20  
divisible systems, 182  
feedback and quality, 182  
assemble-to-order, 34  
assembly line. See mass production.  
assessment, 188–189  
asynchronous cadence, 109  
Austin, Rob, 40  
auto industry. See also specific industries.  
   America, 2–3  
   Japan, 4–7  
   used car sales, 41  
Autodesk, 55  
automating complexity, 72–73  
automating routine tasks, 197–198, 227–228, 231–232  
autonomation (Jidoka), 5–6  
availability of processes, 98

B  
BAA airport management, 217–218, 220–221  
backlog items, 185–186  
balanced scorecards, 144  
barriers  
   eliminating, 210  
   interdepartmental, 122  
batch and queue approach, 88  
Beck, Kent, xx  
Bell, Gordon, 165  
bell curve, and individual performance, 130  
Bell Laboratories, 121  
Benneton, 67  
Beyond the Goal, 230  
big visible charts, 140  
hilling system example, 167–168  
Black Belts, 229  
blame, 143  
Blanc, Honoré, 1–2  
Blenko, Marcia, 57  
BMI, 39  
BMI call center, outsourcing, 215  
Boehm, Barry, 33
INDEX

Boeing
    777 project, 117–120, 140, 230
    787 Dreamliner, 210
    outsourcing, 216–217
Bohnet, Ralph, 167
bonuses as incentives, 145, 146
books and publications
    Beyond the Goal, 230
    Cheaper by the Dozen, 37
    “Collaboration Rules,” 208
    Conquering Complexity in Your Business, 67
    “Do You Have Too Much IT?”, 69
    Estimating and Planning, 232–233
    Fit for Developing Software, 187
    Hidden Value, 146
    The Instructor, the Man and the Job, 234
    The Knowledge-Creating Company, 156
    Lean Software Development: An Agile Toolkit, xxiii
    Lean Solutions, 43
    The Living Company, 141, 225
    The Machine That Changed the World, 11
    Managing the Unexpected, 9
    Product Development for the Lean Enterprise, 15
    Product Development Performance, 13, 52
    “Quality With a Name,” 20
    Taxonomy of Problem Management Activities, 20
    Toyota Production System, 5
    The Toyota Way, 14
    The Ultimate Question, 241
    “When IT’s Customers Are External,” 62–63
    Who has the D?, 57
    Working Effectively with Legacy Code, 167
bottleneck elimination, xix
bottlenecks (Muri), xix
boundaries, organizational
cascading queues, 113–114
cost of crossing, 39–40, 243
lean supply chains, 13
relational contracts, 221
teams, 214
value streams, 84
boundaries, system, 201
Brin, Sergey, 46
building quality in, 25–29
burn-down charts, 140
business case, 240
business intent, testing, 200
business process, 17, 20, 181
business success
    constraints, 153
    responsibility for, 13, 16, 53
    rewards for, 145
C
cadence
    asynchronous, example, 109
cycle time reduction, 108–109
    establishing, 108–109
Cagan, Martin, 53
Canada, 231–232
capable development process, 98
capacity, limiting work to
    cycle time reduction, 110–111
    teams, 134
cascading queues, 113–114
cash stage, 49
cause. See root causes.
champions, 52–57, 133
change
    agents, 229
    management, 25
    scope bloat, 25
    scope control, 25
    tolerance, 182
    waste, 25
change for the better (Kaizen) events, 173–175
change requests, 62
chartering teams, 241
charts, 140
Cheaper by the Dozen, 37
chief architect, 133
chief engineer, 53–55
Christensen, Clayton, 226
Chrysler
    NS minivan, 56
    QFD (quality function deployment) analysis, 56
    shared leadership, 56
churn
    requirements, 24, 91
    test-and-fix, 24
    value streams, 91
    waste, 24
Clark, Kim B., 52
Clark, Mike, 197
ClearStream Consulting, 167–168
Cleland-Huang, Jane, 182
CMM, 124
coches, 133
code complexity, 69
  source of waste, 74–75
technical debt, 150
undeployed, 75
undocumented, 75
unsynchronized, 74
untested, 74
code reviews, 194–195
coffee cup simulation, 10–11
Cohn, Mike, 232
collaboration. See partners; teams.
“Collaboration Rules,” 208
collaboration. See also partners; teams.
collaboration. See also Just-in-Time commitment.
to change, 151
deferring, principle of, 32–33
iterative development, 186
planning as, 33
committees, 209–210
companies
  life expectancy, 225–227
  organizational boundaries
cascading queues, 113–114
cost of crossing, 39–40, 243
lean supply chains, 13
relational contracts, 221
teams, 214
value streams, 84
purpose of, 123
types of, 141
compensation
  alternatives to money, 145–146
  annual raises, 144
  balanced scorecards, 144
  bonuses, 145, 146
  promotion systems, 143–144
  reward basis, 144–145
  span of influence versus span of control, 144–145
  competing on the basis of time, 34
  competitive advantage
  complexity, 69
  customer satisfaction, 241
development speed, 35
expert workforce, 37
feedback, 177
lean principles, 11
management innovation, 124
outsourcing, 215–216
Toyota, 224
users, 189
complete teams, 57–60
complexity
  automating, 72–73
  competitive advantage, 69
cost of, 69–70
  limiting features and functions, 70–71
  minimum useful feature sets, 71–72
  pricing structure, example, 72–73
  prioritizing features, 71–72
  root cause of waste, 67
  software code, 69
concept stage, 46
concurrent development, 182
concurrent engineering, 16
condensing knowledge, 157
configuration management, 201–202
conflict of interest, 215
conquering complexity, 5
Conquering Complexity in Your Business, 67
constraints, 230–233
continuous improvement
  cadence, 168
  complexity reduction, 166
  configuration management, 201
  Deming’s 14 points, 122
development organization objectives, 239
  at PatientKeeper, 98
  principle of, 38
  waste elimination, 166
continuous integration, 202–203
contracts, 218
contracts
  BAA airport management, 217–218, 220–221
  fixed price, 125
  Norwegian Computer Society, 218–219
  NTNU (Norwegian University of Science and Technology), 218–219
  PS 2000, 218–219
  purpose of, 217
  relational, 219–221
  T5 Agreement, 217–218
time and materials, 218
Cook, Scott, 51, 55
costs
  competing on the basis of time, 34
  complexity, 24–25, 69–70
crossing organizational boundaries, 39–40, 243
economies of scale, 5
extra features, 24–25
joint ventures, 220
Keiretsu advantage, 12
INDEX

260

costs (continued):
lifecycle, 20, 70–71
refactoring, 166
of software maintenance, 20–21
standards, 193
support and warranty, 164
synergistic relationships, 221
target, 180, 218–219, 221

counterintuitive concepts
continuous integration, 202
Lean, 11
new paradigms, 11
object orientation, 195
set based development, 161
seven principles, 23
Crawford-Mason, Clare, 125
create knowledge, principle of, 29–32
Critical Chain, 232–233
cross-functional teams, 56, 64, 78, 122
Cunningham, Ward, 187
custom systems. See software development, custom systems.
customer-focused organizations
champions, 52–57
chief engineer, 53–55
complete teams, 57–60
decision making, 57
designing for manufacturability, 58–59
designing for operations, 58–59
development goal, 53
facilitating information flow, 52–60
leadership, 52–57
leadership teams, 55
Murphy’s Law, 59–60
responsibility, 56–57
shared leadership, 56
What can go wrong, will go wrong, 59–60
customers
delight, 49–52. See also Google.
focus on the job, 51–52
Kano model, 49–52
needs, 43
satisfaction, 49–52
satisfaction, as competitive advantage, 241
satisfaction, measurements, 241
service, example, 111–112
understanding, 50
cycle time
measurements, 238–240
PatientKeeper, 97–98
reducing
establishing a cadence, 108–109
evening out work arrival, 103–105
limiting work to capacity, 110–111
minimizing process elements, 105–107
minimizing process size, 107–108
pull scheduling, 112–114
speed, 98–99
utilization and, 102

D
Darwin Information Typing Architecture (DITA), 131
dashboards, 136, 140–141
de Geus, Arie, 141, 225
decisions. See also commitment.
irreversible, 160
key, 162
making, 57
decomposition, optimizing by, 40–41
defects
discovering versus preventing, 27, 82. See also test-driven development.
inspecting for, 27, 82
as management problems, 29
queues, 25–26
rates, 27, 34, 81, 85
seven wastes, 81–82
tracking systems, 27
defer commitment, principle of, 32–33
delays
mapping in value streams, 91
seven wastes, 80–81
delighters, 65
delight customers, 49–52
deliver fast, principle of, 34–35. See also speed.
Dell Computer, 11–13
Deming, W. Edwards
14 points
overview, 122–123
point 12, 210
points 6 and 7, 210
causes of problems, 121, 123–124
choosing suppliers, 122, 123
Deming Cycle, 121
dependence on inspection, 122
fear, 122
inherent system variation, 121
interdepartmental barriers, 122
introduction, 120
leadership, 122
numerical quotas, 123
PDCA (plan, do, check, act), 121, 154–155
pride of workmanship, 123
psychology, 122
purpose of a company, 123
scientific method, 121
slogans, exhortations, and targets, 123
synergy, 121
System of Profound Knowledge, 121
text of knowledge, 121
training, 122, 123
Deming Cycle, 121
democracy principle, Google, 45
Denne, Mark, 182
dependencies, teams, 135
deployment
available to production, 87, 90
average time, 6, 86
concept-to-launch time, 99, 103
cycle time, 170, 238–239
delays, 91
incremental, 178–179
minimum useful feature sets, 71
obsolescence, 91
Polaris project, 178–179
QFD (quality function deployment) analysis, 56
undeployed code, 75
design
of code. See software development.
intent, testing, 200
for manufacturability, 58–59
for operations, 58–59
of products. See Toyota Product Development System; Toyota Production System.
Design for Six Sigma (DFSS), 229
design/build teams, 118, 123, 133
deskilling, 228
deterministic school, 21
detractor, 65, 241
Detroit, 2, 4, 117
developing software. See software development.
development teams
3M, 56–60
capacity, 99
champions, 132
DFSS (Design for Six Sigma), 229
error prevention, 82
expertise, 129–130, 212
goal of, 240
incentives, 123
interaction designers, 189
joined at the hip, 55
maintenance duties, 79
measurements, 237
pride in workmanship, 210
process improvement, 31
pull scheduling, 112–114
rewards, 145
set-based concurrent engineering, 16
size, and technical debt, 153
DFSS (Design for Six Sigma), 229
differentiation, 50
discipline
automating routine tasks, 197–198
code reviews, 194–195
configuration management, 201–202
continuous integration, 202–203
five S’s, 190–192
merging subsystems, 203–204
mistake-proofing, 196–198
nested synchronization, 203–204
Open Source reviews, 196
organizing a workspace, 190–192
pairing, 195–196
shine (seiso), 191–192
sort (seiri), 191–192
standardize (seiketsu), 191–192
standards for software development, 193–196
sustain (shitsuke), 191–192
systematize (seiton), 191–192
test-driven development, 198–201
dispatching, 137–138
DITA (Darwin Information Typing Architecture), 131
divisible systems architecture, 182
Do It Right the First Time, 165
do it right the first time, 29
“Do You Have Too Much IT?”, 69
doctor’s appointments, example, 104–105
documentation, 74, 77
domain, 82, 180, 183
domain models, 185–186
Drucker, Peter, 12–13, 220–221
dual ladder, 143
dysfunctional measurements, 238

e
Easel Corporation, xvii
economic companies, 141
economies of scale, 4, 68
education. See training.
80/20 rule, 25–26
eliminate waste, principle of, 23–25
eliminating barriers, 210
embedded software, 20, 163
empirical school, 21
employees. See partners; people; teams.
engaged thinking people, 35, 37, 117, 237
enterprise software, 20
entrepreneurial leaders, 16, 37, 54
ERP (Enterprise Resource Planning), 231
estimates
as commitments, 232
granular level, 134
implementation effort, 185
stories, 183
tasks, 97
Estimating and Planning, 232–233
Evans, Eric, 186
Evans, Phillip, 208
Excel, 36
excellence principle, Google, 45
exchanging tests, 212
exhortations, 123
exhortations as incentives, 123
expediting projects, 98
experimentation, 171–172
expert technical workforce, 37
expertise, in teams, 129–131
exploratory tests, 201
extra features, as waste, 24–25, 75
F
FAA (Federal Aviation Administration), 119
face-to-face discussion, 78
fail fast, 118–119
fast delivery. See deliver fast; speed.
fear as incentive, 122
feasibility stage, 46–47
Feathers, Michael, 167
features
limiting, 70–71, 165
minimum useful sets, 71–72
prioritizing, 71–72
wastes, 24–25, 75
YAGNI (You Aren’t Going to Need It), 165
FedEx, 34
feedback, and quality
architecture, 182
competitive advantage, 177
iterative development, 183–190
Polaris program, 177–182
release planning, 179–181
financial results. See return on investment.
fire, Aisin plant, 208–209, 211
FIT (Framework for Integrated Tests), 75, 150,
187
Fit for Developing Software, 187
Fitnesse, 150
five S’s, 190–192
fixed price contracts, 125
fixtures, 187
focus on the job, 51–52
Ford, Henry, 2–3
Ford Motor Company, 2–3
14 points of Deming, 122–123
Fowler, Martin, 167
framework for integrated tests. See FIT
(Framework for Integrated Tests).
France, 1–2
Francis, Charles A., 3
frequent integration, 212
Fujimoto, Takahiro, 52
Fujitsu, 39
funding profiles, 61
future blindness, 226
G
games, 17, 48, 181
Gap, 68
Gates, Bill, 36
GE Workout, 173–175
genchi-genbutsu (go, see, confirm), 54
General Motors, 2–3
George, Michael, 67
Gilbreth, Frank, 37–38
Gilbreth, Lillian, 37–38
global networks, 210–214
global teams, 212
global work groups, 212
goal setting, 223
Goldratt, Eliyahu, 230, 232
Google
 corporate philosophy, 44
customer satisfaction, 50
democracy principle, 45
excellence principle, 45
history of, 43–44
Keyhole, 45
maps, 45
page rank system, 48
product development principles, 44–45
product development timeline
 cash stage, 49
concept stage, 46
feasibility stage, 46–47
pilot stage, 48
systems design stage, 47
queuing theory, 101–102
speed principle, 45
startup, 46–47
value principle, 44
workforce utilization, 101–102
Google Earth, 45
Google Local, 45
Green Book, 6

H
hack-a-thon, 152
hacking versus speed, 35
Hamel, Gary, 117, 124–125
handoffs, 77–78
hangers, theft of, 125
hardening software, 150–151
haste makes waste, 35
Heathrow, 217
help desk, BMI, 39
help each other, 35, 127, 129, 183
Hidden Value, 146
history of lean software development
  See Just-in-Time
  See mass production
  See Toyota Product Development System
  See Toyota Production System
H&M, 67
Honda, xxiii, 55
Honeywell, 119–120
HTTPUnit, 150
hypothesis development, 171, 234–241

I
IBM AT cables, 196–198
incentives
  applause, 210
  blame, 143
  individual performance, 142
  performance evaluations, 141–143
  rankings, 142–143
incremental development, dangers of, 164
incremental funding, 61
Inditex, 67, 69
individual performance as incentives, 142
industrial model, 2, 5, 11
infrastructure, outsourcing, 214–215
innovation
  management, 124, 218
  start of, 46
  Web inspired, 233
inspections
  dependence on, 122
discovering defects, 27, 82. *See also* test-driven development.
preventing defects, 27, 82. *See also* test-driven development.
purpose of, 27
types of, 27
The Instructor, the Man and the Job, 234
integration
  continuous, 202–203
  frequent, 212
interaction designers, 55, 130, 189
interchangeable parts, 1–2
interchangeable people, 2–3
interdepartmental barriers, 122
Internet age, and knowledge creation, 159
intrinsic rewards, 146
Intuit
  complete teams, 57–58
  founding of, 51
  leadership teams, 55
  limiting complexity, 70
QuickBooks, 70
Quicken
  introduction of, 51
  leadership teams, 55
  Quickrental Property Management, 57–58
inventory. *See also* Just-in-Time.
coffee cup simulation, 10–11
pull system, 10–11
rocks-and-stream metaphor, 7–8
as waste, 24
irreversible decisions, 160
ISO 9000, 124–125
IT departments
  accountability, 64–65
  business collaboration, 62–65
  cost, 68
  external customers, 62–63
  fixing, 64
guide to the use of technology, 69
  *versus* software companies, 62–65
  we-they model, 63
workload example, 103–104
iterative development
  assessment, 188–189
  commitment, 186
  example, 184
feedback and quality, 183–190
FIT (Framework for Integrated Tests), 187
implementation, 186–188
introduction, 183–184
iterative development (continued):
  overview, 183
  planning, 186
  preparation, 185–186
  stories, 183–186
  story-test driven development, 186
  user interface variation, 189–190

J
Japan. See also Toyota; Toyota Product Development System; Toyota Production System.
  auto industry, 4–7
  textile industry, 3–4
Java, five S’s, 192
Jefferson, Thomas, 1
Jensen, Bent, 80
Jidoka (autonomation), 5–6
JIFFIE, 151
job grades, 143–144
Job Instruction (JI) module, 235–236
Job Methods (JM) module, 235–236
Job Relations (JR) module, 235–236
Johnson, Jim, 24
joined at the hip, 55
joint ventures, 220–221
Jones, Daniel, 43
journey
accommodations, 233
  adopting new technologies, 230–231
  automating routine tasks, 227–228, 231–232
  centering on people, 227–228
  corporate life expectancy, 225–227
  Critical Chain, 232–233
  developing a hypothesis, 234–241
  ERP (Enterprise Resource Planning), 231
  future blindness, 226
  goal setting, 223
  measurement, 237–241
  push versus pull systems, 236–237
  right to think, 237
  road map, 242
  schedules, 228
Six Sigma, 229–230
  Theory of Constraints, 230–233
  thinking, 236–237
  tools versus results, 229–230
  training, 234–236
  the use of technology, 227–228
JR (Job Relations) module, 235–236
Jula, John, 54
junior people, 130–131, 144
JUnit, 150
Juran, J. M., 26
Just-in-Time. See also inventory.
  autonomation (Jidoka), 5–6
  definition, 4
  Green Book, 6
  Just-in-Time flow, 5
  maximizing local efficiencies, 8
  mistake-proof systems, 6–7
  nonstock production, 6
  rocks-and-stream metaphor, 7–8
  stop-the-line culture, 5–6
  zero inspection, 6–7
Just-in-Time commitment. See also commitment.
  dangers of incremental development, 164
  Do It Right the First Time, 165
  example, 167–168
  examples
    medical device interface, 162
    pluggable interfaces, 163
    red-eye reduction, 162–163
    introduction, 159–160
    irreversible decisions, 160
    key decisions, 162
    legacy systems, 166–168
    refactoring, 164–168
    and scientific method, 154
    set-based design, 160–164
    and waste, 164
  YAGNI (You Aren’t Going to Need It), 165
Just-in-Time manufacturing, 4–7

K
Kaizen (change for the better) events, 173–175
Kanban, 10–11, 136, 138–139
kanban cards, 10–11
Kano, Noriaki, 49–52
Kano model, 49–52
Keiretsu, 12–13
Kennedy, Michael, 15
key decisions, 162
Keyhole, 45
knowledge
creation
  A3 reports, 157–158
  condensing knowledge, 157
  in the Internet age, 159
  keeping notebooks, 156–157
  lost knowledge, 155–159
  principle of, 29–32
  problem definition, 152–153
  at Rally Software Development, 149–152
INDEX

265

technical debt, 150
tracking knowledge, 155–159
theory of, 121
knowledge-based engineering, 15
*The Knowledge-Creating Company*, 156

L

large group improvement, 173–175
large-batch software development, 71, 102
last responsible moment, 32, 161, 185
lava lamp, 140, 198
leadership
customer-focused organizations, 52–57
Deming’s points, 122
entrepreneurial, 16, 37, 54
Honda, 55
Intuit, 55
Open Source, 209–210
process, 132–133
Strong Project Leader, 54
team, 55, 132–133
technical, 132–133
traveling team leaders, 213
lean
definition, xxiii
initiatives
first step, 153
initiating. See journey.
reasons for failure, 153
manufacturing
*versus* development, 14
overview, 11–12
principles, competitive advantage, 11. See also seven principles.
production
*See also* lean, software development
*See also* mass production
*See also* Toyota Product Development System
*See also* Toyota Production System
Dell Computer, 11–13
flowchart, 12
Keiretsu, 12–13
knowledge-based engineering, 15
manufacturing, 11–12
manufacturing *versus* development, 14
operations, 11–12
product development, 13–15
Southwest Airlines, 11–12
supply chain, 12–13
Toyota *versus* other vehicle manufacturers, 13
software development
history of
*See* lean, production
*See* mass production
*See* Toyota Product Development System
*See* Toyota Production System
overview, 17
*Lean Solutions*, 43
learn-by-doing, 19
learning. See training.
legacy systems, 166–168
Lexus, 13
lifecycle costs, 20, 70–71
Liker, Jeffrey, 14
limiting work to capacity, 110–111, 134
Linux security breach, example, 207–208, 211
Little’s Law, 100–101
*The Living Company*, 141, 225
L.L. Bean, 34
local efficiencies, 8
looms, automated, 3–4
lost knowledge, 155–159

M

MacCormack, Alan, 30
MacGibbon, Simon, 62
*The Machine That Changed the World*, 11
maintenance
cost of, 20–21
staffing for, 79–80
management
functional, 133
innovation as competitive advantage, 124
people. See people, managing
project, 133. See also project managers.
*Managing the Unexpected*, 9
manufacturing. See also* Toyota Product Development System; Toyota Production System.
*versus* development, 14
Just-in-Time, 4–7
lean, 14
lean production, 11–12
mass production, 12–13
video cassettes, 59
mapping value streams. See value streams.
maps, Google, 45
Marick, Brian, 166, 199
market research, 56, 62–63
market share, 61, 241
Martens, Ryan, 149
mass production
See also lean, production
See also Toyota Product Development System
See also Toyota Production System
American auto industry, 2–3
American System of Manufacture, 1
Ford Motor Company, 2–3
General Motors, 2–3
interchangeable parts, 1–2
interchangeable people, 2–3
Japanese auto industry, 4–7
Japanese textiles, 3–4
Just-in-Time manufacturing, 4–7
and lean manufacturing, 12–13
maximizing local efficiencies, 8
McAfee, Andrew, 69
McCabe Cyclomatic Complexity Index, 194–195
Measure UP, 40–41
measurements
customer satisfaction, 241
cycle time, 238–240
decreasing number of, 40–41
dysfunctional, 238
improving the wrong ones, 237
Measure UP, 40–41
net promoter score, 241
optimize by decomposition, 40–41
raising levels, 40–41
reducing the number of, 238
ROI (return on investment), 240–241
Sloan, Alfred P., 40–41
Sloan’s metrics, 40–41
statistical process control, 120–122
medical device interface example, 162
Meszaros, Gerard, 167
MetaScrum meeting, xvii
metrics. See measurements.
Microsoft, respect for people, 36
Miller, Lynn, 55, 189
mindfulness, 9
mind-meld, 50
minimum useful feature sets, 71–72
Minoura, Teruyuki, 236
mistake-proofing, 6–7, 196–198
money, as incentive, 145–146
Muda (waste), xix
Mugridge, Rick, 187
Mulally, Alan, 118, 123, 140
multitasking, causing waste, 78–80
Mura (stress), xix
Muri (bottlenecks), xix
Murphy’s Law, 59–60
myths
finishing the code, 79
haste makes waste, 35
one best way, 37–38
optimize by decomposition, 40–41
planning is commitment, 33
predictable outcomes, 31–32
specifications reduce waste, 24–25
testing to find defects, 28–29
N
National Center for Manufacturing Sciences (NCMS), 13
nested synchronization, 203–204
net promoter score, 241
New United Motor Manufacturing Incorporated (NUMMI), 226
newspaper, online subscription, 50
no partial credit, 188
no secrets, 118
Nonaka, Ikujiro, 156
nonfunctional requirements, testing, 201
nonstock production, 6
non-value-added waste, 23, 83
Norwegian Computer Society, 218–219
Norwegian University of Science and Technology (NTNU), 218–219
notebooks, keeping, 156–157
NS minivan, 56
numerical quotas, 123
O
Ohno, Taiichi
introduction, 5–6
planning, 33
value streams, 83
waste, 23–25, 75
on the job training, 234–236
one best way, 2, 37–38
one click build, 198
Oobeya, 213
Open Source
chief engineer approach, 54
leadership, 54
reviews, 196
software example, 209–210
Strong Project Leader, 54
operations, lean, 11–12
optimize by decomposition, 40–41
optimize the whole, principle of, 38–41
options-based development, 135
ordinary employees, 117, 227
O'Reilly, Charles, 146
organizational boundaries. See boundaries, organizational.
organizing a workspace, 190–192
organizing work, 138–139
outsourcing
  basic principles, 216–217
  BMI call center, 215
  Boeing, 216–217
  competitive advantage, 215–216
  conflict of interest, 215
  development, 216–217
  infrastructure, 214–215
  introduction, 214
  Procter & Gamble, 216–217
  Toyota, 216–217
  transactions, 215
overproduction, 25, 75
overtime, 110–111

P
Page, Larry, 46
page rank system, Google, 48
pairing, 195–196
Pareto analysis, 26
partially done work, 74–75
partners. See also teams.
  committers, 209–210
  contracts
    BAA airport management, 217–218, 220–221
    Norwegian Computer Society, 218–219
    NTNU (Norwegian University of Science and Technology), 218–219
    PS 2000, 218–219
    purpose of, 217
    relational, 219–221
    T5 Agreement, 217–218
  Deming point 12, 210
  eliminating barriers, 210
  equality of, 213
examples
  3M, 213–214
  Boeing 787 Dreamliner, 210
  Linux security breach, 207–208, 211
  Open Source software, 209–210
  Procter & Gamble, 210
  exchanging tests, 212
  frequent integration, 212
  global networks, 210–214
  global teams, 212
  global work groups, 212
  joint ventures, 220–221
  leaders, 209–210
  Oobeya, 213
outsourcing
  basic principles, 216–217
  BMI call center, 215
  Boeing, 216–217
  development, 216–217
  infrastructure, 214–215
  introduction, 214
  Procter & Gamble, 216–217
  Toyota, 216–217
  transactions, 215
  proxies, 213
  rotating people, 212
  synergy, 207–217
  traveling team leaders, 213
  war room, 213
PatientKeeper
  cycle time, 97–98
  delivery speed, 95–98
  development teams, 97
  introduction of Scrum, xvii
  limiting complexity, 71
  limiting work to capacity, 134
  product managers, 97
  release schedules, 97
PBS documentary, 119
PDCA (plan, do, check, act), 121, 154–155
people, managing
  andon, 139–140
  under the bell curve, 130
  Boeing 777 project, 117–120, 140
  causes of low quality and productivity, 121
  centering on people, 227–228
  choosing suppliers, 122
  compensation
    alternatives to money, 145–146
    annual raises, 144
    balanced scorecards, 144
    bonuses, 145, 146
    promotion systems, 143–144
    reward basis, 144–145
    span of influence versus span of control, 144–145
  dashboards, 136, 140–141
  Deming Cycle, 121
  Deming on, 120–123
  dependence on inspection, 122
  fear, 122
people, managing (continued):

- incentives
  - individual performance, 142
  - performance evaluations, 141–143
  - rankings, 142–143
- inherent system variation, 121
- interdepartmental barriers, 122
- job grades, 143–144
- junior people, 130–131, 144
- kanban, 136, 138–139
- leadership, 122
- numerical quotas, 123
- ordinary employees, 117, 227
- organizing work, 138–139
- PBS documentary, 119
- PDCA (plan, do, check, act), 121, 154–155
- pride of workmanship, 123
- projects versus products, 62
- psychology, 122
- rotating assignments, 212
- scientific method, 121
- self-directing work, 137–141
- sharing early and often, 118
- slogans, exhortations, and targets, 123
- stop-the-line culture, 139–140
- synergy, 121
- System of Profound Knowledge, 121
- testing early, failing fast, 118–119
- theory of knowledge, 121
- training, 122, 123, 129
- trust, 125
- visible signals, 139–140
- visual workspace, 136–141
- wall charts, 140
- why programs fail, 124–125
- Working Together program, 118–120
- performance evaluations as incentives, 141–143
- personnel. See partners; people; teams.
- PERT (Program Evaluation and Review Technique), 179
- Pfeffer, Jeffrey, 146
- pilot stage, 48
- P&L (profit and loss) model, 240
- plan, do, check, act (PDCA), 121, 154–155
- plan-driven methods, 33
- planning
  - as commitment, 33
  - iterative development, 186
- Taiichi Ohno on, 33
- pluggable interfaces example, 163
- Polaris program, 177–182
- policies. See practices; principles.

Post-it Notes, 139
practices. See also principles.
- definition, 19
  - for successful software development, 30
- predictable outcomes, 31–32
- Price, Jerry, 125
- pricing structure, complexity example, 72–73
- pride of workmanship, 123
- principles. See also practices.
  - continuous improvement, 38
  - definition, 19
  - Google
    - democracy principle, 45
    - excellence principle, 45
    - product development principles, 44–45
    - speed principle, 44
  - value principle, 44
  - lean software development. See seven principles.
- learn-by-doing, 19
- of outsourcing, 216–217
- software development, 20–21
- understand-before-doing, 19
- prioritizing features, 71–72
- Prius, 21

problem solving
- analyzing the situation, 169
- defining the problem, 152–153, 169
- disciplined approach, 169–172
- experimentation, 171–172
- first rule, 168
- follow up, 172
- hypothesis generation, 171
- introduction, 168
- Kaizen (change for the better) events, 173–175
- large group improvement, 173–175
- scientific method, 154, 169–172
- standardization, 172
- verifying results, 172
- process cycle efficiency, 85–86, 90–92, 108
- process leadership, 132–133
- processes
  - availability, 98
  - average time, calculating, 100–101
  - capable, 98
  - minimizing elements, 105–107
  - minimizing size, 107–108
  - quality measurement, 99
  - robust, 177
  - too big, 107–108
  - too many things, 105–107
Procter & Gamble, 51, 210, 216–217
product development, lean, 13–15
*Product Development for the Lean Enterprise*, 15
*Product Development Performance*, 13, 52
product managers, 133
product owners, 133
productivity, 28
products
  concept stage, 46
development. *See* software development;
  Toyota Product Development System;
  Toyota Production System.
  *versus* projects, 60–63
specifications
  basis for acceptance tests, 150
  waste reduction, 24–25
profit, definition, 152
profit and loss (P&L) model, 240
profitability, 61, 122, 241–242
Program Evaluation and Review Technique (PERT), 179
programmer tests. *See* unit tests.
programmers. *See* partners; people; teams.
project managers, 42, 127, 133, 237. *See also* management.
projects
  average process time, calculating, 100–101
  average speed, 99–100
  cycle time, 98–99
  dividing work into stories, 99
  expediting, 98
  measuring, 99
  PatientKeeper delivery cycle, 95–98
  process availability, 98
  process capability, 98
  *versus* products, 60–63
  red flags, 98
  setting upper limits, 99
  setting upper size limits, 99
  time delays, 98–99
promotion systems as incentives, 143–144
property tests, 201
Proulx, Tom, 55
proxies, 213
PS 2000 contract, 218–219
psychology, 122
pull scheduling, example, 112–113
pull systems, 10–11, 236–237
push systems, 236–237
Q
QA (Quality Assurance), 89, 96
QFD (Quality function deployment) analysis, 56
  building in, principle of, 25–29
  change tolerance, 182
  discipline
    automating routine tasks, 197–198
    code reviews, 194–195
    configuration management, 201–202
    continuous integration, 202–203
    five S’s, 190–192
    merging subsystems, 203–204
    mistake-proofing, 196–198
    nested synchronization, 203–204
    Open Source reviews, 196
    organizing a workspace, 190–192
    pairing, 195–196
    shine (seiri), 191–192
    sort (seiz), 191–192
    standardize (seiketsu), 191–192
    standards for software development, 193–196
    sustain (shitsuke), 191–192
    systematize (seiton), 191–192
    test-driven development, 198–201
divisible systems architecture, 182
iterative development
  assessment, 188–189
  commitment, 186
  example, 184
  FIT (Framework for Integrated Tests), 187
  implementation, 186–188
  introduction, 183–184
  overview, 183
  planning, 186
  preparation, 185–186
  stories, 183–186
  story-test driven development, 186
  user interface variation, 189–190
robust development processes, 177
role of feedback
  architecture, 182
  iterative development, 183–190
  Polaris program, 177–182
  release planning, 179–181
Quality Assurance (QA), 89, 96
quality function deployment (QFD) analysis, 56
  "Quality With a Name," 20
queuing theory. *See also* speed.
  average process time, calculating, 100–101
  cascading queues, 113–114
queuing theory (continued):
cycle time reduction
establishing a cadence, 108–109
evening out work arrival, 103–105
limiting work to capacity, 110–111
minimizing process elements, 105–107
minimizing process size, 107–108
pull scheduling, 112–114
examples
airport check-in desk, 110
asynchronous cadence, 109
customer service, 111–112
doctor’s appointments, 104–105
IT workload, 103–104
pull scheduling, 112–113
release cycles, 107–108
a seven year list, 106–107
thrashing, 111–112
Google, 101–102
Little’s Law, 100–101
system stability, 101–102
utilization, 101–102
variation, 101–102
QuickBooks, 70
Quicken Rental Property Management, 57–58
R
raises as incentives, 144
Rally Software Development, 149–152
ranking people, 128, 142–143
Raymond, Eric, 54
red-eye reduction example, 162–163
refactoring, 164–168
Reichheld, Fred, 241
relational contracts, 219–221
relearning, 76
release cycles, example, 107–108
release planning, 179–181
remote teams, 212–213
repeatable reliable cycle time, 238
requirements
churn, 24, 91
nonfunctional, 182, 201
overloading, 25
SRS (Software Requirements Specifications), 75
stale, 74
test specs, 82
timing assumptions, 233
too early, 24, 91
respect for people, 3, 36–38
response time
by category, 84
at peak capacity, 101
queue length, 172
reliability, 98
testing, 201
responsibility, 56–57
responsibility-based planning and control, 133–135
retrospectives, 236
return on investment (ROI), 41, 240–241
reversible decisions, 32
rewards. See also compensation; incentives.
    basis for, 144–145
    intrinsic, 146
right to think, 237
risk
    contracting away, 218
    custom software development, 181
    partially done work, 24
    refactoring, 164
river companies, 141
robust development processes, 177
rocks-and-stream metaphor, 7–8
Rogers, Paul, 57
root causes
    failure of lean initiatives, 153
    group improvement failure, 174
    low quality and productivity, 121, 123–124
    of problems, 121, 123–124
    technical debt, 150
    waste, 67
rotating people, 212
S
safety considerations, stop-the-line culture, 9
sales, engineering, development (SED) system, 55
Sapolsky, Harvey, 179
satisfaction, customer, 49–52, 241
schedules
    inventory. See Just-in-Time.
    Kanban, 10–11
    PatientKeeper releases, 97
    philosophy of, 228
    stepping dates, 133–134
    and teams, 134, 135
Schnaith, Kent, 192
Schwaber, Ken, xvii
scientific method
    Deming Cycle, 121
    Just-in-Time commitment, 154
    managing people, 121
    problem solving, 154, 169–172
    steps of, 154
stop-the-line culture, 154
Toyoda, Kiichiro, 154
Toyoda, Sakichi, 154
Toyota Production System, 154
scope bloat, 25
scope control, 25
Scrum
bottleneck elimination, xix
creation of, xvii–xviii
definition, 28
quality improvement, 28
stress avoidance, xix
Type A, xvii
Type B, xvii
Type C, xvii
waste elimination, xix
winning companies, xvii
winning product portfolio, xvii
winning teams, xvii
Scrum Product Owners, 133
ScrumMasters, 133
Sears, 34
SED (sales, engineering, development) system, 55
seiketsu (standardize), 191–192
seiri (sort), 191–192
seiso (shine), 191–192
seiton (systematize), 191–192
self-directing work, 137–141
self-organization, 17, 19, 97
set-based design, 160–164
seven principles
building quality in, 25–29
create knowledge, 29–32
defer commitment, 32–33
deliver fast, 34–35
eliminate waste, 23–25
myths
haste makes waste, 35
one best way, 37–38
optimize by decomposition, 40–41
planning is commitment, 33
predictable outcomes, 31–32
specifications reduce waste, 24–25
testing to find defects, 28–29
optimize the whole, 38–41
respect people, 36–38
seven wastes. See also waste.
defects, 81–82
delays, 80–81
extra features, 75
handoffs, 77–78
partially done work, 74–75
relearning, 76
task switching, 78–80
seven year list, example, 106–107
shared leadership, 56
sharing early and often, 118
Shewhart Cycle, 121
Shimmings, Ian, 41
shine (seiso), 191–192
Shingo, Shigeo
introduction, 6–7
purpose of inspections, 82
seven wastes, 73
types of inspections, 27
ship builders, training, 234–236
shitsuke (sustain), 191–192
Shook, Jim, 35
Shore, Jim, 20
Sienna minivan, 53–55
Silicon Valley Product Group, 53
silos, 40, 131
simulation, kanban cards, 10
single point of responsibility, 65
Six Sigma, 124, 229–230
slack, 15, 88, 102, 112, 134
slipping dates, 133–134
Sloan, Alfred P., 2, 40–41
slogans as incentives, 123
small batches, 15, 74, 101–102, 196
Smalley, Art, 153
Smith, Levering, 178
Sobek, Durwood, 53
software
cost of maintenance, 20–21
development timeline, 20
difficult to change. See legacy systems.
embedded, definition, 20
enterprise, definition, 20
legacy, 166–168
structure of. See architecture, software.
software companies versus internal IT, 62–65
software development
capable processes, 98
concurrent, 182
defect queues, 25–26
detailed design, 29–30
deterministic school, 21
empirical school, 21
handling changes. See change, management.
large-batch approach, 71, 102
outsourcing, 216–217
plan-driven methods, 33
software development (continued):
principles of. See principles; seven principles.
process quality measurement, 99
speed, competitive advantage, 35
speed versus hacking, 35
systematic learning, 31
waterfall model, 22, 29–30
software development, custom systems
accountability, 64–65
beginning/end criteria, 62
change requests, 62
funding profiles, 61
IT departments
accountability, 64–65
fixing, 64
versus software companies, 62–65
we-they model, 63
IT—business collaboration, 62–65
products versus projects, 60–63
software companies versus internal IT, 62–65
staffing, 62
we-they model, 63
Software Requirements Specifications (SRS), 75
sort (seiri), 191–192
Southwest Airlines, 11–12
span of influence versus span of control, 144–145
specialists in teams, 130–131
specification-by-example, 200
specifications, 24–25, 150
speed. See also deliver fast; queuing theory.
average projects, 99
cycle time, 98–99
dividing work into stories, 99
expediting, 98
versus hacking, 35
measuring, 99
PatientKeeper delivery cycle, 95–98
principle of, 45
process availability, 98
process capability, 98
red flags, 98
setting upper limits, 99
time delays, 98–99
unique projects, 100
Spolsky, Joel, 36
Spring, 150
Sprints, at PatientKeeper, xvii
SRS (Software Requirements Specifications), 75
staffing. See partners; people; teams.
Stalk, George, 5, 35
standardization, problem solving, 172
standardize (seiketsu), 191–192
standards for software development, 193–196
statistical process control, 120–122
stealing hangers, 125
stop-the-line culture
andon, 139–140
definition, 5–6
safety considerations, 9
and scientific method, 154
stories
dividing work into, 99
iterative development, 183–186
no partial credit, 188
story tests, 200. See also acceptance tests.
story-test driven development, 186
strangling legacy code, 167
stress (Mura), xix
stress avoidance, xix
Strong Project Leader, 54
suggestion systems, 236
supervisors. See people, managing.
suppliers, choosing, 122, 123
supply chain, lean, 12–13
targets as incentives, 123
task switching, 78–80
Taxonomy of Problem Management Activities, 20
Taylor, Frederick Winslow, 2, 37, 227
TDD (test-driven development). See test-driven development.
teachers. See training.
teams. See also partners.
barriers to, 128
champions, 133
characteristics of, 126–127
chartering, 241
coaches, 133
co-located, 211, 213
complete, 57–60
cross-functional, 56, 64, 78, 122
dependencies, 135
design/build, 118, 123, 133
development
3M, 56–60
capacity, 99
champions, 132
DFSS (Design for Six Sigma), 229
error prevention, 82
expertise, 129–130, 212
goal of, 240
incentives, 123
interaction designers, 189
joined at the hip, 55
maintenance duties, 79
measurements, 237
pride in workmanship, 79
process improvement, 31
pull scheduling, 112–114
rewards, 145
set-based concurrent engineering, 16
size, and technical debt, 153
expertise, 129–131
global, 212
Honda, 55
versus individual efforts, 126
Intuit, 55, 57–58
leadership, 55, 132–133
limiting work to capacity, 134
organizational boundaries, 214
product managers, 133
Quicken, 55
ranking systems, 128
remote, 212–213
responsibility-based planning and control, 133–135
schedules, 134, 135
Scrum Product Owners, 133
ScrumMasters, 133
silos, 131
slipping dates, 133–134
specialists, 130–131
variation, 135
winning, xvii
work breakdown structure, 135
versus workgroups, 126–127, 212
Teamwork is the key..., 56–57
technical debt, 150

technical leadership, 132–133
technical success, 145
technical writers, 75, 130–131
test early, fail fast, 118–119
test harness
acceptance tests, 202
benefits of, 82
legacy systems, 166–167
schedule, 27
unit tests, 200
usability tests, 201
user interface, 151
test-and-fix churn, 24
test-driven development (TDD)
exploratory tests, 201
productivity, 28
property tests, 201
purpose of, 199
story tests, 200
types of tests, 199
unit tests, 200
usability tests, 201
testing
3D modeling, 118
automating, 82
Boeing 777, 118–120
business intent, 200
design intent, 200
to find defects, 28–29. See also test-driven development.
nonfunctional requirements, 201
testing early, failing fast, 118–119
too late, 88, 91
user interface, 150–151, 201
verification, role of, 29
testing early, failing fast, 118–119
tests
acceptance, 150, 186
acceptance-test-driven development, 186
exchanging, 212
programmer. See unit tests.
story-test driven development, 186
unit, 200
usability, 21
textile industry, Japan, 3–4
Theory of Constraints, 230–233
theory of knowledge, 121
thinking, 236–237
thinking tools, 21–22, 195
thrashing, example, 111–112
time, competing on the basis of, 34
timebox, 32, 181
 timelines. See value streams.
too many things in processes, 105–107
too much work. See limiting work to capacity.
tools versus results, 229–230
towering technical competence, 129
Toyoda, Eiji, 5, 226
Toyoda, Kiichiro
  incentives, 141
  introduction, 4
  scientific method, 154
  tracking knowledge, 155
Toyoda, Sakichi
  evolutionary thinking, 226–227
  incentives, 141
  introduction, 3
  scientific method, 154
  tracking knowledge, 155
Toyota
  chief engineer, 53–55
  competitive advantage, 224
  fire at Aisin plant, 208–209, 211
  genchi-genbutsu (go, see, confirm), 54
  versus other vehicle manufacturers, 13
  outsourcing, 216–217
  problem definition, 152–153
  product delivery deadlines, 161
  profits, xxiii
  responsibility, 56–57
  responsibility-based planning and control, 133–135
  set-based design, 161
  Sienna minivan, 53–55
  Smart Car initiative, 224–225
  Teamwork is the key, 56–57
  towering technical competence, 129
  training new engineers, 129

Toyota Product Development System
 See also Just-in-Time manufacturing
 See also mass production
 See also Toyota Production System
 cornerstone elements, 16
 entrepreneurial leadership, 16
 expert engineering workforce, 16
 respect for people, 36–37
 responsibility-based planning and control, 16
 set-based concurrent engineering, 16
 software development philosophy, 21
 study of, 15

Toyota Production System
 See also Just-in-Time manufacturing
 See also mass production
 See also Toyota Product Development System
 automated looms, 3
 automonation (Jidoka), 5–6
 detecting abnormalities. See automonation (Jidoka); stop-the-line culture.
 goals, 152–153
 Japanese auto industry, 4–7
 Just-in-Time flow, 4–5
 overview, 4–7
 push versus pull systems, 236–237
 scientific method, 154
 versus Six Sigma, 229–230
 thinking, 236–237
 value streams, 83
 Toyota Production System, 5
 The Toyota Way, 14
 traceability, 75, 199
 tracking knowledge, 155–159
 tradeoffs, 41, 158, 241
 training
 Allen’s steps, 234–236
 Deming’s points, 122, 123
 on the job, 234–236
 Job Instruction (JI) module, 235–236
 Job Methods (JM) module, 235–236
 Job Relations (JR) module, 235–236
 new engineers, 129
 ship builders, 234–236
 TWI (Training Within Industry), 235–236
 vocational education, 234–236
 Training Within Industry (TWI), 235–236
 transactions, outsourcing, 215
 traveling team leaders, 213
 trust, 125
 Turner, Richard, 33
 Type A Scrum, xvii
 Type B Scrum, xvii
 Type C Scrum, xvii

U
 The Ultimate Question, 241
 uncoded documentation, waste, 74
 undeployed code, waste, 75
 understand-before-doing, 19
 undocumented code, waste, 75
 unit tests, 200
 United Airlines, 117–118
 United Kingdom, 41, 193, 217
 United States
  3M tour, 213
  Deming and, 121
  doctor’s appointments, 104
  invention of interchangeable parts, 1–3
liens registry, 231
Toyota manufacturing, 216, 226
Toyota moves to, 12
unsynchronized code, waste, 74
untested code, waste, 74
unused documentation, waste, 77
US War Production Board, 235
usability tests, 201
used car sales, 41
user interface
  competitive advantage, 189
  iterative design, 189–190
  testing, 150–151, 201
  variation, 189–190
utilization
  and cycle time, 102, 244
  full, 88
  Google workforce, 101–102
  and queuing theory, 101–102
  and variation, 101–114

V
value
  customer-focused organizations
    champions, 52–57
    chief engineer, 53–55
    complete teams, 57–60
    decision making, 57
    designing for manufacturability, 58–59
    designing for operations, 58–59
    development goal, 55
    facilitating information flow, 52–60
    leadership, 52–57
    leadership teams, 55
    Murphy’s Law, 59–60
    responsibility, 56–57
    shared leadership, 56
  What can go wrong, will go wrong, 59–60
customers
  delighting, 49–52. See also Google.
    focus on the job, 51–52
    Kano model, 49–52
    needs, 43
    satisfaction, 49–52
    understanding, 50
value principle, 44
value streams
  churn, 91
  delays, 91
  examples, 85–91
  for future processes, 92
  keeping it simple, 85
mapping, 83–84
owner identification, 84–85
preparation, 83–84
start/stop points, 84
waste diagnosis, 91
Van Schooenderwoert, Nancy, 27
variation
  inherent in the system, 121
  and queuing theory, 101–102
  and utilization, 101–114
variation in teams, 135
verification, and long release cycles, 107–108
verifying results of problem solving, 172
video cassettes, manufacturing, 59
visible signals, 139–140
vision, 16
visual workspace, 136–141
“vital few and trivial many” rule, 26
vocational education, 234–236
voice of the customer, 53, 229
volunteers, 54, 208–210

W
waiting. See delays.
Wake, Bill, 165
wall charts, 140
war room, 213
waste. See also seven wastes.
  80/20 rule, 25–26
  anticipating, 76
  biggest source of, 24–25
  churn, 24
  complexity and, 67, 69–73
  diagnosing. See value streams.
  elimination
    principle of, 23–25
    reducing by specification, 24–25
  Taiichi Ohno on, 23–25
  extra features, 24–25
  inventory as, 24
  Just-in-Time commitment, 164
  lost knowledge, 76
  Muda, xix
  multitasking, 78–80
  non-value-added, 23, 83
  partially done software, 24
  recognizing, 23. See also value streams.
  requirements churn, 24
  root cause, 67
  test-and-fix churn, 24
  uncoded documentation, 74
  undeployed code, 75
waste (continued):
  undocumented code, 75
  unsynchronized code, 74
  untested code, 74
  unused documentation, 77
  “vital few and trivial many” rule, 26
waste (Muda), xix
waterfall development model, 22, 29–30
Weick, Karl E., 9
Welch, Jack, 173
we-they model, 63
What can go wrong, will go wrong, 59–60
“When IT’s Customers Are External,” 62–63
Whitney, Eli, 1
Who has the D?, 57
Wild, Werner, 159
winning companies, xvii
winning product portfolio, xvii
winning teams, xvii

Wolf, Bob, 208
Womack, James, 43
work breakdown structure, 135
workers. See partners; people; teams.
workgroups, 126–127, 212
Working Effectively With Legacy Code, 167
Working Together program, 118–120
Workout, 173–175
write less code, 29, 67–73

Y
YAGNI (You Aren’t Going to Need It), 165
Yamada, Kosaku, 13
Yokoya, Yuji, 53–55

Z
Zara, 67–68
zero inspection, 6–7