

THE LEAN SERIES

ERIC RIES, SERIES EDITOR

Jez Humble, Joanne Molesky & Barry O'Reilly

LEAN ENTERPRISE

How High Performance
Organizations
Innovate at Scale

O'REILLY®

“...destined to be the classic, authoritative reference for how organizations plan, organize, implement, and measure their work.... Any business leader who cares about creating competitive advantage through technology and building a culture of innovation needs to read this book.”

—Gene Kim, co-author of *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*, founder and former CTO of Tripwire, Inc.

“*Lean Enterprise* provides a pragmatic toolkit of strategies and practices for establishing high performing organizations. It should be required reading for every executive who understands that we’re all in the technology business now.”

—Stephen Foreshee-Cain, COO, UK Government Digital Service

“To thrive in the digital world, transformation must be more than technology-driven—everyone within the organization must collectively work together to adapt. This book provides an essential guide for all leaders to change the way they deliver value to customers.”

—Matt Pancino, CEO, Suncorp Business Services

“The approach in this book is both challenging and disciplined, and some organizations will be unable to imagine following this path. But those who make the journey will find it impossible to imagine ever going back—and if they happen to be a competitor, they are well positioned to steal both your market and your people. Ignore this book at your own risk.”

—Mary Poppendieck, co-author of *The Lean Mindset* and the *Lean Software Development* series

US \$24.99

CAN \$26.99

ISBN: 978-1-449-36842-5



9 781449 368425



Twitter: @oreillymedia
facebook.com/oreilly
oreilly.com

Praise for *Lean Enterprise*

“This book is *Reengineering the Corporation* for the digital age. It is destined to be the classic, authoritative reference for how organizations plan, organize, implement, and measure their work. *Lean Enterprise* describes how organizations can win in the marketplace while harnessing and developing the capabilities of employees. Any business leader who cares about creating competitive advantage through technology and building a culture of innovation needs to read this book.”

— Gene Kim, co-author of *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*, founder and former CTO of Tripwire, Inc.

“This book is a godsend for anyone who’s tried to change their organization and heard: ‘It’s OK for the little guy, but we’re too big/regulated/complex to work like that here.’ Humble, Molesky, and O’Reilly have written an easy-to-read guide that demystifies the success of Lean organizations in a way that everyone can understand and apply. *Lean Enterprise* provides a pragmatic toolkit of strategies and practices for establishing high performing organizations. It should be required reading for every executive who understands that we’re all in the technology business now.”

— Stephen Foreshew-Cain, COO, UK Government Digital Service

“To thrive in the digital world, transformation must be more than technology driven—everyone within the organization must collectively work together to adapt. This book provides an essential guide for all leaders to change the way they deliver value to customers.”

— Matt Pancino, CEO, Suncorp Business Services

“This is the book I’ve been waiting for—one that takes on the hardest questions in bringing Lean approaches to the enterprise. The authors provide solutions that are valuable even in low trust environments.”

— Mark A. Schwartz (@schwartz_cio)

“This book integrates into a compelling narrative the best current thinking about how to create great software-intensive products and services. The approach in this book is both challenging and disciplined, and some organizations will be unable to imagine following this path. But those who make the journey will find it impossible to imagine ever going back—and if they happen to be a competitor, they are well positioned to steal both your market and your people. Ignore this book at your own risk.”

— Mary Poppendieck, co-author of *The Lean Mindset* and the Lean Software Development series

“My job is to support people in practicing a scientific pattern, to help reshape thinking and working habits in business, politics, education, and daily life. The 21st century is increasingly demanding a way of working that’s cognitively complex, interpersonal, iterative, and even entrepreneurial. With *Lean Enterprise*, Jez Humble, Joanne Molesky, and Barry O'Reilly explain how software can and is leading the way to transforming our ways of working, which can change our ways of thinking and help us adapt to the emerging world around us.”

— Mike Rother, author of *Toyota Kata*

“Nearly all industries and institutions are being disrupted through the rapid advance of technology, guided by the inspired vision of individuals and teams. This book clearly explains how the disciplines of Lean, Agile, Kata, Lean Startup, and Design Thinking are converging through the unifying principles of an adaptive learning organization.”

— Steve Bell, Lean Enterprise Institute faculty, author of *Lean IT* and *Run Grow Transform*

“Building software the right way is a challenging task in and of itself, but *Lean Enterprise* goes beyond the technology considerations to guide organizations on how to quickly build the right software to deliver expected business results in a low risk fashion. This is a must read for any organization that provides software based services to its customers.”

— Gary Gruver, VP of Release, QE, and Operations for Macys.com

“To compete in the future businesses need to be skilled at understanding their customers and taking the validated learnings to market as quickly as possible.

This requires a new kind of adaptive and learning organization—the lean enterprise. The journey starts here in this book!”

— John Crosby, Chief Product and Technology Officer,
lastminute.com

“Rapid advancements in technology are creating unparalleled rates of disruption. The rules of the disruption game have changed, and many organizations wonder how to compete as new giants emerge with a different approach to serving their customers. This book provides an essential guide to those that have come to the realization that they have to change to regain an innovative competitive advantage but are unsure where to start.”

— Jora Gill, Chief Digital Officer, The Economist

“*Lean Enterprise* was the book I gave my leadership team to get everyone on the same page about how we can challenge the status quo, remove roadblocks, and out-innovate our competition. By leveraging the continual insights we get from co-creating with customers, our people, and data, we now have so many additional new ways to grow our business.”

— Don Meij, CEO, Domino’s Pizza Enterprises Ltd.

“While agile and lean methods have had a big impact on software delivery, their true potential only comes as they have a broader impact on enterprises of all sizes. In this book, Jez, Joanne, and Barry have set out what those changes look like—a realistic vision of how future companies will make today’s look like cassette tape players.”

— Martin Fowler, Chief Scientist, ThoughtWorks

“This is an important book. It takes an informed and informative look at the fundamentals that need to shift to start building organizations capable of continuous learning and improvement. It moves well beyond the technical to the organizational. *Lean Enterprise* is a must-read for existing and emerging leaders seeking to ensure their company’s ongoing success.”

— Jeff Gothelf, author of *Lean UX*,
and Principal of Neo Innovation

“I was telling everyone to get this book for a year before it was finished. It documents the path being taken by the leading lean enterprises and the fat ones will be wiped out by the lean ones in the years to come.”

— Adrian Cockcroft (@adrianco)

Lean Enterprise

Jez Humble, Joanne Molesky, and Barry O'Reilly

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo

O'REILLY®

Lean Enterprise

by Jez Humble, Joanne Molesky, and Barry O'Reilly

Copyright © 2015 Jez Humble, Joanne Molesky, and Barry O'Reilly. All rights reserved.

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (<http://safaribooksonline.com>). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editors: Mary Treseler and Angela

Rufino

Production Editor: Kara Ebrahim

Copyeditor: Dmitry Kirsanov

Proofreader: Alina Kirsanova

Indexers: Dmitry Kirsanov and Alina

Kirsanova

Interior Designer: David Futato

Cover Designer: Ellie Volckhausen

Illustrators: Rebecca Demarest and Peter

Staples

Revision History for the First Edition

2014-12-01: First Release

See <http://oreilly.com/catalog/errata.csp?isbn=9781449368425> for release details.

The O'Reilly logo is a registered trademark of O'Reilly Media, Inc. *Lean Enterprise*, the cover image, and related trade dress are trademarks of O'Reilly Media, Inc.

While the publisher and the authors have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the authors disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-449-36842-5

[CW]

This book is dedicated to all of you who have (to paraphrase Admiral Grace Hopper) asked for forgiveness, not permission, in the pursuit of perfection, and to all the leaders committed to creating organizations where everybody knows what the right thing is, and you don't need anyone's permission to do it.

Contents

Preface.....	XIII
--------------	------

PART I: ORIENT

<i>Chapter 1</i>	
Introduction	5
<i>Chapter 2</i>	
Manage the Dynamics of the Enterprise Portfolio	21

PART II: EXPLORE

<i>Chapter 3</i>	
Model and Measure Investment Risk.....	45
<i>Chapter 4</i>	
Explore Uncertainty to Detect Opportunities	63
<i>Chapter 5</i>	
Evaluate the Product/Market Fit	87

PART III: EXPLOIT

<i>Chapter 6</i>	
Deploy Continuous Improvement	111
<i>Chapter 7</i>	
Identify Value and Increase Flow	133
<i>Chapter 8</i>	
Adopt Lean Engineering Practices	155
<i>Chapter 9</i>	
Take an Experimental Approach to Product Development	171
<i>Chapter 10</i>	
Implement Mission Command	189

PART IV: TRANSFORM

<i>Chapter 11</i>	
Grow an Innovation Culture	209
<i>Chapter 12</i>	
Embrace Lean Thinking for Governance, Risk, and Compliance	231
<i>Chapter 13</i>	
Evolve Financial Management to Drive Product Innovation	245
<i>Chapter 14</i>	
Turn IT into a Competitive Advantage	265
<i>Chapter 15</i>	
Start Where You Are	283

Bibliography	297
Index	303

Preface

Software is eating the world.

— Marc Andreessen

In an industrial company, avoid software at your own peril . . . a software company could disintermediate GE someday, and we're better off being paranoid about that.

— Jeff Immelt

You are a fool if you do just as I say. You are a greater fool if you don't do as I say. You should think for yourself and come up with better ideas than mine.

— Taiichi Ohno, Workplace Management

In this book we show how to grow organizations which can innovate rapidly in response to changing market conditions, customer needs, and emerging technologies.

Companies live and die on their ability to discover new businesses and create ongoing value for customers. This has always been true, but never more so than in the past few years. Competitive pressure is increasing, fueled by rapid changes in technology and society. As Deloitte's Shift Index shows, the average life expectancy of a Fortune 500 company has declined from around 75 years half a century ago to less than 15 years today. Professor Richard Foster of Yale University estimates that "by 2020, more than three-quarters of the S&P 500 will be companies that we have not heard of yet."¹ The long-term survival of

¹ <http://www.bbc.co.uk/news/business-16611040>

any enterprise depends on its ability to understand and harness the cultural and technical forces that continue to accelerate innovation cycles.

First, the Internet and social media have provided consumers with powerful tools to inform the decisions they make. These tools also give smart organizations new ways to discover and engage with users and customers. Enterprises that use design thinking and user experience (UX) design strategically to delight customers at each step of their interaction with the organization have thrived: research shows companies which apply UX design in this way experience faster growth and higher revenues.²

Second, advances in technology and process have made it possible to build, evolve, and scale disruptive products and services rapidly and with little capital investment. Small teams across the world prototype new software-based products in days or weeks, using free or cheap services and infrastructure, and then rapidly evolve those that gain traction. In the near future, the ubiquity of cheap, powerful networked embedded devices will enable us to prototype and evolve a wider variety of products cheaply on similarly short cycles. As 3D printing becomes cheaper and faster and begins to handle a wider variety of materials, we will create and deliver an enormous variety of customized products on demand.

Software has three characteristics which enable this kind of rapid innovation. First, it's relatively inexpensive to prototype and evolve ideas in software. Second, we can actually use such prototypes from an early stage in their evolution. Finally, in the course of creating these prototypes, we can discover a great deal about what customers find valuable and incorporate it back into our design—accelerating the rate at which we can test new ideas with users, collect feedback, and use it to improve our products and businesses.

Meanwhile, the relentless march of miniaturization (embodied in Moore's Law)³ has enabled incredibly powerful computers to become tiny and find their way into everything, with software at center stage. In a Forbes article titled "Now Every Company Is A Software Company," David Zanca, senior vice president for information technology at FedEx, describes himself as running "a software company inside of FedEx." Venkatesh Prasad, senior technical leader at Ford, describes his company as a maker of "sophisticated computers-on-wheels." Ben Wood of CCS Insight notes that Nokia "went through this incredible decade of innovation in hardware, but what Apple saw was that all you needed was a rectangle with a screen, and the rest was all about the

² *Evaluation of the Importance of Design*, Danish Design Center, 2006.

³ In 1965 Gordon Moore, co-founder of Intel, predicted that the density of integrated circuits would double approximately every two years.

software.”⁴ As a result of this shift in thinking about software, companies, including IT outsourcing pioneers GE and GM, are taking software development back in-house. As we discuss in Chapter 15, the UK government has followed suit. As reported by *The Economist*:⁵

GM’s reasons for doing this may well apply to many other firms too. “IT has become more pervasive in our business and we now consider it a big source of competitive advantage,” says Randy Mott, GM’s Chief Information Officer, who has been responsible for the reversal of the outsourcing strategy. While the work was being done by outsiders, he said most of the resources that GM was devoting to IT were spent on keeping things going as they were rather than on thinking up new ways of doing them. The company reckons that having its IT work done mostly in-house and nearby will give it more flexibility and speed and encourage more innovation.

The business world is moving from treating IT as a utility that improves internal operations to using rapid software- and technology-powered innovation cycles as a competitive advantage. This has far-reaching consequences. The traditional program and project management models we have used for IT are unsuited to rapid innovation cycles. However, they are deeply embedded in the way we manage everything from operations and customer service to budgeting, governance, and strategy. The elements of a suitable product-centric paradigm that works at scale have all emerged in the last 10 years, but they have not yet been connected and presented in a systematic way. This book aims to fill this gap, providing inspiration from organizations that have successfully adopted these ideas. More importantly, we have made a detailed inquiry into the culture of high performance, which is the critical factor enabling rapid innovation at scale.

Why Did We Write This Book?

All of the authors are experienced working in both enterprises and startups, and we have set out to present a pragmatic and systematic approach to innovation and transformation that works effectively in an enterprise context. We have addressed not just how high-performing organizations develop products, but how companies that are working towards higher performance can adopt these techniques in an incremental, iterative, low-risk way.

⁴ <http://www.bbc.co.uk/news/technology-23947212>; in our opinion, this is the key insight behind Microsoft’s acquisition of Nokia.

⁵ *The Economist Special Report: Outsourcing and Offshoring*, 406, no. 8819, 19 January 2013.

We wrote the book because of our frustration at the state of the industry. The techniques and practices we describe are not new, and they are known to work. However, they are not yet mainstream, and are often implemented piece-meal, leading to local, rather than systemic, improvements. As a result, companies toil at building—at huge cost—products, services, and businesses that do not deliver the expected value to customers.

When *Continuous Delivery* (Addison-Wesley) and *The Lean Startup* (Crown Business) were published, we saw an enormous amount of demand from people working in enterprises who wanted to adopt the practices described in these books. A large number of companies have achieved measurable benefit from using the practices we discuss, resulting in delivery of higher-quality products to market faster, increased customer satisfaction, and higher returns on investment. This comes with reduced cost and risk as well as happier employees who are no longer working unsustainable hours and have the opportunity to harness their creativity and passion at work.

However, everyone finds it difficult to implement these ideas successfully. In most cases it was impossible to realize anything more than incremental improvements because only part of the organization changed—and that part still needed to work with the rest of the organization, which expected them to behave in the traditional way. Thus we describe how successful companies have rethought everything from financial management and governance, to risk and compliance, to systems architecture, to program, portfolio, and requirements management in the pursuit of radically improved performance.

This book presents a set of patterns and principles designed to help you implement these ideas. We believe that every organization is different and will have different needs, so we don't provide rules on how to implement particular practices. Instead, we describe a heuristic approach to implementation that emphasizes the importance of experimentation in order to learn how your organization can best adopt these ideas and improve. This approach takes longer, but it has the advantages of showing measurable benefits faster and reducing the risk of change. It also enables your organization and people to learn for themselves what works best.

We hope you will find value in this book. The most dangerous attitude would be: “These are good ideas, *but they cannot work in our organization.*” As Taiichi Ohno, the father of the Toyota Production System, said:⁶

Whether top management, middle management, or the workers who actually do the work, we are all human, so we're like walking

⁶ [ohno12]

misconceptions, believing that the way we do things now is the best way. Or perhaps you do not think it is the best way, but you are working within the common sense that “We can’t help it, this is how things are.”

You will face obstacles adopting the ideas in this book. When you read the case studies, you will likely see reasons why the described approach may not work in your organization. Do not turn obstacles into objections. Treat what you read here as an inspiration for your own efforts, not as recipes to be followed without deviation. Look for obstacles constantly and treat them as opportunities to experiment and learn. To quote Ohno again:⁷

Kaizen [improvement] opportunities are infinite. Don’t think you have made things better than before and be at ease...This would be like the student who becomes proud because they bested their master two times out of three in fencing. Once you pick up the sprouts of kaizen ideas, it is important to have the attitude in our daily work that just underneath one kaizen idea is yet another one.

Opportunities to improve lie everywhere—not just in the products or services we build but in the way we behave and interact and, most importantly, in the way we think.

Who Should Read This Book?

We wrote this book primarily for leaders and managers. The book focuses on principles and patterns that can be applied in any domain in any type of organization.

Our intended audience includes:

- Executives interested in strategy, leadership, organization culture, and good governance
- Directors of IT, both for applications and for infrastructure and operations
- Anyone working in program or project management, including members of the PMO
- People in finance and accounting or in governance, regulation, and compliance who are involved in delivery

⁷ [ohno12]

- CMOs, product managers, and others involved in designing products and services that involve software development

Anyone working on delivery teams should also find this book valuable—but don’t expect any deep discussion of engineering practices, such as how to write maintainable functional acceptance tests, automate deployment, or manage configuration. Those topics are discussed in much more depth in *Continuous Delivery*.

This book is particularly targeted at people working in medium and large organizations who realize they must think differently about strategy, culture, governance, and the way they manage products and services in order to succeed. That’s not to say that smaller organizations won’t find the book useful—just that some of the material may not be applicable to them at this stage in their evolution.

One of our goals was to keep the book relatively short, concise, and practical. In order to do that, we decided not to spend a lot of time discussing the theoretical models that drive the principles and practices we describe. Instead, we have presented some foundational principles from these fields so you can understand the basic theoretical underpinnings; then we describe the practical applications of these theories. We also provide references to further reading for those who are interested.

We are also careful not to offer detailed guidance on which software tools to use and how to use them. This is for two reasons. First, we think that tool choice is actually not a tremendously important decision (so long as you avoid the bad ones). Many organizations moving to agile methodologies spend an undue amount of time on tool choice hoping to magically solve their underlying problems. But the most common failure mode for such organizations is their inability to change their organizational culture, not the availability of good tools. Secondly, information on particular tools and processes quickly goes out of date. There are plenty of good tools (including many open source ones) and literature on how to use them. In this book we focus on strategies to help your organization succeed, regardless of the tools you choose.

Conspectus

Part I of the book introduces the main themes of the book: culture, strategy, and the lifecycle of innovations. In **Part II** we discuss how to explore new ideas to gather data so you can quickly evaluate which ones will provide value or see a sufficiently rapid uptake. **Part III** covers how to exploit validated ideas—those that emerge from the crucible of exploration—at scale, and also presents a systematic approach to improving the way we run large programs of work. Finally, **Part IV** shows how enterprises can grow an environment that fosters

learning and experimentation, with a focus on culture, governance, financial management, IT, and strategy.

Everybody should read **Part I**. Readers should then feel free to dip into the chapters that interest them. However it's worth reading **Chapter 3**, **Chapter 6**, and **Chapter 7** before proceeding to **Part IV** since it builds on concepts presented in those chapters.

Safari® Books Online

NOTE

Safari Books Online is an on-demand digital library that delivers expert content in both book and video form from the world's leading authors in technology and business.

Technology professionals, software developers, web designers, and business and creative professionals use Safari Books Online as their primary resource for research, problem solving, learning, and certification training.

Safari Books Online offers a range of product mixes and pricing programs for organizations, government agencies, and individuals. Subscribers have access to thousands of books, training videos, and prepublication manuscripts in one fully searchable database from publishers like O'Reilly Media, Prentice Hall Professional, Addison-Wesley Professional, Microsoft Press, Sams, Que, Peachpit Press, Focal Press, Cisco Press, John Wiley & Sons, Syngress, Morgan Kaufmann, IBM Redbooks, Packt, Adobe Press, FT Press, Apress, Manning, New Riders, McGraw-Hill, Jones & Bartlett, Course Technology, and dozens more. For more information about Safari Books Online, please visit us online.

How to Contact Us

Please address comments and questions concerning this book to the publisher:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

We have a web page for this book, where we list errata, examples, and any additional information: <http://bit.ly/lean-enterprise-book>.

To comment or ask technical questions about this book, send email to *book-questions@oreilly.com*.

For more information about our books, courses, conferences, and news, see our website at <http://www.oreilly.com>.

Find us on Facebook: <http://facebook.com/oreilly>

Follow us on Twitter: <http://twitter.com/oreillymedia>

Watch us on YouTube: <http://www.youtube.com/oreillymedia>

Acknowledgments

Many people have contributed to this book. In particular, we are deeply grateful to the following people who provided detailed reviews of early drafts or individual chapters (alphabetically by first name): Adrian Cockcroft, Amy McLeod, Andy Pittaway, Bas Vodde, Ben Williams, Bjarte Bogsnes, Brett Ansley, Carmen Cook, Charles Betz, Chris Cheshire, Courtney Hemphill, Dan North, Darius Kumana, David Tuck, Don Reinertsen, Gary Gruver, Gene Kim, Ian Carroll, James Cook, Jean-Marc Domaingue, Jeff Gothelf, Jeff Patton, Jim Highsmith, Joe Zenevitch, John Allspaw, John Crosby, Jonathan Thoms, Josh Seiden, Kevin Behr, Kief Morris, Kraig Parkinson, Lane Halley, Lee Nicholls, Lindsay Ratcliffe, Luke Barrett, Marc Hofer, Marcin Floryan, Martin Fowler, Matt Pancino, Michael Orzen, Mike Rother, Pat Kua, Randy Shoup, Ranjan Sakalley, Salim Virani, Steve Bell, Tom Barker, Tristan Kromer, and Will Edel-muth. Thank you so much. The ideas we present came from a wide variety of sources, and were winnowed and refined through innumerable workshops, talks, and discussions with people working in an enormous variety of organizations across the world. Thanks to all of you who participated in those discussions and gave us the benefit of your experiences and feedback. We'd like to extend special thanks to our fabulous editorial and production team at O'Reilly: Mary Treseler, Angela Rufino, Allyson MacDonald, Kara Ebrahim, and Dan Fauxsmith. Special thanks are also due to Peter Staples for creating almost all of the gorgeous diagrams in the book. Steve Bell, John Kordyback, Scott Buckley, and Gareth Rushgrove provided case studies for this book: thanks so much for your contributions and insight. Finally, Dmitry Kirsanov and Alina Kirsanova did characteristically thorough, detailed, and high-quality work copyediting, proofreading, and indexing the book—thank you.

Jez started working on this book as an excuse to stay home after his second daughter, Reshma, was born. Reshma and her sister, Amrita, have taught him the joy of disruption throughout by playing pranks and co-creating many new adventures that provoked both new insights and helpless laughter. Rani, his beautiful, brilliant wife, kept it real throughout even when it felt relentless, for which she has his undying gratitude, love, and admiration. He thanks his mum for her encouragement and support, particularly when he had to write during visits. Jez would like to thank his co-authors Joanne and Barry for moderating his command-and-control tendencies and making this book a truly

collaborative exercise. It would have been a very different—and much poorer—book without you. He would like to thank his colleagues at Chef for providing inspiration and support, and for living the dream of stirring up delight in the pursuit of a world-class product and customer experience. He also wants to thank his previous employer, ThoughtWorks, for providing a unique, mindful home for innovators and tinkerers, many of whose ideas populate these pages. Finally, special thanks to Chris Murphy, Chad Wathington, David Rice, Cyndi Mitchell, Barry Crist, and Adam Jacob for their support of this book.

Joanne really didn't understand what she had agreed to when Jez Humble and Martin Fowler convinced her to collaborate on a book about the next steps for Continuous Delivery. As time progressed (over two and a half years) and the book evolved into what it is today, there are a lot of people who provided support, encouragement, and complete trust in her capabilities to finish this work. John, Joanne's husband, lifetime partner, and best friend, provided encouragement and unending understanding during those guilt-ridden weekends and evenings when “the book” distracted her from fun activities. Her colleagues and the leadership team at ThoughtWorks provided all that she needed to research and write this work, in particular David Whalley, Chris Murphy, and the ThoughtWorks Australia leadership team who hired her—because they understood how important it is for something as command-and-control as security, risk, and compliance to fit with agile and lean delivery practices. Last, but not least, she would like to acknowledge her co-authors and good friends Barry and Jez, who taught her about perseverance, collaboration, and true trust in each other.

Barry could not have written this book without Qiu Yi, his life editor, partner, and wife. Her passion, persistence, and patience smooths his edges. Her compassion knows no end. His parents, Niall and Joan, have always believed in him, providing support and making personal sacrifices to enable him to reach for his goals. He could not ask for better role models; their principles and values have shaped his own, and for that he is grateful. He misses his brothers and sisters. The time they spend together is always precious and too short. His entire family is close to his heart and never far from his thoughts. He has been inspired by many friends, colleagues, and storytellers in his life and career; their conversations, lessons, and knowledge is captured here. Thank you for exposing him to it. When he wrote his first blog and pressed publish, he never imagined the outcome would lead him here. The encouragement, collaboration, and calibration of Jez and Joanne have taught him much more than how to craft ideas into words—he's grown with their guidance.

PART I

ORIENT

The purpose of an organization is to enable ordinary human beings to do extraordinary things.

— Peter Drucker

Shareholder value is the dumbest idea in the world...[it is] a result, not a strategy...Your main constituencies are your employees, your customers, and your products.¹

— Jack Welch

We begin by offering our definition of an enterprise: “a complex, adaptive system composed of people who share a common purpose.” We thus include non-profits and public sector companies as well as corporations. We will go into more detail on complex, adaptive systems in [Chapter 1](#). However, the idea of a common purpose known to all employees is essential to the success of an enterprise. A company’s purpose is different from its vision statement (which describes what an organization aspires to become) and its mission (which describes the business the organization is in). Graham Kenny, managing director of consultancy *Strategic Factors*, describes the purpose of an organization as what it does *for someone else*, “putting managers and employees *in*

¹ <http://on.ft.com/1zmWBMD>

*customers' shoes.*² He cites as examples the Kellogg food company ("Nourishing families so they can flourish and thrive") and the insurance company IAG ("To help people manage risk and recover from the hardship of unexpected loss"), to which we add our favorite example: SpaceX, "founded in 2002 by Elon Musk to revolutionize space transportation and ultimately make it possible for people to live on other planets."³

Creating, updating, and communicating the company's purpose is the responsibility of the enterprise's executives. Their other responsibilities include creating a strategy through which the company will achieve its purpose and growing the culture necessary for that strategy to succeed. Both strategy and culture will evolve in response to changes in the environment, and leaders are responsible for directing this evolution and for ensuring that culture and strategy support each other to achieve the purpose. If leaders do a good job, the organization will be able to adapt, to discover and meet the changing customer needs, and to remain resilient to unexpected events. This is the essence of good governance.

In the context of corporations, the idea of a common purpose other than profit maximization may seem quaint. For many years, the conventional wisdom held that corporate executives should focus on maximizing shareholder value, and this goal was reinforced by compensating executives with stocks.⁴ However, these strategies have a number of flaws. They create a bias towards short-term results (such as quarterly earnings) at the expense of longer-term priorities such as developing the capabilities of employees and the relationships with customers. They also tend to stifle innovation by focusing on tactical actions to reduce costs in the short term at the expense of riskier strategies that have the potential to provide a higher payoff over the lifetime of the organization, such as research and development or creating disruptive new products and services. Finally, they often ignore the value of intangibles, such as the capabilities of employees and intellectual property, and externalities such as the impact on the environment.

Research has shown that focusing only on maximizing profits has the paradoxical effect of *reducing* the rate of return on investment.⁵ Rather, organizations

2 <http://bit.ly/1zmWARB>

3 In the copious free time left over from SpaceX, Musk co-founded Tesla Motors along with "a group of intrepid Silicon Valley engineers who set out to prove that electric vehicles could be awesome."

4 This strategy originates from Jensen and Meckling's "Theory of the Firm" (*Journal of Financial Economics*, 3, no. 4, 1976).

5 John Kay's *Obliguity* (Penguin Books) provides detailed research and analysis supporting what he describes as the "profit-seeking paradox."

succeed in the long term through developing their capacity to innovate and adopting the strategy articulated by Jack Welch in the above epigraph: focusing on employees, customers, and products. **Part I** of this book sets out how to achieve this.

Introduction

It's possible for good people, in perversely designed systems, to casually perpetrate acts of great harm on strangers, sometimes without ever realizing it.

— Ben Goldacre

On April 1, 2010, California's only motor vehicle plant, New United Motor Manufacturing, Inc. (NUMMI), shut down. NUMMI, which opened in 1984, had been a joint venture between GM and Toyota. Both companies stood to benefit from the partnership. Toyota wanted to open a plant in the US to escape import restrictions threatened by the US Congress in reaction to the inexorably falling market share of US auto manufacturers. For GM, it was a chance to learn how to build small cars profitably and to study the Toyota Production System (TPS) that had enabled Japanese auto manufacturers to consistently deliver the highest quality in the industry at costs that undercut those of US manufacturers.¹

For the joint venture, GM chose the site of their shuttered Fremont Assembly plant. GM's Fremont plant was one of their worst in terms of both the quality of the cars produced and the relationship between managers and workers. By the time the plant closed in 1982, labor relations had almost completely broken down, with workers drinking and gambling on the job. Incredibly, Toyota agreed to the demand of United Auto Workers' negotiator Bruce Lee to rehire

¹ The story of the NUMMI plant is covered comprehensively in *This American Life*, episode 403: <http://www.thisamericanlife.org/radio-archives/episode/403/>, from which all the direct quotes are taken.

the union leaders from Fremont Assembly to lead the workforce at NUMMI. The workers were sent to Toyota City in Japan to learn the TPS. Within three months, the NUMMI plant was producing near-perfect quality cars—some of the best quality in America, as good as those coming from Japan—at much lower cost than Fremont Assembly had achieved. Lee had been right in his bet that “it was the system that made it bad, not the people.”

Much has been written about the TPS, but one recurring theme, when you listen to the Fremont Assembly workers who ended up at NUMMI, is teamwork. It might seem banal, but it was an incredibly powerful experience for many of the UAW employees. The TPS makes building quality into products the highest priority, so a problem must be fixed as soon as possible after it’s discovered, and the system must then be improved to try and prevent that from happening again. Workers and managers cooperate to make this possible. The moment a worker discovers a problem, he or she can summon the manager by pulling on a cord (the famous *andon* cord). The manager will then come and help to try and resolve the problem. If the problem cannot be resolved within the time available, the worker can stop the production line until the problem is fixed. The team will later experiment with, and implement, ideas to prevent the problem from occurring again.

These ideas—that the primary task of managers is to help workers, that workers should have the power to stop the line, and that they should be involved in deciding how to improve the system—were revolutionary to the UAW employees. John Shook, the first American to work in Toyota City, who had the job of training the NUMMI workers, reflects that “they had had such a powerful emotional experience of learning a new way of working, a way that people could actually work together collaboratively—as a team.”

The way the TPS works is in sharp contrast to the traditional US and European management practice based on the principles of Frederick Winslow Taylor, the creator of *scientific management*. According to Taylor, the job of management is to analyze the work and break it down into discrete tasks. These tasks are then performed by specialized workers who need understand nothing more than how to do their particular specialized task as efficiently as possible. Taylorism fundamentally thinks of organizations as machines which are to be analyzed and understood by breaking them down into component parts.

In contrast, the heart of the TPS is creating a high-trust culture in which everybody is aligned in their goal of building a high-quality product on demand and where workers and managers collaborate across functions to constantly improve—and sometimes radically redesign—the system. These ideas from the TPS—a high-trust culture focused on continuous improvement (*kaizen*), powered by alignment and autonomy at all levels—are essential to building a large organization that can adapt rapidly to changing conditions.

A key part of the success of the TPS is in its effect on workers. Taylorism makes workers into cogs in a machine, paid simply to perform preplanned actions as quickly as possible. The TPS, instead, requires workers to pursue mastery through continuous improvement, imbues them with a higher purpose—the pursuit of ever-higher levels of quality, value, and customer service—and provides a level of autonomy by empowering them to experiment with improvement ideas and to implement those that are successful.

Decades of research have shown that these *intrinsic motivators* produce the highest performance in tasks which require creativity and trial-and-error—where the desired outcome cannot be achieved simply by following a rule.² In fact, extrinsic motivators such as bonuses and rating people in performance reviews actually *decrease* performance in such nonroutine work.³ Rick Madrid, who worked at the Fremont plant both before and during the NUMMI era, says of the TPS that “it changed my life from being depressed, bored—and like my son said, it changed my attitude. It changed me all for the better.” Giving people pride in their work rather than trying to motivate them with carrots and sticks is an essential element of a high-performance culture.⁴

Although the principles at the heart of the TPS might seem relatively straightforward, they were very hard to adopt. Indeed, GM utterly failed in taking what it had achieved at NUMMI and reproducing it in other GM plants. Some of the biggest obstacles were changes to the organizational hierarchy. The TPS does away with the concept of seniority in which union workers are assigned jobs based on how many years of service they have, with the best jobs going to the most senior. Under the TPS, everybody has to learn all the jobs required of their team and rotate through them. The TPS also removes the visible trappings and privileges of management. Nobody wore a tie at the NUMMI plant—not even contractors—to emphasize the fact that everybody was part of the same team. Managers did not receive perks accorded to them at other GM plants, such as a separate cafeteria and car park.

Finally, attempts to improve quality ran up against organizational boundaries. In the TPS, suppliers, engineers, and workers collaborate to continuously

2 Behavioral scientists often classify work into two types: routine tasks where there is a single correct result that can be achieved by following a rule are known as *algorithmic*, and those that require creativity and trial-and-error are called *heuristic*.

3 Decades of studies have repeatedly demonstrated these results. For an excellent summary, see [pink].

4 Indeed one of W. Edwards Deming’s “Fourteen Points For The Transformation Of Management” is “Remove barriers that rob people in management and in engineering of their right to pride of workmanship. This means, *inter alia*, abolishment of the annual or merit rating and of management by objective” [deming], p. 24.

improve the quality of the parts and to make sure workers have the tools they need to do their job. This worked at NUMMI because the engineers were in-house and the parts came from Japanese suppliers that had a collaborative relationship with Toyota. In the US supply chain, things were different. If the parts that came in to GM assembly plants were of poor quality, or didn't fit, there was simply no mechanism to fix the problem.

Ernie Schaefer, manager of GM's Van Nuys plant—which faced many of the same problems as Fremont Assembly—describes what was different about NUMMI: “You can see a lot of things different. But the one thing you don't see is the system that supports the NUMMI plant. I don't think, at that time, anybody understood the large nature of this system. General Motors was a kind of throw it over the wall organization. Each department, we were very compartmentalized, and you design that vehicle, and you'd throw it over the wall to the manufacturing guys.” This is the legacy of a Taylorist management approach. The TPS exists—and can only succeed—within an ecosystem of organizational culture, supplier relations, financial management, HR, and governance designed around its philosophy.

GM tried to implement the TPS at Van Nuys, but failed. Workers and managers rebelled in the face of changes in status and behavior that were required of them, despite the threat of closure (which was ultimately carried out). According to Larry Spiegel, a veteran of NUMMI who had been sent to Van Nuys to help implement the TPS, people at the plant simply didn't believe the threats to shut it down: “There were too many people convinced that they didn't need to change.”

This lack of urgency acted as a barrier to adoption across GM—and is perhaps the biggest obstacle to organizational change in general.⁵ The US division of GM took about 15 years to decide they needed to seriously prioritize implementing the TPS, and a further 10 years to actually implement it. By this time any competitive advantage they could have gained was lost. GM went bankrupt and was bailed out by the US government in 2009, at which point it pulled out of NUMMI. Toyota shut down the NUMMI plant in 2010.

The story of NUMMI is important because it illustrates the main concern of this book—growing a lean enterprise, such as Toyota—and many of the common obstacles. Toyota has always been very open about what it is doing, giving public tours of its plants, even to competitors—partly because it knows that what makes the TPS work is not so much any particular practices but the

⁵ John Kotter, author of *Leading Change*, says, “a majority of employees, perhaps 75 percent of management overall and virtually all of the top executives, need to believe that considerable change is absolutely essential” [kotter], p. 51.

culture. Many people focus on the practices and tools popularized by the TPS, such as the *andon* cords. One GM vice president even ordered one of his managers to take pictures of every inch of the NUMMI plant so they could copy it precisely. The result was a factory with *andon* cords but with nobody pulling them because managers (following the principle of extrinsic motivation) were incentivized by the rate at which automobiles—of *any* quality—came off the line.

A Lean Enterprise Is Primarily a Human System

As the pace of social and technological change in the world accelerates, the lean approach pioneered by Toyota becomes ever more important because it sets out a proven strategy for thriving in uncertainty through embracing change. The key to understanding a lean enterprise is that it is primarily a *human* system. It is common for people to focus on specific practices and tools that lean and agile teams use, such as Kanban board, stand-up meetings, pair programming, and so forth. However, too often these are adopted as rituals or “best practices” but are not seen for what they really are—*countermeasures* that are effective within a particular context in the pursuit of a particular goal.

In an organization with a culture of continuous improvement, these countermeasures emerge naturally within teams and are then discarded when they are no longer valuable. The key to creating a lean enterprise is to enable those doing the work to solve their customers’ problems in a way that is aligned with the strategy of the wider organization. To achieve this, we rely on people being able to make local decisions that are sound at a strategic level—which, in turn, relies critically on the flow of information, including feedback loops.

Information flow has been studied extensively by sociologist Ron Westrum, primarily in the context of accidents and human errors in aviation and health-care. Westrum realized that safety in these contexts could be predicted by organizational culture, and developed a “continuum of safety cultures” with three categories:⁶

Pathological organizations are characterized by large amounts of fear and threat. People often hoard information or withhold it for political reasons, or distort it to make themselves look better.

Bureaucratic organizations protect departments. Those in the department want to maintain their “turf,” insist on their own rules, and generally do things by the book—*their* book.

⁶ [westrum-2014]

Generative organizations focus on the mission. How do we accomplish our goal? Everything is subordinated to good performance, to doing what we are supposed to do.

These cultures process information in different ways. Westrum observes that “the climate that provides good information flow is likely to support and encourage other kinds of cooperative and mission-enhancing behavior, such as problem solving, innovations, and interdepartmental bridging. When things go wrong, pathological climates encourage finding a scapegoat, bureaucratic organizations seek justice, and the generative organization tries to discover the basic problems with the system.” The characteristics of the various types of culture are shown in [Table 1-1](#).

Table 1-1. How organizations process information

Pathological (power-oriented)	Bureaucratic (rule-oriented)	Generative (performance-oriented)
Low cooperation	Modest cooperation	High cooperation
Messengers shot	Messengers neglected	Messengers trained
Responsibilities shirked	Narrow responsibilities	Risks are shared
Bridging discouraged	Bridging tolerated	Bridging encouraged
Failure leads to scapegoating	Failure leads to justice	Failure leads to enquiry
Novelty crushed	Novelty leads to problems	Novelty implemented

Westrum’s typology has been extensively elaborated upon, and has a visceral quality that will appeal to anybody who has worked in a pathological (or even bureaucratic) organization. However, some of its implications are far from academic.

In 2013, PuppetLabs, IT Revolution Press, and ThoughtWorks surveyed 9,200 technologists worldwide to find out what made high-performing organizations successful. The resulting *2014 State of DevOps Report* is based on analysis of answers from people working in a variety of industries including finance, telecoms, retail, government, technology, education, and healthcare.⁷ The headline result from the survey was that strong IT performance is a competitive advantage. Analysis showed that firms with high-performing IT organizations were

⁷ [forsgren]

twice as likely to exceed their profitability, market share, and productivity goals.⁸

The survey also set out to examine the cultural factors that influenced organizational performance. The most important of these turned out to be whether people were satisfied with their jobs, based on the extent to which they agreed with the following statements (which are strongly reminiscent of the reaction of the NUMMI workers who were introduced to the Toyota Production System):

- I would recommend this organization as a good place to work.
- I have the tools and resources to do my job well.
- I am satisfied with my job.
- My job makes good use of my skills and abilities.

The fact that job satisfaction was the top predictor of organizational performance demonstrates the importance of intrinsic motivation. The team working on the survey wanted to look at whether Westrum's model was a useful tool to predict organizational performance.⁹ Thus the survey asked people to assess their team culture along each of the axes of Westrum's model as shown in **Table 1-1**, by asking them to rate the extent to which they agreed with statements such as "On my team, failure causes enquiry."¹⁰ In this way, the survey was able to measure culture.

Statistical analysis of the results showed that team culture was not only strongly correlated with organizational performance, it was also a strong predictor of job satisfaction. The results are clear: a high-trust, generative culture is not only important for creating a safe working environment—it is the foundation of creating a high-performance organization.

Mission Command: An Alternative to Command and Control

High-trust organizational culture is often contrasted to what is popularly known as "command and control": the idea from scientific management that

⁸ The survey measured organizational performance by asking respondents to rate their organization's relative performance in terms of achieving its profitability, market share, and productivity goals. This is a standard scale that has been validated multiple times in prior research. See [widener].

⁹ In the interests of full disclosure, Jez was part of the team behind the *2014 State of DevOps Report*.

¹⁰ This method of measuring attitudes quantitatively is known as a Likert scale.

the people in charge make the plans and the people on the ground execute them—which is usually thought to be modelled on how the military functions. In reality, however, this type of command and control has not been fashionable in military circles since 1806 when the Prussian Army, a classic plan-driven organization, was decisively defeated by Napoleon’s decentralized, highly motivated forces. Napoleon used a style of war known as *maneuver warfare* to defeat larger, better-trained armies. In maneuver warfare, the goal is to minimize the need for actual fighting by disrupting your enemy’s ability to act cohesively through the use of shock and surprise. A key element in maneuver warfare is being able to learn, make decisions, and act faster than your enemy—the same capability that allows startups to disrupt enterprises.¹¹

Three men were especially important to the reconstruction of the Prussian Army following its defeat by Napoleon: Carl von Clausewitz, David Scharnhorst, and Helmuth von Moltke. Their contributions not only transformed the military doctrine; they have important implications for people leading and managing large organizations. This particularly applies to the idea of *Auftragstaktik*, or Mission Command, which we will explore here. Mission Command is what enables maneuver warfare to work at scale—it is key to understanding how enterprises can compete with startups.

Following the eventual defeat of Napoleon, General David Scharnhorst was made Chief of the newly established Prussian General Staff. He put together a reform commission which conducted a postmortem and began to transform the Prussian Army. Scharnhorst noted that Napoleon’s officers had the authority to make decisions as the situation on the ground changed, without waiting for approval through the chain of command. This allowed them to adapt rapidly to changing circumstances.

Scharnhorst wanted to develop a similar capability in a systematic way. He realized this required the training of an independent, intelligent cadre of staff officers who shared similar values and would be able to act decisively and autonomously in the heat of battle. Thus military schools were set up to train staff officers, who for the first time were accepted from all social backgrounds based on merit.

In 1857, Helmuth von Moltke, perhaps best known for his saying “no plan survives contact with the enemy,” was appointed Chief of the General Staff of the Prussian Army. His key innovation, building on the military culture established by Scharnhorst, was to treat military strategy as a series of options which were to be explored extensively by officers in advance of the battle. In

¹¹ As we discuss in [Chapter 3](#), this concept is formalized in John Boyd’s OODA (observe-orient-decide-act) loop, which in turn inspired Eric Ries’ build-measure-learn loop.

1869 he issued a directive titled “Guidance for Large Unit Commanders” which sets out how to lead a large organization under conditions of uncertainty.

In this document, von Moltke notes that “in war, circumstances change very rapidly, and it is rare indeed for directions which cover a long period of time in a lot of detail to be fully carried out.” He thus recommends “not commanding more than is strictly necessary, nor planning beyond the circumstances you can foresee.” Instead, he has this advice: “The higher the level of command, the shorter and more general the orders should be. The next level down should add whatever further specification it feels to be necessary, and the details of execution are left to verbal instructions or perhaps a word of command. This ensures that everyone retains freedom of movement and decision within the bounds of their authority...The rule to follow is that an order should contain all, but also only, what subordinates cannot determine for themselves to achieve a particular purpose.”

Crucially, orders always include a passage which describes their *intent*, communicating the *purpose* of the orders. This allows subordinates to make good decisions in the face of emerging opportunities or obstacles which prevent them from following the original orders exactly. Von Moltke notes that “there are numerous situations in which an officer must act on his own judgment. For an officer to wait for orders at times when none can be given would be quite absurd. But as a rule, it is when he acts in line with the will of his superior that he can most effectively play his part in the whole scheme of things.”

These ideas form the core of the doctrine of *Auftragstaktik*, or Mission Command, which, in combination with the creation of a professionally trained cadre of staff officers who understood how to apply the doctrine operationally, was adopted by multiple elite military units, including the US Marine Corps as well as (more recently) NATO.

The history of the Prussian Army’s development of *Auftragstaktik* is described in more detail in Stephen Bungay’s treatise on business strategy, *The Art of Action* (from which the above quotations from “Guidance for Large Unit Commanders” are taken).¹² Bungay develops a theory of directing strategy at scale which builds on the work of Scharnhorst, von Moltke, and another Prussian general, Carl von Clausewitz. As a 26-year old, Clausewitz had fought against Napoleon in the fateful battles of Jena and Auerstadt. He subsequently served on Scharnhorst’s reform commission and bequeathed us his unfinished *magnum opus*, *On War*. In this work he introduces the concept of the “fog of war”—the fundamental uncertainty we face as actors in a large and rapidly

12 [bungay]

changing environment, with necessarily incomplete knowledge of the state of the system as a whole. He also introduces the idea of *friction* which prevents reality from behaving in an ideal way. Friction exhibits itself in the form of incomplete information, unanticipated side effects, human factors such as mistakes and misunderstandings, and the accumulation of unexpected events.

Friction and Complex Adaptive Systems

Clausewitz' concept of friction is an excellent metaphor to understand the behavior of complex adaptive systems such as an enterprise (or indeed any human organization). The defining characteristic of a complex adaptive system is that its behavior at a global level cannot be understood through Taylor's reductionist approach of analyzing its component parts. Rather, many properties and behavior patterns of complex adaptive systems "emerge" from *interactions* between events and components at multiple levels within the system. In the case of open systems (such as enterprises), we also have to consider interactions with the environment, including the actions of customers and competitors, as well as wider social and technological changes.¹³ Friction is ultimately a consequence of the human condition—the fact that organizations are composed of people with independent wills and limited information. Thus friction cannot be overcome.

Bungay argues that friction creates three gaps. First, a *knowledge gap* arises when we engage in planning or acting due to the necessarily imperfect state of the information we have to hand, and our need to make assumptions and interpret that information. Second, an *alignment gap* is the result of people failing to do things as planned, perhaps due to conflicting priorities, misunderstandings, or simply someone forgetting or ignoring some element of the plan. Finally, there is an *effects gap* due to unpredictable changes in the environment, perhaps caused by other actors, or unexpected side effects producing outcomes that differ from those we anticipated. These gaps are shown in Figure 1-1.

¹³ For those interested in different types of systems and how to make sense of them, we recommend studying Dave Snowden's Cynefin framework: <http://www.youtube.com/watch?v=N7oz366X0-8>.

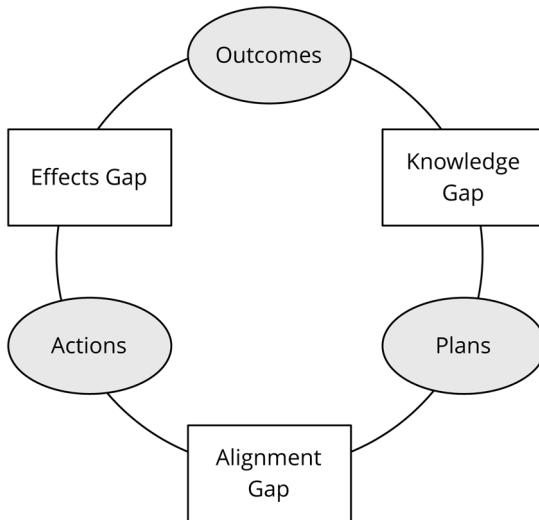


Figure 1-1. Gaps in complex adaptive systems, from *The Art of Action: How Leaders Close the Gaps between Plans, Actions, and Results* by Stephen Bungay (reprinted by permission of Nicholas Brealey Publishing)

Bungay then goes on to describe the usual scientific management remedy applied by enterprises, the alternative proposed by the doctrine of *Auftragstaktik*, and his own interpretation of Mission Command as applied to business, which he terms “directed opportunism.” These are shown in Table 1-2.

Table 1-2. The three gaps, and how to manage them

	Effects gap	Knowledge gap	Alignment gap
<i>What is it?</i>	The difference between what we expect our actions to achieve and what they actually achieve	The difference between what we would like to know and what we actually know	The difference between what we want people to do and what they actually do
<i>Scientific management remedy</i>	More detailed controls	More detailed information	More detailed instructions

	Effects gap	Knowledge gap	Alignment gap
<i>Auftragstaktik</i> remedy	"Everyone retains freedom of decision and action within bounds"	"Do not command more than is necessary or plan beyond the circumstances you can foresee"	"Communicate to every unit as much of the higher intent as is necessary to achieve the purpose"
<i>Directed opportunism</i> remedy	Give individuals freedom to adjust their actions in line with intent	Limit direction to defining and communicating the intent	Allow each level to define how they will achieve the intent of the next level up, and "backbrief"

It is crucial to understand that when we work in a complex adaptive system where friction dominates, the scientific management remedies *cannot work*. In fact, they make things worse. Creating ever more detailed plans delays the feedback that would tell us which of our assumptions are invalid. Complex sets of rules and controls punish the innocent but can be evaded by the guilty, all the while destroying morale, innovation, and entrepreneurialism. Intelligence gathering fails in the face of bureaucratic or pathological organizations which hide or distort information in order to protect their turf. Organizations unable to escape the grip of scientific management are perfect targets to be disrupted by organizations that understand how to move fast at scale.

Create Alignment at Scale Following the Principle of Mission

The most important concern leaders and managers operating within a complex adaptive system face is this: how can we enable people within the organization to make good decisions—to act in the best interests of the organization—given that they can *never* have sufficient information and context to understand the full consequences of their decisions, and given that events often overtake our plans?

In *The Principles of Product Development Flow*,¹⁴ Donald Reinertsen presents the *Principle of Mission*, based on the doctrine of Mission Command, in which we “specify the end state, its purpose, and the minimum possible constraints.” According to the Principle of Mission, we create alignment not by making a detailed plan of how we achieve our objective but by describing the *intent* of our mission and communicating *why* we are undertaking it.

The key to the Principle of Mission is to create alignment and enable autonomy by setting out clear, high-level target conditions with an agreed time

¹⁴ [reinertsen]

frame—which gets smaller under conditions of greater uncertainty—and then leaving the details of *how* to achieve the conditions to teams. This approach can even be applied to multiple levels of hierarchy, with each level reducing the scope while providing more context. In the course of the book, this principle is applied in multiple contexts:

Budgeting and financial management

Instead of a traditional budgeting process which requires all spending for the next year to be planned and locked down based on detailed projections and business plans, we set out high-level objectives across multiple perspectives such as people, organization, operations, market, and finance that are reviewed regularly. This kind of exercise can be used at multiple levels, with resources allocated dynamically when needed and the indicators reviewed on a regular basis.

Program management

Instead of creating detailed, upfront plans on the work to be done and then breaking that work down into tiny little bits distributed to individual teams, we specify at the program level only the measurable objectives for each iteration. The teams then work out how to achieve those objectives, including collaborating with other teams and continuously integrating and testing their work to ensure they will meet the program-level objectives.

Process improvement

Working to continuously improve processes is a key element of the TPS and a powerful tool to transform organizations. In [Chapter 6](#) we present the Improvement Kata in which we work in iterations, specifying target objectives for processes and providing the people who operate the processes the time and resources to run experiments they need to meet the target objectives for the next iteration.

Crucially, these mission-based processes must *replace* the command and control processes, not run alongside them. This requires people to behave and act in different ways and to learn new skills. It also requires a cultural change within the organization, as we discuss in [Chapter 11](#). Discussing how to apply Mission Command in business, Stephen Bungay reflects on a culture that enables Mission Command—which, not coincidentally, has the same characteristics that we find in the generative organizations described by Westrum in [Table 1-1](#):

The unchanging core is a holistic approach which affects recruiting, training, planning, and control processes, but also the culture and values of an organization. Mission Command embraces a conception of leadership which unsentimentally places human beings at its center. It crucially depends on factors which do not appear on the balance sheet

of an organization: the willingness of people to accept responsibility; the readiness of their superiors to back up their decisions; the tolerance of mistakes made in good faith. Designed for an external environment which is unpredictable and hostile, it builds on an internal environment which is predictable and supportive. At its heart is a network of trust binding people together up, down, and across a hierarchy. Achieving and maintaining that requires constant work.¹⁵

Your People Are Your Competitive Advantage

The story of the Fremont Assembly site doesn't stop with NUMMI. It is in fact the locus of two paradigm shifts in the US auto manufacturing industry. In 2010, the NUMMI plant was purchased by Tesla Motors and became the Tesla Factory. Tesla uses continuous methods to innovate faster than Toyota, discarding the concept of model years in favor of more frequent updates and in many cases enabling owners of older cars to download new firmware to gain access to new features. Tesla has also championed transparency of information, announcing it will not enforce its patents. In doing so, it echoes a story from Toyota's origins when it used to build automatic looms. Upon hearing that the plans for one of the looms had been stolen, Kiichiro Toyoda is said to have remarked:

Certainly the thieves may be able to follow the design plans and produce a loom. But we are modifying and improving our looms every day. So by the time the thieves have produced a loom from the plans they stole, we will have already advanced well beyond that point. And *because they do not have the expertise gained from the failures it took to produce the original*, they will waste a great deal more time than us as they move to improve their loom. We need not be concerned about what happened. We need only continue as always, making our improvements.¹⁶

The long-term value of an enterprise is not captured by the value of its products and intellectual property but rather by its ability to continuously increase the value it provides to customers—and to create *new* customers—through innovation.

A key premise of this book—supported by the experience of companies such as Tesla, among many, many others—is that the flexibility provided by software can, when correctly leveraged, accelerate the innovation cycle. Software can

15 [bungay], p. 88.

16 [rother-2010], p. 40, emphasis ours.

provide your enterprise with a competitive advantage by enabling you to search for new opportunities and execute validated opportunities faster than the competition. The good news is that these capabilities are within the reach of all enterprises, not just tech giants. The data from the *2014 State of Devops Report* shows that 20% of organizations with more than 10,000 employees fall into the high-performing group—a smaller percentage than smaller companies, but still significant.

Many people working in enterprises believe that there is some essential difference between them and tech giants such as Google, Amazon, or Netflix that are held up as examples of technology “done right.” We often hear, “that won’t work here.” That may be right, but people often look in the wrong places for the obstacles that prevent them from improving. Skeptics often treat size, regulation, perceived complexity, legacy technology, or some other special characteristic of the domain in which they operate as a barrier to change. The purpose of this chapter is to show that while these obstacles are indeed challenges, the most serious barrier is to be found in organizational culture, leadership, and strategy.

Many organizations try to take shortcuts to higher performance by starting innovation labs, acquiring startups, adopting methodologies, or reorganizing. But such efforts are neither necessary nor sufficient. They can only succeed if combined with efforts to create a generative culture and strategy across the whole organization, including suppliers—and if this is achieved, there will be no need to resort to such shortcuts.

The second chapter of this book describes the principles that enable organizations to succeed in the long term by balancing their portfolio of products. In particular, we distinguish two independent types of activity in the product development lifecycle: *exploring* new ideas to gather data and eliminate those that will not see rapid uptake by users, and *exploiting* those that we have validated against the market. **Part II** of the book discusses how to run the explore domain, with **Part III** covering the exploit domain. Finally, **Part IV** of the book shows how to transform your organization focusing on culture, financial management, governance, risk, and compliance.

Manage the Dynamics of the Enterprise Portfolio

The purpose of a business is to create a customer.

— Peter Drucker

In this chapter we will examine the lifecycle of businesses and how companies can balance the *exploration* of new business models with the *exploitation* of proven ones. We'll need this distinction in order to understand where lean startup practices and principles can be applied in an enterprise context and how they can be used as the basis of managing an innovation portfolio.

In his book *Diffusion of Innovations*, Everett Rogers describes the cycle through which all successful technologies and ideas progress, shown in [Figure 2-1](#).¹ Over time, all successful ideas, whether technologies, product categories, business models, or even methodologies, progress from being scarce and unevenly distributed to eventually becoming a commodity. They then form building blocks for new, higher-level, more valuable innovations. Of course the time it takes for innovations to progress through the various stages of the cycle can vary substantially.

¹ Rogers' work, detailed in [\[rogers\]](#), was in fact derived from research into the adoption of technology by farmers in Iowa.

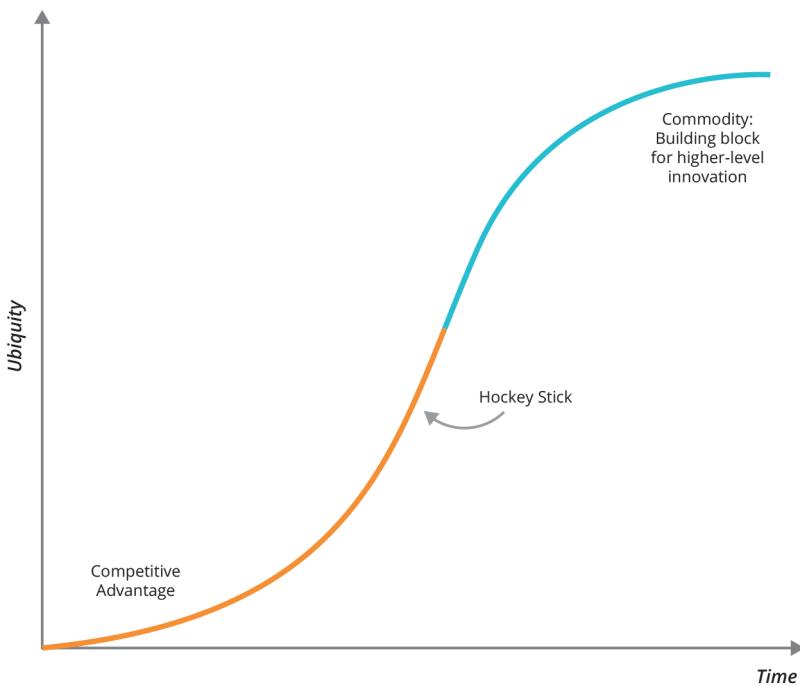


Figure 2-1. The S-curve which shows the lifecycle of innovations

Rogers believed people could be classified into groups based on how they respond to innovation, as shown in [Figure 2-2](#). Initially, new technologies and ideas are experimented with and tested by innovators, which form the smallest group of the overall population. As innovators discover technologies that provide competitive advantage (most will not), these technologies are taken up by the early adopters. In this way, success in each group leads to further diffusion through the other groups. Rogers' ideas were popularized and built upon by Geoffrey Moore, who introduced the concept of the “chasm,” a logical divide between uptake by early adopters and the early majority. This chasm was inspired by Moore's observation that many innovations flounder once they are no longer seen as a source of competitive advantage by visionaries, but are not yet sufficiently established to be seen as a safe bet or proven practice by people in the early majority.

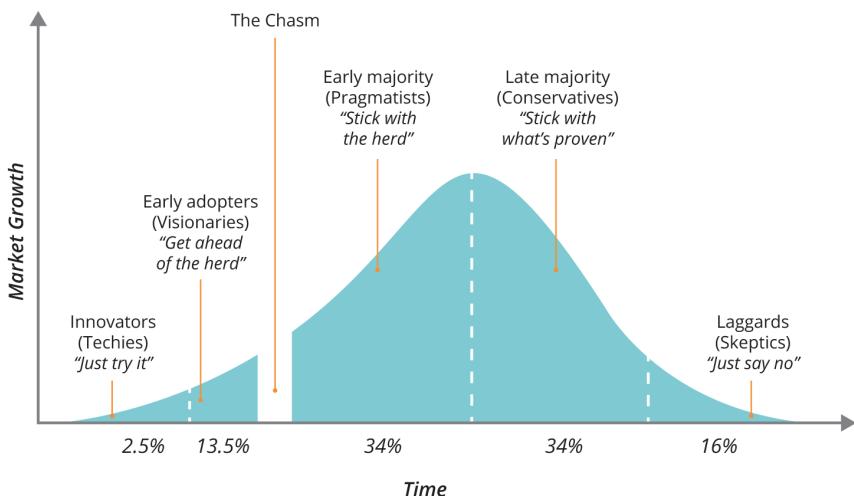


Figure 2-2. Technology adoption lifecycle, from *Dealing with Darwin* by Geoffrey A. Moore, 2006 (used by permission of Portfolio, an imprint of Penguin Group (USA) LLC)

Once the market has assimilated a disruptive new technology or idea, a whole range of product offerings gets spawned. Moore's take on the product category lifecycle is shown in Figure 2-3. A successful product category will initially see high growth (stage B), followed by a mature market (stage C) in which consolidation takes place. Growth in mature markets is typically driven by acquiring competitors and new customers as well as by efficiency gains. Finally, product categories decline (stage D). At any point, a category can be disrupted by some new innovation—indeed an innovation is defined as “disruptive” based on its effect on existing product categories and business models. Even in the face of disruption, it’s sometimes possible to maintain a lucrative niche market; for instance, feature phones are still an important category in many countries, and IBM still has a profitable mainframe business.

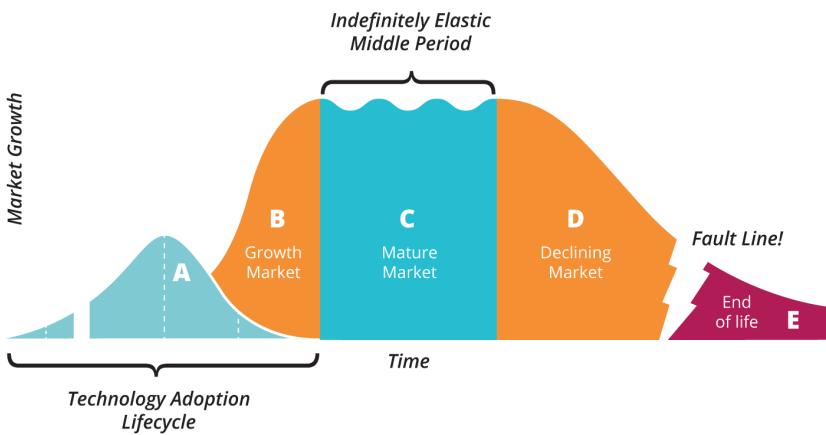


Figure 2-3. Category maturity lifecycle, from *Dealing with Darwin* by Geoffrey A. Moore, 2006 (used by permission of Portfolio, an imprint of Penguin Group (USA) LLC)

The first point to observe is that products at different stages of maturity vary significantly in terms of how they are managed, developed, marketed, and funded. For example, in a mature market we have a good understanding of our customers and the value they get from the product; acquiring new customer or selling new products to existing ones is well understood, and new products in the category typically contain only incremental innovations. For new categories, the opposite is true.

While there is a lot of detail involved in understanding these different stages of the lifecycle, such as whether our customers are other businesses or consumers, we can get to the important discussion by drastically simplifying the problem and looking at two key activities that all enterprises will engage in: *exploring* new product categories and business models, and *exploiting* the proven ones.² Steve Blank refers to these activities as “search” and “execute” in the context of customer development.³

Startups begin by exploring new opportunities through business model innovation: they search for a new business model that is aligned with the founders’ purpose and vision, delivers value for customers, and can drive the profitability and growth of the organization. Once it is found, the business model is *exploited* by growing and scaling it, finding ways to drive down costs, improve

² This distinction was first proposed by James March in his paper “Exploration and Exploitation in Organizational Learning” [march].

³ [blank]

efficiency, and increase market share and customer base. However, every business model is ultimately transient: eventually, every business model and product category will be disrupted by new ones—it is only a matter of time.

Exploring new opportunities and exploiting existing ones are fundamentally different strategies requiring different structure, competencies, processes, and mindset. It is hard to overemphasize this key point: management practices that are effective in the exploit domain will lead to failure if applied to exploring new opportunities—and vice versa. The differences between these two domains are listed in [Table 2-1](#).

Table 2-1. Explore versus exploit

	Explore	Exploit
<i>Strategy</i>	Radical or disruptive innovation, new business model innovation	Incremental innovation, optimizing existing business model
<i>Structure</i>	Small cross-functional multiskilled team	Multiple teams aligned using Principle of Mission
<i>Culture</i>	High tolerance for experimentation, risk taking, acceptance of failure, focus on learning	Incremental improvement and optimization, focus on quality and customer satisfaction
<i>Risk management</i>	Biggest risk is failure to achieve product/market fit	A more complex set of trade-offs specific to each product/service
<i>Goals</i>	Creating new markets, discovering new opportunities within existing markets	Maximizing yield from captured market, outperforming competitors
<i>Measure of progress</i>	Achieving product/market fit	Outperforming forecasts, achieving planned milestones and targets

Startups that discover a successful business model and cross the chasm often find it hard to transition into the next stage: executing and scaling in a growth market. Meanwhile, organizations that succeed in transforming themselves into engines of execution often lose their ability to explore new business models. Eric Ries wrote himself an imaginary memo capturing this shift in mindset:

Dear Eric, thank you for your service to this company. Unfortunately, the job you have been doing is no longer available, and the company you used to work for no longer exists. However, we are pleased to offer you a new job at an entirely new company, that happens to contain all the same people as before. This new job began months ago, and you are already failing at it. Luckily, all the strategies you've

developed that made you successful at the old company are entirely obsolete. Best of luck!⁴

A key goal of successful portfolio management in an enterprise is understanding how to balance exploring new businesses with exploiting proven existing business models—and how to transition businesses successfully from one state to the other. Leaders must understand the difference between these two domains and be able to operationalize the very different mindsets and strategies that govern them.

Exploring New Ideas

Less than 50% of startups are alive five years after they start.⁵ In a similar way, enterprises waste enormous amounts of money on trying to grow new businesses, creating little to no value for customers.⁶ Of course it's impossible to know in advance whether or not a new business will be successful, but Eric Ries' *The Lean Startup* details a method for working in conditions of extreme uncertainty. The Lean Startup methodology applies within the enterprise context just as it does in the world of startups, so long as we are clear on its purpose: *to discover and operationalize new and potentially disruptive business models, and to quickly discard those that will not work.*

Every entrepreneur, whether they work in a startup or an enterprise, has a vision of their business and the impact it will have on legions of grateful, adoring customers. For this vision to become reality, there are two key assumptions that must be tested: the *value* hypothesis and the *growth* hypothesis. The value hypothesis asks whether our business actually provides value for users by solving a real problem. If so, we can say we have found a *problem/solution fit*. The growth hypothesis tests how fast we can acquire new customers and whether we have what Steve Blank calls a repeatable and scalable sales process—in other words, if our customer base can rapidly move up the “hockey stick” in [Figure 2-1](#) and whether we have a sufficiently low customer acquisition cost. If we pass these tests, we have a *product/market fit* and can proceed to the final two stages in Steve Blank’s customer development process: *customer creation*, where we launch our business in earnest, followed by *company building* where we attempt to cross the chasm.⁷

⁴ <http://bit.ly/1v6Y8YI>

⁵ <http://bit.ly/1v6YfTX>; these numbers vary by industry, with the 5-year survival rate for infotech businesses being substantially lower than education and health: <http://bit.ly/1v6YeIN>.

⁶ Hard data is harder to come by, but circumstantial evidence is everywhere.

⁷ <http://bit.ly/1v6Y8YI>

In the Lean Startup methodology, we take a systematic approach by working through this process iteratively. We start by working out what we need to *learn* by creating a value hypothesis. We then decide what to *measure* in order to test that hypothesis. We then design an experiment, called the *minimum viable product*, which we *build* in order to gather the necessary data from real customers to determine if we have a product/market fit.

The trick is to invest a minimum of work to go through this cycle. Since we are operating in conditions of extreme uncertainty, we expect that our value hypothesis will be incorrect. At this point we *pivot*, coming up with a new value hypothesis based on what we learned, and go through the process again. We keep doing this until we either achieve a product/market fit, decide to stop experimenting, or funding dries up. The amount of time we have before the money runs out is known as the *runway*, and the goal is to pivot as frequently as possible in order to find a product/market fit while we still have runway left.

An important characteristic of the Lean Startup method is that experiments are cheap and quick to run compared to building a complete product. In general, we are able to build a minimum viable product and gather data in the space of hours, days, or weeks rather than months or years, using small, cross-functional teams that are focused on executing the build-measure-learn feedback loop shown in [Figure 2-4](#). We expect that many experiments will fail but a few will succeed; however, by being rigorous in following the steps above, every iteration will result in *validated learning*. Validated learning means that we test—to the necessary degree of precision and no further—the key assumptions behind our business model to understand whether or not it would succeed, and then made the decision to persevere, pivot, or stop.

The Lean Startup process being relatively cheap, in an enterprise context we can pursue multiple possible business models simultaneously using the Principle of Optionality.

NOTE

What Is an Option?

Purchasing an option gives us the right, but not the obligation, to do something in the future (typically to buy or sell an asset at a fixed price). Options have a price and an expiry date. Concert tickets, an agreement to go out for dinner with someone, and a decision to fund the development of a new product are all examples of options.

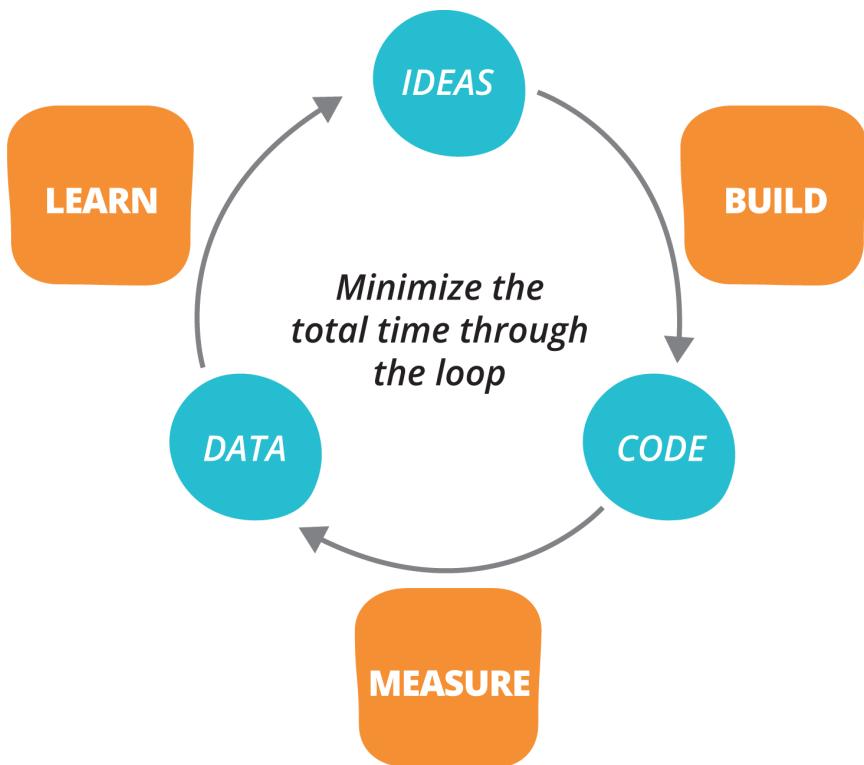


Figure 2-4. The build-measure-learn loop

Investing a fixed amount of time and money to investigate the economic parameters of an idea—be it a business model, product, or an innovation such as a process change—is an example of using optionality to manage the uncertainties of the decision to invest further. We limit our maximum investment loss (“downside”) on any individual idea, with the expectation that a small number of ideas will pay off big time, and offset or negate investments in those that did not, as shown in Figure 2-5.⁸ Optionality is a powerful concept that lets you defer decisions on how to achieve a desired outcome by exploring multiple possible approaches simultaneously.

⁸ This idea of option-like trial and error, or tinkering, is explored in Nassim Taleb’s *Antifragile* [taleb], p. 181ff. Using the language of Dave Snowden’s Cynefin framework, options are a way to make experiments “safe to fail” by designing them so as to limit the possible negative outcomes associated with failure. For more on the application of options to IT management, read *Commitment* (Hathaway Te Brake Publications) by Olav Maassen et al.

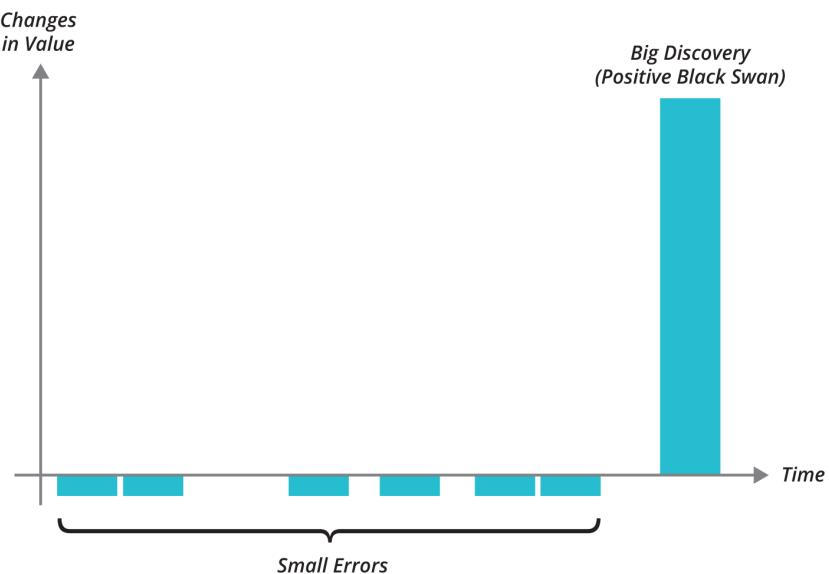


Figure 2-5. The principle of optionality, from *Antifragile: Things That Gain From Disorder* by Nassim Nicholas Taleb, 2012 (used by permission of Random House)

TIP

Effectuation

Limiting the downside and making sure every decision creates at least *some* upside (even if it is just new information) is one of several techniques that entrepreneurs apply in situations of uncertainty, where simple cause-effect (algorithmic) reasoning is an inappropriate way to manage risk. In her book *Effectuation: Elements of Entrepreneurial Expertise* (Edward Elgar Publishing), cognitive scientist Dr Saras Sarasvathy describes a framework for entrepreneurship based on research into how entrepreneurs work in real life.⁹

Limiting initial investment and creating resource scarcity is essential to managing the risk of innovation. Given that most innovative ideas we have will not succeed, we must come up with simple, quick experiments to eliminate bad ideas rapidly and cheaply.

Consider the case of the ARM CPU that is at the heart of almost every mobile device today. The first version of this processor was designed in Cambridge, UK, in the 1980s by two people, Sophie Wilson and Steve Furber. It went from

⁹ For a three-page guide to the effectuation framework, visit <http://bit.ly/1v6YjmK>.

a concept to a production-ready design in 18 months.¹⁰ When their boss, Herman Hauser, was asked how they did it, he said, “When we decided to do a microprocessor, in hindsight, I think I made two great decisions. I trusted the team, and gave them two things that Intel and Motorola had never given their people: the first was no money and the second was no people. They had to keep it simple.”¹¹

The concepts at the heart of the Lean Startup are designed to rapidly evaluate business models through identifying and testing assumptions in a scientific way. Thus they have application beyond the creation of new businesses. For example, the principles of constraining time and resources, thus limiting downside, and building a minimum viable product to test your value hypothesis as soon as possible with real customers should be applied at the start of *every* endeavor. We should use this approach to explore *all* new ideas which have unknowns, uncertainties, and therefore risks—whether it’s delivering new products, replacing existing systems, adopting new tools, processes, or methodologies, or implementing commercial off-the-shelf software solutions (COTS). Whenever you hear of a new IT project starting up with a large budget, teams of tens or hundreds of people, and a timeline of many months before something actually gets shipped, you can expect the project will go over time and budget and not deliver the expected value.

WARNING

Apply Lean Startup to Internal IT Projects

Lean Startup principles are just as important for internal software engineering projects, including services and platforms such as private clouds, systems replacements, and so forth. Enormous initiatives, with roadmaps of months or even years, constantly pop up for these types of projects, with lip service paid to working incrementally to solve a real (internal) customer problem. In fact, teams building these systems are often dismissive of their customers’ needs and preferences—we often hear statements such as “we know what they need better than them.” Projects run in this way, without regularly delivering incremental value to their customers in order to get feedback, are an appalling waste of time and resources and rarely achieve their intent, outcome, or objectives. But there are other serious negative consequences: internal systems that are painful to use make employees frustrated, impact morale and their ability to do their work effectively. Apart from underperformance costs, businesses create systems that add further complexity to already enormously complex production environments. The inevitable outcome is “shadow IT”—teams deserting the services approved or maintained by the IT department in favor of something better so they can get their work done.

10 <http://bit.ly/1v6Ynmw>

11 <http://bit.ly/1v6YoH7>

Organizations tend to start new projects with large teams both because they assume (wrongly) that this will help finish sooner and because they use processes, such as annual budgeting cycles, that stimulate land grabs for favored projects and resources. In building complex systems, however, these forces inevitably lead to system bloat, increased complexity and dependency management, inefficiency, and poor quality. Establishing and trying to maintain effective communication within large teams consumes enormous amounts of capacity on large projects. Meanwhile, the systems created grow rapidly in an uncontrolled and undirected manner.

In this environment it is extremely hard to establish effective feedback loops to determine whether what is being built is valuable and aligned to the product or project vision. Often it's not even possible to integrate the components into a working system for much of the project—and when we try to do that, we find a myriad of problems that must be addressed to get the system into a working state, let alone released. It has been our experience, as reflected in [Table 1-2](#), that adding more upfront planning to this process tends to make the eventual outcome worse, not better.

No major piece of work should be fully funded before we have evidence to support the business and economic model on which it is based, and this exploration must be done with small, cross-functional teams with a limited runway, as described in [Part II](#).

When Exploring New Business Models, Minimize Investment in Software Development

One large retail organization we worked with wanted to open a store in a new market—their first international expansion. The IT team were given eight weeks to adapt their point-of-sale system to work in the new country, calculating a different sales tax and using a different currency. We estimated that changing the existing system to work in multiple currencies and tax regimes would be a substantial multi-month IT project requiring significant investment. Forced to seek options to validate that the solution was actually possible, the team hard-coded the new sales tax into the existing mainframe system and implemented a simple proxy that replaced the currency symbols in real time for systems in the new store. Although the international expansion was ultimately cancelled as a result of the 2008 financial crisis, the initial software part of the project validated the proposed solution with minimal investment, before an investment into a fully functioning and robust long-term solution was agreed.

A final note on exploration. In this chapter we focus on the diffusion of innovation as it applies to products, but exactly the same principles apply to organizational change. Many enterprises try to roll out new methodologies, practices, processes, and tools across the entire organization in one go, ignoring the

fact that people respond to such innovations in different ways and that there is no one-size-fits-all approach to adoption. It is common to see this kind of “big bang” approach fail to achieve expected results, or be quietly abandoned to give way to another new initiative attempting to address the failings of the last.

We should explore and experiment with radical process changes—known as *kaikaku* in Lean terminology—in the same way we explore potential new business models. That is, we should try them out with a relatively small, cross-functional part of the organization, with people that fall in the “innovator” category. These people must be interested in the proposed process experiments and have the necessary skills to run them. For a change that proves to be valuable, this team can help other groups adopt it so it “crosses the chasm” within the wider organization until it becomes the standard way to work. However, process improvement does not stop here. As we discuss in [Chapter 6](#), all teams will still make continuous, incremental process improvements, known as *kaizen*, as part of their daily work to reduce waste and increase throughput and quality. Organizational culture will be discussed in more detail in [Chapter 11](#).

Exploiting Validated Business Models

Enterprises are optimized to exploit business models that have crossed the chasm—it’s what they are designed to do. However, it’s very common for engineering work to be the bottleneck when evolving existing products and introducing new products within the category being exploited.

Projects form the basis of the traditional paradigm for carrying out work in the enterprise. A project typically requires a business case to be written to gain a budget allocation, which in turn leads to a large amount of upfront planning, design, and analysis. The various departments must then coordinate the work and execute the plan. Success of a project is measured by completing the original plan on time and budget. Sadly, however, whether the project “succeeds” according to these criteria is irrelevant and insignificant when compared to whether we actually created value for customers and for our organization.

Data gathered from evolving web-based systems reveals that the plan-based approach to feature development is very poor at creating value for customers and the organization. Amazon and Microsoft (along with many startups) use a technique called A/B testing to gather data on whether a feature will actually deliver value to users before it gets built in full. Ronny Kohavi, who directed Amazon’s Data Mining and Personalization group before joining Microsoft as General Manager of its Experimentation Platform, reveals the “humbling statistics”: 60%–90% of ideas *do not improve the metrics they were intended to improve*. Based on experiments at Microsoft, 1/3 of ideas created a statistically significant positive change, 1/3 produced no statistically significant difference,

and 1/3 created a statistically significant negative change.¹² All of the ideas tested were thought to be good ones—but neither intuition nor expert opinion are good gauges of the value our ideas have for users.

The project paradigm exacerbates this problem. Projects typically take so long to go from start to finish that stakeholders try and ram as many features as possible into each one, mindful of the fact that it will be hard to get features added once the project is complete. Furthermore, the planning process happens when we have the least information and understanding of project risks—right at the beginning. Due to a cognitive bias known as the *planning fallacy*, executives tend to “make decisions based on delusional optimism rather than on a rational weighing of gains, losses, and probabilities. They overestimate benefits and underestimate costs. They spin scenarios of success while overlooking the potential for mistakes and miscalculations. As a result, they pursue initiatives that are unlikely to come in on budget or on time or to deliver the expected returns—or even to be completed.”¹³

As we execute the project, we discover new information—but since nobody wants their features cut, new information generally leads to more work, which is known as “scope creep.” Donald Reinertsen describes the vicious cycle of adding more scope as we run projects and discover more information as the “large batch death spiral”—which, combined with the planning fallacy, means projects will overrun both their budget and due date in proportion to their size. This is an important argument for working in small batches.¹⁴

All of these features and added scope means that projects typically add a tremendous amount of complexity to production environments, which—as we discuss in [Chapter 14](#)—is not typically accounted for as part of the project planning process. This complexity leads to higher costs and unplanned work in the operations department, and adds significantly to the cost and effort required to execute future projects.

Finally, because the project approach judges people according to whether work is completed on time and on budget, not based on the value delivered to customers, productivity gets measured based on *output* rather than *outcomes*. This drives several damaging behaviors. Product people become judged on

12 [kohavi]

13 [kahneman], p. 252. The planning fallacy is relied upon by many service providers who submit rock-bottom bids for the initial, predefined, contractual services (especially when contracts are awarded to the lowest bidder) and then make their profit through change requests for which customers pay a premium.

14 Reinertsen devotes a whole chapter ([reinertsen], Chapter 5) of his book to the case for reducing batch sizes.

their ability to create comprehensive specification documents and well-crafted business cases, not on whether the products and features they come up with deliver value to users. Developers are rewarded for having code completed on their developer workstations, but not for integrating it into a working, tested system that can survive real-world usage at scale. We create an unsustainable “hero culture” that rewards overwork and high utilization (making sure everybody is busy) rather than doing the least possible work to achieve the desired outcomes.

High utilization means work involving collaboration takes longer to complete, because the people you need to work with are always busy with other priorities. In order to meet ever more serious deadlines, people fail to carry out maintenance and process improvement work (such as automation) that would increase quality and throughput. This, in turn, drives up the cost of doing further work, increasing the pressure on the organization to “work harder” and fueling a vicious cycle of overwork.

John Seddon, author of *Freedom from Command & Control* (Productivity Press), states that “dysfunctional behavior is ubiquitous and systemic, not because people are wicked, but because the requirement to serve the hierarchy competes with the requirement to serve customers...people’s ingenuity is engaged in survival, not improvement.”

How do we extricate ourselves from this downward spiral? In **Part III** of this book, we describe how to run large-scale programs of work in the exploit domain, using the following principles:

1. *Define, measure, and manage outcomes rather than output.* Applying the Principle of Mission, we specify “true north” for our program of work—our ideal stakeholder outcomes. Then, at the program level, we work iteratively, specifying for each iteration the measurable program-level outcomes we want to achieve. *How* to achieve these outcomes is delegated to teams working within the program. Based on the feedback from real customers after each iteration, we work to improve quality of demand, improve speed, and improve quality of outcomes.
2. *Manage for throughput rather than capacity or utilization.* We implement Kanban principles by making all work visible and limiting work in process. We then aim to stop starting work and start *finishing* it as soon as possible. We continuously undertake process improvement work to reduce lead time—the time it takes to deliver work—throughout the system. We use continuous delivery and work in small increments to make it cheap and low risk to deliver work in small batches with easier feedback loops.
3. *Ensure people are rewarded for favoring a long-view system-level perspective over pursuing short-term functional goals.* People should be rewarded

for continuous and effective (win-win) collaboration, for *minimizing* the amount of work required to achieve the desired outcomes, and for reducing the complexity of the systems we create to enable these outcomes. People should not be punished when failures occur; rather, we must build a culture of experimentation and collaboration, design systems which make it safe to fail, and put in place processes so we can learn from our mistakes and use this information to make our systems more resilient.

Balancing the Enterprise Portfolio

The key to managing an enterprise business portfolio, as with any financial investment, is to use an economic model. However, this is not a widespread practice. In a survey of 161 global business decision makers shown in [Figure 2-6](#), only 24% of respondents reported using an economic model to make investment decisions in products and services. Astonishingly, 13% admitted that the most highly paid person's opinion (known as the HiPPO method) is the primary deciding factor.¹⁵ 47% reported using the only slightly less embarrassing method of decision by committee.

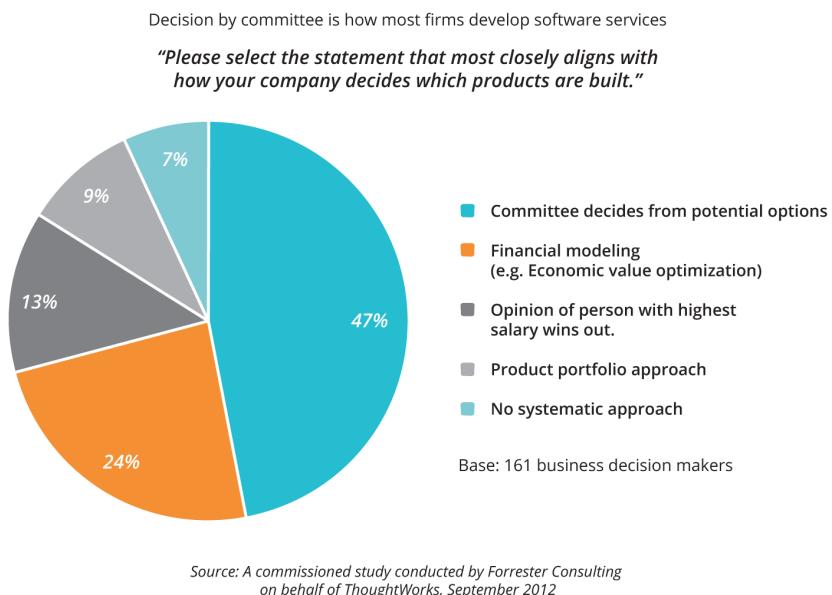


Figure 2-6. How do enterprises make investment decisions?

¹⁵ This term was coined by Ronny Kohavi, partner architect at Microsoft.

In *Escape Velocity: Free Your Company's Future from the Pull of the Past*,¹⁶ Geoffrey Moore presents a “growth/materiality matrix” for visualizing existing investment decisions, shown in Figure 2-7, and describes how it allows us to distinguish between companies which have an effective portfolio strategy and those which do not. The y axis of the diagram measures whether a particular business is material to you relative to others, where “material” means that it generates 5–10% or more of the total revenue or profit. The x axis measures the growth rate of the business in absolute terms.

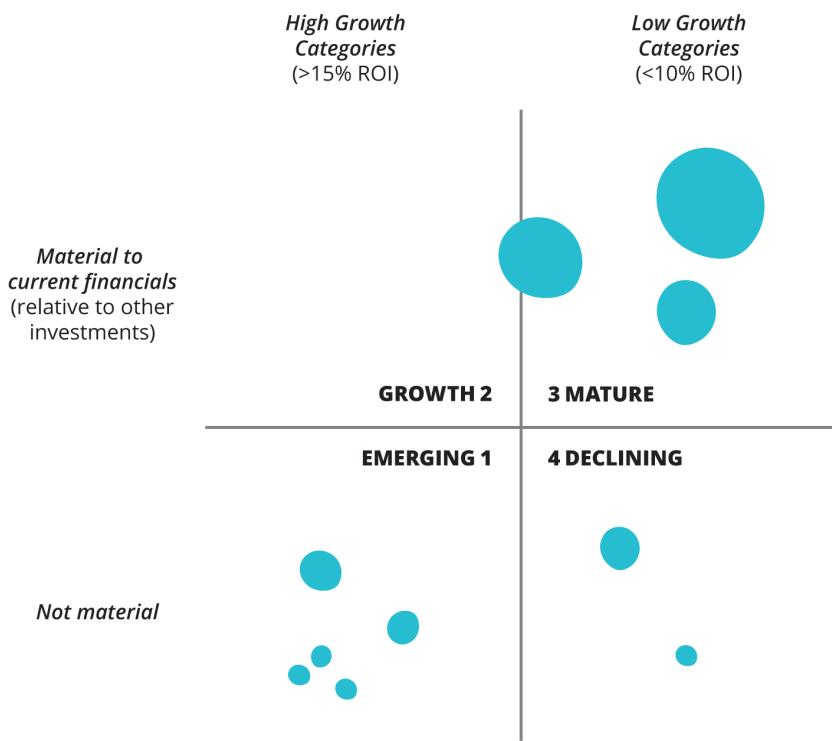


Figure 2-7. The growth/materiality matrix for portfolio management, from *Escape Velocity: Free your Company's Future from the Pull of the Past* by Geoffrey A. Moore, 2011 (used by permission of HarperCollins Publishers LLC)

Many organizations—in 2014, think Microsoft, IBM, and HP—have market-leading franchises in quadrant 3, corresponding to stage C (a mature market) in Figure 2-3. But, as shown in Figure 2-7, none has been able to develop

¹⁶ [moore]

(rather than acquire) a major franchise in stage B (corresponding to quadrant 2) despite substantial R&D investments which have led to new businesses in quadrant 1. In contrast, Amazon, Google, and Apple have each created businesses in the past decade which have grown rapidly to become material to the enterprise.

In order to understand why so many companies fail to create businesses in quadrant 2, we must understand the dynamics of the enterprise portfolio. This is described in the *three horizon* model presented in [baghai], shown in Figure 2-8. Horizon 1 consists of your set of core product categories and businesses (corresponding to quadrant 3 in Figure 2-7).

MANAGING A PORTFOLIO

The Three Horizons Model

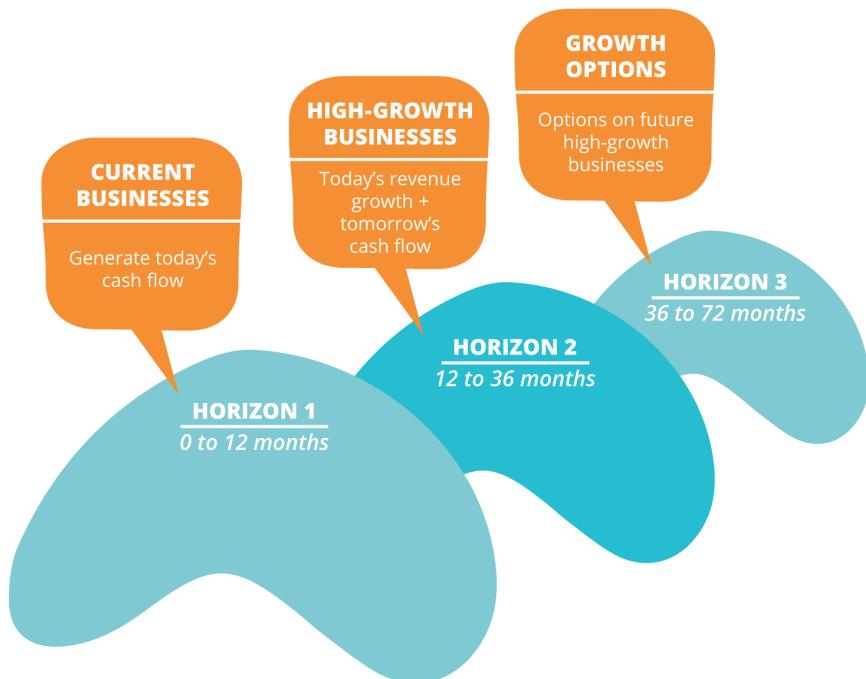


Figure 2-8. *Three Horizons*, from *Escape Velocity: Free your Company's Future from the Pull of the Past* by Geoffrey A. Moore, 2011 (used by permission of HarperCollins Publishers LLC)

Investments in horizon 1 businesses will deliver results in the same year, and typically take the form of developing existing products and launching new ones within the existing categories. Horizon 2 is the set of emerging businesses which will form the core business of the future. These require significant investment and the attention of sales and marketing divisions to succeed, but will not deliver the same levels of returns as horizon 1 investments.

Horizon 3 is the domain of the Lean Startup where we experiment with new business models and attempt to create a product/market fit for new businesses. We aim to invest enough time and money to create a runway to discover a product/market fit before making further investments. We then either move the idea into horizon 2 or shelve it, perhaps until market conditions or advances in technology make it favorable to try again.

There are three significant problems conspiring to kill businesses that make it into horizon 2. First, they require substantial investment in terms of research, sales, and marketing resources without delivering corresponding revenue returns—the metric by which these departments are usually measured. Second, each of the three horizons requires very different management and support practices in order to succeed, as shown in [Table 2-2](#). Blindly applying a consistent approach to each will result in failure. Finally, as Clayton Christensen discusses in *The Innovator's Dilemma* (Harper Business), profitable enterprises are often reluctant to cannibalize their profits and market share by launching disruptive new products—those that might threaten their existing bottom line and market valuation.

Table 2-2. Three horizons

	Horizon 1 (0-12 months)	Horizon 2 (12-36 months)	Horizon 3 (36-72 months)
Goals	Maximize economic returns	Cross the chasm, start contributing significant revenues	Create a new business
Key metrics	Revenue versus plan, market share, profitability	Rate of sales, target accounts	Buzz / word-of-mouth popularity (consumer), name-brand customers (enterprise)

We frequently see these forces conspiring to prevent businesses from making it through horizon 2, even in enterprises which do an excellent job of both exploring and exploiting business models. Often, other companies eventually bring them to market with devastating results. Xerox PARC invented the modern GUI (as well as many other elements of modern computing), but the “toner heads” at Xerox’s head office “had no clue about a computer or what it

could do” and so it was ultimately Apple and Microsoft who brought computers into people’s living rooms instead.¹⁷

Photography giant Kodak, which filed for bankruptcy in 2012, actually invented the digital camera. Steve Sasson and his team from the Kodak Apparatus Division Research Laboratory created the breakthrough innovation in 1975.¹⁸ However, the team was met with puzzled managers who could not comprehend why customers would ever want to view photographs on a monitor. Their business was optimized for developing photographs—making paper, film, and other supplies—not capturing memories.

WARNING

Why You Cannot Simply Hire or Acquire Your Way to Innovation

A number of enterprises have been acquiring startups in an attempt to pick on a current trend and accelerate innovation—perhaps to diversify and balance their portfolio, to turn themselves into “innovation labs,” or to tick a box in stage A, quadrant 1, or horizon 3. We have seen at close hand the poor outcomes this creates, with these acquisitions failing to produce the expected return and the senior staff leaving as soon as they could exercise their options. The problems occur when the acquired company—working on a horizon 3 or 2 product—is subjected to the horizon 1 governance, financial targets, and management structures of the acquiring enterprise, completely destroying its ability to innovate. Sometimes people from the parent organization are rotated through the innovation lab in the hope this will magically teach them how to innovate in a different horizon, instead of simply giving them culture shock. Acqui-hiring frequently fails for the same reason: taking great people and putting them into a pathological or bureaucratic culture does not change the culture—it breaks the people. The solution is to do the hard work to transform the culture of your own organization and grow effective leadership and management appropriate to each horizon—which will, incidentally, remove the need to hire in or acquire innovators.

Our hypothesis is that organizations survive and grow in the medium and long term by balancing the ability to continuously explore potential new business models with effective exploitation of existing ones. Indeed, one of the hallmarks of a truly adaptive and resilient organization is that it continually disrupts its own existing business models in search of future opportunities and new markets and customers.

For example, Amazon’s pursuit of electronic books and the production of the Kindle disrupted what was then its primary business model of selling physical

17 Steve Jobs, *The Lost Interview*.

18 <http://bit.ly/1v6YwGv>

books. The development of the Amazon Marketplace enabled other vendors to leverage Amazon's infrastructure and potentially undercut products sold by Amazon. 3M defines its strategy as constant new product innovation and sets targets for the percentage of revenue from products introduced in the past five years at 30%, which it exceeded in 2008.¹⁹ Inge G. Thulin, President and CEO of 3M, expects that number to reach 40% by 2017.²⁰

Intuit uses a simple model to balance horizons 1, 2, and 3, as shown in **Table 2-3**. Google follows a similar model, but with different allocations: 70% to Horizon 1, 20% to Horizon 2, and 10% to Horizon 3.²¹

Table 2-3. Intuit innovation horizons and metrics²²

	Existing businesses	Adolescent businesses	Ideas
<i>Investment</i>	60%	30%	10% of operating expenses, funded quarterly based on validated learning
<i>Metrics</i>	Growing category, share, net promoter, revenue	Growth, increasing efficiency (will lead to profitability)	Love metrics based on delivering customer benefit, active product usage, proactive word of mouth
<i>Example products</i>	TurboTax, Mint	QuickBooks Online Accounting	SnapTax

The most important point to bear in mind when balancing horizons is that unless senior leadership takes an active role in managing investments, including putting in place appropriate management practices for different horizons and paying attention to how management is incentivized, core businesses will always find a way to use their corporate clout to sideline and ultimately neutralize the other horizons. If the cultural and management barriers are simply

19 <http://for.tn/11ixTko>

20 <http://www.cnbc.com/id/100801531>

21 <http://cnmon.ie/1v6YHBA>

22 <http://bit.ly/1v6YI8Q>

too strong for this kind of “ambidextrous” approach, the alternative is to spin off a maximally independent business unit.

How Aetna Created New Companies to Disrupt Its Core Businesses

Aetna, like all the players in the US healthcare market, knew that the Obama administration’s Affordable Care Act represented both a serious risk and a significant opportunity. 160 years old at the time the bill was signed into law, Aetna decided to create a new company called Healthagen, “a separate organization, separately capitalized, separately compensated, and separately managed, so they’re not subject to the same management process at Aetna” with the purpose of disrupting the healthcare provider market with new technology and business models. Healthagen has a goal to drive \$1.5bn-\$2bn of revenue per year initially.²³

Aetna has also created another subsidiary along similar lines to create a consumer marketplace and drive private exchange models. Mark Bertolini, Aetna’s Chairman, President, and CEO, states that his goal is to build a technology-based competitive ecosystem that will disrupt Aetna’s own core business.

Conclusion

Every idea has its own lifecycle, with successful ideas creating competitive advantage for early adopters and ultimately becoming the building blocks for higher-level innovations. Enterprises must ensure they have a pipeline of new ideas to provide the basis for growth in future years. Effective enterprise portfolio management requires that we create and apply an economic model to balance investments across all three horizons. For further reading on portfolio management, we recommend Geoffrey Moore’s *Escape Velocity: Free Your Company’s Future from the Pull of the Past*.

We expect that there will be several ideas for new businesses incubating in horizon 3. Since we cannot predict which ones will be successful, we apply the Principle of Optionality and assume that many will fail but a few will succeed. We apply the Lean Startup methodology to rapidly pivot through business models for these businesses until the teams exploring them run out of resources or discover a product/market fit and gain traction. Most ideas will never make it to horizon 2, but those that do require a fundamentally different management approach. In horizon 3, we care mainly about finding a product/market fit, but in horizon 2 we need to identify and manage to a wider set of risks specific to our business. Instead of business model innovation, we switch to incremental innovation, which requires a different set of skills.

23 <http://bit.ly/1v6YM8m>

Too many enterprises kill innovation by trying to manage horizon 2 and 3 investments using the strategies of horizon 1. In the rest of this book we will mainly ignore horizon 1 (although many of the principles and techniques described in **Part III** can be usefully applied to that domain). **Part II** of this book deals with the explore domain that we will leverage within horizon 3 investments. **Part III** discusses how to move fast at scale in the exploit domain, using the same lean principles that have been applied in manufacturing for decades to continuously drive higher quality and lower costs. In **Part IV** we discuss how to transform your enterprise, starting with growing an innovation culture.

Questions for readers:

- What framework does your organization use to balance your portfolio of exploring new business models, exploiting validated ones, and developing core businesses?
- Is there a place where you can see this portfolio at a glance?
- What performance metrics are used to measure the health of activities in each of these domains?
- What is the percentage ratio for investments in horizons 1, 2, and 3 in your organization? Is it intentional or accidental? What do you think should it be?
- How does senior leadership ensure that investments in horizons 2 and 3 are nurtured, and that transitions between horizons are managed in a way that will maximize the relevant performance metrics for each individual investment?

PART II

EXPLORE

The best lack all conviction, while the worst / Are full of passionate intensity.

— W. B. Yeats

When faced with a new opportunity or a problem to be solved, our human instinct is to jump straight to a solution without adequately exploring the problem space, testing the assumptions inherent in the proposed solution, or challenging ourselves to validate the solution with real users.

We can see this instinct at work when we design new products, add new features to existing products, address process and organizational problems, begin projects, or replace existing systems. It is the force that leads us towards buying expensive tools that purport to solve all of our use cases, rolling out a new methodology or organizational refresh across the whole company, or investing in “bet the company” programs of work.

Worse, we often fall in love with our own solutions, and then fall prey to the sunk cost fallacy when we ignore evidence that should cause us to question whether we should continue to pursue them. When combined with a position of power, these forces can have catastrophic consequences—one of our colleagues was nearly fired by a client for having the temerity to ask about the business case behind a particular project.

If we had one superpower, it would be to magically appear whenever a problem or new opportunity was under discussion. Our mission would be to prevent anybody from commencing a major program to solve the problem or pursue the opportunity until they do the following:

- Define the measurable business outcome to be achieved
- Build the smallest possible prototype capable of demonstrating measurable progress towards that outcome
- Demonstrate that the proposed solution actually provides value to the audience it is designed for

Since we are only mortal, we trust that you will keep a copy of this book to hand to wield at the appropriate moment.

In this part, we discuss how to explore opportunities and problem spaces by taking a scientific and systematic approach to problem solving. By taking an experimental approach, we can effectively manage the risks and enable teams to make better decisions and judgements under the uncertainty that is inherent in innovation.

Model and Measure Investment Risk

Doubt is not a pleasant condition, but certainty is absurd.

— Voltaire

For enterprises experimenting with new business models and products, as for startups, the biggest risk is a failure to create something that actually delivers value to users. The Lean Startup framework allows us to rapidly discard ideas that do not deliver value or will not be adopted sufficiently quickly so we don't waste our resources on them. However, the principles behind the Lean Startup can be applied to *all* kinds of activities within the enterprise, such as building internal tools, process improvement, organizational change, systems replacement, and GRC (governance, risk, and compliance) programs.

In this chapter we present the principles and concepts that enable us to take a systematic approach to managing the risk of planned work, by gathering information to reduce uncertainty. This framework forms the basis of a practical approach to exploring new opportunities we present throughout the rest of **Part II**.

Model Investment Risk

Typically in enterprises we must build a business case along with a plan to support it before we can get approval to proceed. This usually involves a team of people creating a detailed document that estimates the value the proposed initiative will create. The business case describes the required resources, dependencies, and finally a beautifully crafted set of numbers detailing the planned work with costs, key metrics, a resource plan, and timeframes. Depending on

the level of detail and the estimated investment required, this process can take weeks or months to complete.

An important goal of the planning process is to support an investment decision. In order to make this decision, we need to have a good understanding of the *risks* involved with the investment. Following Douglas Hubbard, we define risk as “a state of uncertainty where some of the possibilities involve a loss, catastrophe, or other undesirable outcome,” and the *measurement* of risk as “a set of possibilities, each with quantified probabilities and quantified losses.”¹ For example, “We believe there is a 50% chance the project will be cancelled, with a loss of \$2m in development work.”

In *How to Measure Anything*, Hubbard discusses his work analyzing business cases for IT investments:²

Each of these business cases had 40 to 80 variables, such as initial development costs, adoption rate, productivity improvement, revenue growth, and so on. For each of these cases, I ran a macro in Excel that computed the information value for each variable. I used this value to figure out where to focus measurement efforts. When I ran the macro that computed the value of information for each of these variables, I began to see this pattern: 1) The vast majority of variables had an information value of zero...2) The variables that had high information values were routinely those that the client never measured. 3) The variables that clients used to spend the most time measuring were usually those with a very low...information value.

Take the example of estimating development costs in order to put together business cases to obtain project approval. This usually involves analyzing months’ worth of future work, breaking it into small pieces, and estimating the effort required for each piece. However, as Hubbard notes, “Even in projects with very uncertain development costs, we haven’t found that those costs have a significant information value for the investment decision...The single most important unknown is whether the project will be canceled...The next most important variable is utilization of the system, including how quickly the system rolls out and whether some people will use it at all.”³

Thus the business case essentially becomes a science fiction novel based in an universe that is poorly understood—or which may not even exist! Meanwhile significant time is wasted on detailed planning, analysis, and estimation, which

1 Definitions are taken from [hubbard], p. 50.

2 [hubbard], p. 111.

3 http://www.cio.com/article/119059/The_IT_Measurement_Inversion

provides large amounts of information with extremely limited value. According to research by Donald Reinertsen, author of *The Principles of Product Development Flow: Second Generation Lean Product Development*,⁴ it's typical for 50% of total product development time to be spent in such "fuzzy front end" activities. Naturally, this leads to poor investment decisions and needlessly long product development cycles. This creates multiple negative outcomes:

- Long product development cycles dramatically reduce the potential return on investment we can achieve from successful new products.
- Most perniciously, long development cycles delay the time it takes to get customer feedback on whether we are building something valuable.
- Typical market research activities are poor at predicting a product/market fit, especially in new product categories. Research said that minivans and iPods would not be successful.
- In the absence of good data, people tend to get their pet projects funded. Particularly in enterprise IT, we often see spectacular amounts of money poured down the drain on systems replacement projects—even (perhaps especially?) in organizations operating in highly regulated sectors.

There are two factors we care about in a business plan. The first is the sensitivity of the key metric to the various variables in the business case. The second is the level of uncertainty in the variables to which the key metric is sensitive. Given distributions and ranges for the key variables, a simple but powerful approach is to perform a Monte Carlo simulation to work out the possible outcomes. This will allow us to find the variables to which we need to pay attention in order to make good investment decisions.

To run a Monte Carlo simulation, we use a computer to create thousands of randomized scenarios based on the distribution shape and ranges for the input variables, and then compute the value of the metric we are interested in for each scenario. The output of a Monte Carlo simulation is a histogram, with the number of scenarios for each range on the *y*-axis, and the ranges on the *x*-axis. You can perform a Monte Carlo simulation using Excel, or use one of a number of existing custom tools.⁵ The output of a Monte Carlo simulation for a business case might look something like Figure 3-1. As Hubbard notes, the uncertainty in ROI for IT programs tends to be very high and increases with the duration of the program.

⁴ [reinertsen]

⁵ See <http://www.howtomeasureanything.com> for an example. For an introduction to Monte Carlo simulation for business models, see <http://bit.ly/1vKoXBE>.

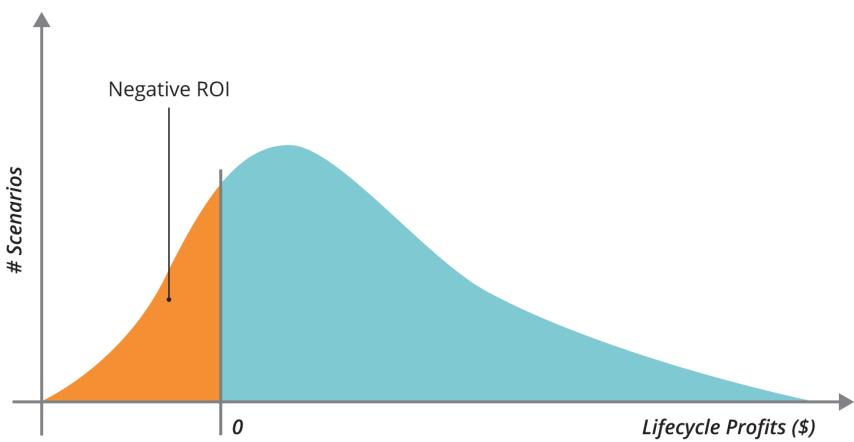


Figure 3-1. Output of a Monte Carlo simulation

As you can verify by doing a Monte Carlo simulation on your own business cases, ROI in IT programs is not very sensitive to cost, but rather to whether the program will be cancelled and to the utilization of the resulting system. These variables depend primarily on whether we have *built the right thing*. However, the standard enterprise planning process provides *almost no validation of this*.

Let us be absolutely clear. *In most enterprises, around 30%–50% of the total time to market is spent on activity which provides almost zero value in terms of mitigating the risks of our investments.* This near-zero-value activity is mostly driven by financial management and planning processes. In our experience, the fuzzy front end presents the biggest opportunity for radical process improvement (*kaikaku*) in enterprises. We can drastically reduce the required time, and make better decisions, by taking a systematic approach to risk management. In this chapter, we discuss how to attack the fuzzy front end for new businesses and new products. In [Chapter 7](#), we show how to change the way program-level feature backlogs are managed.

Applying the Scientific Method to Product Development

The way the world tells you whether what you are doing is valuable is whether they send you money.

— Donald Reinertsen

When there is a large amount of uncertainty in the key metric we care about, we begin by identifying the variables with the highest information value—the riskiest assumptions. These are the ones to which our outcome metric is most

sensitive. In the case of both business model innovation and product development, Donald Reinertsen comments that “unit sales are where the bodies are buried.”

The most inefficient way to test a business model or product idea is to plan and build a complete product to see whether the predicted market for it really exists. Yet this is exactly what we do once we have an approved business case. Part of the problem is the language we use to describe the product development process. For example, consider the term “requirements.” Whose requirements are they? Are they user requirements? In *Lean IT*, Steve Bell and Mike Orzen comment that “users are often unable to articulate exactly what they need, yet they often seem insistent about what they don’t want...once they see it.”⁶

We should stop using the word “requirements” in product development, at least in the context of nontrivial features. What we have, rather, are *hypotheses*. We *believe* that a particular business model, or product, or feature, will prove valuable to customers. But we must test our assumptions. We can take a scientific approach to testing these assumptions by running experiments.

In the case of business model and product innovation, the Lean Startup movement provides us with a framework for operating in conditions of extreme uncertainty. In *Running Lean* (O'Reilly), Ash Maurya explains how to execute a Lean Startup model:

- Do not spend a lot of time creating a sophisticated business model. Instead, design a simplified *business model canvas* which captures and communicates the key operating assumptions of your proposed business model.
- Gather information to determine if you have a problem worth solving—meaning that it is both solvable *and* people will pay for it to be solved. If both of these conditions obtain, we have achieved a *problem/solution fit*.
- Then, design a *minimum viable product* (MVP)—an experiment designed to maximize learning from potential early adopters with minimum effort. In the likely case that the results of the MVP invalidate your product hypothesis, *pivot* and start again. Continue this process until you decide to abandon the initial problem, run out of resources, or discover a *product/market fit*. In the latter case, exit the explore phase and proceed to exploit the validated model.

⁶ [bell], p. 48.

- Throughout this process, update the business model canvas based on what you learn from talking to customers and testing MVPs.

We present this approach in detail in [Chapter 4](#).

There are two key innovations in this model. First, we stop using detailed planning as a way to manage risk. Instead, we find customers and run cheap experiments to discover if our proposed business model or product is actually valuable to them. Second, rather than creating only one plan, we iterate by running a *series* of experiments in order to discover a product/market fit, since we expect that in conditions of uncertainty our first idea is very unlikely to bear fruit.

A common objection to these principles is that such experiments cannot possibly be representative of a complete product. This objection is based on a false understanding of measurement. The purpose of measurement is not to gain certainty but to *reduce uncertainty*. The job of an experiment is to gather *observations that quantitatively reduce uncertainty*.⁷ The key principle to bear in mind is this: *when the level of uncertainty of some variable is high, we need very little information to significantly reduce that uncertainty*.

NOTE

Definition of Measurement

Measurement: A quantitatively expressed reduction of uncertainty based on one or more observations.⁸

This definition may seem counterintuitive unless you have experience running experiments in a scientific context. In experimental science, the result of a measurement is never simply a single value. It is, rather, a *probability distribution* which represents the range of possible values, as shown in [Figure 3-2](#). Any measurement that doesn't indicate the precision of the result is considered practically meaningless. For example, a measurement of my position with a precision of 1 meter is far more valuable than that same position with a precision of 500 miles. The point of investing in measurement in a scientific context is to *reduce our uncertainty* about the actual value of some quantity. Thus, in particular, if we express our estimates as precise numbers (as opposed to ranges), we are setting ourselves up for failure: the chance of us meeting a date 6 months in the future *precisely to the day* is practically zero.

⁷ [\[hubbard\]](#), p. 23.

⁸ Ibid.

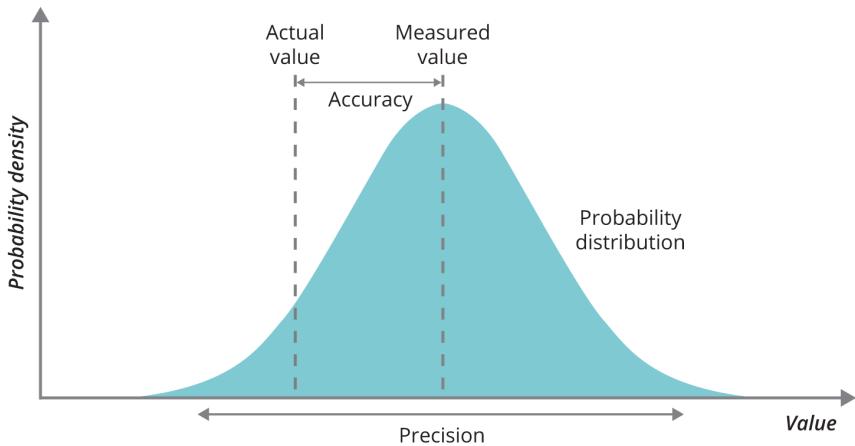


Figure 3-2. Accuracy and precision

A minimum viable product can be thought of as a way to conduct a relatively cheap measurement so as to reduce our uncertainty concerning our key metric. This is what makes an MVP such a good investment. Typically, putting together a business plan and requirements for a significant initiative takes weeks or months in an enterprise context. In the same amount of time, by following the Lean Startup model, we could run multiple experiments, learn from real customers, and emerge with a superior, battle-tested plan based on evidence. Let's examine the differences between these two approaches when we need to make an investment decision, as shown in [Table 3-1](#).

Table 3-1. Traditional product lifecycle versus Lean Startup lifecycle

	Traditional project-planning process	Lean Startup discovery process
<i>What data do we have to make the investment decision?</i>	A business plan based on a set of untested hypotheses and assumptions, backed by case studies and market research.	Real data based on evidence compiled from a working product or service tested with real customers.
<i>What happens next?</i>	We must create detailed requirements, if we haven't already, and then start a project to build, integrate, test, and finally release the system.	We already have a validated MVP which we can build upon immediately with new features and enhancements based on customer feedback.

Traditional project-planning process	Lean Startup discovery process
<i>When do we find out if the idea is any good (i.e., will it get a good return on investment)?</i>	Once the project is complete and the product or service is live. We already have this evidence based on the data we have collected.

As discussed in [Chapter 2](#), an important factor in the success of the Lean Startup approach is to limit the size of the explore team and the resources available to them (including time). This encourages people to apply their creativity and focus on learning rather than pursuing “perfect” solutions. There are no awards for elegance of software design or automated test coverage in an MVP—the more skeletal, the better, provided we can gather the information we need. Many of the “war stories” exchanged by Lean Startup practitioners describe the ingenious shortcuts they took in the pursuit of validated learning.

Of course a reasonable question is: given that product development is effectively a form of discovery, how much time and money should we spend on validated learning? Game theory actually provides a formula for the *expected value of information* (EVI). A detailed discussion of how to calculate this number is beyond the scope of this book, but it is covered in Hubbard’s *How to Measure Anything*.⁹ The EVI gives us an upper bound on how much we should be prepared to pay to gather the information in question. If the cost of performing a measurement is much less than the EVI (say, an order of magnitude less), it is clearly worth performing the measurement. Thus, the more risky and expensive the project in question, the more value you get for your money by pursuing a Lean Startup approach.

⁹ [\[hubbard\]](#), Chapter 7.

NOTE

Expected Value of Information

Hubbard defines the value of information as follows: “Roughly put, the value of information equals the chance of being wrong times the cost of being wrong. The cost of being wrong—that is, what is lost if your decision doesn’t work out—is called an opportunity loss. For a simplistic example, say you’re considering investing \$1 million in a new system. It promises a net \$3 million gain over three years. (For our example’s sake, it’ll either be completely successful or a total bomb.) If you invest but the system fails, your mistake costs you \$1 million. If you decide not to invest and you should have, the mistake costs you \$3 million. When we multiply the opportunity loss by the chance of a loss, we get the expected opportunity loss (EOL). Calculating the value of information boils down to determining how much it will reduce EOL.”¹⁰

In reality, the success of a product is rarely a binary outcome. If we return to the example of the predicted ROI for a business case illustrated in [Figure 3-1](#), we get the EOL by calculating the area of the shaded part of the curve, which represents the scenarios in which we lose money on our investment. In other words, we sum up the ROI at each point multiplied by the probability of that outcome. Assuming we had perfect information on the exact outcome in ROI, that could potentially be worth as much as the EOL we have just calculated. Since an MVP will typically provide less than perfect information, the EOL represents an upper bound on what we should spend on the runway for discovering a product/market fit.¹¹

Applying the Lean Startup Approach Internally Within Enterprises

The Lean Startup model isn’t limited to new product development. It can be used for any kind of new work in an enterprise context, including systems replacement, building internal tools and products, process innovation, and evaluating commercial off-the-shelf software (COTS). In all cases, we begin by stating the *measurable customer outcome* that we wish to achieve. We can define our goal in terms of our immediate downstream customer, such as our colleague who will use the tool, process, or COTS. For example, for an internal test automation tool, we might aim to reduce the lead time for full regression testing to 8 hours.

To determine if we have a problem/solution fit, we look for a customer willing to work with us to pilot the new system, tool, process, or software. This is a critical step which is often skipped by enterprises. Indeed for internal tools it’s common to *mandate* their use—a disastrous policy which often results in enormous amounts of waste, unhappy users, and little value to the organization. The process of finding customers and figuring out a real problem they will pay you to solve (even if the payment takes the form of

¹⁰ <http://bit.ly/1v6YRcp>

¹¹ On his website, <http://howtomeasureanything.com>, Hubbard provides a spreadsheet that helps you calculate the value of information.

time and feedback rather than money)—thereby obtaining a problem/solution fit—is essential to developing internal tools, purchasing COTS, or internal systems replacement. Mandating the use of a particular solution makes it much harder to gather feedback on whether that solution actually provides value.

Once we have a pilot team, we design and execute a minimum viable product. This may be a prototype of a tool designed to help just one team, or an implementation of a COTS package to solve a problem for just one team or for a single business process for that team. The hardest part here is to limit scope so as to solve a real problem but deliver something in the space of days or weeks, rather than months. The worst thing we can do is disappear to design the perfect tool or adoption strategy, without continually delivering value to real users and gathering feedback from them throughout the process. It's essential to be disciplined about time-boxing this activity and to focus on solving a real and an urgent problem as soon as possible.

The measure of success—and whether or not we should proceed—is whether our users find the MVP good enough to use of their own free will and whether we actually meet the measurable customer outcome we set out to achieve. If not, we need to pivot and return to the beginning.

Principles for Exploration

In [Chapter 1](#), we showed how small, highly motivated forces were able to defeat larger, better trained enemies through a style of war known as maneuver warfare. “Disruption” is a word that is currently ubiquitous to the point of cliché, but in the context of maneuver warfare, the chief exponent of the idea of disrupting your opponent’s decision-making process was Colonel John Boyd of the US Air Force. In his career as a fighter pilot and instructor, Boyd was famous for never losing his bet that he could win any dogfight—from a position of disadvantage—within 40 seconds, and also for co-creating the energy-maneuverability theory of aircraft performance that led to the design of the F-16 fighter jet. However, his best-known creation is the “OODA loop,” a model (shown in [Figure 3-3](#)) of how humans interact with their environment which forms the basis of Boyd’s theory of maneuver warfare. OODA stands for *observe, orient, decide, act*, the four activities that comprise the loop.

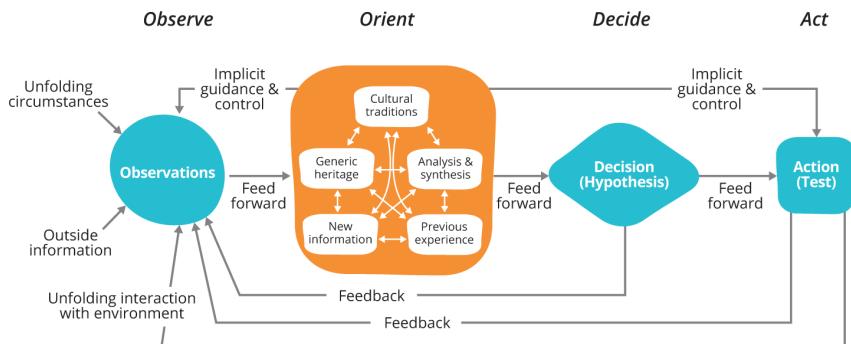


Figure 3-3. The OODA loop

A common misconception (primarily by people who have not actually seen the diagram) is that these activities are carried out one after the other in a loop, and that disruption is achieved by going through the cycle faster than your opponent. There are two important flaws with this interpretation. First, in reality both humans and organizations are performing all of these activities simultaneously, and there are multiple feedback and feed-forward loops between each of them. Second, it is often advantageous to *delay* making decisions until the “last responsible moment” (which we can analyze using optionality and Cost of Delay, see [Chapter 7](#)).

To truly understand the diagram, we must start with *orientation*. Boyd’s insight here is that our observations, decisions, and actions are all contingent upon our current orientation, which is in turn determined by a complex series of factors including our genetics, our habits and experiences, and the cultures within which we grew up and are currently operating, as well as the information we have to hand. The second thing to note about the diagram is that there are two mechanisms of influence: one is the feedback and feed-forward loops, and the other is “implicit guidance and control.”

Psychology tells us that our actions can be shaped either by IGT (implicit guidance and control) or by feed-forward from a conscious decision. Implicit guidance and control in humans is provided by a system in the mind, called *System 1*, which “operates automatically and quickly, with little or no effort and no sense of voluntary control.” Conscious decisions are made by *System 2* which “allocates attention to the effortful mental activities that demand it, including complex computations. The operations of System 2 are often associated with the subjective experience of agency, choice, and concentration.”¹² Equally, IGT

¹² [\[kahneman\]](#), pp. 20–21. These names were coined by Stanovich and West in [\[stanovich\]](#).

affects how we *observe* things, for example our tendency to ignore information that contradicts our beliefs (this is known as *confirmation bias*).

Both of these mechanisms exist at the organizational level. In terms of action, organizations use the implicit guidance and control mechanism when they delegate decision-making using decentralized command and the Principle of Mission, relying on a shared understanding of their goals along with alignment across the organization to ensure that people act in the interests of the wider organization. However, some actions (particularly those involving compliance) must be taken using the explicit feed-forward mechanism.

Implicit guidance and control also govern how organizations observe. Generative cultures create monitoring systems and visible displays that enable people throughout the organization to rapidly access relevant information—which, in turn, changes their orientation. Changes in orientation will cause us to update what we measure and how information flows through the organization. In pathological and bureaucratic organizational cultures, measurement is used as a form of control, and people hide information that challenges existing rules, strategies, and power structures. As Deming said, “whenever there is fear, you get the wrong numbers.”

When Boyd talks about “operating inside” an opponent’s OODA loop, he means *understanding* our opponent’s loop and how it determines their actions. Then you can use that knowledge against them:

The basic pattern is simple: An organization uses its better understanding of—clearer awareness of—the unfolding situation to set up its opponent by employing actions that fit with the opponent’s expectations, which Boyd, following Sun Tzu, called the *zheng*. When the organization senses (viz. from its previous experiences, including training) that the time is ripe, it springs the *qi*, the unexpected, extremely rapidly. The primary reason for implicit guidance when engaged with opponents is that explicit instructions—written orders, for example—would take too much time. As Boyd put it, “The key idea is to emphasize implicit over explicit in order to gain a favorable mismatch in friction and time (i.e., ours lower than any adversary’s) for superiority in shaping and adapting to circumstances.”¹³

The OODA model can also be applied in the context of customer engagement: “Instead of surprise → shock → exploitation, as in war and the martial arts,

¹³ This quote and the OODA loop diagrams in this section were taken from Chet Richards’ excellent discussion of the OODA loop: http://www.jvminc.com/boydsrealooda_loop.pdf. Chinese words have been updated to use pinyin.

zheng/qi could operate as something more like surprise → delight → fascination → become more committed customers. Apple plays this game, the ‘pursuit of wow!’ as Tom Peters once described it, very well.”¹⁴

Boyd refers to the implicit guidance and control pathways within an organization, determined by its culture and existing institutional knowledge and processes, as its *repertoire*. We have discussed how organizations apply their existing repertoire to disrupt competitors, but in order to improve performance and avoid disruption, we must be constantly creating new repertoire of our own. This can take the form of process improvement, evolution of existing products, or creation of new businesses and new products. This loop is also represented in the OODA model, as shown in Figure 3-4.

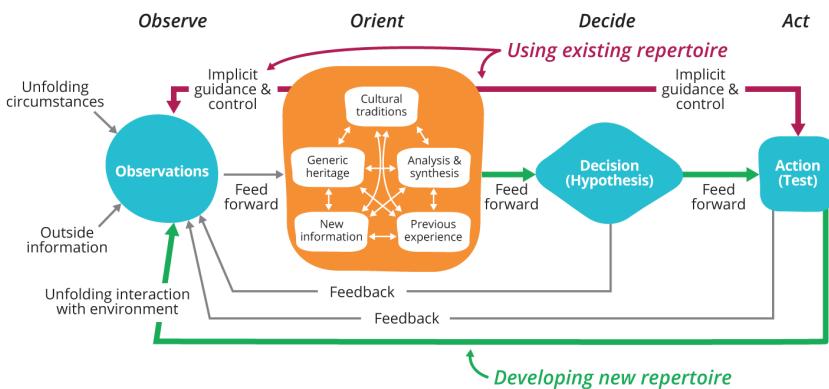


Figure 3-4. Creating new repertoire

The repertoire generation loop is more or less a statement of the scientific method, in which we create new hypotheses based on observation and synthesis, design experiments to test these hypotheses, and then update or discard our theories (which form part of our orientation) based on the results of the experiment. This loop, in turn, inspired Eric Ries’ build-measure-learn loop (Figure 2-4) which shows how to create new repertoire in the form of new business models, products, and features. The build-measure-learn loop seems straightforward, but is hard to adopt in practice due to its combination of a scientific approach (building to learn) with an engineering mindset (learning to build).

14 Chet Richards, op. cit.

For process improvement (discussed in [Chapter 6](#)) and for changing organizational culture (discussed in [Chapter 11](#)), we can use a loop known as the Deming cycle, shown in [Figure 3-5](#).

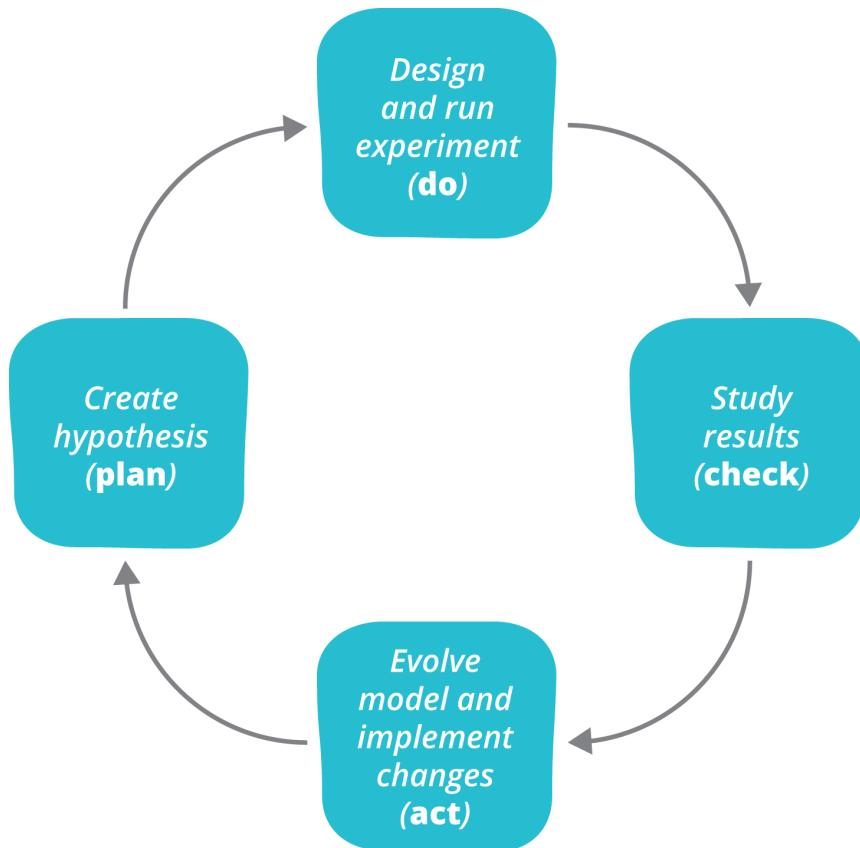


Figure 3-5. The Deming cycle

The key to being successful with these cycles (and the scientific method in general) is to use them *systematically* and *continuously*. Applying them *systematically* means using them as a general tool to explore *all* types of risk, ensuring that the expense of running an experiment is commensurate with the value of the information we will discover. Applying them *continuously* means doing it as often as possible (as Mike Roberts says, “Continuous means much more often than you think”), with a focus on getting through the loop in the shortest possible time. The most important question to ask in the context of repertoire generation is: how fast can we learn? While we may not immediately *release* the results of our learning exercises to the wider world—when to launch your

product is a matter of strategy—we should learn and test our assumptions with real users as frequently as possible.

When everybody in the organization has been trained to employ the scientific approach to innovation as part of their daily work, we will have created a generative culture. We achieve this by practicing the experimental approach until it becomes habitual, part of our repertoire, using the Improvement Kata described in [Chapter 6](#). That is what allows an organization to adapt rapidly to its changing environment. Toyota calls this “building people before building cars.”¹⁵

Scientific Management Versus the Scientific Method

It’s essential to distinguish between Taylor’s scientific management, discussed in [Chapter 1](#), and the experimental approach we describe here. In scientific management, analysis is performed and decisions are taken by management, with the people who do the work functioning more or less as automatons. In the experimental approach, the job of leadership and management is to design, evolve, and operate a system in which the people doing the work have the necessary skills and resources to run their own experiments, thus individually and collectively learning, developing, and growing their knowledge.

As shown in [Table 3-2](#), applying the scientific method to product development is fundamentally different from the traditional plan-based approach and requires different skills and behaviors. It is not that the traditional project life-cycle is bad—it can be effective in projects where the thing to be built has been built many times before and the risks are well understood. But traditional project management is the wrong model for conditions of uncertainty, such as new product development or any kind of custom software development.

15 [liker]

Table 3-2. Traditional project planning versus Lean Startup

Skill or behavior	Traditional planning approach	Experimental approach
<i>Changes to the plan</i>	Changes to the plan once it has been agreed upon are considered problematic and indicate a failure in the process.	We expect that the initial plan will not survive contact with real customers, and aim to invalidate it and pivot as quickly as possible.
<i>Skills required</i>	Requirements gathering, analysis, determination of costs, resource and dependency planning, ability to gain political support	Designing experiments and performing measurements, data collection and analysis, ability to work effectively in cross-functional teams and communicate with wider organization
<i>How success is measured</i>	Whether the plan is approved and funded	How fast we can go through learning cycles and exit the explore phase, either by de prioritizing or canceling the work, or proceeding to the exploit phase
<i>How we achieve compliance</i>	Were the appropriate processes correctly followed and have the necessary signoffs been gathered?	Did we identify the actual risks to stakeholders and gather the relevant information to effectively manage them?

The biggest obstacles to taking a scientific approach to product development and organizational change are cultural and organizational, as we discuss in [Part IV](#). In most cases, organizations have simply never taken an experiment-based approach and lack the skills and experience to implement it. In the context of product development, understanding how to design and execute experiments and analyze data is both hard and critically important—and yet they are not part of the core curriculum in most MBA programs or courses in software design and analysis. In bureaucratic and pathological organizations, an experimental approach may also challenge existing power structures and cultural norms.

Conclusion

We have laid out the foundations of a scientific approach to exploring new work—whether it's new business models and products, internal enterprise work such as building new tools, or adopting new processes. When we have a shared understanding of what we mean by risk, measurement, and uncertainty, we can apply the principles and practices of the Lean Startup movement. These provide a superior way to manage the investment decision risks than traditional planning activities.

Our ability to compete is based on creating a common orientation across the organization and enabling the people doing the work to constantly create and practice new repertoire through a process of experimentation. These activities enable us to more effectively detect and analyze changes in our environment, to get inside the decision-making processes of other organizations, and to act—to better serve our customers and shape our environment. Boyd's OODA model shows that adaptation to our environment is a continuous and ongoing process—for organizations as much as for people.

Questions for readers:

- How does your organization or department model investment risks in your business plan? What data is it based on?
- What are the variables in the plan with the highest information value? What measurements have been made to reduce the uncertainty in these variables?
- How confident are you that people will find the work you are currently doing valuable? What evidence do you have to support your decision?
- How often have you tried out the product you are working on with any of its intended users? What did you change as a result?

Explore Uncertainty to Detect Opportunities

It was darkness which produced the lamp. It was fog that produced the compass. It was hunger that drove us to exploration.

— Victor Hugo

In this chapter we will cover practices to support the principles, discussed in [Chapter 3](#), of exploring opportunities in conditions of extreme uncertainty, especially when considering new business models or products. We introduce the concept of *Discovery* to show how to quickly map out a business hypothesis to create a shared understanding of a problem and engage stakeholders from across the organization to buy in and align to our vision.

We will share concrete tools and techniques to safely create and test hypotheses to solve real business problems identified and validated in our customer development process.

Then, we will describe how to use a disciplined, scientific, evidence-based approach to experimentation to answer the fundamental question—not “*can* we build it?” but “*should* we build it?”

We will discuss how to test the riskiest assumptions of our hypothesis and generate empirical data to support our decision to pivot, persevere, or stop by creating safe-to-fail experiments using MVPs. Our purpose is to base further investment and portfolio management decisions on evidence, not science fiction. We will execute on opportunities by building the right thing at the right time and stop wasting people’s time on ideas that are not valuable.

Discovery

Discovery is a rapid, time-boxed, iterative set of activities that integrates the practices and principles of design thinking and Lean Startup. We use it intensively at the beginning of the explore phase of a new initiative.

In *Lean UX: Applying Lean Principles to Improve User Experience*, Jeff Gothelf and Josh Seiden state, “Design thinking takes a solution-focused approach to problem solving, working collaboratively to iterate an endless, shifting path toward perfection. It works towards product goals via specific ideation, prototyping, implementation, and learning steps to bring the appropriate solution to light.”¹

By combining the principles of design thinking with Lean Startup practices, we can build a continuous feedback loop with real users and customers into our development cycle. The principle is to invest the minimum amount of effort to get the maximum amount of learning, and to use the outcomes of our experiments as the base for our decision to pivot, persevere, or stop.

NOTE

Customers and Users

Although we often use the terms interchangeably, it is useful to distinguish between the *customers* of a product or service, who pay for it or invest in its development, and the *users*. Users do not pay for the product, but they contribute a great deal of value to the organization that builds the product, and often to the product itself (social networks are one obvious example). In an enterprise, people are *required* to use particular systems in order to get their work done, and organizations suffer real negative consequences when systems are hard to use. It’s essential to engage both customers and users as key stakeholders in the co-creation of products, services, or improvement opportunities.

During Discovery, we create a collaborative and inclusive environment for a small cross-functional, multidisciplinary team to explore a business, product, or improvement opportunity. The team should be fully dedicated and co-located to maximize the speed of learning and the effectiveness of real-time decision making. It must assume ownership of delivery and be empowered to make the necessary decisions to meet the objectives of the initiative.

When forming a team, it is key to keep the group small, including only the competencies required to explore the problem domain. Large teams are ill-equipped for rapid exploration and cannot learn at the speed required to be successful. The group must know their limitations and boundaries, taking

¹ [\[gothelf\]](#), Preface.

responsibility to reach out and engage others outside the group for input and collaboration when appropriate.

The final—and too often forgotten—members of the team are customers and users. It is easy to fall into the trap of seeing them as simply a consumer of the solution we have created. In fact, they are critical stakeholders. Their input is the key ingredient and the most objective measure of how valuable our solution is or can be. Through the feedback they provide, customers and users are co-creators of value for any solution. Their needs must always be the focal point for everything we do.

Creating a Shared Understanding

When you want to build a ship, do not begin by gathering wood, cutting boards, and distributing work, but awaken within the heart of man the desire for the vast and endless sea.

— Attributed to Antoine de Saint-Exupéry

When starting a new piece of work, it is imperative that the group creates an environment maximizing the potential of everyone involved. Based on the new information they are discovering, people learn, change, and improve when they are involved in a process that is energizing, interactive, and adaptive.

As Dan Pink argues in *Drive*,² there are three key elements to consider when building an engaged and highly motivated team. First, success requires a shared sense of purpose in the entire team. The vision needs to be challenging enough for the group to have something to aspire to, but clear enough so that everyone can understand what they need to do. Second, people must be empowered by their leaders to work autonomously to achieve the team objectives. Finally, people need the space and opportunity to master their discipline, not just to learn how to achieve “good enough.”

The process of shaping the vision begins by clearly articulating the problem that the team will try to solve. This essential step is often overlooked, or we assume everyone knows what the problem is. The quality of a problem statement increases our team’s ability to focus on what really matters—and, more importantly, ignore what does not. By developing our team’s shared understanding of our goals and what we aim to accomplish, we improve our ability to generate better solutions.

² [pink]

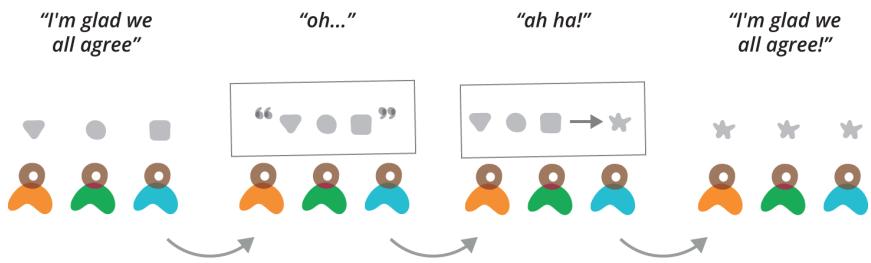


Figure 4-1. Building a shared understanding as a team

TIP

Go Gamestorming

Gamestorming by David Gray et al.³ and the supporting *Go Gamestorming* Wiki,⁴ contain numerous games that encourage engagement and creativity while bringing structure and clarity to collaborative ideation, innovation, and improvement workshops.

One of the fundamental techniques of Discovery is the use of visual artefacts, models, and information radiators to communicate and capture group learnings. Using graphical templates and exercises to externalize ideas helps our team to articulate, debate, and evolve concepts and ideas to form consensus (see Figure 4-1). It also helps to depersonalize and anonymize thoughts so we can safely debate *ideas*, not *individuals*—minimizing egos, HiPPOs (highest paid person's opinions), and extroverts' attempts to run the show.

Structured Exploration of Uncertainty

If you want to have good ideas you must have many ideas.

— Linus Pauling

When exploring uncertainty, it is important to start broad—to generate as many ideas as possible to cycle through before narrowing our focus on where we will start.

lastminute.com is a travel retailer in Europe, operating in a highly competitive industry with major players and new startups trying to disrupt the travel marketplace every day. In order to stay relevant, the company needs to innovate faster and smarter than their competitors. They invited their customers to

³ [gray]

⁴ Go Gamestorming Wiki, <http://www.gogamestorm.com>

become part of the innovation process. For two days, they ran co-creation workshops that generated over 80 new ideas for online products aligned to their business goals. The team then set up an innovation lab in a hotel lobby for a week, rapidly experimenting with each idea to discard it or validate it as a viable customer problem to implement. Within days, the team identified three winning ideas to invest further effort in developing—resulting in an over 100 percent increase in conversion for their product.⁵

Divergent thinking is the ability to offer different, unique, or variant ideas adherent to one theme; convergent thinking is the ability to identify a potential solution for a given problem. We start exploration with divergent thinking exercises designed to generate multiple ideas for discussion and debate. We then use convergent thinking to identify a possible solution to the problem. From here, we are ready to formulate an experiment to test it (see [Figure 4-2](#)).

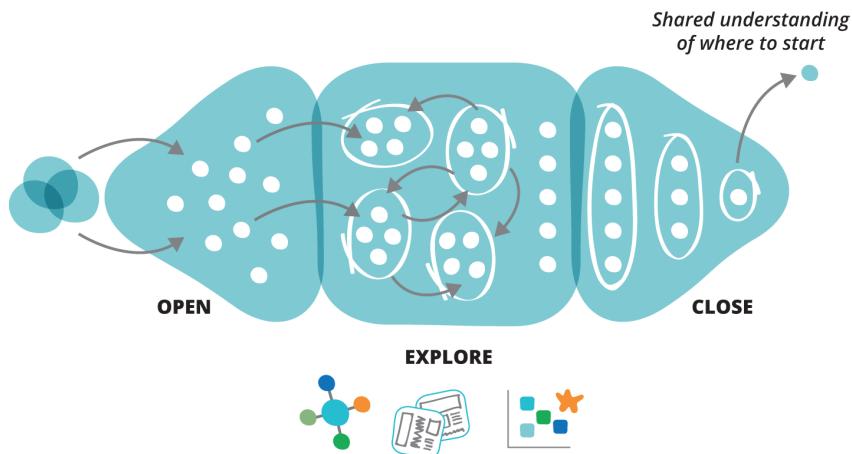


Figure 4-2. Structured exploration with divergent and convergent thinking

What Business Are We In?

Business models are transient and prone to disruption by changes in the competitive environment, advances in design and technology, and wider social and economic change. Organizations that misjudge their purpose, or cannot sense and adapt to these changes, will perish.

Organizations can be rendered obsolete by competitors that solve the same problem with an alternate or superior offering for their customers. Business

⁵ lastminute.com Innovation Lab, <https://www.youtube.com/watch?v=r64rrgbcEHo>

definition and identification of future opportunities must be continually challenged and ever evolving. Allowing complacency to sneak in due to current success is the quickest path to failure for tomorrow. We only need to cite examples such as Blockbuster versus Netflix or HMV and Tower Records versus iTunes, YouTube, and Spotify to illustrate the point that no business model or competitive advantage is indefinitely sustainable.

Winning organizations continually experiment and test theories to learn what works and what does not, recognizing that the ones that do could have a massive impact on the business' future fortunes.

Understanding Our Business Problem to Inform Our Business Plan

As Steve Blank, author of *The Four Steps to the Epiphany*⁶ and *The Startup Owner's Manual* (K & S Ranch), says:

A business plan is the execution document that existing companies write when planning product-line extensions where customer, market, and product features are known. The plan is an operating document and describes the execution strategy for addressing these “knowns.”

The primary objective of a new business initiative is to validate its business model hypotheses (and iterate and pivot until it does). Search versus execution is what differentiates a new venture from an existing business unit. Once a business model is validated, then it should move into execution mode. It's at this point the business needs an operating plan, financial forecasts, and other well-understood management tools.⁷

It is critical to consider many different business models in the early stages of a new initiative. We don't want to commit to a plan until we test the business model hypothesis and have evidence that we are on the correct path. The team must identify the riskiest assumptions of our hypothesis, devise experiments to test those assumptions, and increase the information we can gain to reduce uncertainty. The only assumption that always holds true is that no business plan survives first contact with customers.

The *Business Model Canvas*, shown in [Figure 4-3](#), was created by Alex Osterwalder and Yves Pigneur along with 470 co-creators as a simple, visual business model design generator. It is a strategic management and entrepreneurial tool that enables teams to describe, design, challenge, invent, and pivot

⁶ [blank]

⁷ Steve Blank, <http://steveblank.com/2012/03/05/search-versus-execute>

business models. Instead of writing a business plan, which can become a lengthy process, we outline multiple possible models—each time-boxed to 30 minutes—on a canvas.

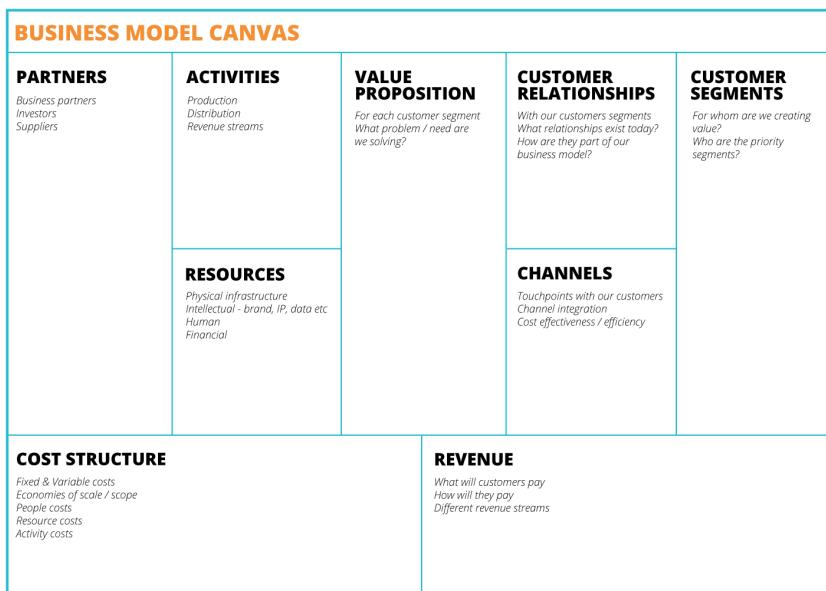


Figure 4-3. The Business Model Canvas

The Business Model Canvas, freely available at <http://www.businessmodelgeneration.com/canvas>, outlines nine essential components of an organization's conceptual business model:

Customer segment

Who are we targeting to create value for? Who are our customers?

Value proposition

What problems are we going to solve to create value for our customers?

Channels

Through what channels are we aiming to reach out to our target customers?

Customer relationships

What type of relationship does each of our customers expect us to create and maintain with them?

Activities

What activities will be required to support our value propositions?

Resources

What resources, people, technology, and support will be needed for the business to operate?

Partnerships

Who do we need to build partnerships with? Who are our key suppliers or who could be needed to provide support resources or activities for our value proposition?

Cost

What are the most important inherent costs with our business?

Revenue

For what value are our customers willing to pay? How much and how often?

By populating the individual elements of the template, we are prompted to consider any potential idea in terms of the entire business' component building blocks. By populating the entire template, we are encouraged to think in a holistic manner about how these pieces fit together to support the greater opportunity. It is key to remember that each component of the canvas represents a set of hypotheses and associated assumptions that require validation to prove that our business model is sound.

Beyond the template itself, Osterwalder also came up with four levels of strategic mastery of competing on business models to reflect the strategic intent of an organization:

Level 0 Strategy

The Oblivious focus on product/value propositions alone rather than the value proposition *and* the business model.

Level 1 Strategy

The Beginners use the Business Model Canvas as a checklist.

Level 2 Strategy

The Masters outcompete others with a superior business model where all building blocks reinforce each other (e.g., Toyota, Walmart, Dell).

Level 3 Strategy

The Invincible continuously self-disrupt while their business models are still successful (e.g., Apple, Amazon).

Our ability to recognize what strategy we are pursuing when creating business models is the first step towards creating a shared understanding of what innovation approach will be most effective in helping us achieve our goals.

The primary objective of the Business Model Canvas is to externalize the business hypothesis and make its assumptions clear so we can identify and validate the main risks. The canvas provides a framework for understanding of each business model, in terms that are understood by all, thus building a shared sense of ownership and enabling collaboration throughout the organization. The Business Model Canvas differs from other canvases listed in [Table 4-1](#) in that it doesn't assume that product/market fit is the riskiest hypothesis that must be tested first.

There are a number of canvas created by others that focus on product development, as shown in [Table 4-1](#).

Table 4-1. Visual ideation canvases

Name	Purpose
<i>The Lean Canvas</i> ⁸	Makes the assumption that product/market fit is the riskiest hypothesis that must be tested.
<i>The Opportunity Canvas</i> ⁹	Focuses discussions about what we're building and why, then helps you understand how satisfying those specific customers and users furthers the organization's overall strategy.
<i>Value Proposition Canvas</i> ¹⁰	Describes how our products and services create customer gains and how they create benefits our customers expect, desire, or would be interesting in using.

Understanding Our Customers and Users

The single most important thing to remember about any enterprise is that there are no results inside its walls. The result of a business is a satisfied customer.

— Peter Drucker

In order for any product or solution to be successful, people must want to use it, and indeed pay money for it. For a team to build a solution that addresses a

⁸ The Lean Canvas, <http://www.leancanvas.com>

⁹ The Opportunity Canvas, <http://comakewith.us/tag/opportunity-canvas>

¹⁰ Value Proposition Canvas, <http://bit.ly/1v6Z5Ae>

real problem or need, it is essential to understand who we are trying to reach and why we are targeting them.

Put a Face to Your Customer and User

A *persona* is a representation of the problems, needs, goals, and behavior of a hypothesized group of customers or users. Personas are based on relevant information and insight known to the creators. They are essentially collections of assumptions that must be tested and refined throughout our customer development process.

When creating a persona, remember the following points:

- Define and brainstorm your initial persona very quickly to get alignment across the team.
- Iteratively redefine your persona based on evidence from user research, testing, and feedback during the customer development cycle.
- Continually realign the persona and the business/product vision as the product starts to emerge.

Personas are just a starting point that we use to create a shared understanding of our customers or users. They are never truly objective or empirical; that is not their purpose. We use personas to create empathy with our targeted group's problems and move the conversation from what our own individual preferences may be to what the selected persona would perceive to be valuable—their *Jobs-To-Be-Done*.

Having empathy for customers and users is a powerful force. When we empathize, we enhance our ability to receive and process information.¹¹ Empathy in design requires deliberate practice. We must design experiments and interaction opportunities to connect with our customers and users in meaningful ways and challenge our assumptions, preconceptions, and prejudices. We need to assume the role of an interested inquirer, trying to understand the challenges they experience.

Creating a balance between empathizing with an experience and analyzing the situation allows us to understand our customers' and users' feelings and perspectives. We can then use that understanding to guide our identification of solution hypotheses and commence the experimentation process.

¹¹ IDEO: Empathy on the Edge, <http://bit.ly/1v6ZlPI>

TIP

Go, Look, See

The design company IDEO,¹² famous for creating the original Apple mouse, runs workshops in which teams completely immerse themselves in the context where the envisioned product or service will be used. Their developers read everything of interest about the markets, observe and interview future users, research offerings that will compete with the new product, and synthesize everything they have learned into pictures, models, and diagrams. The result is insights into customers and users that are tested, improved, or abandoned throughout the iterative development process.

At Toyota, *genchi genbutsu* ("go and see") allows leaders to identify existing safety hazards, observe machinery and equipment conditions, ask about the practiced standards to gain knowledge about the work status, and build relationships with employees. The objective of *genchi genbutsu* is to go to the *gemba* (workplace) to understand the value stream and its problems rather than review reports or make superficial comments.

Similarly, *getting out of the building* (a phrase popularized by entrepreneur and author Steve Blank) is a customer development technique to get feedback and focus early product development efforts around the early adopters through frequent qualitative inquiry (including structured interviews) with multiple potential customers.

People who cannot temporarily let go of their role and status, or set aside their own expertise and opinions, will fail to develop empathy with others' conflicting thoughts, experiences, or mental models. The ability to listen and ask the right questions becomes a powerful skill, and the insights it brings are the foundation of effective problem solving and experimentation.

Turning Insights and Data into Unfair Advantage

The ability to discover and leverage critical insights is essential to high-performing organizations. We used to live in a relatively small data universe with high costs associated with collection, storage, and processing of data. The big data movement has provided technologies and techniques for reviewing, processing, and correlating large existing data sets. Organizations can gain additional value from insights into how and why their customers are interacting with their products and solutions. We can detect weak signals that tell us what is working well—or not so well—and use that information to improve existing services or create new offerings. When combined, software, analytics, and data form a key pillar of our organization's intellectual capital.

Access to, and understanding of, existing customers is a significant competitive advantage that established organizations have over startups. Startups face the

¹² IDEO, <http://www.ideo.com>

challenge of gaining market reach and traction due to the lack of access to known customer data. On the other hand, established organizations have existing market and customer data that can be reused and leveraged to unearth new opportunities.

Organizations are now able to ask questions such as, “Why are customers canceling their memberships?” or “How are customers related to one another?”, and run quick and inexpensive experiments to test their hypotheses based on existing data. This is a powerful technique to remove decision bias from our prioritization process and enable data-driven decisions.

Data analytics enables us to invert the discovery process—to look at how customers are using existing services and to do forward projections for new business model, product, or service opportunities.

NOTE

How Companies Mine Data to Discover Your Secrets

In *The Power of Habit* (Random House), Charles Duhigg writes: “Almost every major retailer, from grocery chains to investment banks to the U.S. Postal Service, has a ‘predictive analytics’ department devoted to understanding not just consumers’ shopping habits but also their personal habits, so as to more efficiently market to them.”

Target used this data to a particularly discomfiting effect in order to identify and market to pregnant women. When you’re pregnant, you need to prepare for your new child by buying lots of stuff. Target wanted to encourage pregnant families to do most of their shopping at Target, potentially capturing them as major customers for life. They analyzed their existing customer data to find a way to identify women in their second trimester of pregnancy who could be targeted for offers.

Target was able to identify changes in buying patterns for 25 key products, including nutritional supplements, cotton balls, and unscented lotion, that accurately predicted not only pregnancy but also due date. As a result, they were able to send pregnant women relevant coupons—advertently disguised amongst other vanilla offers so the women wouldn’t realize they were being targeted—to encourage them to do their pre-baby shopping at Target.¹³

Big data is a tool, not a solution. Crucially, it does not replace empathy. We still need human intuition and innovation to improve the problem definition and identify customer and user needs and problems, so as to form hypotheses that can be tested against the data. Cross-functional teams, personas, and user interviews are all powerful tools that enable us to design experiments more effectively and rapidly. We need to learn how to listen and learn from data

13 <http://onforb.es/1v6ZqCZ>

through unbiased analysis—otherwise our data is useless: “Data, like a flashlight, is only as useful as the person wielding it and the person interpreting what it shows.”¹⁴

Using Insight to Inform Hypotheses and Experiments

During Discovery, numerous members of the cross-functional team will have—and should be encouraged to share—interesting and valuable insights into the organization, customers, business, channels, or markets. By sharing these insights with the team, we can generate new perspectives and inspiration for new products or solutions.

Ask those involved in Discovery to share whatever interesting insights and data they have to inform, create, or challenge problem statements based on a number of perspectives, using the canvas shown in [Figure 4-4](#). For example:

Customer

What specific information does the group have about existing customers? What are their usage and engagement behaviors? How can those insights help to shape future opportunities within existing product offers?

Market trends

Industry trends of the market we are attempting to enter are key to understanding how and where opportunities exist—for example, mobile technologies, location-based services, mobile payments. What are the market trends for the product we are creating? How do we measure against them?

Organization

What specific information does the group have about our organization? Where is the organization focusing its efforts? What is the impact of those efforts? How much of the wider competitive landscape does it cover? Where is the organization most effective?

You will not believe!

Every company has individuals that are willing to share interesting and astounding facts about the business or its customer base. How can we test if they are true and/or offer opportunities to create new value propositions as a result?

¹⁴ Scott Berkun, <http://scottberkun.com/2013/danger-of-faith-in-data>

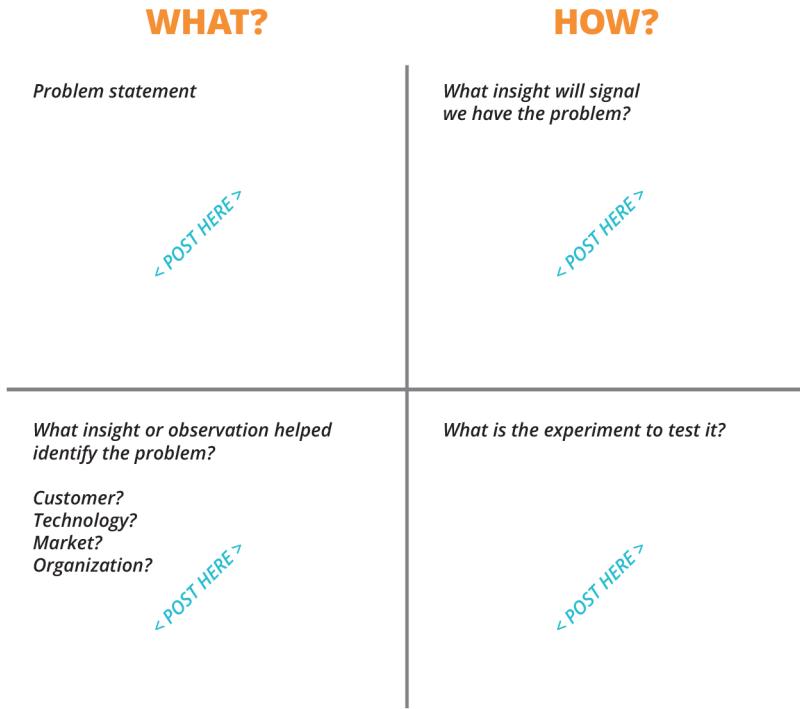


Figure 4-4. The Problem Statement Canvas

By making this information visible and discussing it, we can attempt to identify new business models and value propositions appropriate for the business, given its current constraints and identified problem statements.

Accelerate Experimentation with MVPs

The Lean Startup movement challenges the assumption that customers must have all imaginable features available in a product before they will start to use it. Eric Ries coined the term *minimum viable product* (MVP) to describe a strategy of investing a minimal amount of resources to test the underlying assumptions of our hypotheses with customers. The objective is to eliminate the waste generated by overengineered solutions and accelerate our learning by testing a solution with early customers as soon as possible.

An MVP enables us to use a minimum amount of effort to generate the maximum amount of learning when experimenting with customers. The goal of using an MVP is to execute an experiment that tests the assumptions of our hypotheses as cheaply, quickly, and effectively as possible, in order to learn if

our solution addresses the customer problem we have identified. It eliminates those parts of the solution hypothesis that create unnecessary complexity and consume excessive resources when experimenting with our initial targeted customers. The outcome of the experiment is learning, which enables us to make an evidence-based decision to persevere with our existing business model, pivot to explore a new way to achieve our vision, or stop.

It's important to distinguish between an MVP in Eric Ries' sense and the initial public release of a product, which increasingly takes the form of a public "beta" (Figure 4-5).

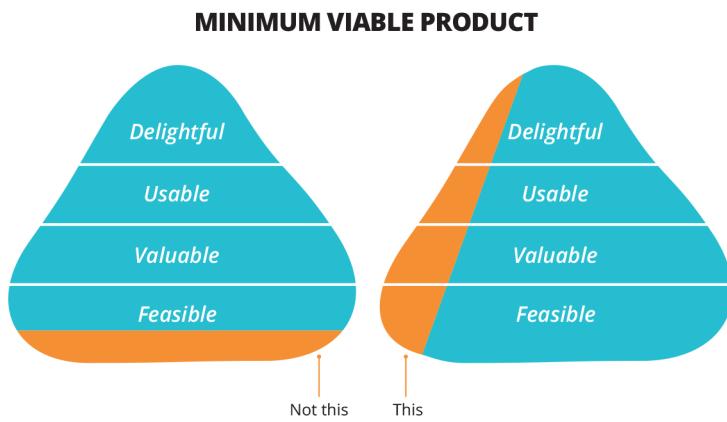


Figure 4-5. Minimum Viable Product: build a slice across instead of one layer at a time¹⁵

Confusingly, people often refer to any validation activity anywhere along on this spectrum as an MVP, overloading the term and understanding of it in the organization or wider industry. Marty Cagan, author of *Inspired: How to Create Products Customers Love* and ex-SVP for eBay,¹⁶ notably uses the term "MVP test" to refer to what Eric Ries calls an MVP. Cagan defines an MVP as "the smallest possible product that has three critical characteristics: people choose to use it or buy it; people can figure out how to use it; and we can deliver it when we need it with the resources available—also known as valuable, usable, and feasible," to which we add "delightful," since design and aesthetics are also as essential for an MVP as for a finished product, as shown in

¹⁵ Diagram inspired by Jussi Pasanen, with acknowledgments to Aarron Walter, Ben Tollady, Ben Rowe, Lexi Thorn, and Senthil Kugalur.

¹⁶ [cagan]

Figure 4-5.¹⁷ Make sure that your team and stakeholders are clear on their definition of MVP.

Should We Build It, Not Can We Build It?

JustGiving is an online fundraising platform that has raised over £2 billion for charities. JustGiving wanted to explore new business models to fund community initiatives that are not necessarily affiliated with a charity.

They formed a small co-located team to rapidly experiment with customers, running sessions with a prototyped version of a crowdfunding platform complete with real community projects that were seeking support. Based on the positive reaction from customers, they proceeded to create a concierge MVP, launching the same trusted community projects with real customers while manually handling back-office tasks such as project setup, payment processing, and collection, to see how the product would perform in the market.

Within seven weeks from the start of the initiative, JustGiving had validated a repeatable business model that they could start to scale into a business in its own right. The product has now become YIMBY¹⁸ with success stories that include purchasing basketball wheelchairs for teams, tools to expand a community garden, and saving the 140-year-old Kettering Town Football Club.

MVPs, as shown in [Table 4-2](#), do not guarantee success; they are designed to test the assumptions of a problem we wish to solve without overinvesting. By far the most likely outcome is that we learn our assumptions were invalid and we need to pivot or stop our approach. Our ultimate goal is to minimize investment when exploring solutions until we are confident we have discovered the *right product*—then, exploit the opportunity by adding further complexity and value to build the *product right*.

Table 4-2. An example set of types of MVPs

Name	What it is	Pros	Cons	Examples
Paper	Throwaway hand-sketched drawings of an interface to use as prototypes, or illustrative examples of a design	Speed, visual, creates shared understanding	Limited interaction, does not test usability or hypothesis	Diagrams, wireframes, sketches

¹⁷ <http://www.svpg.com/minimum-viable-product>

¹⁸ Yes In My Back Yard, <https://www.justgiving.com/yimby>

Name	What it is	Pros	Cons	Examples
Interactive prototype	Clickable, interactive mockup of a prototype or design	Tests design and usability, iterates solutions at speed, uses qualitative customers interviews	Does not test hypothesis or supporting technology	HTML or clickable mockups, videos
Concierge	A personal service instead of a product, which manually guides the customer through a process using the same proposed steps to solve the customer problem in the digital product. The name is derived from hotel concierge.	Reduces complexity, supports generative research, validates assumptions qualitatively with a small investment	Limited scalability, is manual and resource intensive, customer is aware of human involvement	AirBnB founders offering air beds to customers during a Democratic National convention; Collision installation for Stripe ¹⁹
Wizard of Oz	Real working product however behind the scenes all product functions are carried out manually unknown to person using the product	A working solution from customer perspective, a person in the role of the wizard can gain valuable insights from the close involvement; enables evaluative research for price points and validation of value proposition	Limited scalability due to a higher commitment of resources; person in the role of the wizard must appreciate the functionality of the proposed solution; difficult to evaluate systems with a large graphical interface component	Tony Hsieh purchasing shoes for initial customers of Zappos.com

¹⁹ Paul Graham, <http://paulgraham.com/ds.html>

Name	What it is	Pros	Cons	Examples
Micro-niche	Reduce all product features to the bare minimum, socialize and drive paid-for traffic to the product to find out if customers are interested or willing to pay for it	A highly focused test dedicated to any specific topic, takes minimal effort	Needs financial investment to drive traffic, there is competition for keywords and customer click-throughs	http://whatkatewore.com
Working software	Fully functioning working product to address a customer problem, instrumented to measure customer behavior and interactions	Tests hypothesis in a real environment, validates assumptions qualitatively	Expensive, needs investment in people and tools	A/B testing, conversion funnels, referral optimization

How Do Our Vision and MVP Work Together?

Cagan stresses that vision and MVP are intimately related but not the same. Cagan defines vision as the shared understanding that “describes the types of services you intend to provide, and the types of customers you intend to serve, typically over a 2–5 year timeframe.”²⁰ As such, it provides a roadmap and context for MVPs, and we should be prepared to create many MVPs as we search for a repeatable and scalable customer development process aligned with our vision.

Early evangelists, particularly in enterprises, should buy into our entire vision, not just our first MVP experiment. They will need to hear what our organization plans to deliver over the next 6 to 18 months. They are bought into the vision of what we are trying to achieve and so are able to fill in the gaps in our solution as they feel the pain of the problem we are trying to solve. By offering a taste of the solution we are aiming to build, we give them evidence that it works and provide an opportunity for feedback on the solution we are building.

Leveraging business interaction and engagement is very important at the early stages of a new initiative. Feedback and evidence gained through the use of an MVP provide better insight and learning into customer behavior than aggregate measures of success such as total revenue or total transaction value. The

²⁰ Marty Cagan, <http://svpg.com/product-market-fit-vs-product-vision>

MVP allows us to focus on the right thing to build and provides valuable information on how to evolve, adapt, and pivot to meet customer needs discovered through experimenting, as shown in [Figure 4-6](#).

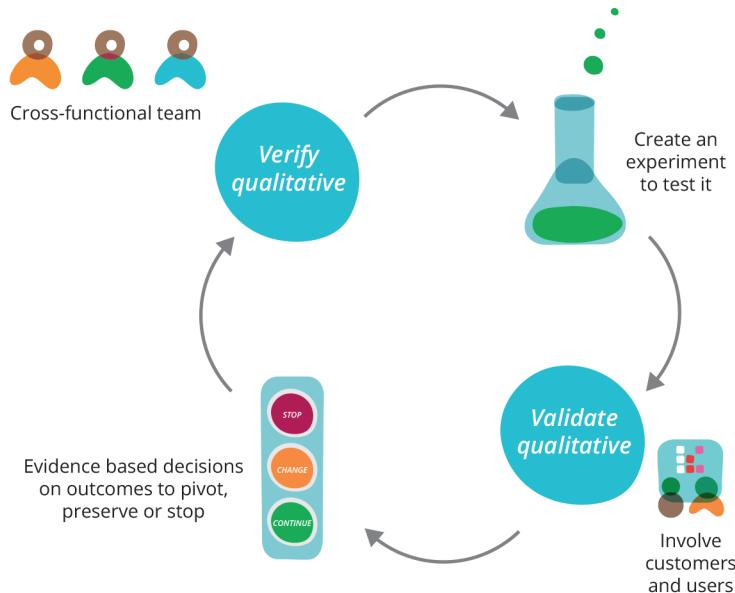


Figure 4-6. *The MVP mindset and experiment evaluation loop*

Starting with the question of what we want to learn from the experiment, we can define how we will observe and measure it, and finally create the cheapest, quickest, and simplest MVP to test our assumptions, measure the effect, and use that learning to formulate next steps.

A fundamental point with new initiatives is preserving cash and iterating rapidly while teams are testing hypotheses to identify a *repeatable* solution. Once those fundamentals are understood, and we achieve a product/market fit, cash preservation becomes less important than spending and we can begin to create a *scalable* solution.

The One Metric That Matters

When designing MVPs to experiment, it is important to identify one key metric that will tell us if the assumptions in our hypothesis are valid. *Lean Analytics* (O'Reilly) authors Alistair Croll and Benjamin Yoskovitz introduced the concept of *One Metric That Matters* (OMTM). OMTM is a single metric that we prioritize as the most important to drive decisions depending on the stage of our product lifecycle and our business model. It is not a single measure that we

will use throughout our product lifetime: it will change over time depending on the problem area we wish to address.

We focus on One Metric That Matters to:

- Answer the most pressing question we have by linking it to the assumptions in the hypothesis we want to test
- Create focus, conversation, and thought to identify problems and stimulate improvement
- Provide transparency and a shared understanding across the team and wider organization
- Support a culture of experimentation by basing it on rates or ratios, not averages or totals, relevant to our historical dataset

It should not be a lagging metric such as return on investment (ROI) or customer churn, both of which measure output after the fact. Lagging indicators become interesting later when we have achieved a product/market fit. By initially focusing on leading metrics, we can get an indication of what is likely to happen—and address a situation quicker to try and change the outcomes going forward. For example, customer complaints are often a leading indicator of churn. If customer complaints are rising, we can expect that customers will leave and the churn will increase. Our OMTM should always evolve as we learn more about the problem we want to solve.

The purpose of the OMTM is to gain objective evidence that the changes we are making to our product are having a measurable impact on the behavior of our customers. Ultimately we are seeking to understand:

- Are we making progress (the *what*)?
- What caused the change (the *why*)?
- How do we improve (the *how*)?

Founder of Intuit Scott Cook says that founders should focus on “love metrics,” for example, how much people love the product, or how often they come back, or how delighted they are in the early stages. “If you’re not getting high activity from the users you already have, it’s time to pivot.” Choosing OMTM provides clarity, alignment, and focus for teams, thus enabling effective decision-making, especially during early stage initiatives.

TIP

Use A3 Thinking as a Systematic Method for Realizing Improvement Opportunities

A3 Thinking is a logical problem-solving tool to capture critical information and define the focus and constraints of the team. Later, it becomes a measure to test our outcomes against. An A3 report (so called because it fits on a piece of A3-size paper), composed of seven elements, embodies the Plan-Do-Check-Act cycle of experimentation:

Background

Capture the critical information to understand the extent and importance of the problem. Tying the background to the goal statement reduces waste by limiting opportunities to focus on wrong areas.

Current condition and problem statement

This is the problem the business stakeholder wants to address, in simple understandable terms and not as a lack-of-solution statement. For example, avoid statements like “Our problem is we need a Content Management System.”

Goal statement

How will we know that our efforts were successful at the end of implementation? Ideally we need one key metric for success. For example, “Our goal is to reduce system failures compared to the previous test results of 22 major issues; our target is to reduce this by 20%.”

Root-cause analysis

Detail the hypothesis and assumptions, or a set of experiments performed to test for cause and effect.

Countermeasures

List the steps of an experiment to be implemented to test the hypothesis.

Check/confirmation effect

Define a method for assessing if the countermeasures have had an effect.

Follow up actions and report

Identify further steps and share what you learned with the team and wider organization.

For more on A3 Thinking, read *Understanding A3 Thinking: A Critical Component of Toyota’s PDCA Management System* by Durward K. Sobek II and Art Smalley.²¹ Other examples include the elevator pitch²² and the Five Ws and One H (Who, What, Where, When, Why, and How).

²¹ [sobek]

²² <http://www.gogamestorm.com/?p=125>

Remember, metrics are meant to hurt—not to make us feel like we are winning. They must be actionable and trigger a change in our behavior or understanding. We need to consider these two key questions when deciding on what our OMTM will be:

What is the problem we are trying to solve?

Product development

Are we attempting to create new products or services that involve customers? How will we know that we are engaging them and they are interested in our product?

Tool selection

Are we attempting to select a tool for using in the organization? How will we know that it is the best tool for the process?

Process improvement

Are we attempting to improve our internal capability and efficiency? How will we know if our changes are having the desired impact?

What stage of the process are we at?

Problem validation

Are we trying to identify that a problem exists by talking to people to see if they are experiencing the pain of the issue we're trying to solve?

Solution validation

Are our people demonstrating alignment and buy-in for the problem we are aiming to solve through qualitative interviews?

MVP validation

Are we creating experiments to quantitatively prove that our solution is working to solve the problem we have identified?

OMTM is a helpful tool for simplifying the complexity of analytics. It specifically tells us if our solution is succeeding or not. Once we have defined the key metric on which to focus, we can identify supporting metrics that provide insight into other areas and support decision-making.

As a good example of OMTM, at LinkedIn, the team does not talk about “total page views” but only “profile views”—the number of people using LinkedIn who search for and find other people, and the number of LinkedIn profiles they viewed.²³

23 <https://medium.com/what-i-learned-building/ab24a585b5ea>

Conclusion

Discovery allows us to safely explore opportunities in conditions of extreme uncertainty—especially in new product development and business model innovation. Discovery concepts and tools let us invest the minimum amount of effort to obtain the maximum amount of learning to make measurable progress towards exploiting validated opportunities. Discovery creates a clear vision and a shared understanding of the problem we are trying to solve within our organization.

We must adopt the mindset in which all our ideas are hypotheses based on assumptions that must be tested, and that most of these assumptions will be proved wrong. By basing our decision-making on information gleaned from fast, inexpensive experiments using MVPs, we can make better investment decisions. The earlier we can pivot or fold on bad ideas, the less time and resources we waste, and the more we can devote to ideas that will deliver value to our customers—or create new ones.

Questions for readers:

- What is your current business hypothesis and how would you create an experiment using an MVP to test it?
- Do you ask “*should* we build it” before pursuing “*can* we build it”?
- What experiments would your team perform and what evidence would they gather to decide when to pivot, persevere, or stop?
- What is your *One Metric That Matters*?

Evaluate the Product/Market Fit

The Edge...there is no honest way to explain it because the only people who really know where it is are the ones who have gone over it.

— Hunter S. Thompson

In this chapter we will discuss how to identify when a product/market fit has been achieved and how to exit the explore stage and start exploiting our product with its identified market. We'll show how to use customized metrics to understand whether we are achieving measurable business outcomes while continuing to solve our customers' problems by engaging them throughout our development process.

We will cover how organizations set themselves up for success with the right strategy, structure, and support, and how they find internal and external customers to provide valuable feedback and insight as they grow their product. We will address how to leverage existing capabilities, services, and practices to scale our product while seeking internal advocates within the organization to collaborate with. Finally, we'll describe the metrics and the engines of growth that help us manage the transition between business model horizons as we begin to scale our solution.

Innovation Accounting

It is not enough to do your best; you must know what to do, and then do your best.

— W. Edwards Deming

We live in a world of data overload, where any argument can find supporting data if we are not careful to validate our assumptions. Finding information to support a theory is never a problem, but testing the theory and then taking the correct *action* is still hard.

As discussed in [Chapter 3](#), the second largest risk to any new product is building the wrong thing. Therefore, it is imperative that we don't overinvest in unproven opportunities by doing the wrong thing the right way. We must begin with confidence that we are actually doing the right thing. How do we test if our intuition is correct, especially when operating in conditions of extreme uncertainty?

Eric Ries introduced the term *innovation accounting* to refer to the rigorous process of defining, experimenting, measuring, and communicating the true progress of innovation for new products, business models, or initiatives. To understand whether our product is valuable and hold ourselves to account, we focus on obtaining admissible evidence and plotting a reasonable trajectory while *exploring* new domains.

Traditional financial accounting measures such as operating performance, cash flow, or profitability indicator ratios like return on investment (ROI)—which are not designed for innovation—often have the effect of stifling or killing new products or initiatives. They are optimized, and more effective, for *exploiting* well-understood domains or established business models and products. By definition, new innovations have a limited operating history, minimal to no revenue, and require investment to start up, as shown in [Figure 5-1](#). In this context, return on investment, financial ratio analysis, cash flow analysis, and similar practices provide little insight into the value of a new innovation nor enable its investment evaluation against the performance of well-established products through financial data comparison alone.

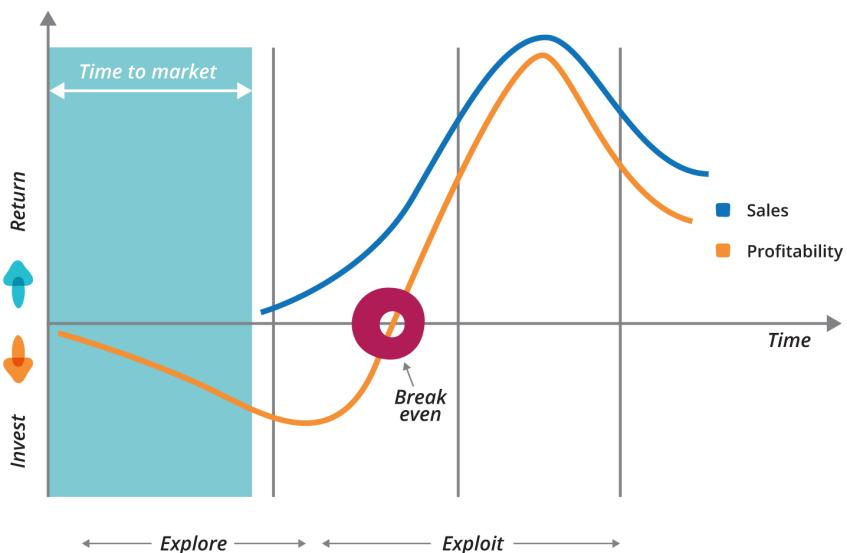


Figure 5-1. Profitability-to-sales ratios for early-stage innovations

When *exploring*, accounting must not be ignored or deemed irrelevant. It simply needs to be interpreted differently to measure the outcomes of innovation and early-stage initiatives. Our principles of accounting and measurement for innovation must address the following goals:

- Establish accountability for decisions and evaluation criteria
- Manage the risks associated with uncertainty
- Signal emerging opportunities and errors
- Provide accurate data for investment analysis and risk management
- Accept that we will, at times, need to move forward with imperfect information
- Identify ways to continuously improve our organization's innovation capability

WARNING

Measurement Fallacy

“What you measure is what you get”—Kaplan and Norton.¹

One of the key ideas of Eric Ries’ *The Lean Startup* is the use of actionable metrics. He advocates that we should invest energy in collecting the metrics that help us make decisions. Unfortunately, often what we tend to see collected and socialized in organizations are *vanity metrics* designed to make us feel good but offering no clear guidance on what *action* to take.

In *Lean Analytics*, Alistair Croll and Benjamin Yoskovitz note, “If you have a piece of data on which you cannot act, it’s a vanity metric...A good metric changes the way you behave. This is by far the most important criterion for a metric: what will you do differently based on changes in the metric?”² Some examples of vanity metrics and corresponding actionable metrics are shown in **Table 5-1.**^{3 4 5}

Table 5-1. Examples of vanity versus actionable metrics

Vanity	Actionable
Number of visits. Is this one person who visits a hundred times, or a hundred people visiting once?	Funnel metrics, cohort analysis. We define the steps of our conversion funnel, then group users and track their usage lifecycle over time.
Time on site, number of pages. These are a poor substitute for actual engagement or activity unless your business is tied to this behavior. They address volumes, but give no indication if customers can find the information they need.	Number of sessions per user. We define an overall evaluation criterion for how long it should take for a session (or action) to complete on the site, then measure how often users perform it successfully.
Emails collected. A big email list of people interested in a new product may be exciting until we know how many will open our emails (and act on what’s inside).	Email action. Send test emails to a number of registered subscribers and see if they do what we tell them to do.

1 “The Balanced Scorecard—Measures That Drive Performance,” p. 70, <http://bit.ly/1vt3X2Q>

2 [croll], p. 13.

3 Ash Maurya, <http://bit.ly/1v6ZG4L>

4 Dan McClure, <http://bit.ly/1vt4925>

5 Ronny Kohavi, <http://bit.ly/1v6ZHpn>

Vanity	Actionable
Number of downloads. While it sometimes affects your ranking in app stores, downloads alone don't lead to real value.	User activations. Identify how many people have downloaded the application and used it. Account creations and referrals provide more evidence of customer engagement.
Tool usage reflects the level of standardization and reuse in the enterprise tool chain.	Tooling effect is the cycle time from check-in to release in production for a new line of code.
Number of trained people counts those who have been through Kanban training and successfully obtained certification.	Higher throughput measures that high-value work gets completed faster leading to increased customer satisfaction.

In *How to Measure Anything*, Douglas Hubbard recommends a good technique for deciding on a given measure: “If you can define the outcome you really want, give examples of it, and identify how those consequences are observable, then you can design measurements that will measure the outcomes that matter. The problem is that, if anything, managers were simply measuring what seemed simplest to measure (i.e., just what they currently knew how to measure), not what mattered most.”⁶

By combining the principle of actionable metrics with Hubbard’s recommendation for how to create the measures that matter most, we can go beyond traditional internal efficiency and financial measurement to focus on value from the perspective of the stakeholders that matter most—our customers.

Dan McClure’s “pirate metrics”⁷ are an elegant way to model any service-oriented business, as shown in Table 5-2 (we have followed Ash Maurya in putting revenue before referral). Note that in order to use pirate metrics effectively, we must always measure them by *cohort*. A cohort is a group of people who share a common characteristic—typically, the date they first used your service. Thus when displaying funnel metrics like McClure’s, we filter out results that aren’t part of the cohort we care about.

⁶ [hubbard], p. 37.

⁷ Pirate Metrics, <http://slidesha.re/1v6ZL8B>

Table 5-2. Pirate metrics: AARRR!

Name	Purpose
Acquisition	Number of people who visit your service
Activation	Number of people who have a good initial experience
Retention	Number of people who come back for more
Revenue	Number of people from the cohort who engage in revenue-creating activity
Referral	Number of people from the cohort who refer other users

Measuring pirate metrics for each cohort allows you to measure the effect of changes to your product or business model, if you are pivoting. Activation and retention are the metrics you care about for your problem/solution fit. Revenue, retention, and referral are examples of *love metrics*—the kind of thing you care about for evaluating a product/market fit.⁸ In [Table 5-3](#) we reproduce the effect on pirate metrics of both incremental change and pivoting for Votizen’s product.⁹ Note that the order and meaning of the metrics are subtly different from [Table 5-2](#). It’s important to choose metrics suitable for your product (particularly if it’s not a service). Stick to actionable ones!

Table 5-3. Effect of incremental change and pivots on Votizen’s pirate metrics

Metric	Interpretation	v.1	v.1.1	v.2	v.3	v.4
Acquisition	Created account	5%	17%	42%	43%	51%
Activation	Certified authenticity	17%	90%	83%	85%	92%
Referrals	Forwarded to friends	—	4%	54%	52%	64%
Retention	Used system at least thrice	—	5%	21%	24%	28%
Revenue	Supported causes	—	—	1%	0%	11%

In order to determine a product/market fit, we will also need to gather other business metrics, such as those shown in [Table 5-4](#). As always, it’s important

⁸ Ash Maurya has a good blog post on pirate metrics, cohorts, and problem/solution fits: <http://bit.ly/1v6ZG4L>.

⁹ By David Binetti, <http://slidesha.re/1v6ZQZZ>

not to aim for unnecessary precision when gathering these metrics. Many of these growth metrics should be measured on a per-cohort basis, even if it's just by week.

Table 5-4. Horizon 3 growth metrics

Measure	Purpose	Example calculation
Customer acquisition cost	How much does it cost to acquire a new customer or user?	Total sales and marketing expenses divided by number of customers or users acquired
Viral coefficient (K)	A quantitative measure of the virality of a product	Average number of invitations each user sends multiplied by conversion rate of each invitation
Customer lifetime value (CLV)	Predicts the total <i>net</i> profit we will receive from a customer	The present value of the future cash flows attributed to the customer during his/her entire relationship with the company ¹⁰
Monthly burn rate	The amount of money required to run the team, a runway for how long we can operate	Total cost of personnel and resources consumed

Which metrics we care about at any given time will depend on the nature of our business model and which assumptions we are trying to validate. We can combine the metrics we care about into a scorecard, as shown in Figure 5-2.¹¹

Customer success metrics provide insight into whether customers believe our product to be valuable. Business metrics, on the other hand, focus on the success of our own business model. As we noted before, collecting data is never an issue for new initiatives; the difficulties lie in getting actionable ones, achieving the right level of precision, and not getting lost in all the noise.

To help us improve, our dashboard should only show metrics that will trigger a change in behavior, are customer focused, and present targets for improvement. If we are not inspired to take action based on the information on our dashboard, we are measuring the wrong thing, or have not drilled down enough to the appropriate level of actionable data.

¹⁰ The standard definition of CLV and many other sales and marketing metrics are given in [farris].

¹¹ Thanks to Aaron Severs, founder of hirefrederick.com, for inspiration and permission to use this diagram.

Stakeholder	Measure	Current	Target	Trend
Customer	% users that complete sales flow	30%	45%	
	% retention	20%	25%	
	Net Promoter Score	44	60	
Business	% visits to sign up for service	20%	25%	
	% conversion to paying customers	15%	20%	
	Customer acquisition costs	\$0.25	\$0.05	
	Life time customer vale	\$0.30	\$0.80	
	% attrition	30%	15%	

Figure 5-2. Example innovation scorecard

In terms of governance, the most important thing to do is have a regular weekly or fortnightly meeting which includes the product and engineering leads within the team, along with some key stakeholders from outside the team (such as a leader in charge of the Horizon 3 portfolio and its senior product and engineering representatives). During the meeting we will assess the state of the chosen metrics, and perhaps update on which metrics we choose to focus on (including the One Metric That Matters). The goal of the meeting is to decide whether the team should persevere or pivot, and ultimately to decide if the team has discovered a product/market fit—or, indeed, if it should stop and focus on something more valuable. Stakeholders outside the team need to ask tough questions in order to keep the team honest about its progress.

Energizing Internal Advocates in the Enterprise

Innovation in large, bureaucratic organizations is challenging because they are inherently designed to support stability, compliance, and precedence over risk taking. Leaders that have risen to the top could do it because they have worked the system as it has existed to date. Therefore, we need to be careful that any critiques do not become focused on individuals or their behavior within the system. We need to seek out collaborators and co-creators across the organization without causing alienation, to gain further support for our efforts, and to cross the chasm to the next stage of the adoption curve within the organization. Ultimately, we will need to identify change agents in the areas where we need change to be successful. The best ammunition here is *demonstrable evidence* that our efforts are achieving measurable business outcomes.

Without doubt there are people in our organization who are frustrated and curious for change. However, they seek safety, context, and cover to act before they are willing to become champions of an initiative. Energizing and engaging these people is key. As they become early adopters of our ideas and initiatives, they will provide a feedback loop enabling us to iterate and improve our product. They are also our sponsors within the wider organization. In bureaucratic environments, people tend to protect their personal brand and not back the losing horse. Our goal is to give them the confidence, resources, and evidence that encourages them to be advocates for our initiative throughout the organization.

Do Things That Don't Scale

Even when we have validated the most risky assumptions of our business model, it is important that we continue to focus on the same principles of simplicity and experimentation. We must continue to optimize for learning and not fall into simply delivering features. The temptation, once we achieve traction, is to seek to automate, implement, and scale everything identified as “requirements” to grow our solution. However, this should not be our focus.

In the early stages, we must spend less time worrying about growth and focus on significant customer interaction. We may go so as far as to only acquire customers individually—too many customers too early can lead to a lack of focus and slow us down. We need to focus on finding passionate early adopters to continue to experiment and learn with. Then, we seek to engage similar customer segments to eventually “cross the chasm” to wider customer acquisition and adoption.

This is counterintuitive to the majority of initiatives in organizations. We are programmed to aim for explosive growth, and doing things that don’t scale doesn’t fit with what we have been trained to do. Also, we tend to measure our required level of service, expenses, and success in relation to the revenue, size, and scope of more mature products in our environment or competitive domain.

We must remember that we are still in the *formative* stage of our discovery process, and don’t want to overinvest and commit to a solution too early. We continually test and validate the assumptions from our business model through market experiments at every step. If we have identified one key customer with a problem and can act on that need, we have a viable opportunity to build something many people want. We don’t need to engage every department, customer segment, or market to start. We just need a focused customer to co-create with.

Once leaders see evidence of rampant growth with us operating with unscalable processes, we’ll easily be able to secure people, funding, and support to

build robust solutions to handle the flow of demand. Our goal should be to create a *pull* system for customers that want our product, service, or tools, not *push* a mandated, planned, and baked solution upon people that we must “sell” or require them to use.

Customer Intimacy

By deliberately narrowing our market to prioritize quality of engagement and feedback from customers, we can build intimacy, relationships, and loyalty with our early adopters. People like to feel part of something unique and special.

Developing Empathy with Customers: Sometimes the Answer Is Inside the Building

The Royal Pharmaceutical Society knew that their clinical drug database was the best in the world. They also knew that there must be many more uses for it than just a stack of printed books. But where should they start? Instead of guessing, or building an expensive platform for products, or trying to sign a deal without a product, they used their other major asset: a building full of pharmacists. Through rapid prototyping, user testing with pharmacists working for the society, and product research with nearby pharmacies, they were quickly able to focus on an app to check for potential interactions between prescribed drugs. There are huge opportunities in licensing the data for international use. By starting with an app that they themselves would use, they were able to understand what international customers might want and to build a great marketing tool.

By keeping our initial customer base small—not chasing vanity numbers to get too big too fast—we force ourselves to keep it simple and maintain close contact with our customers every step of the way. This allows teams more time with customers to listen, build trust, and ensure early adopters that we’re ready to help. Remember, reaching big numbers is not a big win; meeting unmet needs and delighting customers is.

Build a Runway of Questions, Not Requirements

The instinct of product teams, once a problem or solution validation is achieved, is to start building all the requirements for a scalable, fully functioning, and complete solution based on the gaps in their MVPs. The danger with this approach is that it prevents us from evolving the product based on feedback from customers.

In the early stage we are still learning, not earning. Therefore it is important that we do not limit our options by committing time, people, and investment to building features that may not produce the desired customer outcomes. We

must accept that everything is an assumption to be tested, continually seek to identify our area of most uncertainty, and formulate experiments to learn more. To hedge our bets with this approach, leverage things that don't scale—build a runway with scenarios for how we may continue to build out our product.

Our runway should be a list of hypotheses to test, *not* a list of requirements to build. When we reward our teams for their ability to deliver requirements, it's easy to rapidly bloat our products with unnecessary features—leading to increased complexity, higher maintenance costs, and limited ability to change. Features delivered are not a measure of success, *business outcomes* are. Our runway is a series of questions that we need to test to reduce uncertainty and improve our understanding of growth opportunities.

Create a Story Map to Tell the Narrative of the Runway of Our Vision

Story maps are tool developed by Jeff Patton, explained in his book, *User Story Mapping*. As Patton states, "Your software has a backbone and a skeleton—and your map shows it."

Story maps help with planning and prioritizing by visualizing the solution as a whole (see [Figure 5-3](#)). Story mapping is not designed to generate stories or create a release plan—it is about understanding customers' objectives and jobs-to-be-done. Story maps provide an effective means to communicate the narrative of our solution to engage the team and wider stakeholders and get their feedback. By going through story maps and telling the story of the solution, we ensure that we have not missed any major components. At the same time, we maximize learning by identifying the next riskiest hypothesis to test while minimizing waste and overengineered solutions that do not fit customer needs as defined in our MVP.

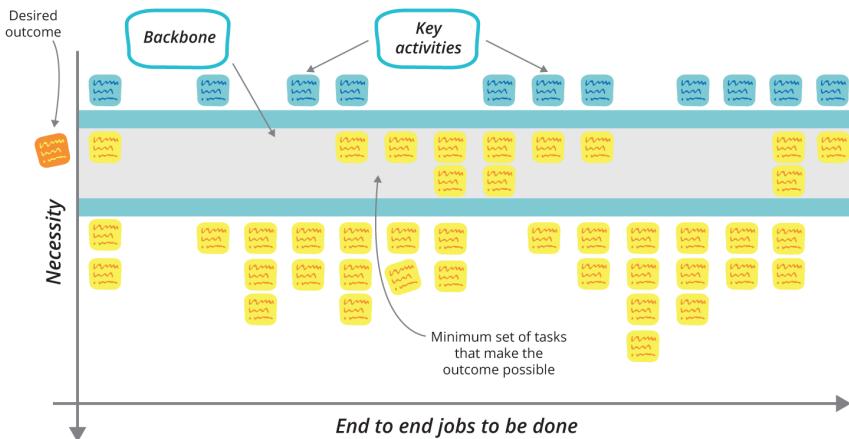


Figure 5-3. A user story map

When we start to harden, integrate, and automate our product, it impacts our ability to rapidly adapt to what we are discovering, often limiting our responsiveness and ability to change. Within Horizon 3, we must continuously work to avoid product bloat by leveraging existing services, capabilities, or manual processes to deliver value to users. Our aim is not to remove ourselves from users. We want to ensure that we are constantly interacting. If we optimize only for building without constantly testing our assumptions with our customers, we can miss key pain points, experiences, and successes—and that is often where the real insights are.

If we want to learn, we must have empathy for our users and experience their pain. When we find a customer with a problem that we can solve manually, we do so for as long as possible. When our customers' quality of service is compromised or we cannot handle the level of demand, we consider introducing features to address the bottlenecks that have emerged through increased use of the product.

NOTE

Leverage Frugal Innovation

Unscalable techniques and practices are not only a necessity—they can be a catalyst for change in an organization's culture. Proving it is possible to test our ideas quickly, cheaply, and safely gives others in the organization encouragement and confidence that experimentation is possible, the result being a lasting change for the better in our culture.

Engineering Practices for Exploring

In general, we favor the principles of The Toyota Production System by “building quality into” software, discussed at length in [Chapter 8](#). However, when exploring, there is a tension between the need to experiment by building MVPs, and building at high levels of quality through practices such as test automation.

When we start working on validating a new product idea or a new feature in an existing product, we want to try out as many ideas as fast as possible. Ideally we will do this without writing any software at all. But for the software we do write, we don’t want to spend a ton of time building acceptance tests and refactoring our code. We will (as Martin Fowler puts it) deliberately and prudently accumulate technical debt in order to run experiments and get validation.¹²

However, if our product is successful, we will hit a brick wall with this approach. Perhaps a year or two in (depending on our pain threshold), changes will become onerous and time consuming and the product will become infested with defects and suffer from poor performance. We may even get to the stage where we consider a Big Rewrite.

Our advice is this. There are two practices that should be adhered to from the beginning that will allow us to pay down technical debt later on: continuous integration and a small number of basic unit and user-journey tests. The moment a product (if we are in Horizon 3) or feature (in Horizon 2) goes from being an experiment to validated, we need to start aggressively paying down technical debt. Typically that means adding more user-journey tests, employing good architectural practices such as modularization, and making sure all new code written on the feature uses test-driven development (good engineers will already use TDD).

Having forced ourselves to do something that should be unnatural to engineers—hack out embarrassingly crappy code and get out of the building to get validation from early on—we must then pull the lever hard in the other direction, kill the momentum, and transition our focus from building the right thing to building the thing right. Needless to say, this requires extreme discipline.

Choosing at what point in the lifecycle of our product or feature to pay down our technical debt is an art. If you find (as many do) that you’ve gone too far down the path of accumulating technical debt, consider the alternatives to the Big Rewrite described in [Chapter 10](#).

¹² <http://martinfowler.com/bliki/TechnicalDebtQuadrant.html>

Engines of Growth

In *The Lean Startup*, Eric Ries argues that there are three key strategies for growth—choose one:

Viral

Includes any product that causes new customers to sign up as a necessary side effect of existing customers' normal usage: Facebook, MySpace, AIM/ICQ, Hotmail, Paypal. Key metrics are acquisition and referral, combined into the now-famous viral coefficient.

Pay

Is when we use a fraction of the lifetime value of each customer and flow that back into paid acquisition through search engine marketing, banner ads, public relations, affiliates, etc. The spread between your customer lifetime value and blended customer acquisition cost determines either your profitability or your rate of growth, and a high valuation depends on balancing these two factors. Retention is the key goal in this model. Examples are Amazon and Netflix.

Sticky

Means something causes customers to become addicted to the product, and no matter how we acquire a new customer, we tend to keep them. The metric for sticky is the “churn rate”—the fraction of customers in any period who fail to remain engaged with our product or service. This can lead to exponential growth. For eBay, stickiness is the result of the incredible network effects of their business.

For enterprises, however, there are further growth options to consider:

Expand

Is building an adaptive initial business model that we could simply evolve and expand further by opening up new geographies, categories, and adjacencies. Amazon has executed this strategy excellently, moving from selling books to an e-commerce store offering new retail categories. With this growth strategy, the initial targeted market should be large enough to support multiple phases of growth over time.

Platform

Once we have a successful core product, we transform it into a platform around which an “ecosystem” of complementary products and services is developed by both internal and external providers. Microsoft did this with Windows by creating MS Office, Money, and other support packages, including those developed by external vendors. Other platform examples include Apple’s AppStore, Salesforce’s Force.com, and Amazon’s Marketplace and Web Services offerings.

Great products, tools, and practices, both internal and external, have always spread by word of mouth due to their truly compelling value proposition and a brand that customers are proud to advocate. If our growth is derived from our customers, then it will happen without us having to invest. If not, we will be limited by the effort required to manually discover, convert, and service our customers.

Ultimately, our product is the key driver of growth. If we build a truly compelling solution that addresses a customer need and that they really love, they will use it. More impressively, they will become advocates and encourage others to use it—creating the best sales team we could wish to hope for to enable success.

Transitioning Between Horizons to Grow and Transform

In [Chapter 2](#), we mentioned that organizations must manage all three horizons concurrently. The ability to recognize, transition, and convert initiatives through these cycles, as shown in [Figure 5-4](#), holds the key for the future success, relevance, and longevity of the organization.

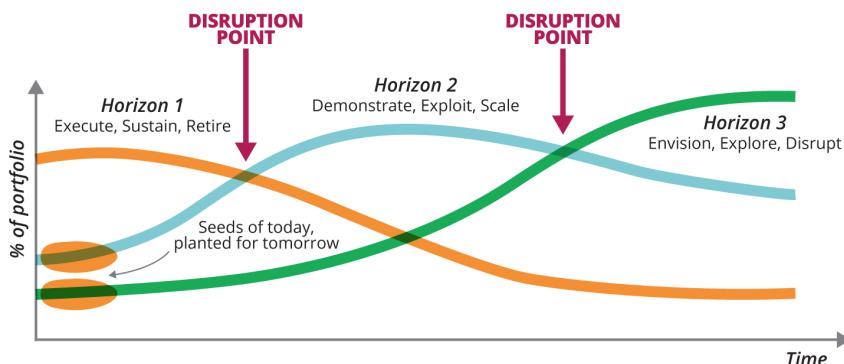


Figure 5-4. Percentage of product portfolio for the three innovation horizons over time

As we describe in [Chapter 3](#), it is Horizons 2 and 3 that need the most leadership support. These horizons contain much more uncertainty and lower revenue, so they can be easily crushed if not managed independently of Horizon 1.

We must be aware of the pitfalls at each stage, including transitioning at a wrong time and selecting a wrong strategy for each horizon.

Lean Development and Lean Operations, by Steve Bell

Lean thinking is usually associated with operations, as it originated with the Toyota Production System (TPS) and has been widely adopted in manufacturing settings.¹³ But Toyota's long-term success is equally due to their application of lean principles to quickly and efficiently develop desirable new products of high quality and a reasonable cost. Toyota has demonstrated that an enterprise that adopts lean thinking seamlessly across development *and* operations can gain a lasting competitive advantage.

Lean development and operations are interrelated and complementary, but very different in nature. *Lean operations* emphasizes standardization and reduction of waste, uncertainty, and variation in order to create efficient processes that produce consistent, quality products. In contrast, *lean development* utilizes uncertainty and variation early in the design process to learn from experiments, especially from failures—which is the most effective way to solve problems and drive innovation.

Yet here is the paradox: lean development, which requires variation and uncertainty, relies on standardized work methods to formulate hypothesis for innovation and run consistent and repeatable experiments that minimize waste and time, while maximizing creativity and value.

For example, lean development rigorously and continuously engages the Voice of the Customer at the *gemba* (the place, both physical and virtual, where the work is done) utilizing iterative, often set-based design to accelerate learning and quickly create a product with a valid *value hypothesis*. Other standardized lean practices like A3 problem solving, visual management, and value stream mapping are useful in a development setting, improving speed to market while reducing R&D cost and enterprise risk.

Once there is a viable new product or service, the enterprise may utilize its lean operations capabilities to quickly and efficiently bring it to market, validating the *growth hypothesis*. This is where many lean startups either lose the market to fast followers or are acquired by larger enterprises with lean operations capability to rapidly reach the market, exploit early profits, and achieve brand domination. While acquiring startups as a source of innovation is certainly a viable strategy for larger enterprises, most would also like to enhance their internal innovation capability as well.

Lean development thus creates innovative products and services that flow through lean operations and into the hands of the customer as a continuous value stream, in the same spirit as continuous delivery (DevOps) does in a software context. When an enterprise is able to integrate and exploit this rapid flow of ideas to value, profits from mature products can fund continuous innovation, creating a virtuous cycle illustrated in [Figure 5-5](#).

¹³ Although Lean originated in manufacturing, it has evolved into most sectors, including health-care, financial services, transportation, construction, education, and the public sector.

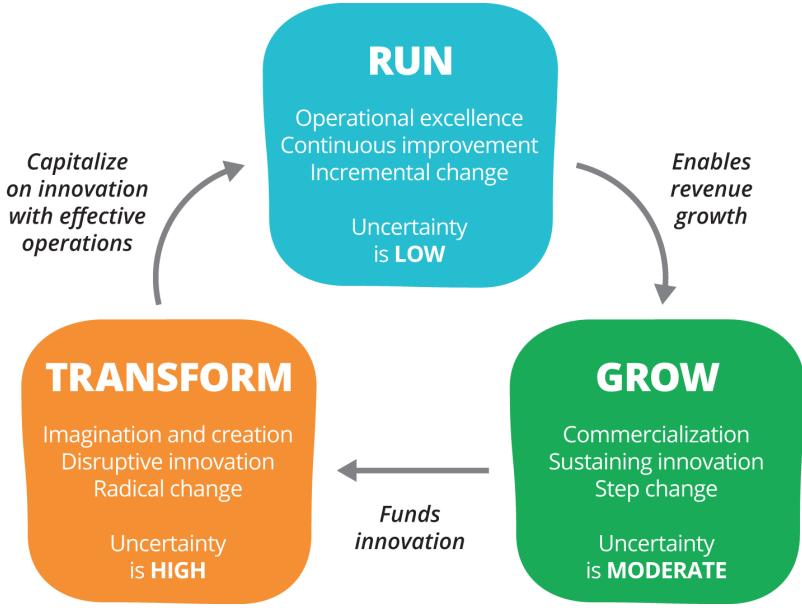


Figure 5-5. The virtuous cycle of innovation, by Steve Bell

When attempting to cross Horizon 3, indicators of customer satisfaction and continued engagement are important signals to monitor for future growth. Once we have found customers, learned how to address their needs, and are confident of meeting their demand, we should seek to expand the customer market by geography, channel, or offerings.

While exploring, we are testing a fit between product and market, typically through bespoke solutions for our initial customers. Exploiting is about finding an offering and business model that appeals to a broader customer set.

The five critical enablers of growth when transitioning from explore to exploit are:

Market

It is imperative to select the right market. Ideally, there are plenty of potential customers that will support our growth aspirations; we must identify the elements that made us successful with our early adopters, and then seek to find similar but larger groups to engage with. The insights we have learned by working with our early adopters are key to informing this decision. Early adopters are also likely to spread their experiences with our

product by word of mouth, eventually driving the product to “cross the chasm” to wider adoption.

Monetization model

We must decide what is the best way to capture the value created by our offering, as it essentially defines what will drive revenue for our business model. It is also difficult to change later.

Customer adoption

How will we get customers on board with the product? We must be careful not to make major product or pricing concessions to any individual group to win over a large account. We must remain true to our product vision and manage the tension and demands of any single customer group that could limit further growth.

Forget “big bang” launches

Play it safe: continue to test and validate the product, contain the fire, work with smaller samples of customers. Build momentum through alpha and beta product launches with targeted customer segments. As we gain more confidence, understanding, and success, we broaden our customer base. Ideally, we want customers to come to us with problems to solve so we don’t have to push new products on them.

Team engagement

We must do all we can to keep the team together to protect culture, learning speed, and acquired knowledge. We do not want to build a wall between innovation and operation teams. Collaboration, directed towards organization learning and development, is key to making an innovation culture stick as we start to scale and hire new team members.

When considering process improvements and tool selection, similar principles apply to identifying target users, evaluation and capture of benefit, user adoption, avoiding “big bang” rollouts, and team engagement.

Innovation Takes Time: From Auction to Marketplace

Amazon auctions (later known as zShops) were launched in March 1999 in response to the success of eBay. The site was promoted heavily from the home, category, and individual product pages. Despite the promotion, one year after launch it had only achieved a 3.2% share of the online auction market compared to 58% for eBay, and subsequently declined.

In November 2000, zShops was renamed to “Amazon Marketplace,” offering competitive prices on products available through third-party sellers alongside the standard product listings. The strategy, initially driven by the need to compete with eBay, was adjusted to align to Amazon’s strategic focus on low pricing.

Extending the model further, Amazon introduced selling used products through the seller marketplace, providing another revenue stream without any impact to its supply chain. Advertising, packing, and shipping are handled exclusively by sellers, with Amazon taking a cut of the transaction for providing the sales channel with minimal cost.

In 2012, Amazon's Marketplace service produced 12% of revenues¹⁴ with total unit sales increasing 32% from the previous year.¹⁵

By reconsidering how we define and measure validated learning, we can start to test and communicate if and when our initiatives are getting traction. By continually experimenting with our customers and moving our *One Metric That Matters* as cheaply and quickly as possible, we can limit our investment, reduce associated risks, and maximize learning. An evidence-based approach to product development provides safety, context, and cover to act for stakeholders—and is a catalyst for change in the larger organization.

Conclusion

Innovation accounting provides a framework to measure progress in the context of Horizon 3—that is, under conditions of extreme uncertainty. It is designed to gather leading indicators of the future growth of the idea, so that we can eliminate those that will not succeed in Horizon 2.

We have identified the three key areas to consider during this stage. First, we must find customers to act as co-creators of value. We use their feedback to experiment and refine our value proposition before aiming for a wider market. Second, we focus on learning rather than revenue by taking a narrow customer focus and validating each assumption our solution makes. We do not need to build requirements; we need to answer questions about the desired functionality of our product. Finally, we focus on user engagement over quick financial gain—with more satisfied users there will come revenue (or whatever value we hope to create for our organization). As we improve our understanding of our users and the product opportunity, we can decide on a monetization model to ensure the ongoing success for the product.

Most ideas will not achieve a product/market fit. For those that do, a metamorphosis is required. The behaviors and management principles required to succeed in Horizon 2 are fundamentally different from those that govern Horizon 3. **Part III** presents how to grow an organization focused on building the product right, now that we have confidence that we are building the right product.

14 <http://bit.ly/1v700QY>

15 <http://bit.ly/1v701og>

Questions for readers:

- What customer and business metrics would be on your innovation scorecard?
- Who are the key stakeholders, and what is their influence for each stage of the adoption curve for your product? How do you plan to engage them and create alignment?
- What experiments do you plan to run to test and validate your business model hypothesis with customers? How will you visualize and prioritize them?
- How can you gather data to test with your identified market as cheaply and quickly as possible?
- What are your criteria for moving a product from Horizon 3 to Horizon 2?

PART III

EXPLOIT

Prediction is difficult, especially about the future.

— Niels Bohr

In [Part II](#), we showed how to explore new opportunities—whether potential products or internal tools and services. In this part, we discuss how to exploit validated ideas. As discussed in [Chapter 2](#), these two domains demand a completely different approach for management and execution. However, both are necessary—and indeed complementary—if we are to effectively balance our enterprise portfolio and adapt to a constantly changing business environment.

We hope you are reading this part because you have successfully exited the explore domain—but it's just as likely that you are here because you participate in a large program of work in an enterprise which has been set up in the traditional way. Thus, this part of the book primarily describes how to change the way we lead and manage such large-scale programs of work in a way that empowers employees and dramatically increases the rate at which we can deliver valuable, high-quality products to customers. But before we can begin, we must understand our current condition.

In an enterprise context, planned work is usually prioritized through a centralized or departmental planning and budgeting process. Approved projects then go through the development process before going live or being released to manufacturing. Even in organizations which have adopted “agile” development methods, the value stream required to deliver a project often resembles

Figure III-1, which we describe as “water-scrum-fall.”¹ In cases where one or more of these phases are outsourced, we must also go through a procurement process before we can proceed to the design and development phases following approval. Because this process is so onerous, we tend to batch up work, creating large programs which further exacerbate the problems with the project paradigm.

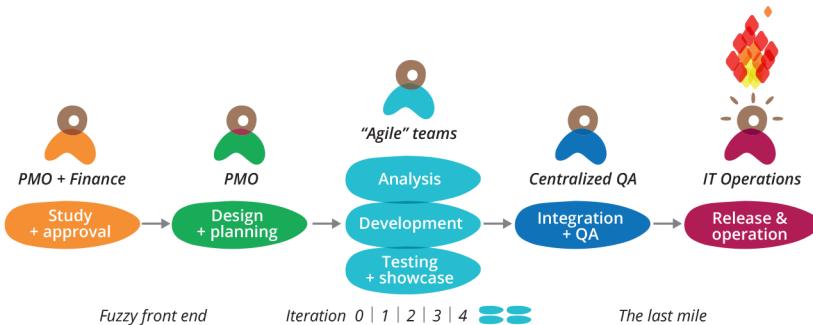


Figure III-1. Water-scrum-fall

This project-based paradigm for carrying out software development at scale has its origins in the post-WWII United States military-industrial complex, where software was crucial for building a new generation of airplanes, missile systems, and spacecraft that had essentially one customer: the US government. It’s no coincidence that the term “software engineering” was coined at a 1968 NATO conference which was convened to work out how to formalize large-scale software development.²

The traditional centralized phase-gate project paradigm was designed in a simpler era. Products could not deliver value until they were fully manufactured, they didn’t need to change substantially over the course of their lifecycle, and we had a high level of confidence that we would not need to change the specification significantly in response to new information discovered in the course of building the product.

None of these criteria apply to software-based systems today, and the power of software derives from the fact that it’s cheap to prototype and change. In particular, since we are so frequently wrong about what users of our products and systems will find valuable, planning out big programs of work months in

¹ The term “water-scrum-fall” was coined by Forrester Research. A *value stream* is defined as “the sequence of activities an organization undertakes to deliver on a customer request” [martin], p. 2. We cover value streams in [Chapter 7](#).

² <http://homepages.cs.ncl.ac.uk/brian.randell/NATO>

advance leads to an enormous amount of waste and bad blood. Instead of trying to get better at predicting the future, we should improve our ability to adapt rapidly and effectively to new information.³

The modern, *lean-agile* paradigm we present for running large-scale programs of work discussed in this part is the result of working with and studying a number of organizations that need to get software development off the critical path. They want to move fast at scale, detect weak signals in the market, and exploit them rapidly. This is what allows them to provide better customer service, to reduce the cost of creating and evolving products, and to increase quality and stability of their services.

There are several frameworks that deal with scaling agile software development methods. In general, these frameworks take small teams practicing Scrum and add more structures on top to coordinate their work. However, these teams are still embedded within a phase-gate program and portfolio management process that is more or less unchanged from the traditional project management paradigm. They still apply top-down thinking and tend to batch up work into releases with long cycle times, thus limiting the use of the information collected to guide future decisions. Our approach differs in several important respects from these frameworks, as well as from more traditional phase-gate frameworks.

The most important difference is that instead of presenting a particular set of processes and practices to implement, we focus on implementing continuous improvement at the senior leadership level to drive the *evolution* of your organization and the processes you use. Continuous improvement cannot be at the edges of our “big diagram”: we put it front and center. This reflects the fact that there is no one-size-fits-all solution and that every organization faces a different set of circumstances. Every organization will take its own path to address changes, aligned to its own business objectives; to create lasting results, we must enable teams to try things out and learn what works and what doesn’t for themselves.

In the following chapters, we will present the following principles for lean-agile product development at scale:

- Implement an iterative continuous improvement process at the leadership level with concise, clearly specified outcomes to create alignment at scale, following the Principle of Mission.
- Work scientifically towards challenging goals, which will lead you to identifying and removing—or avoiding—no-value-add activity.

³ This is a key claim of Nassim Taleb’s body of work.

- Use continuous delivery to reduce the risk of releases, decrease cycle time, and make it economic to work in small batches.
- Evolve an architecture that supports loosely coupled customer-facing teams which have autonomy in *how* they work to achieve the program-level outcomes.
- Reduce batch sizes and take an experimental approach to the product development process.
- Increase and amplify feedback loops to make smaller, more frequent decisions based on the information we learn from performing our work to maximize customer value.

We'll also provide several examples of enterprises that have leveraged these principles to create a lasting competitive advantage, and describe how they transformed themselves in the process.

Deploy Continuous Improvement

The paradox is that when managers focus on productivity, long-term improvements are rarely made. On the other hand, when managers focus on quality, productivity improves continuously.

— John Seddon

In most enterprises, there is a distinction between the people who build and run software systems (often referred to as “IT”) and those who decide what the software should do and make the investment decisions (often called “the business”). These names are relics of a bygone age in which IT was considered a cost necessary to improve efficiencies of the business, not a creator of value for external customers by building products and services. These names and the functional separation have stuck in many organizations (as has the relationship between them, and the mindset that often goes with the relationship). Ultimately, we aim to remove this distinction. In high-performance organizations today, people who design, build, and run software-based products are an integral part of business; they are given—and accept—responsibility for customer outcomes. But getting to this state is hard, and it’s all too easy to slip back into the old ways of doing things.

Achieving high performance in organizations that treat software as a strategic advantage relies on *alignment* between the IT function and the rest of the organization, along with the ability of IT to *execute*. It pays off. In a report for the *MIT Sloan Management Review*, “Avoiding the Alignment Trap in Information Technology,” the authors surveyed 452 companies and discovered that

high performers (7% of the total) spent a little less than average on IT while achieving substantially higher rates of revenue growth.¹

However, *how* you move from low performance to high performance matters. Companies with poor alignment and ineffective IT have a choice. Should they pursue alignment first, or try to improve their ability to execute? The data shows that companies whose IT capabilities were poor achieve worse results when they pursue alignment with business priorities before execution, even when they put significant additional investment into aligned work. In contrast, companies whose engineering teams do a good job of delivering their work on schedule and simplifying their systems achieve better business results with much lower cost bases, even if their IT investments aren't aligned with business priorities.

The researchers concluded that to achieve high performance, companies that rely on software should focus first and foremost on their ability to execute, build reliable systems, and work to continually reduce complexity. Only then will pursuing alignment with business priorities pay off.

However, in every team we are always balancing the work we do to improve our capability against delivery work that provides value to customers. In order to do this effectively, it's essential to manage both kinds of work at the program and value stream levels. In this chapter we describe how to achieve this by putting in place a framework called *Improvement Kata*. This is the first step we must take to drive continuous improvement in our execution of large scale programs. Once we have achieved this, we can use the tools in the following chapters to identify and remove no-value-add activity in our product development process.

The HP LaserJet Firmware Case Study

We will begin with a case study from the HP LaserJet Firmware team, which faced a problem with both alignment and execution.² As the name suggests, this was a team working on embedded software, whose customers have no desire to receive software updates frequently. However, it provides an excellent example of how the principles described in the rest of **Part III** work at scale in a distributed team, as well as of the economic benefits of adopting them.

HP's LaserJet Firmware division builds the firmware that runs all their scanners, printers, and multifunction devices. The team consists of 400 people distributed across the USA, Brazil, and India. In 2008, the division had a

1 [\[schpilberg\]](#)

2 This case study is taken from [\[gruver\]](#), supplemented by numerous discussions with Gary Gruver.

problem: they were moving too slowly. They had been on the critical path for all new product releases for years, and were unable to deliver new features: “Marketing would come to us with a million ideas that would dazzle the customer, and we’d just tell them, ‘Out of your list, pick the two things you’d like to get in the next 6–12 months.’” They had tried spending, hiring, and outsourcing their way out of the problem, but nothing had worked. They needed a fresh approach.

Their first step was to understand their problem in greater depth. They approached this by using *activity accounting*—allocating costs to the activities the team is performing. **Table 6-1** shows what they discovered.

Table 6-1. Activities of the HP LaserJet Firmware team in 2008

% of costs	Activity
10%	Code integration
20%	Detailed planning
25%	Porting code between version control branches
25%	Product support
15%	Manual testing
~5%	Innovation

This revealed a great deal of no-value-add activity in their work, such as porting code between branches and detailed upfront planning. The large amount spent on current product support also indicated a problem with the quality of the software being produced. Money spent on support is generally serving *failure demand*, as distinct from *value demand* was only driving 5% of the team’s costs.³

The team had a goal of increasing the proportion of spending on innovation by a factor of 10. In order to achieve that goal, they took the bold but risky decision to build a new firmware platform from scratch. There were two main architectural goals for the new “FutureSmart” platform. The first goal was to increase quality while reducing the amount of manual testing required for new

³ The distinction between failure demand and value demand comes from John Seddon, who noticed that when banks outsourced their customer service to call centers, the volume of calls rose enormously. He showed that up to 80% of the calls were “failure demand” of people calling back because their problems were not solved correctly the first time [seddon].

firmware releases (a full manual testing cycle required six weeks). The team hoped that this goal could be achieved through:

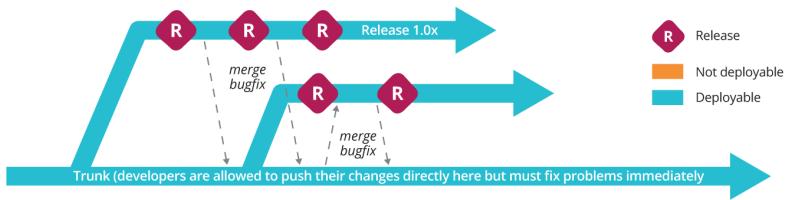
- The practice of continuous integration (which we describe in [Chapter 8](#))
- Significant investment in test automation
- Creating a hardware simulator so that tests could be run on a virtual platform
- Reproduction of test failures on developer workstations

Three years into the development of the new firmware, thousands of automated tests had been created.

Second, they wanted to remove the need for the team to spend time porting code between branches (25% of total costs on the existing system). This was caused by the need to create a branch—effectively a copy of the entire code-base—for every new line of devices under development. If a feature or bug-fix added to one line of devices was required for any others, these changes would need to be merged (copied back) into the relevant code branches for the target devices, as shown in [Figure 6-1](#). Moving away from branch-based development to trunk-based development was also necessary to implement continuous integration. Thus the team decided to create a single, modular platform that could run on any device, removing the need to use version control branches to handle the differences between devices.

The final goal of the team was to reduce the amount of time its members spent on detailed planning activities. The divisions responsible for marketing the various product lines had insisted on detailed planning because they simply could not trust the firmware team to deliver. Much of this time was spent performing detailed re-plans after failing to meet the original plans.

Furthermore, the team did not know how to implement the new architecture, and had not used trunk-based development or continuous integration at scale before. They also understood that test automation would require a great deal of investment. How would they move forward?



In trunk-based development (above), developers make their changes directly to trunk.

In a typical non-trunk-based development style (below), developers typically make changes to long-lived branches which are then stabilized before being released.

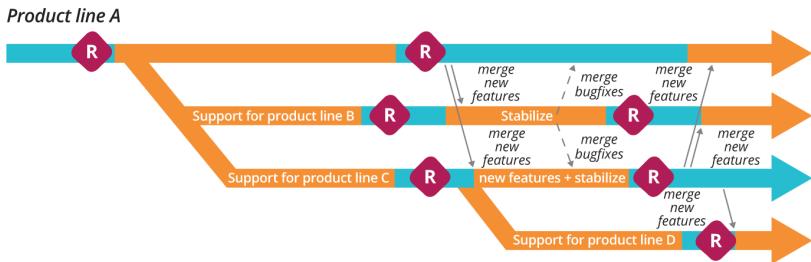


Figure 6-1. Branching versus trunk-based development

It's all too easy to turn a sequence of events into a story in an attempt to explain the outcome—a cognitive bias that Nassim Taleb terms the *narrative fallacy*. This is, arguably, how methodologies are born. What struck us when studying the FutureSmart case were the similarities between the program management method of FutureSmart's engineering management team and the approach Toyota uses to manage innovation as described in Mike Rother's *Toyota Kata: Managing People for Improvement, Adaptiveness, and Superior Results*.⁴

Drive Down Costs Through Continuous Process Innovation Using the Improvement Kata

The Improvement Kata, as described by Mike Rother, is a general-purpose framework and a set of practice routines for reaching goals where the path to the goal is uncertain. It requires us to proceed by iterative, incremental steps, using very rapid cycles of experimentation. Following the Improvement Kata also increases the capabilities and skills of the people doing the work, because it requires them to solve their own problems through a process of continuous experimentation, thus forming an integral part of any learning organization.

⁴ [rother-2010]

Finally, it drives down costs through identifying and eliminating waste in our processes.

The Improvement Kata needs to be first adopted by the organization's management, because it is a management philosophy that focuses on developing the capabilities of those they manage, as well as on enabling the organization to move towards its goals under conditions of uncertainty. Eventually, everybody in the organization should be practicing the Improvement Kata habitually to achieve goals and meet challenges. This is what creates a culture of continuous improvement, experimentation, and innovation.

To understand how this works, let's examine the concept of *kata* first. A kata is "a routine you practice deliberately, so its pattern becomes a habit."⁵ Think of practicing scales to develop muscle memory and digital dexterity when learning the piano, or practicing the basic patterns of movement when learning a martial art (from which the term derives), or a sport. We want to make continuous improvement a habit, so that when faced with an environment in which the path to our goal is uncertain, we have an instinctive, unconscious routine to guide our behavior.

In Toyota, one of the main tasks of managers is to teach the Improvement Kata pattern to their teams and to facilitate running it (including coaching learners) as part of everyday work. This equips teams with a method to solve their own problems. The beauty of this approach is that if the goal or our organization's environment changes, we don't need to change the way we work—if everybody is practicing the Improvement Kata, the organization will automatically adapt to the new conditions.

The Improvement Kata has four stages that we repeat in a cycle, as shown in [Figure 6-2](#).

⁵ [rother]

THE FOUR STEPS OF THE IMPROVEMENT KATA MODEL

A systematic, scientific pattern of working

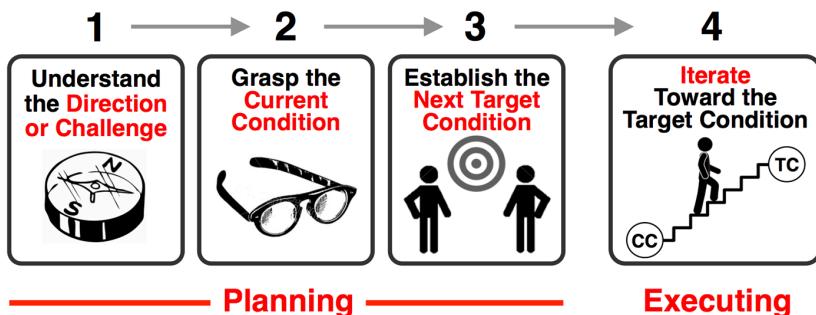


Figure 6-2. The Improvement Kata, courtesy of Mike Rother

Understand the Direction

We begin with understanding the direction. Direction is derived from the vision set by the organization's leadership. A good vision is one that is inspiring—and, potentially, unattainable in practice. For example, the long-term vision for Toyota's production operations is “One-piece flow at lowest possible cost.” In *Leading Lean Software Development*, Mary and Tom Poppendieck describe Paul O’Neill setting the objective for Alcoa to be “Perfect safety for all people who have anything to do with Alcoa” when he became CEO in 1987.⁶

People need to understand that they must always be working towards the vision and that they will never be done with improvement. We will encounter problems as we move towards the vision. The trick is to treat them as obstacles to be removed through experimentation, rather than objections to experimentation and change.

Based on our vision and following the Principle of Mission, we must understand the direction we are working in, at the level of the whole organization and at the value stream level. This challenge could be represented in the form of a future-state value stream map (see Chapter 7 for more on value stream mapping). It should result in a measurable outcome for our customers, and we should plan to achieve it in six months to three years.

⁶ [poppendieck-09], Frame 13, “Visualize Perfection.”

Planning: Grasp the Current Condition and Establish a Target Condition

After we have understood the direction at the organizational and value stream levels, we incrementally and iteratively move towards it at the process level. Rother recommends setting target conditions with a horizon between one week and three months out, with a preference for shorter horizons for beginners. For teams that are using iterative, incremental methods to perform product development, it makes sense to use the same iteration (or sprint) boundaries for both product development and Improvement Kata iterations. Teams that use flow-based methods such as the Kanban Method (for which see [Chapter 7](#)) and continuous delivery (described in [Chapter 8](#)) can create Improvement Kata iterations at the program level.

As with all iterative product development methods, Improvement Kata iterations involve a planning part and an execution part. Here, planning involves grasping the current condition at the process level and setting a target condition that we aim to achieve by the end of the next iteration.

Analyzing the current condition “is done to obtain the facts and data you need in order to then describe an appropriate next target condition. What you’re doing is trying to find the current pattern of operation, so you can establish a desired pattern of operation (a target condition).” The target condition “describes in measurable detail how you want a process to function...[It is] a description and specification of the operating pattern you want a process or system to have on a future date.”⁷

The team grasps the current condition and establishes a target condition together. However, in the planning phase the team does not plan how to *move* to the target condition. In the Improvement Kata, people doing the work strive to achieve the target condition by performing a series of experiments, not by following a plan.

A target condition identifies the process being addressed, sets the date by which we aim to achieve the specified condition, and specifies measurable details of the process as we want it to exist. Examples of target conditions include WIP (work in progress) limits, the implementation of Kanban or a continuous integration process, the number of good builds we expect to get per day, and so forth.

⁷ [rother]

Getting to the Target Condition

Since we are engaging in process innovation in conditions of uncertainty, we cannot know in advance how we will achieve the target condition. It's up to the people doing the work to run a series of experiments using the Deming cycle (plan, do, check, act), as described in [Chapter 3](#). The main mistakes people make when following the Deming cycle are performing it too infrequently and taking too long to complete a cycle. With Improvement Kata, everybody should be running experiments on a daily basis.

Each day, people in the team go through answering the following five questions:⁸

1. What is the target condition?
2. What is the actual condition now?
3. What obstacles do you think are preventing you from reaching the target condition? Which one are you addressing now?
4. What is your next step? (Start of PDCA cycle.) What do you expect?
5. When can we go and see what we learned from taking that step?

As we continuously repeat the cycle, we reflect on the last step taken to introduce improvement. What did we expect? What actually happened? What did we learn? We might work on the same obstacle for several days.

This experimental approach is already central to how engineers and designers work. Designers who create and test prototypes to reduce the time taken by a user to complete a task are engaged in exactly this process. For software developers using test-driven development, every line of production code they write is essentially part of an experiment to try and make a unit test pass. This, in turn, is a step on the way to improving the value provided by a program—which can be specified in the form of a target condition, as we describe in [Chapter 9](#).

The Improvement Kata is simply a generalization of this approach to improvement, combined with applying it at multiple levels of the organization, as we discuss when presenting strategy deployment in [Chapter 15](#).

How the Improvement Kata Differs from Other Methodologies

You can think of the Improvement Kata as a *meta-methodology* since it does not apply to any particular domain, nor does it tell you what to do. It is not a playbook; rather, as with the Kanban Method, it teaches teams how to *evolve*

⁸ [rother]

their existing playbook. In this sense, it differs from other agile frameworks and methodologies. With the Improvement Kata, there is no need to make existing processes conform to those specified in the framework; process and practices you use are *expected* to evolve over time. *This* is the essence of agile: teams do not become agile by adopting a methodology. Rather, true agility means that teams are constantly working to evolve their processes to deal with the particular obstacles they are facing at any given time.

NOTE

Single-Loop Learning and Double-Loop Learning

Changing the way we think and behave in reaction to a failure is crucial to effective learning. This is what distinguishes *single-loop learning* from *double-loop learning* (see [Figure 6-3](#)). These terms were coined by management theorist Chris Argyris, who summarizes them as follows: "When the error detected and corrected permits the organization to carry on its present policies or achieve its present objectives, then that error-and-correction process is single-loop learning. Single-loop learning is like a thermostat that learns when it is too hot or too cold and turns the heat on or off. The thermostat can perform this task because it can receive information (the temperature of the room) and take corrective action. Double-loop learning occurs when error is detected and corrected in ways that involve the modification of an organization's underlying norms, policies and objectives."⁹ Argyris argues that the main barrier to double-loop learning is defensiveness when confronted with evidence that we need to change our thinking, which can operate at both individual and organizational levels. We discuss how to overcome this anxiety and defensiveness in [Chapter 11](#).

⁹ [\[argyris\]](#), pp. 2–3.

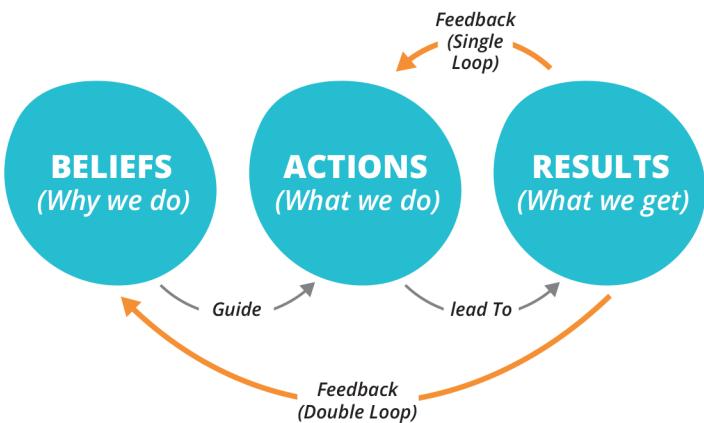


Figure 6-3. Single-loop and double-loop learning

When you practice the Improvement Kata, process improvement becomes planned work, similar to building product increments. The key is that we don't plan *how* we will achieve the target condition, nor do we create epics, features, stories, or tasks. Rather, the team works this out through experimentation over the course of an iteration.

Deploying the Improvement Kata

Rother's work on the Improvement Kata was a direct result of his enquiry into how people become managers at Toyota. There is no formal training program, nor is there any explicit instruction. However, to become a manager at Toyota, one must have first worked on the shop floor and therefore participated in the Improvement Kata. Through this process, managers receive implicit training in how to manage at Toyota.

This presents a problem for people who want to learn to manage in this way or adopt the Improvement Kata pattern. It is also a problem for Toyota—which is aiming to scale faster than is possible through what is effectively an apprenticeship model for managers.

Consequently, Rother presents the Coaching Kata in addition to the Improvement Kata. It is part of deploying the Improvement Kata, but it is also as a way to grow people capable of working with the Improvement Kata, including managers.

Rother has made a guide to deploying the Improvement Kata, *The Improvement Kata Handbook*, available for free on his website at <http://bit.ly/11iBzIY>.

How the HP LaserJet Team Implemented the Improvement Kata

The direction set by the HP LaserJet leadership was to improve developer productivity by a factor of 10, so as to get firmware off the critical path for product development and reduce costs.¹⁰ They had three high-level goals:

1. Create a single platform to support all devices
2. Increase quality and reduce the amount of stabilization required prior to release
3. Reduce the amount of time spent on planning

They did not know the details of the path to these goals and didn't try to define it. The key decision was to work in iterations, and set target conditions for the end of each four-week iteration. The target conditions for Iteration 30 (about 2.5 years into the development of the FutureSmart platform) are shown in [Figure 6-4](#).

The first thing to observe is that the target conditions (or “exit criteria” as they are known in FutureSmart) are all measurable conditions. Indeed, they fulfill all the elements of SMART objectives: they are specific, measurable, achievable, relevant, and time bound (the latter by virtue of the iterative process). Furthermore, many of the target conditions were not focused on features to be delivered but on attributes of the system, such as quality, and on activities designed to validate these attributes, such as automated tests. Finally, the objectives for the entire 400-person distributed program for a single month was captured in a concise form that fit on a single piece of paper—similar to the standard A3 method used in the Toyota Production System.

How are the target conditions chosen? They are “aggressive goals the team feels are possible and important to achieve in 4 weeks...We typically drive hard for these stretch goals but usually end up hitting around 80% of what we thought we could at the beginning of the month.”¹¹ Often, target conditions would be changed or even dropped if the team found that the attempt to achieve them results in unintended consequences: “It’s surprising what you learn in a month and have to adjust based on discovery in development.”¹²

¹⁰ [\[gruver\]](#), p. 144.

¹¹ [\[gruver\]](#), p. 40.

¹² Ibid.

Rank	Theme	Exit Criteria
		Objective met / <i>Objective not met</i>
0	Quality threshold	P1 issues open < 1 week L2 test failure 24 hour response
1	Quarterly bit release	A) <i>Final P1 change requests fixed</i> B) Reliability error rate at release criteria
2	New platform stability and test coverage	A) Customer Acceptance Test 100% passing B) All L2 test pillars 98% passing C) L4 test pillars in place D) L4 test coverage for all Product Turn On requirements E) 100% execution of L4 tests on new products
3	Product Turn On dependencies and key features	A) Print for an hour at speed to finisher with stapling B) Copy for an hour <i>at speed</i> C) <i>Enable powersave mode</i> D) Manufacturing nightly test suite execution E) Common Test Library support for four-line control panel display
4	Build for next-gen products	A) <i>End-to-end system build on new processor</i> B) <i>High-level performance analysis on new processor</i>
5	Fleet integration plan	Align on content and schedule for "slivers" of end-to-end agile test with system test lab

Figure 6-4. Target conditions for iteration 30¹³

¹³ Gruver, Gary, Young, Mike, Fulghum, Pat. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*, 1st Edition, (c) 2013. Reprinted by permission of Pearson Education, Inc. Upper Saddle River, NJ.

WARNING

What Happens When We Do Not Achieve Our Target Conditions?

In bureaucratic or pathological organizational cultures, not achieving 100% of the specified target conditions is typically considered a failure. In a generative culture, however, we *expect* to not be able to achieve all our target conditions. The purpose of setting aggressive target conditions is to reveal obstacles so we can overcome them through further improvement work. Every iteration should end with a retrospective (described in [Chapter 11](#)) in which we investigate how we can get better. The results form part of the input for the next iteration's target conditions. For example, if we fail to achieve a target condition for the number of good builds of the system per day, we may find that the problem is that it takes too long to provision test environments. We may then set a target condition to reduce this in the next iteration.

This approach is a common thread running through all of Lean Thinking. The subtitle of Mary and Tom Poppendieck's book *Leading Lean Software Development* reads: "Results are not the point." This is a provocative statement that gets to the heart of the lean mindset. If we achieve the results by ignoring the process, we do not learn how to improve the process. If we do not improve the process, we cannot repeatably achieve better results. Organizations that put in place unmodifiable processes that everybody is required to follow, but which get bypassed in a crisis situation, fail on both counts.

This adaptive, iterative approach is not new. Indeed it has a great deal in common with what Tom Gilb proposed in his 1988 work *Principles of Software Engineering Management*:¹⁴

We must set measurable objectives for each next small delivery step. Even these are subject to constant modification as we learn about reality. It is simply not possible to set an ambitious set of multiple quality, resource, and functional objectives, and be sure of meeting them all as planned. We must be prepared for compromise and trade-off. We must then design (engineer) the immediate technical solution, build it, test it, deliver it—and get feedback. This feedback must be used to modify the immediate design (if necessary), modify the major architectural ideas (if necessary), and modify both the short-term and the long-term objectives (if necessary).

¹⁴ [\[gilb-88\]](#), p. 91.

Designing for Iterative Development

In large programs, demonstrating improvement within an iteration requires ingenuity and discipline. It's common to feel we can't possibly show significant progress within 2–4 weeks. Always try to find something small to bite off to achieve a little bit of improvement, instead of trying to do something you think will have more impact but will take longer.

This is not a new idea, of course. Great teams have been working this way for decades. One high-profile example is the Apple Macintosh project where a team of about 100 people—co-located in a single building—designed the hardware, operating system, and applications for what was to become Apple's breakthrough product.

The teams would frequently integrate hardware, operating system, and software to show progress. The hardware designer, Burrell Smith, employed programmable logic chips (PALs) so he could prototype different approaches to hardware design rapidly in the process of developing the system, delaying the point at which it became fixed—a great example of the use of optionality to delay making final decisions.¹⁵

After two years of development, the new firmware platform, FutureSmart, was launched. As a result, HP had *evolved* a set of processes and tools that substantially reduced the cost of no-value-add activities in the delivery process while significantly increasing productivity. The team was able to achieve “predictable, on-time, regular releases so new products could be launched on time.”¹⁶ Firmware moved off the critical path for new product releases for the first time in twenty years. This, in turn, enabled them to build up trust with the product marketing department.

As a result of the new relationship between product marketing and the firmware division, the FutureSmart team was able to considerably reduce the time spent on planning. Instead of “committing to a final feature list 12 months in advance that we could never deliver due to all the plan changes over the time,”¹⁷ they looked at each planned initiative once every 6 months and did a 10-minute estimate of the number of months of engineering effort required for a given initiative, broken down by team. More detailed analysis would be performed once work was scheduled into an iteration or mini-milestone. An example of the output from one of these exercises is shown in [Figure 6-5](#).

¹⁵ http://folklore.org/StoryView.py?story=Macintosh_Prototypes.txt

¹⁶ [gruver], p. 89.

¹⁷ [gruver], p. 67.

High-Level Estimate – FW Engineering Months														
Rank	Initiative	Component 1 (25-30)	Component 2 (20-25)	Component 3 (30-40)	Component 4 (30-40)	Component 5 (20-30)	Component 6 (20-30)	Component 7 (20-30)	Component 8 (15-25)	Component 10 (40-50)	Component 11 (20-30)	Component 12 (20-30)	other teams	TOTAL
1	Initiative A		21		5	3		1						30
2	Initiative B	3						4					17	24
3	Initiative C		5						2	1	1			9
4	Initiative D						10		2	2	2			16
5	Initiative E				20					3	5			28
6	Initiative F	23						5	6			2		36
7	Initiative G								2					2
8	Initiative H									5				5
9	Initiative I									3				3
10	Initiative J	20	27			17			39	17	21	9		150
11	Initiative K		3	30		3		3	14				12	65
12	Initiative L								2					2
13	Initiative M	3					10		6	6	6			31
		29	25	51	30	20	25	23	12	74	26	38	59	401

Figure 6-5. Ballpark estimation of upcoming initiatives¹⁸

This is significantly different from how work is planned and estimated in large projects that often create detailed functional and architectural epics which must be broken down into smaller and smaller pieces, analyzed in detail, estimated, and placed into a prioritized backlog *before* they are accepted into development.

Ultimately the most important test of the planning process is whether we are able to keep the commitments we make to our stakeholders, including end users. As we saw, a more lightweight planning process resulted in firmware development moving off the critical path, while at the same time reducing both development costs and failure demand. Since we would expect failure demand to *increase* as we increase throughput, this is doubly impressive.

¹⁸ Gruver, Gary, Young, Mike, Fulghum, Pat. *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*, 1st Edition, (c) 2013. Reprinted by permission of Pearson Education, Inc. Upper Saddle River, NJ.

Three years after their initial measurements, a second activity-accounting exercise offered a snapshot of the results the FutureSmart team had achieved with their approach, shown in [Table 6-2](#).

Table 6-2. Activity of the HP LaserJet Firmware Team in 2011

% of costs	Activity	Previously
2%	Continuous integration	10%
5%	Agile planning	20%
15%	One main branch	25%
10%	Product support	25%
5%	Manual testing	15%
23%	Creating and maintaining automated test suites	0%
~40%	Innovation	~5%

Overall, the HP LaserJet Firmware division changed the economics of the software delivery process by adopting continuous delivery, comprehensive test automation, an iterative and adaptive approach to program management, and a more agile planning process.

Economic Benefits of HP FutureSmart's Agile Transformation

- Overall development costs were reduced by ~40%.
- Programs under development increased by ~140%.
- Development costs per program went down 78%.
- Resources driving innovation increased eightfold.

The most important point to remember from this case study is that the enormous cost savings and improvements in productivity were only possible on the basis of a large and ongoing *investment* made by the team in test automation and continuous integration. Even today, many people think that Lean is a management-led activity and that it's about simply *cutting costs*. In reality, it requires *investing* to remove waste and reduce failure demand—it is a worker-led activity that, ultimately, can continuously drive down costs and improve quality and productivity.

Managing Demand

Up to now, we've been discussing how to improve the throughput and quality of the delivery process. However, it is very common for this kind of improvement work to get crowded out by business demands, such as developing new features. This is ironic, given that the whole purpose of improvement work is to increase the rate at which we can deliver as well as the quality of what gets delivered. It's often hard to make the outcome of improvement work tangible—which is why it's important to make it visible by activity accounting, including measuring the cycle time and the time spent serving failure demand such as rework.

The solution is to use the same mechanism to manage both demand and improvement work. One of the benefits of using the Improvement Kata approach is that it creates alignment to the outcomes we wish to achieve over the next iteration across the whole program. In the original Improvement Kata, the target conditions are concerned with process improvement, but we can use them to manage demand as well.

There are two ways to do this. In organizations with a generative culture (see [Chapter 1](#)), we can simply specify the desired business goals as target conditions, let the teams come up with ideas for features, and run experiments to measure whether they will have the desired impact. We describe how to use impact mapping and hypothesis-driven development to achieve this in [Chapter 9](#). However, more traditional enterprises will typically have a backlog of work prioritized at the program level by its lines of business or by product owners.

We can take a few different approaches to integrating a program-level backlog with the Improvement Kata. One possibility is for teams working within the program to deploy the Kanban Method, as described in [Chapter 7](#). This includes the specification of work in process (WIP) limits which are owned and managed by these teams. New work will only be accepted when existing work is completed (where “completed” means it is at least integrated, fully tested with all test automation completed, and shown to be deployable).

TIP

Managing Cross-Cutting Work

Implementing some features within a program will involve multiple teams working together. To achieve this, the HP FutureSmart division would set up a small, temporary “virtual” feature team whose job is to coordinate work across the relevant teams.

The HP FutureSmart program, some of whose teams were using Scrum, took the approach of specifying a target velocity at the *program* level. Work adding up to the target velocity was accepted for each iteration, approximating a WIP limit. In order to implement this approach, all work was analyzed and estimated at a high level before being accepted. Analysis and estimation was kept to the bare minimum required to be able to consistently meet the overall program-level target conditions, as shown in [Figure 6-5](#).

WARNING

Do Not Use Team Velocity Outside Teams

It is important to note that specifying a target velocity at the program level does *not* require that we attempt to measure or manage velocity at the team level, or that teams must use Scrum. Program-level velocity specifies the expected work capacity of all teams based on high-level estimates, as shown in [Figure 6-5](#). If a team using Scrum accepts work based on these high-level feature specifications, they then create lower-level stories with which to work.

Scrum's team-level velocity measure is not all that meaningful outside of the context of a particular team. Managers should *never* attempt to compare velocities of different teams or aggregate estimates across teams. Unfortunately, we have seen team velocity used as a measure to compare productivity between teams, a task for which it is neither designed nor suited. Such an approach may lead teams to "game" the metric, and even to stop collaborating effectively with each other. In any case, it doesn't matter how many stories we complete if we don't achieve the business outcomes we set out to achieve in the form of program-level target conditions.

In this and the next chapter, we describe a much more effective way to measure progress and manage productivity—one that does not require all teams to use Scrum or "standardize" estimates or velocity. We use activity accounting and value stream mapping (described in [Chapter 7](#)) to measure productivity, and we use value stream mapping combined with the Improvement Kata to increase it—crucially, at the value stream level rather than at the level of individual teams. We measure and manage progress through the use of target conditions at the program level, and if we need to increase visibility, we reduce the duration of iterations.

Creating an Agile Enterprise

Many organizations look to try and adopt agile methods to improve the productivity of their teams. However, agile methods were originally designed around small, cross-functional teams, and many organizations have struggled to use these methods at scale. Some frameworks for scaling agile focus on creating such small teams and then adding structures to coordinate their work at the program and portfolio level.

Gary Gruver, Director of Engineering for FutureSmart, contrasts this approach of “trying to enable the efficiencies of small agile teams in an enterprise” with the FutureSmart team’s approach of “trying to make an enterprise agile using the basic agile principles.”¹⁹ In the FutureSmart approach, while the teams ran within tight guide rails in terms of engineering practices (which we discuss in more detail in [Chapter 8](#)), there was relatively little attention paid to whether they had, for example, implemented Scrum at the team level. Instead, teams have relative autonomy to choose and evolve their own processes, provided they are able to meet the program-level target conditions for each iteration.

This required that engineering management had the freedom to set their own program-level objectives. That is, they didn’t have to get budget approval to pay for process improvement work such as test automation or building out the toolchain for continuous integration. Indeed, the business wasn’t even consulted on this work. All business demand was also managed at the program level. Notably, product marketing requests always went through the program-level process, without feeding work directly to teams.

Another important consideration is the way enterprises treat metrics. In a control culture, metrics and targets are often set centrally and never updated in response to the changes in behavior they produce. Generative organizations don’t manage by metrics and targets. Instead, the FutureSmart management “use[s] the metrics to understand where to have conversations about what is not getting done.”²⁰ This is part of the strategy of “Management by Wandering Around” pioneered by HP founders Bill Hewlett and Dave Packard.²¹ Once we discover a problem, we ask the team or person having a hard time what we can do to help. We have discovered an opportunity to improve. If people are punished for failing to meet targets or metrics, one of the fallouts is that they start manipulating work and information to look like they are meeting the targets. As FutureSmart’s experience shows, having good real-time metrics is a better approach than relying on scrums, or scrums of scrums, or Project Management Office reporting meetings to discover what is going on.

Conclusion

The Improvement Kata provides a way to align teams and, more generally, organizations by taking goals and breaking them down into small, incremental outcomes (target conditions) that get us closer to our goal. The Improvement

19 [\[gruver\]](#), Chapter 15.

20 [\[gruver\]](#), p. 38.

21 Perhaps it’s better characterized as “Management by Wandering Around and Asking Questions.” In the Toyota Production System, this is known as a *gemba* walk.

Kata is not just a meta-methodology for continuous improvement at the enterprise and program level; it is a way to push ownership for achieving those outcomes to the edges of the organization, following the Principle of Mission. As we show in [Chapter 9](#), it can also be used to run large programs of work.

The key characteristics of the Improvement Kata are its iterativeness and the ability to drive an experimental approach to achieve the desired target conditions, which makes it suitable for working in conditions of uncertainty. The Improvement Kata is also an effective way to develop the capabilities of people throughout the enterprise so they can self-organize in response to changing conditions.

The FutureSmart case study shows how a large, distributed team applied the Improvement Kata meta-method to increase productivity eightfold, improving quality and substantially reducing costs. The processes and tools the team used to achieve this transformation changed and evolved substantially over the course of the project. This is characteristic of a truly agile organization.

Implementing an enterprise-level continuous improvement process is a prerequisite for any ongoing large-scale transformation effort (such as adopting an agile approach to software delivery) at scale. True continuous improvement never ends because, as our organization and environment evolve, we find that what works for us today will not be effective when conditions change. High-performance organizations are constantly evolving to adapt to their environment, and they do so in an organic way, not through command and control.

Questions for readers:

- Do you know how much time your engineering organization is spending on no-value-add activities and servicing failure demand versus serving value demand, and what the major sources of waste are?
- Must engineering teams get permission to invest in work that reduces waste and no-value-add activity across the value stream as a whole, such as build, test, and deployment automation and refactoring? Are such requests denied for reasons such as “there is no budget” or “we don’t have time”?
- Does everyone within the organization know the short- and long-term outcomes they are trying to achieve? Who decides these outcomes? How are they set, communicated, reviewed, and updated?
- Do teams in your organization regularly reflect on the processes they use and find ways to experiment to improve them? What feedback loops are in place to find out which ideas worked and which didn’t? How long does it take to get this feedback?

Identify Value and Increase Flow

There is nothing so useless as doing efficiently that which should not be done at all.

— Peter Drucker

The measure of execution in product development is our ability to constantly align our plans to whatever is, at the moment, the best economic choice.

— Donald Reinertsen and Stefan Thomke

Most enterprises are drowning in a sea of overwork, much of which provides little value to customers. In addition to improving existing products and delivering new ones, every enterprise has several initiatives and strategic projects in play at any given time, and every day unplanned work arrives to distract us from achieving our goals. A common response to this problem is attempting to increase utilization (work harder), improve efficiency (work faster), and cut costs using outdated and counterproductive management processes. Lean Thinking provides a proven alternative which “can be summarized in five principles: precisely specify *value* by specific product, identify the *value stream* for each product, make value *flow* without interruptions, let the customer *pull* value from the producer, and pursue *perfection*.¹”¹

In the previous chapter we showed how to implement a program-level continuous improvement strategy to improve productivity and quality and reduce costs. In this chapter we show how the five lean principles were adopted by

¹ [womack]

Maersk to reduce the cycle time of new features by over 50% while simultaneously increasing quality and return on investment.

The Maersk Case Study

In “Black Swan Farming using Cost of Delay,”² Joshua J. Arnold and Özlem Yüce discuss how they approached reducing cycle time in Maersk Lines, the world’s largest shipping company. Maersk’s IT department had an annual IT budget of over \$150m, with much of its development carried out by globally distributed outsourcing vendors. They faced a large amount of demand and slow time-to-market: in 2010, median lead time for a feature was 150 days, with 24% of requirements taking over a year to deliver (from conception to software in production). At the point of analysis, in October 2010, more than 2/3 of the 4,674 requirements identified as being in process were in the “fuzzy front end,” waiting to be analyzed in detail and funded. In one case, “a feature that took only 82 hours to develop and test took a total of 46 weeks to deliver end-to-end. Waiting time consumed over 38 weeks of this,” mostly in the fuzzy frontend ([Figure 7-1](#)).

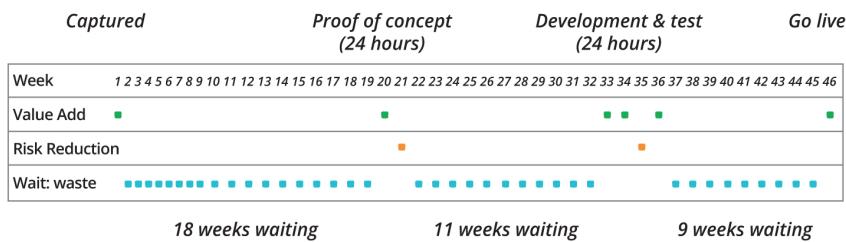


Figure 7-1. Value stream map of a single feature delivered through a core system at Maersk (courtesy of Joshua J. Arnold and Özlem Yü)

Based on the desired outcomes of “more value, faster flow, and better quality,” Arnold and Yüce chose eight goals for all teams:

1. Get to initial prioritization faster
 2. Improve prioritization using Cost of Delay
 3. Pull requirements from Dynamic Priority List
 4. Reduce the size of requirements
 5. Quickly get to the point of writing code
 6. Actively manage work in progress

² <http://costofdelay.com>; [arnold].

7. Enable faster feedback
8. Enable smooth, sustainable flow

Previously, features had always been batched up into projects, resulting in many lower-value features being delivered along with a few high-value ones. The HiPPO method (highest paid person's opinion) was used to decide which features were high-value, and a great deal of effort was spent trying to find the "right ideas" and analyzing them in detail so as to create projects, get approval, and justify funding.

Arnold and Yüce implemented a new process for managing requirements. They created a backlog of features—initially at the project level, but later at the program and portfolio levels—called the Dynamic Priority List. When new features were proposed, they would be quickly triaged, causing the backlog to be reprioritized. When development capacity became available, the highest priority feature would be "pulled" from the list.

Features were prioritized using the Cost of Delay method (described in detail later in the chapter) which estimates the value of a feature in dollars by calculating how much money we lose by *not* having the feature available when we need it. Using this approach, we can determine the impact of time on value and make prioritization decisions on an economic basis. For example, the cost of delay for the feature shown in [Figure 7-1](#) was roughly \$210,000 per week, meaning that the delay incurred by having the feature wait in queues for 38 weeks cost \$8M. Putting in the extra effort to calculate a dollar value is essential to reveal assumptions, come to a shared understanding, and move away from relying on the most senior person in the room making the prioritization call.

The actual number used to prioritize features is known as *cost of delay divided by duration* (or "CD3"). It is calculated as cost of delay for a feature divided by the amount of time we estimate it will take to develop and deliver that feature. This takes into account the fact that we have limited people and resources available to complete work, and that if a particular feature takes a long time to develop it will "push out" other features. Logically, if we have two features with the same cost of delay but one will take twice as long as another to develop, we should develop the shorter-duration feature first. One of the impacts of accounting for duration is that it encourages people to break work down into smaller, more valuable pieces, which in turn increases the CD3 score.

Implementing the Dynamic Priority List and using CD3 to schedule work helped the team to achieve several other goals on their list, such as faster initial prioritization, reducing the size of requirements, writing code more quickly, and creating a smoother flow. By July 2011, median cycle time had been

reduced by about 50% on the two services piloted. (One of the pilot services was a centralized SAP accounting system.) Arnold and Yüce present two factors causing the reduction in cycle time: increased urgency generated by the Cost of Delay calculation exercises, and decreased batch size caused by people breaking work into smaller chunks to increase the CD3. Furthermore, customer satisfaction increased significantly on the pilot projects.

Perhaps most interestingly, calculating the cost of delay clarified which work was most important. In the two systems analyzed, the distribution of cost of delay followed a power law curve. The cost of delay per week numbers for the features in the pilot service, shown in [Figure 7-2](#), make it abundantly clear which three requirements should be prioritized above others. These requirements were not identified as being of the highest priority before the cost of delay was calculated.

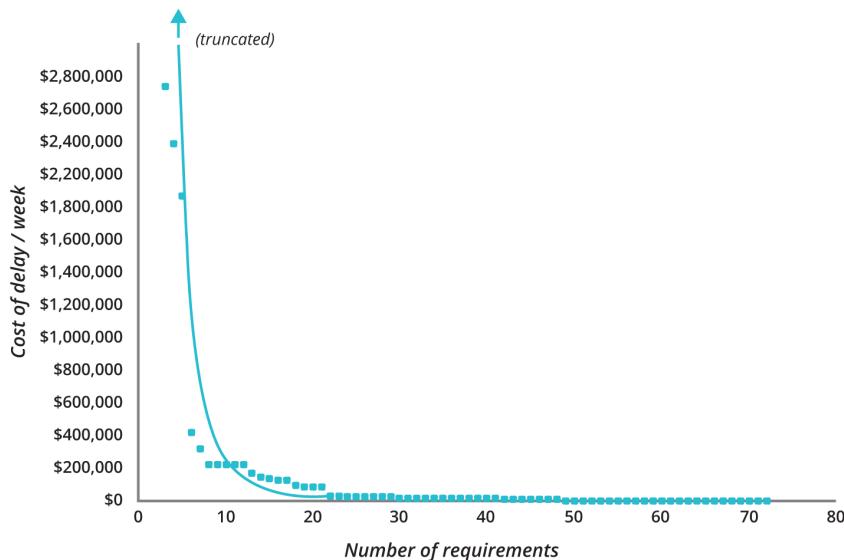


Figure 7-2. CD3 per feature (courtesy of Joshua J. Arnold and Özlem Yüce)

The Maersk case study demonstrates the importance of using a flow-based approach to product development instead of large batches of work delivered in projects, and of using the Cost of Delay—not intuition or HiPPO—to measure the relative priority of work to be done.

Increase Flow

As we discussed in [Chapter 6](#), we want to improve the performance of the delivery process before we tackle improving alignment. However, if we want to

see substantial improvements in performance, we need to start by choosing the right places to focus our efforts. It's common to see large organizations waste a lot of effort making changes to processes or behaviors that are highly visible or easy to change but not a major contributor to the overall problem. We need to begin any improvement effort by understanding where the problems arise and making sure they are understood at all levels of the organization. Only then will we have the right context to determine what to do next.

Map Your Product Development Value Streams

The best way to understand where problems start is by performing an activity called *value stream mapping*.³ Every organization has many value streams, defined as the flow of work from a customer request to the fulfillment of that request. Each value stream will cross multiple functions within an organization, as shown in Figure 7-3.

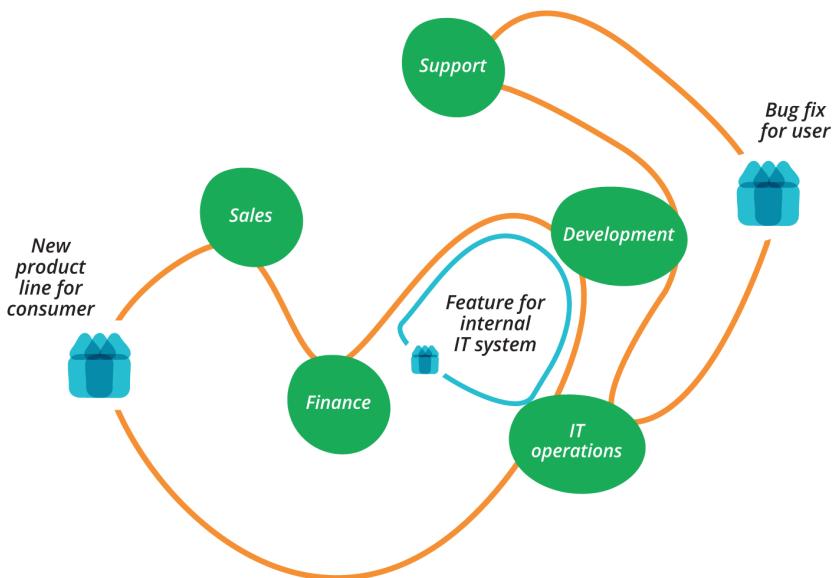


Figure 7-3. Value streams passing through departments

In the context of exploiting validated ideas for software, the value streams we care about are related to product development, from taking an idea or

³ Value stream mapping was first described in [rother-2009] and is the subject of an excellent book by Karen Martin and Mike Osterling [martin].

customer request for a feature or bug fix to delivering it to users. Every product or service will have its own value stream.

To begin, we select the product or service we want to study, and map the *existing* value stream to reflect the current condition. To avoid the common mistake of trying to improve through local optimization, it's essential to create a *future-state* value stream that represents how we want the value stream to flow at some future point—typically in one to three years. This represents our target condition. The current and future value streams can then be used as the basis for improvement work by applying the Improvement Kata across the scope of the entire value stream, as shown in Figure 7-4.

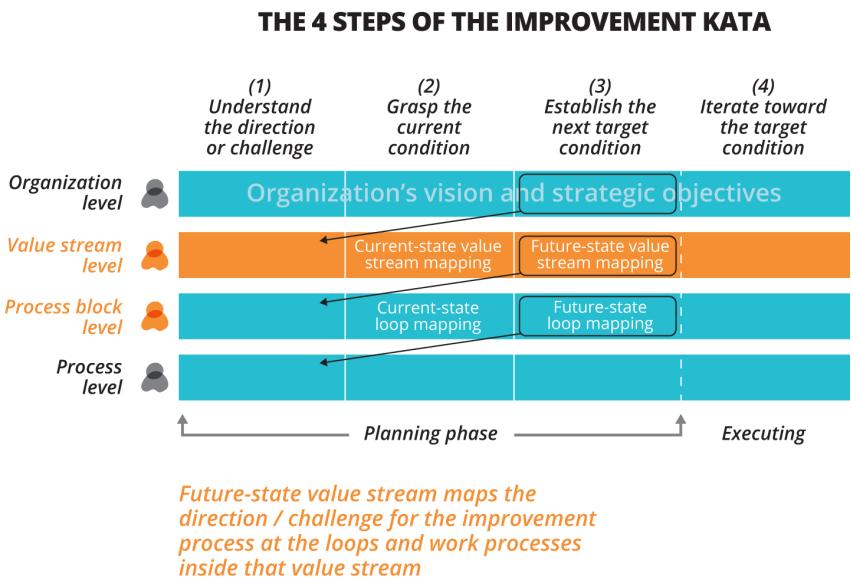


Figure 7-4. Value stream mapping in the context of the Improvement Kata

To run a value stream mapping exercise, we must gather people from every part of the organization involved in the value stream. In the case of product design and delivery, this might include the product's business unit, product marketing, design, finance, development, QA, and operations. Most importantly, the value stream mapping team must include those who are able to authorize the kind of change required to achieve the future-state value stream. Often, getting all the stakeholders to commit to spending 1–3 days together in a single room at the same time is the hardest part of the process. Aim for the smallest possible team that fulfills these criteria—certainly no more than 10 people.

Performing value stream mapping involves defining, on a large surface (**Figure 7-5**), the various process blocks of the product's delivery. How you slice and dice the value stream into process blocks (also known as *value stream loops*) is a bit of an art. We want enough detail to be useful, but not so much that it becomes unnecessarily complex and we get lost arguing about minutiae. Martin and Osterling suggest aiming for between 5 and 15 process blocks.⁴ For each process block within the value stream, we record the activity and the name of the team or function that performs it.

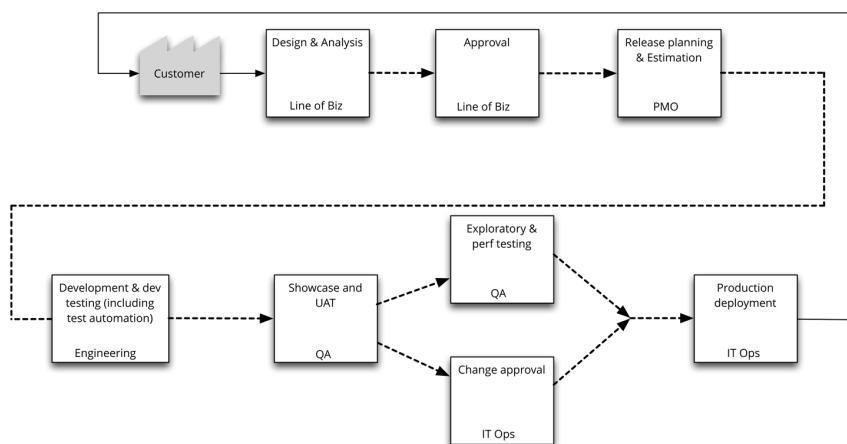


Figure 7-5. Outline of a value stream map showing process blocks

Once we have a block diagram, we gather the data necessary to understand the state of work within the value stream. We want to know the number of people involved in each process and any significant barriers to flow. We also note the amount of work within each process block, as well as queues between blocks. Finally, we record three key metrics: lead time, process time, and percent complete and accurate, as shown in **Table 7-1**.

Table 7-1. Metrics for value stream mapping

Metric	What it measures
Lead time (LT)	The time from the point a process accepts a piece of work to the point it hands that work off to the next downstream process

⁴ [martin], p. 63.

Metric	What it measures
Process time (PT)	The time it would take to complete a single item of work if the person performing it had all the necessary information and resources to complete it and could work uninterrupted
Percent complete and accurate (%C/A)	The proportion of times a process receives something from an upstream process that it can use without requiring rework

When mapping a value stream, we always record the state of the processes as they are on the day we perform the exercise. It's extremely tempting to record numbers representing an ideal or best case state rather than the typical state—but looking at what the numbers are right now helps to keep people on track. Wherever possible, the team should actually go to the places where the work is done and ask the people doing it for the real numbers. This helps the team to experience the different environments where works take place across the value stream.

The output of a simple value stream mapping exercise for a single feature that goes through a relatively straightforward product development value stream is shown in [Figure 7-6](#). If it proves useful, we could go into more detail on each of the stages of the process and state what happens when a process rejects input as incomplete or inaccurate. This is particularly important when the ratio of lead time to process time is large or when the downstream process has an unusually poor %C/A.

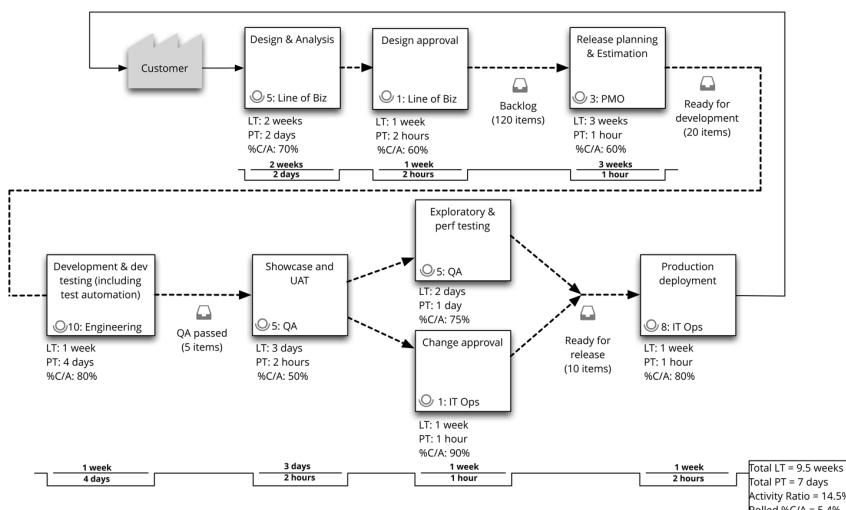


Figure 7-6. Example value stream map of a feature

Running this exercise for the first time in an organization is always enlightening. People are invariably surprised—and often shocked—by how processes in which they do not participate actually work and are impacted by their work. We have seen arguments break out! Ultimately, by producing a better idea of how work moves through the organization, value stream mapping increases alignment, empathy, and shared understanding between the stakeholders.

Perhaps the most valuable metric when performing this exercise is %C/A. It's very common to discover that a great deal of time is wasted on failure demand such as rework: developers discover flaws in the design, testers are given builds that cannot run or be deployed, customers ask for changes when they see features showcased, and critical defects or performance problems are discovered in production or reported by users. Facilitators of these exercises should come armed with questions to discover and capture rework, such as:

- At which points do we discover problems in the design?
- What happens in this case?
- Who is involved in that step?
- How do handoffs work?
- At what point do we discover whether the feature actually delivers the expected value to customers?
- Where are architectural problems (such as performance and security) discovered?
- What is the effect on lead time and quality?

These issues should be captured on the value stream map, their probability recorded in the form of %C/A in the processes that discover them and (where possible) attributed to the part of the value stream where they were actually caused.

The total amount of waste in an enterprise value stream is usually very sobering. While everybody has an intuitive grasp that enterprise value streams are inefficient, seeing the whole value stream from idea to measurable customer outcome often reveals staggering amounts of waste. This waste manifests itself in the percentage of time that is not value-adding, in how often work is sitting idle in queues, and crucially in the %C/A numbers that show us where we have failed to build in quality during upstream processes.

Finally, value stream mapping reveals the folly of local optimizations. In almost every case we have seen, making one process block more efficient will have a minimal effect on the overall value stream. Since rework and wait times are some of the biggest contributors to overall delivery time, adopting “agile”

processes within a single function (such as development) generally has little impact on the overall value stream, and hence on customer outcomes.

In most cases we need to rethink our whole approach to delivering value by transforming the entire value stream, starting by defining the measurable customer and organizational outcomes we wish to achieve through our redesign. In order to mitigate the disruption of this kind of change, we usually limit our efforts to a single product or set of capabilities within a product—one which would most benefit customers and the organization as a whole.

We then create a future-state value stream map which describes how we want the value stream to function in the future. The goal of this design activity is to improve performance. Martin and Osterling define optimal performance as “delivering customer value in a way in which the organization incurs no unnecessary expense; the work flows without delays; the organization is 100 percent compliant with all local, state and federal laws; the organization meets all customer-defined requirements; and employees are safe and treated with respect. In other words, the work should be designed to eliminate delays, improve quality, and reduce unnecessary cost, effort, and frustration.”⁵

There is of course no “right answer” to creating a future-state value stream map, but a good rule of thumb is to aim to significantly reduce lead time and improve rolled %C/A (indicating we have done a better job of building in quality). It’s important for participants in this exercise to be bold and consider radical change (*kaikaku*). Achieving the future state will almost certainly require some people to learn new skills and change the work they are doing, and some roles (but not the people who perform them) will become obsolete. For this reason, as we discuss in [Chapter 11](#), it’s essential to provide support for learning new skills and behaviors, and to communicate widely and frequently that nobody will be punished for carrying out improvement work—otherwise you are likely to experience resistance.

At this stage, don’t try to guess *how* the future state will be achieved: focus on the target conditions to achieve. Once the current and future-state value stream maps are ready, we can use the Improvement Kata to move towards the future state. In [Chapter 6](#) we described the use of the Improvement Kata to drive continuous improvement at the program level. The target conditions for the future-state value stream map should be fed into program-level Improvement Kata cycles. However, where value streams extend beyond the teams involved in programs of work—perhaps into IT operations and business units—we need to also establish Improvement Kata cycles at the value stream level, with owners who establish and track key performance indicators and monitor progress.

⁵ [martin], p. 101.

WARNING

Organizations using the phase-gate paradigm (described in [Figure III-1](#) at the beginning of [Part III](#)) will find the principles described in the following chapters increasingly hard to implement without fundamentally changing their organizational structure. The Improvement Kata described in [Chapter 6](#) can (and should) be implemented everywhere as it makes almost no presuppositions about organizational structure. We have just discussed how to map and improve the flow of work through your organization, which will enable you to begin incrementally changing the form of your organization and the roles of your people. [Chapter 8](#) discusses lean engineering practices that enable faster, higher-quality delivery at lower cost. If your organization outsources software engineering, your outsourcing partners will need to implement these practices, and this is likely to require changes to your relationship, including potentially contractual changes. [Chapter 9](#), which describes an experimental approach to product development, requires that designers, engineers, testers, infrastructure specialists, and product people work collaboratively in very short iterations. This is extremely hard to do when any of these functions are outsourced; it's marginally easier when all teams are internal, but still requires everyone to work in a coordinated way.

Everyone can, and should, begin the journey we describe in [Part III](#). Be forewarned: implementing the entire program will be disruptive to most organizations and take years of investment and experimentation to achieve. Do not try to implement the entire program across all of your organization quickly—use value stream mapping and break down future-state value stream maps into blocks to do it iteratively and incrementally, value stream by value stream.

Limit Work in Process

If our goal is to increase the flow of high-value work through the product development value stream, value stream mapping represents an essential first step. However, we must take further steps to manage the flow of work through the system so as to decrease lead times and increase predictability.

In the context of product development, the Kanban Method provides principles and practices to support this goal, as described in David J. Anderson's *Kanban: Successful Evolutionary Change for your Technology Business*.⁶ First, we must visualize the flow of work through the value stream: we take the current-state value stream map and translate it to a physical or virtual board with columns representing process blocks and queues between blocks. We then create a card for each piece of work currently passing through the value stream, as shown in [Figure 7-7](#). These cards are moved across the board as work progresses through the value stream.

⁶ [anderson]

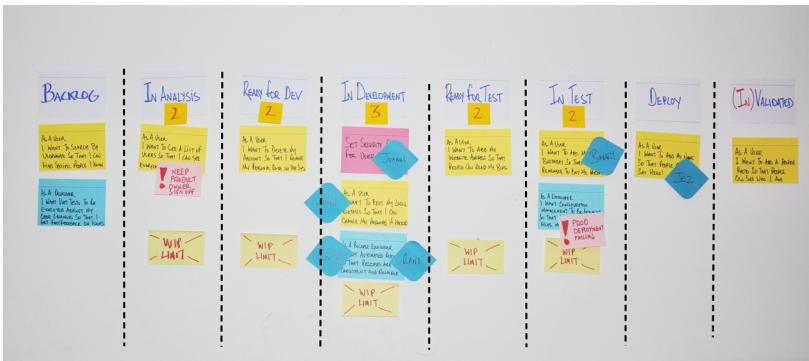


Figure 7-7. An example of a Kanban board

We can visualize the dynamics of the value stream by creating a *cumulative flow diagram* that shows the amount of work in each queue and process block over time. An example of a cumulative flow diagram is shown in Figure 7-8. It clearly shows the relationship between work in process (WIP) and lead time: as we reduce WIP, lead time falls.⁷

In the Maersk case study, we discussed two ways to reduce the size of the batches of work that move through the product development value stream: reduce the size of requirements and unbundle projects into requirements that can be prioritized independently. Limiting WIP is another powerful way to reduce batch size. Since reducing batch sizes is the most important factor in systematically increasing flow and reducing variability, and has important second-order effects such as improving quality and increasing trust between stakeholders, we should pursue these practices relentlessly, and measure our progress.

⁷ These two quantities are in fact causally related; in the mathematical field called Queue Theory this is known as Little's Law.

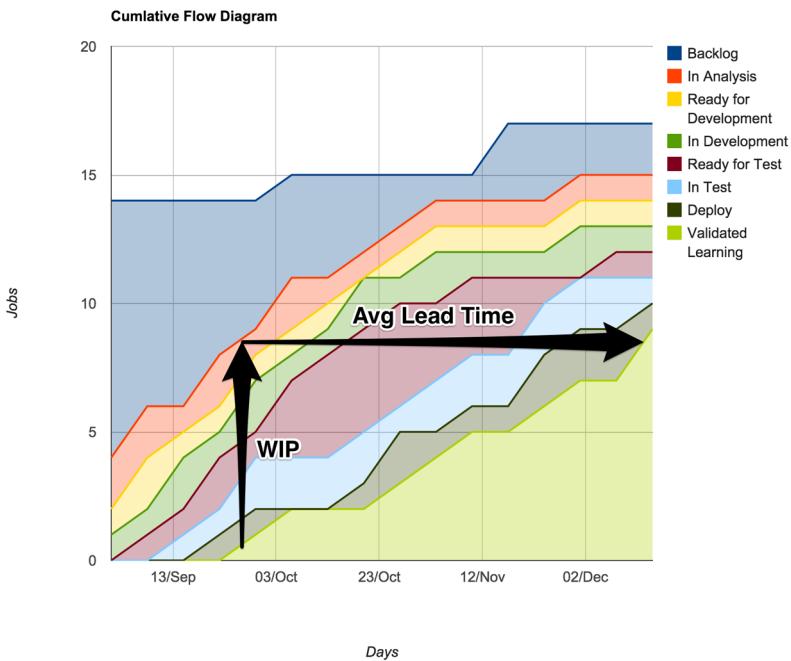


Figure 7-8. A cumulative flow diagram

The Kanban Method offers a comprehensive way to manage the flow of work through the product development value stream by using the following practices:

- *Visualize workflow* by creating a board showing the current work in process within the value stream in real time.
- *Limit work in process* by setting WIP limits for each process block and queue within a value stream, and updating them in order to trade off lead time against utilization (how busy people are).
- *Define classes of service* for different types of work and the processes through which they will be managed, to ensure that urgent or time-sensitive work is prioritized appropriately.
- *Create a pull system* by agreeing on how work will be accepted into each process block when capacity becomes available—perhaps by setting up a regular meeting where stakeholders decide what work should be prioritized based on available capacity.

- Hold regular “*operational reviews*” for the stakeholders within each process block to analyze their performance and update WIP limits, classes of service, and the method through which work is accepted.

WARNING

WIP Limits Should Hurt

Part of the purpose of WIP limits is to reveal opportunities for improvement. Imposing WIP limits will focus attention on work which is blocked or hard to complete, since our inability to complete it prevents us picking up new work. At this point, it’s tempting to relax WIP limits to make sure “something is getting done.” It’s essential to avoid this temptation and address the sources of the problem instead.

The Kanban Method follows four principles of continuous improvement designed to minimize resistance to change:

- Start with what you do now
- Agree to pursue incremental, evolutionary change
- Initially, respect current roles, responsibilities, and job titles
- Encourage acts of leadership at all levels

The Kanban Method is a powerful tool to improve performance and increase quality and trust at the team level in an environment where there is no buy-in for continuous improvement at the senior management level. In such a situation, we strongly recommend that teams deploy the Kanban Method as their first step to get better. Once you have demonstrated measurable improvement, you still need to pursue enterprise-level continuous improvement since, in a typical enterprise context, adopting Kanban at the team level is likely to lead only to incremental improvements.

TIP

Managing Work in Process at the Enterprise Level

The primary goal of limiting WIP is to *finish* work to a sufficiently high level of quality so as to increase throughput. Reducing lead times in this way *requires* that there be sufficient slack in the system to manage the WIP effectively. Slack is also essential to provide time for process improvement work. Since 20th-century Taylorist theories of management emphasize maximizing employee utilization, this requires a significant change in thinking for many organizations.

In enterprises, one indicator of too much WIP is the number of people assigned to more than one project. This pernicious practice inevitably leads to longer lead times and lower quality due to constant context switching. Instead of assigning people to multiple projects, have a centralized team that can provide extra specialist support to teams on demand, but do not assign these people to any teams and carefully monitor their utilization to keep it well below 100%.⁸

Cost of Delay: A Framework for Decentralizing Economic Decisions

One of the biggest problems in product development is delivering valuable features so late that they no longer provide a competitive advantage. As the Maersk case study shows, a major contributor to this problem is batching up features into projects and delivering the results to customers after months of waiting. Value stream mapping often reveals—as it did at Maersk—that much of the time spent waiting was in the analysis phase, with features sitting in the product backlog waiting to be analyzed, estimated, approved, and prioritized for development.

As we saw in [Chapter 3](#), the information provided by these activities is of very low value. Frameworks such as Scrum recommend to prioritize the backlog by business value but offer very little guidance on how to calculate it. Prioritizing by business value also fails to make explicit the time sensitivity of work. However, a powerful framework for making rational prioritization decisions based on economics exists in the form of Cost of Delay.⁹ The team at Maersk reduced cycle times for high-value features through a combination of unbundling features from projects and using Cost of Delay as a lightweight method to identify and prioritize the work with the highest opportunity cost.

To use Cost of Delay, we begin by deciding on the metric we are trying to optimize across our value stream. For organizations engaged in product

⁸ Paweł Brodzinski has a good article on WIP limits for portfolio management: <http://brodzinski.com/2014/06/portfolio-management.html>.

⁹ The concept of Cost of Delay was invented by Don Reinertsen—see [\[reinertsen\]](#).

development, this is typically lifecycle profit, but a logistics company might use a metric such as cost per ton mile (the amount it costs to move one ton a mile). When presented with a decision, we look at all the pieces of work affected by that decision and calculate how to maximize our One Metric That Matters (see [Chapter 4](#)) given the various options we have. For this, we have to work out, for each piece of work, what happens to our key metric when we delay that work (hence “cost of delay”).

Let’s start with a naively simple example to outline the mechanics. Say we have two pieces of work that we could begin when we arrive on Monday morning. We are assured (quite forcefully) that they are both of the highest priority. What should we do?

We start by calculating how much it will cost us if we do not do the work. Task A is to upgrade a core piece of software to a new version that supports encrypting credit card data to meet a compliance deadline, which is two weeks from now. We will be fined \$50,000 per working day that we are not in compliance. The cost of not doing this work is zero up until the point at which the penalty kicks in, and the cost of delay for the work is \$250,000 per week after the deadline. Task A will take two weeks.

Task B is to complete a key feature required by prospective customers, which we have already advertised will be ready one week from now. We expect we will close \$100,000 of business per week when we release this new feature. Furthermore, one of our competitors is right behind us, and we believe they will release a new version of their software with this feature one month from now. Task B takes one week to complete.

The arithmetic is simple, and our options are shown in [Figure 7-9](#). If we perform Task A first, then we delay Task B by two weeks, costing us \$200,000. If we perform Task B first, we delay Task A by a week, costing us \$250,000. Thus we should perform Task A first.

Task A: 2 weeks, CoD \$250k / week
Task B: 1 weeks, CoD \$100k / week



Figure 7-9. How do we prioritize Tasks A and B with Cost of Delay?

We can also calculate what happens if we try and do both tasks simultaneously. Assuming we assign half our capacity to each, it will take us two weeks to complete Task B and three weeks to complete Task A. That leads to a total delay cost of \$350,000. This shows we should still perform Task A first before Task B.

TIP

Use CD3 to Encourage Smaller Blocks of Work

Applying the CD3 method as described in the Maersk case study, Task A's CD3 is 125,000, whereas Task B's CD3 is 100,000, which tells us that Task A is higher priority. Suppose we have an alternative to Task B: Task C. Task C will provide the same value to 80% of the customers who wanted the feature delivered by Task B (\$80,000 per week), but we estimate that completing C will take half of the time of B (.5 week). Task C's CD3 will be 160,000, making it higher priority than Task A. Using CD3 consistently has an important side effect—it encourages us to break work into smaller and more valuable chunks.

There are several consequences to using Cost of Delay. By calculating cost of delay for each feature, we no longer rely solely on a product owner to estimate the business value for the stories in the backlog, which is a poor way to prioritize since this person must constantly recalculate in order to take into account the time sensitivity of business value. Instead, given that we have limited capacity, we think of prioritization as choosing what to delay.

When development capacity becomes available, the team simply picks the item with the highest delay cost *at that time*. This is a key advantage of using Cost of Delay: following the Principle of Mission, it allows everybody in the organization to make rational, transparent economic decisions without the need for command-and-control mechanisms such as onerous reviews, approvals, and prioritization by the most senior person in the room. We separate out several concerns which can each be addressed independently at the correct level of the organization:

1. *What is the economic metric we are trying to optimize?* (Remember the One Metric That Matters from [Chapter 4](#).) Communicating this metric is the responsibility of leadership.
2. *What is the impact on this metric of delaying each piece of work?* Calculating this delay cost is the key goal of analysis.
3. *How should we schedule and prioritize work?* This can be determined autonomously by teams given the information from points 1 and 2 above.

Furthermore, Cost of Delay provides us with an *economic argument* for limiting work in process. As we saw in the example above, the delay cost of trying to perform both A and B simultaneously was greater than completing the individual tasks sequentially.

Of course, the example above is very simplistic. First of all, we assumed that the cost of delay would remain constant over time. This is rarely the case in real life. For example, the cost of delay for Task A rises to \$50,000 per day right after the last day we can start the upgrade work to complete it in time to

meet the external deadline. But for Task B, the amount of business we can close is likely to be time sensitive, given that our competitors will soon have a similar feature.

The time sensitivity of the cost of delay is captured in an *urgency profile*. Urgency profiles that might be used in real life for Task A and Task B are shown in Figure 7-10. Cost of delay, on the y axis, represents the amount of money it costs *per unit time* to delay the work. To calculate the total *delay cost*, we measure the shaded area under the graph. Although in theory there are many possible urgency profiles, in general almost all tasks in a given organization will fit a handful of standard profiles. These can be modeled within a Kanban system by using classes of service.

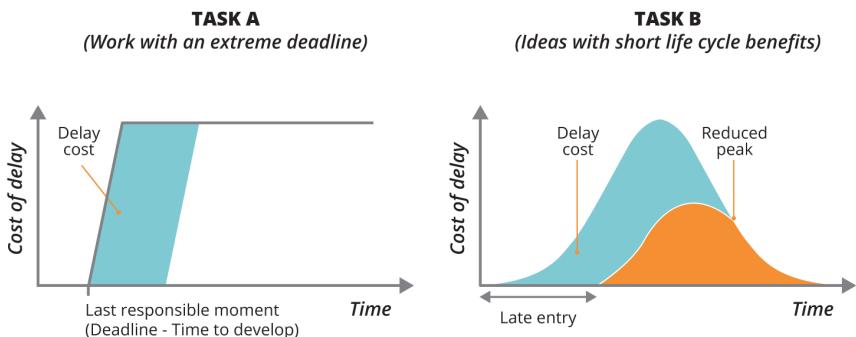


Figure 7-10. Urgency profiles for Task A and Task B

The second problem we face is that in many cases, it is extremely hard to get a precise dollar number for cost of delay. To arrive at a number, we typically make several assumptions and include multiple factors. For example, not completing Task A on time might lead to a loss of customers' confidence in our ability to keep their data safe, which could impact future sales. It's important to make these assumptions explicit, visible, and recorded alongside the work being discussed so we can validate them. The most important thing to bear in mind is that we are aiming for *accuracy*, not *precision*, in our estimates. If the uncertainty around the cost of delay is very high, this means we are in the “explore” domain and must test our assumptions, perhaps using one of the techniques, such as modeling a key metric with Monte Carlo simulations or testing business hypothesis with MVPs, discussed in Part II.

One of the major benefits of Cost of Delay is that instead of arguing about answers to the questions we pose, we can reason about the assumptions we make in our models and focus on validating them. Cost of Delay enables a key cultural change—from political fights over whose work is more important to exposing and validating our assumptions and their effect on economic

variables. This does require a certain level of organizational maturity. As with all process changes, we recommend starting off with a product where people actually *want* to try using Cost of Delay, and providing the necessary support to experiment with it.

Applying Cost of Delay across the enterprise allows us to decentralize decision making and bring decisions under economic control. To do this, we have to change the way we think about managing work, particularly the role of the group responsible for prioritizing decisions at the portfolio and program level (often known as the project management office, or PMO). In the Cost of Delay model, the key responsibility of this group changes from making the actual decisions to creating and updating the *framework* for making decisions: working with finance and individual projects to create standard urgency profiles and templates, gathering and analyzing data, rolling out cost of delay across the organization, and creating feedback loops to continuously improve the quality of the decision-making process. The actual decisions can then be made by teams on the ground, based on the cost of delay information updated when there is new work or when the assumptions underlying existing calculations change. This is a large change that will require executive support, a pilot, and improvements based on learning before it is ready to be adopted by the organization as a whole.

Finally, there are things that you should not do with Cost of Delay. Avoid simply adding cost of delay on top of existing prioritization methods. Its purpose is to improve the quality of demand by enabling us to identify and avoid lower-value work whilst shortening lead times for high-value work. If we turn Cost of Delay into a heavyweight process that sits alongside existing processes, we will not achieve that goal. Cost of Delay provides the biggest payoff when queues are large—in other words, when there is a long lead time. It is essentially a *countermeasure* that becomes valuable when there is too much work in the system. If you don’t have large queues and too much work, using Cost of Delay will likely provide little overall value.

In the context of software-based products and services, implementing lean engineering practices is essential to reducing lead times overall and gathering rapid customer feedback as early as possible in the product lifecycle. The ultimate solution for the deficit of throughput—and thus productivity—is to reduce lead times overall by reducing batch sizes, managing work in process, and reducing the cost of delivering value to customers. In terms of measuring value, there is no substitute for shipping and getting real customer feedback.

Conclusion

In high-performing organizations, leadership and management has a sharp focus on the *value* the organization is creating for its customers. Working

scientifically toward challenging goals, which leads to identifying and removing or avoiding no-value-add activity, is the essence of Lean Thinking, and this requires a significant mindset change for most organizations. Value stream mapping is a powerful tool to visualize our work, so we know our current conditions and create a shared understanding of where we are and where we want to get to. An effective value stream mapping exercise can be truly eye-opening by allowing teams to see, for the first time, the flow of work through the organization in response to customer demand, and their true contribution within it. The output of a value stream mapping exercise is then used to set target conditions for the Improvement Kata described in [Chapter 6](#). We can then use the Kanban Method to manage the flow of work through the value stream, set and update WIP limits and classes of service, and create a pull system to increase flow.

Improving the flow of work through the organization is one side of the coin. The other is making sure we are working on the right things. Cost of Delay provides a way to measure the value of time, enabling teams to make transparent prioritization decisions. By quantifying the value of the work we are doing, we can avoid doing low-value work. If we limit work in process across the value stream and only work on the highest-value tasks, we can rapidly reduce time-to-market for work which is the highest value to our customers.

Questions for readers:

- How is work prioritized in your team or organization? Do you use an economic model or do HiPPOs decide?
- How would you try to implement an economic model for your current backlog? Why not look at the items within your current iteration queue and estimate the Cost of Delay for each?
- What is your current lead time for changes? How could you reduce batch sizes in order to decrease lead time?
- How do you discover the economic impact of your work once it is delivered? Do the team only focus on input and output metrics, such as velocity? How could you work with finance, product management, or other departments to understand business outcomes and bottom-line impacts of your work?
- Do you batch features up into projects as part of your planning and funding processes? How might you unbundle projects and move to a model where you incrementally fund and deliver only the high-value work (we discuss financial management in [Chapter 13](#))? How would you coordinate work across value streams?

Adopt Lean Engineering Practices

Cease dependence on mass inspection to achieve quality. Improve the process and build quality into the product in the first place.

— W. Edwards Deming

An effective innovation capability relies on being able to frequently test ideas with real users. Crucially, the rate at which we can learn, update our product or prototype based on feedback, and test again, is a powerful competitive advantage. This is the value proposition of the lean engineering practices we describe in this chapter. Andy Hertzfeld, one of the engineers who worked on the original Apple Macintosh, notes that “instead of arguing about new software ideas, we actually tried them out by writing quick prototypes, keeping the ideas that worked best, and discarding the others. We always had something running that represented our best thinking at the time.”¹

In many organizations, getting software deployed in an integrated production-like environment is a process that still takes days or even weeks. But organizations that treat software as a competitive advantage rather than a necessary evil invest substantially in reducing this lead time. For a sense of what’s possible at scale, in May of 2011, Amazon achieved a mean time between deployments to *production systems* of 11.6 seconds, with up to 1,079 such deployments in a single hour, aggregated across the thousands of services that comprise Amazon’s platform. Some of these deployments affected upwards of

¹ <http://bit.ly/1v70zdR>

10,000 hosts.² Amazon, of course, is subject to regulations such as Sarbanes-Oxley and PCI-DSS.

A major reason Amazon has invested in this capability is to make it extremely cheap and low-risk for employees to design and run safe-to-fail online experiments of the type we describe in [Chapter 9](#) to gather data from real users. In many cases, running an experiment doesn't require going through a bureaucratic change request process. This gives Amazon's cross-functional delivery teams the ability to test out wild ideas—safe in the knowledge that if something goes wrong, the experiment can be turned off with only a tiny percentage of users impacted for a very short time.

Despite the name, continuous delivery is not about deploying to production multiple times a day. The goal of continuous delivery is to make it safe and economic to work in small batches. This in turn leads to shorter lead times, higher quality, and lower costs. It's for these reasons that the HP FutureSmart team rearchitected their firmware from scratch to minimize the lead time between code check-in and validated, releasable software. Finally, continuous delivery results in boring, safe, push-button deployments rather than long, painful ordeals that must be performed outside of business hours.

This chapter is aimed at readers who wish to understand the principles and practices behind continuous delivery. For those who want just the high-level picture, we present an executive summary of lean engineering practices in the next section. Readers may then skip to the final section of this chapter.

The Fundamentals of Continuous Delivery

Continuous delivery is the ability to get changes—experiments, features, configuration changes, bug fixes—into production or into the hands of users safely and quickly in a sustainable way. Let's examine each of those requirements.

Safely

In order to ensure deployments are safe, we construct a *deployment pipeline* which subjects each proposed change to a battery of automated tests of several different types, followed by manual validations such as exploratory testing and usability testing. We then enable push-button deployments of validated builds to downstream test and staging environments, and ultimately to production, release to manufacturing, or an app store (depending on the type of software). A major goal of the deployment pipeline is to detect and reject changes that are risky, contain regressions, or take us outside the envelope of acceptable performance. As a *byproduct* of

² According to Jon Jenkins' talk at Velocity 2011, "Velocity Culture (the unmet challenge in Ops)."

implementing a deployment pipeline, we get an audit trail of where each change has been introduced, what tests have been run against it, which environments it has passed through, who deployed it, and so forth. This information is invaluable as evidence for compliance.

Quickly

We must constantly monitor and reduce the *lead time* for getting changes into the hands of users. Mary and Tom Poppendieck ask, “How long would it take your organization to deploy a change that involves just one single line of code?”³ We reduce lead time by working to *simplify* and *automate* the build, deploy, test, and release process. We must be able to spin up test environments on demand, deploy software packages to them, and run comprehensive automated tests of several varieties rapidly in parallel on a grid of compute resources. Using this process, it is possible to get a high level of confidence that our software is releasable. Typically this involves architecting (or rearchitecting) with testability and deployability in mind. An important side effect of this work is that the product team can get rapid feedback on the quality of their work, and problems are found soon after they are introduced rather than in later integration and testing phases when they are more expensive to fix.

Sustainably

The point of all this is to make it economically viable to work in small batches. The reason large batches of work are released infrequently is because corralling releases is painful and expensive. The mantra of continuous delivery is: “If it hurts, do it more often, and bring the pain forward.” If integration, testing, and deployment are painful, we should aim to perform them every time anybody checks anything into version control. This reveals the waste and inefficiency in our delivery process so we can address it through continuous improvement. However, to make it economic to work in small batches, we need to invest in extensive test and deployment automation and an architecture that supports it.

There are two golden rules of continuous delivery that must be followed by everybody:

1. The team is not allowed to say they are “done” with any piece of work until their code is in trunk on version control and releasable (for hosted services the bar is even higher—“done” means deployed to production). In *The Lean Startup*, Eric Ries argues that for new features that aren’t simple

³ [poppendieck-06], p. 59.

user requests, the team must also have run experiments on real users to determine if the feature achieves the desired outcome.

2. The team must prioritize keeping the system in a deployable state over doing new work. This means that if at any point we are not confident we can take whatever is on trunk in version control and deliver it to users through an automated, push-button process, we need to stop working and fix that problem.⁴

We should emphasize that following these steps consistently will be hard and require discipline—even for small, experienced teams.

TIP

Enforcing Your Definition of “Done”

The HP FutureSmart managers had a simple rule to help enforce these golden rules. Whenever anybody wanted to demonstrate a new feature (which was required to be able to declare it “done”), they would ask if the code had been integrated into trunk, and if the new functionality was going to be demonstrated from a production-like environment by running automated tests. The demonstration could only proceed if the answer was “yes” to both questions.

In [Chapter 6](#) we discussed the enormous increases in quality, productivity, and reductions in cost the HP FutureSmart team was able to achieve. These improvements were made possible by the team putting continuous delivery principles at the heart of their rebuild. The FutureSmart team eliminated the integration and testing phases from their software development process by building integration and testing into their daily work. It was also possible to shift priorities rapidly in response to the changing needs of product marketing and users:⁵

We know our quality within 24 hours of any fix going into the system...and we can test broadly even for small last-minute fixes to ensure a bug fix doesn’t cause unexpected failures. Or we can afford to bring in new features well after we declare “functionality complete”—or in extreme cases, even after we declare a release candidate.

Let’s look at the engineering patterns that enabled the HP FutureSmart team to achieve their eightfold productivity increase.

⁴ This is the concept of *jidoka* in the Toyota Production System as applied to software delivery.

⁵ [\[gruver\]](#), p. 60.

Continuous Integration and Test Automation

In many development teams, it is common for developers to work on long-lived branches in version control. On small, experienced co-located teams this can be made to work. However, the inevitable outcome of scaling this process is “integration hell” where teams spend days or weeks integrating and stabilizing these branches to get the code released. The solution is for all developers to work off trunk and to integrate their work into trunk at least once per day. In order to be able to do this, developers need to learn how to break down large pieces of work into small, incremental steps that keep trunk working and releasable.

We validate that trunk is working by building the application or service every time a change to it is made in version control. We also run unit tests against the latest version of the code, and give the team feedback within a few minutes if the build or test process fails. The team must then either fix the problem or—if the problem cannot be fixed in a few minutes—revert the change. Thus we ensure that our software is always in a working state during the development process.

Continuous integration is the practice of working in small batches and using automated tests to detect and reject changes that introduce a regression. It is, in our opinion, the most important technical practice in the agile canon, and it forms the foundation of continuous delivery, for which we require in addition that each change keeps the code on trunk releasable. However, that can be hard to adopt for teams that are not used to it.

In our experience, people tend to fall into two camps: those who can’t understand how it is possible (particularly at scale) and those who can’t believe people could work in any other way. We assure you that it is possible, both at small scale and large scale, whatever your domain.

Let’s first address the scale problem with two examples. First, the HP FutureSmart case study demonstrates continuous integration being effective with a distributed team of 400 people working on an embedded system. Second, we’ll note that almost all of Google’s 10,000+ developers distributed over 40 offices work off a single code tree. Everyone working off this tree develops and releases from trunk, and all builds are created from source. 20 to 60 code changes are submitted every minute, and 50% of the codebase changes every month.⁶ Google engineers have built a powerful continuous integration system

⁶ <http://bit.ly/1v70LcY>

that, in 2012, was running over 4,000 builds and 10 million test suites (approximately 60 million tests) every day.⁷

Not only is continuous integration possible on large, distributed teams—it is the only process that is known to scale effectively without the painful and unpredictable integration, stabilization, or “hardening” phases associated with other approaches, such as release trains or feature branches. Continuous delivery is designed to eliminate these activities.

Fundamentals of Test Automation

As can be seen from the Google and HP FutureSmart examples, continuous integration relies on comprehensive test automation. Test automation is still controversial in some organizations, but it is impossible to achieve short lead times and high-quality releases without it. Test automation is an important and complex topic about which many good books have been written,⁸ but here are some of the most important points:

- Test automation is emphatically not about reducing the number of testers—but test automation does change the role and the skills required of testers. Testers should be focused on exploratory testing and working with developers to create and curate suites of automated tests, not on manual regression testing.
- It is impossible to evolve high-quality automated test suites unless testers collaborate with developers in person (irrespective of team or reporting structures). Creating maintainable suites of automated tests requires strong knowledge of software development. It also requires that the software be designed with test automation in mind, which is impossible when developers aren’t involved in testing.
- Test automation can become a maintenance nightmare if automated test suites are not effectively curated. A small number of tests that run fast and reliably detect bugs is better than a large number of tests that are flaky or constantly broken and which developers do not care about.
- Test automation must be designed with parallelization in mind. Running tests in parallel enables developers to get fast feedback and prevents bad practices such as dependencies between tests.
- Automated tests complement other types of testing such as exploratory testing, usability testing, and security testing. The point of automated testing is to validate core functionality and detect regressions so we don’t waste time trying to manually test (or deploy) versions of the software that contain serious problems.
- Reliable automated tests require comprehensive configuration and infrastructure management. It should be possible to create a production-like virtual test environment on demand, either within the continuous integration environment or on a developer workstation.

⁷ <http://bit.ly/1v70NBG>

⁸ We recommend [freeman] and [crispin].

- Only spend time and effort on test automation for products or features once they have been validated. Test automation for experiments is wasteful.

The main objection to continuous integration comes from developers and their managers. Breaking every new feature or rearchitecturing effort into small steps is harder than completing it in isolation on a branch, and takes longer if you are not used to the discipline of working in small batches. That means it may take longer, at first, to declare stories “dev complete.” This may, in turn, drive the development velocity down and create the impression that the team’s efficiency has decreased—raising the blood pressure of development managers.

However, we should not be optimizing for the rate at which we declare things “done” in isolation on a branch. We should optimize for the *overall* lead time—the time it takes us to deliver valuable software to users. Optimizing for “dev complete” time is precisely what causes “integration hell.” A painful and unpredictable “last mile” of integration and testing, in turn, perpetuates the long release cycles that are a major factor in project overruns, poor quality software, higher overall costs, and dissatisfied users.

Are You Really Doing Continuous Integration?

Continuous integration (CI) is hard, and in our experience most teams that say they are practicing it actually aren’t. Achieving CI is not simply a case of installing and running a CI tool; it is a mindset. One of our favorite papers on CI discusses how to do it without any CI tool at all—using just an old workstation, a rubber chicken, and a bell (of course you’ll need more than that on a large development team, but the principles are the same at scale).⁹

To find out if you’re really doing CI, ask your team the following questions:

- Are all the developers on the team checking *into* trunk (not just merging from trunk into their branches or working copies) at least once a day? In other words, are they doing trunk-based development and working in small batches?
- Does every change to trunk kick off a build process, including running a set of automated tests to detect regressions?
- When the build and test process fails, does the team fix the build within a few minutes, either by fixing the breakage or by reverting the change that caused the build to break?

If the answer to any of these questions is “no,” you aren’t practicing continuous integration. In particular, reverting bad changes is an insufficiently practiced technique. At

⁹ James Shore’s *Continuous Integration on a Dollar a Day*.

Google, for example, *anyone* is empowered to revert a bad change in version control, even if it was made by someone on a different team: they prioritize keeping the system working over doing new work.

Of course if you are in-flight working on a large application and using lots of branches, it's not easy to move to continuous integration. In this situation, the goal should be to push teams towards working on trunk, starting with the most volatile branches. In one large organization, it took a year to go from 100 long-lived branches down to about 10–15.

The Deployment Pipeline

Recall the second golden rule of continuous delivery: we must prioritize keeping the system working over doing new work. Continuous integration is an important step towards this goal—but, typically, we wouldn't feel comfortable exposing to users software that has only passed unit tests.

The job of the deployment pipeline is to evaluate every change made to the system, to detect and reject changes which carry high risks or negatively impact quality, and to provide the team with timely feedback on their changes so they can triage problems quickly and cheaply. It takes every check-in to version control, creates packages from that version that are deployable to any environment, and performs a series of tests against that version to detect known defects and to verify that the important functionality works. If the package passes these tests, we should feel confident deploying that particular build of the software. If any stage of the deployment pipeline fails, that version of the software cannot progress any further, and the engineers must immediately triage to find the source of the problem and fix it.

Even the simplest deployment pipeline, such as that shown in [Figure 8-1](#) (a more complex deployment pipeline is shown in [Figure 8-2](#)), enables members of the team to perform push-button deployments of builds that have passed CI to production-like exploratory testing or user acceptance testing environments. It should be possible to provision test environments and deploy any good CI build to them using a fully automated process. This same process should be used to deploy to production.

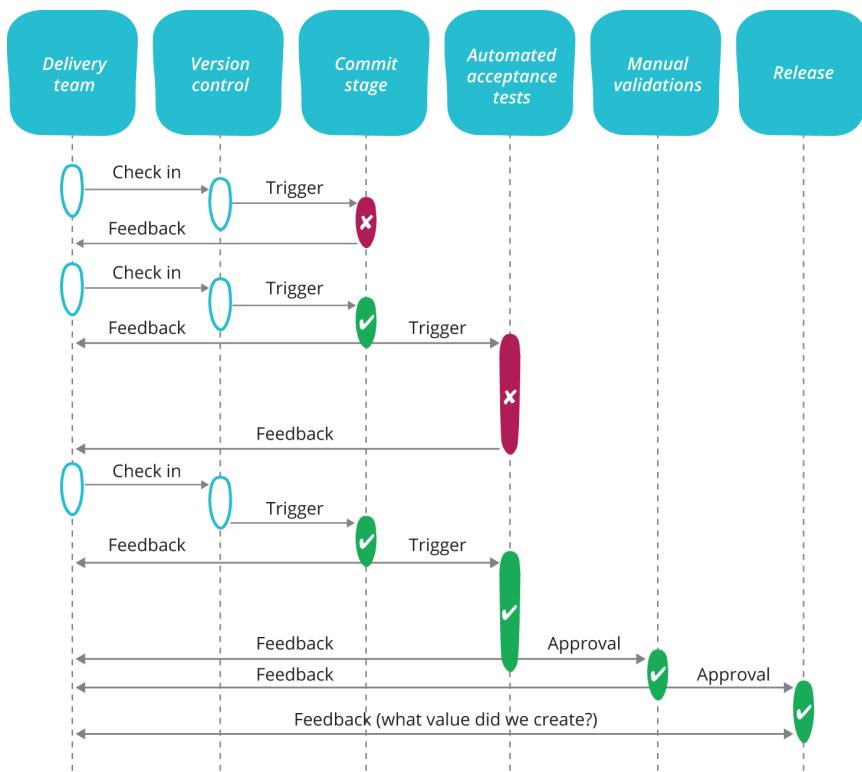


Figure 8-1. Changes moving through a simple deployment pipeline

The deployment pipeline connects together all the steps required to go from check-in to deployment to production (or distribution to an app store). It also connects all the people involved in delivering software—developers, testers, release engineers, and operations—which makes it an important communication tool.

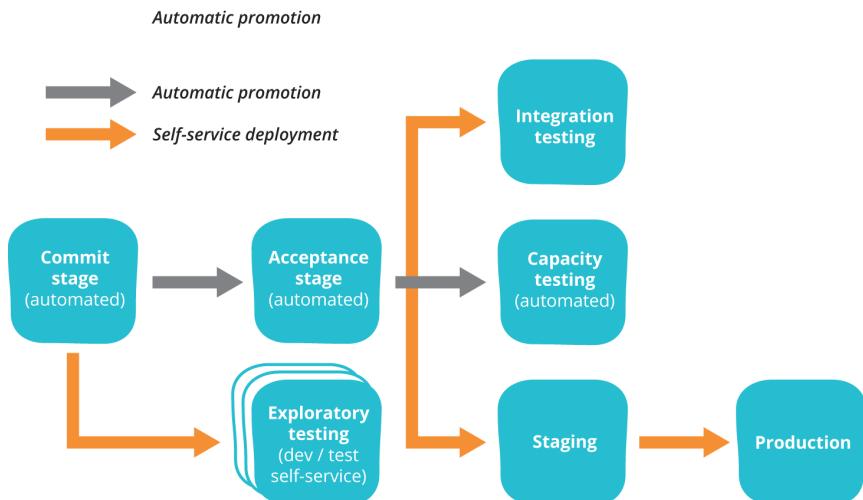


Figure 8-2. A more complex deployment pipeline

The FutureSmart Deployment Pipeline

The FutureSmart team's deployment pipeline allows a 400-person distributed team to integrate 100–150 changes—about 75–100 *thousand* lines of code—into trunk on their 10-million-line codebase every day. Each day, the deployment pipeline produces 10–14 good builds of the firmware out of Level 1. All changes—including feature development and large-scale changes—are made on trunk. Developers commit into trunk several times every week.

All changes to any system—or the environments it runs in—should be made through version control and then promoted via the deployment pipeline. That includes not just source and test code but also database migrations and deployment and provisioning scripts, as well as changes to server, networking, and infrastructure configurations.

The deployment pipeline thus becomes the record of which tests have been run against a given build and what the results were, what builds have been deployed to which environments and when, who approved promotion of a particular build and when, what exactly the configuration of every environment is—and indeed the whole lifecycle of code and infrastructure changes as they move through various environments.

This, in turn, means that a deployment pipeline implementation has several other important uses besides rejecting high-risk or problematic changes to the system:

- You can gather important information on your delivery process, such as statistics of the cycle time of changes (the mean, the standard deviation), and discover the bottlenecks in your process.
- It provides a wealth of information for auditing and compliance purposes. Auditors love the deployment pipeline because it allows them to track every detail of exactly which commands were run on which boxes, what the results were, who approved them and when, and so forth.
- It can form the basis of a lightweight but comprehensive change management process. For example, Australia's heavily regulated National Broadband Network telco used a deployment pipeline to automatically submit change management tickets when changes were made to the production infrastructure, and to automatically update their CMDB when provisioning new systems and performing deployments.¹⁰
- It enables team members to perform push-button deployments of the build of their choice to the environment of their choice. Tools for implementing deployment pipelines typically allow for such approvals to be issued on per-environment basis and for workflows around build promotion to be enforced.

Continuous Delivery and Change Control

Many enterprises have traditionally used change advisory boards or similar change control systems as a way to reduce the risk of changes to production environments. However, the *2014 State of Devops Report*,¹¹ which surveyed over 9,000 individuals across many industries, discovered that approval processes external to development teams do little to improve the stability of services (measured in terms of time to restore service and percentage of failed changes), while acting as a significant drag on throughput (measured in terms of lead time for changes and change frequency). The survey compared external change approval processes with peer-review mechanisms such as pair programming or the use of pull requests. Statistical analysis revealed that when engineering teams held themselves accountable for the quality of their code through peer review, lead times and release frequency improved considerably with negligible impact on system stability. Further data from the report, which supports the use of the techniques discussed in this chapter, is presented in Chapter 14.

The data suggests that it is time to reconsider the value provided by heavyweight change control processes. Peer review of code changes combined with a deployment pipeline provide a powerful, safe, auditable, and high-performance replacement for external approval of changes. The National Broadband Network case study (referenced

¹⁰ See <http://puppetlabs.com/blog/a-deployment-pipeline-for-infrastructure/>

¹¹ [forsgren]

above) shows one method to implement a lightweight change control process which is compatible with frameworks such as ITIL in a regulated environment. For more on compliance and risk management, see [Chapter 12](#).

Implementing continuous delivery requires thinking carefully about systems architecture and process and doing a certain amount of upfront planning. Any manual activities which are repeated should be considered potential waste and thus candidates for simplification and automation. This includes:

Build

It should be possible to create packages from source, deployable to any environment, in a single step using a script that is stored in version control and can be run by any developer.

Provisioning

Anybody should be able to self-service a test environment (including network configuration, host configuration, any required software and applications) in a fully automated fashion. This process should also use information and scripts that are kept in version control. Changes to environment configuration should always be made through version control, and it should be cheap and painless to kill existing boxes and re-provision from source.

Deploy

Anybody should be able to deploy application packages to any environment they have access to using a fully automated process which uses scripts kept in version control.

Test

It should be possible for any developer to run the complete automated test suite on their workstation, as well as any selected set of tests. Test suites should be comprehensive and fast, and contain both unit and acceptance-level tests.

We require, as a foundation for automation, excellent configuration management. In particular, everything required to reproduce your production system and to build, test, and deploy your services needs to be in version control. That means not just source code but build, test, and deployment scripts, infrastructure and environment configuration, database schemas and migration scripts, as well as documentation.

Decouple Deployment and Release

The most important principle for doing low-risk releases is this: decouple *deployment* and *release*. To understand this principle, we must first define these terms.

Deployment is the installation of a given version of a piece of software to a given environment. The decision to perform a deployment—including to production—should be a purely technical one. *Release* is the process of making a feature, or a set of features, available to customers. Release should be a purely business decision.

Often, these two terms are treated as synonyms—that is, we use deployment as our primary mechanism for performing releases. This has a very serious negative consequence: it couples the technical decision to deploy to the business decision to release. This is a major reason why organizational politics gets injected into the deployment process, to the detriment of all.

There are a number of techniques for deploying software to a production environment safely without making its functionality available to users—so we can validate that our system behaves correctly. The simplest—and one of the most powerful—is blue-green deployments (sometimes known as black-red deployments). This pattern requires two separate production environments, code-named blue and green. At any time, only one of these is live; in [Figure 8-3](#), it's green.

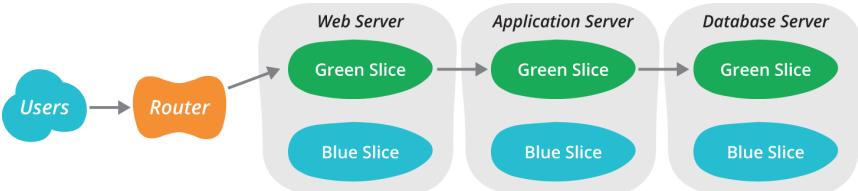


Figure 8-3. Blue-green deployments

When we want to release a new version of our service, we deploy the packages with the new features to the environment that is not currently live (blue in this example) and test it at our leisure. The release process then simply changes the router to point to the blue environment; to roll back, we point the router back to the green environment. A more sophisticated variation gradually ramps up traffic to the blue environment over time.

Crucially for companies with painful deployment process who cannot release during peak hours, blue-green deployments allow the deployment process to be done safely during normal business hours, *days* before a planned release if necessary. The much simpler release process (and rollback, if necessary) can

then be performed at off-peak hours remotely by a much smaller group of people.

Some organizations use their main and backup data centers for their blue and green environments, thus verifying that they can perform a hot disaster-recovery process every time they deploy. However, the blue and green environments do not have to be physically segregated. They can be virtual or logical environments running on the same physical infrastructure (especially since the non-live environment typically consumes very little resources).

Deployment and release can also be decoupled at the feature or component level, instead of the system level, using a technique known as “dark launching.” In his talk on the Facebook release process, release manager Chuck Rossi says that all the major features that will launch in the next six months are *already* in production—you just can’t see them yet. Developers protect new features with “feature flags” so that administrators can dynamically grant access to particular sets of users on a per-feature basis. In this way, features can be made available first to Facebook staff, then to a small set of users as part of an A/B test (see [Chapter 9](#)). Validated features can then be slowly ramped up to 100% of the user base—and switched off under high load or if a defect is found. Feature toggles can also be used to make different feature sets available to different groups of users from a single platform.

TIP

Dark Launching for Mobile Apps

Instead of launching new mobile apps directly to an app store, create a separate brand to deploy and validate them before launching them under your official brand.

Conclusion

Continuous delivery represents an alternative to large-batch development and release processes. It has been adopted by many large engineering organizations across different domains, including heavily regulated industries such as financial services. Despite its origins in web services, this engineering paradigm has been successfully applied to packaged software, firmware, and mobile development. It enables organizations to respond rapidly to changing customer needs and increase software quality while reducing both the risk of release and the cost of software development.

Culture also plays an important role in enabling continuous delivery. A culture in which interactions between development, operations, and information security teams are generally win-win is highly correlated with high performance, as is a culture that is at the “generative” end of Westrum’s typology ([Chapter 1](#)).

As organizations work to implement continuous delivery, they will have to change the way they approach version control, software development, architecture, testing, and infrastructure and database management. Figure 8-4 is synthesized from our study of a number of different organizations.¹²

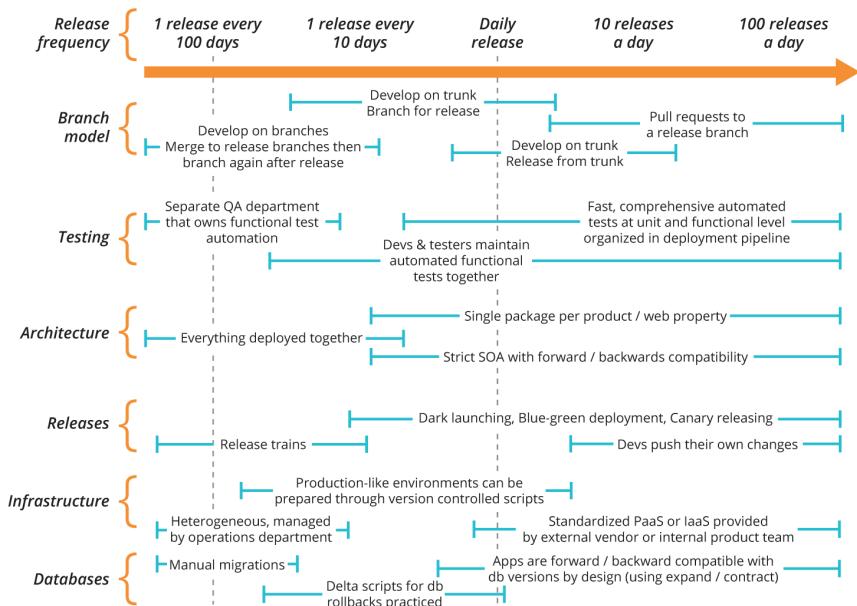


Figure 8-4. Deployment g-forces, courtesy of Paul Hammant

Of course all these areas are interrelated. For example, building a maintainable, comprehensive, automated test suite requires an architecture which allows software to be deployed on local developer workstations, which in turn requires that production-like environments can be set up by version-controlled scripts. Working out what to attack first, in the case of existing systems, can be complex. We discuss evolutionary architectural change in Chapter 10.

We strongly recommend that you start by implementing comprehensive configuration management, continuous integration, and trunk-based development. Also important is creating a culture of test automation with developers, which in turn requires that test environments can be provisioned on demand. In our experience, attempts to attack problems in release or operations, discussed in

¹² This diagram is adapted from one by Paul Hammant, <http://paulhammant.com/2013/03/13/facebook-tbd-take-2>.

Chapter 14, cannot produce significant improvement without continuous integration, test automation, and automated environment provisioning.

Questions for readers:

- What is your definition of “done” in order for a feature to be accepted? Must it—at the very least—be integrated into trunk and demonstrated from a production-like environment by running automated tests?
- Are you practicing continuous integration as we define it in this book? How would you work to introduce it?
- Are the relationships between developers, testers, and IT operations personnel collaborative or adversarial? What steps might you take to improve them?
- Are your production deployments painful, “big bang” events that involve planned outages outside of business hours? How could you change them so as to perform more of the work within normal business hours?

Take an Experimental Approach to Product Development

The difficulty in defining quality is to translate future needs of the user into measurable characteristics, so that a product can be designed and turned out to give satisfaction at a price the user will pay.

— Walter Shewhart

Up to now, we have spent the whole of [Part III](#) showing how to improve the speed at which we can deliver value to customers. In this chapter, we switch focus to discuss *alignment*—how to use the capability we have developed to make sure we are building the right things for customers, users, and our organization.

In [Chapter 7](#), we showed how to use the Cost of Delay to prioritize work. In an organization where IT is essentially a service provider, this is an effective way to avoid working on low-value tasks that consume precious time and resources. However, in high-performance organizations, projects and requirements are not tossed over the wall to IT to build. Rather, engineers, designers, testers, operations staff, and product managers work in partnership on creating high-value outcomes for customers, users, and the organization as a whole. Furthermore, these decisions—made locally by teams—take into account the wider strategic goals of the organization.

In [Chapter 6](#) we described the Improvement Kata, an iterative approach to process improvement in which we set target conditions for the next iteration and then let teams decide what work to do in order to achieve those target conditions. The key innovation we present in this chapter is to use the same process to manage product development. Instead of coming up with

requirements or use cases and putting them into a backlog so that teams build them in priority order, we describe, in measurable terms, the *business outcomes* we want to achieve in the next iteration. It is then up to the teams to discover ideas for features which will achieve these business outcomes, test them, and build those that achieve the desired outcomes. In this way, we harness the skill and ingenuity of the entire organization to come up with ideas for achieving business goals with minimal waste and at maximum pace.

As an approach to running agile software development at scale, this is different from most frameworks. There's no program-level backlog; instead, teams create and manage their own backlogs and are responsible for collaborating to achieve business goals. These goals are defined in terms of target conditions at the program level and regularly updated as part of the Improvement Kata process (see [Chapter 6](#)). Thus the responsibility for achieving business goals is pushed down to teams, and teams focus on business outcomes rather than measures such as the number of stories completed (team velocity), lines of code written, or hours worked. Indeed, the goal is to *minimize* output while maximizing outcomes: the fewer lines of code we write and hours we work to achieve our desired business goals, the better. Enormous, overly complex systems and burned-out staff are symptoms of focusing on output rather than outcomes.

One thing we don't do in this chapter (or indeed this book) is prescribe what processes teams should use to manage their work. Teams can—and should—be free to choose whatever methods and processes work best for them. Indeed, in the HP FutureSmart program, different teams successfully used different methodologies and there was no attempt to impose a “standard” process or methodology across the teams. What is important is that the teams are able to work together effectively to achieve the target conditions.

Therefore, we don't present standard agile methods such as XP, Scrum, or alternatives like Kanban. There are several excellent books that cover these methods in great detail, such as David Anderson's *Kanban: Successful Evolutionary Change for Your Technology Business*,¹ Kenneth S. Rubin's *Essential Scrum: A Practical Guide to the Most Popular Agile Process* (Addison-Wesley), and Mitch Lacey's *The Scrum Field Guide: Practical Advice for Your First Year* (Addison-Wesley). Instead, we discuss how teams can collaborate to define approaches to achieve target conditions, then design experiments to test their assumptions.

The techniques described in this chapter require a high level of trust between different parts of the organization involved in the product development value

¹ [\[anderson\]](#)

stream, as well as between leaders, managers, and those who report to them. They also require high-performance teams and short lead times. Thus, unless these foundations (described in previous chapters in this part) are in place, implementing these techniques will not produce the value they are capable of.

Using Impact Mapping to Create Hypotheses for the Next Iteration

The outcome of the Improvement Kata’s iteration planning process (described in [Chapter 6](#)) is a list of measurable target conditions we wish to achieve over the next iteration, describing the *intent* of what we are trying to achieve and following the Principle of Mission (see [Chapter 1](#)). In this chapter, we describe how to use the same process to drive product development. We achieve this by creating target conditions based on customer and organizational outcomes as part of our iteration planning process, in addition to process improvement target conditions. This enables us to use program-level continuous improvement for product development too, by adopting a *goal-oriented* approach to requirements engineering.

Our product development target conditions describe customer or business goals we wish to achieve, which are driven by our product strategy. Examples include increasing revenue per user, targeting a new market segment, solving a given problem experienced by a particular persona, increasing the performance of our system, or reducing transaction cost. However, we do not propose solutions to achieve these goals or write stories or features (especially not “epics”) at the program level. Rather, it is up to the teams within the program to decide how they will achieve these goals. This is critical to achieving high performance at scale, for two reasons:

- The initial solutions we come up with are unlikely to be the best. Better solutions are discovered by creating, testing, and refining multiple options to discover what best solves the problem at hand.
- Organizations can only move fast at scale when the people building the solutions have a deep understanding of both user needs and business strategy and come up with their own ideas.

A program-level backlog is not an effective way to drive these behaviors—it just reflects the almost irresistible human tendency to specify “the means of doing something, rather than the result we want.”²

² [\[gilb-88\]](#), p. 23.

TIP

Getting to Target Conditions

Goal-oriented requirements engineering has been in use for decades,³ but most people are still used to defining work in terms of features and benefits rather than measurable business and customer outcomes. The features-and-benefits approach plays to our natural bias towards coming up with solutions, and we have to think harder to specify the attributes that an acceptable solution will have instead.

If you have features and benefits and you want to get to target conditions, one simple approach is to ask *why* our customers care about a particular benefit. You may need to ask “*why*” several times to get to something that looks like a real target condition.⁴ It’s also essential to ensure that target conditions have *measurable* acceptance criteria, as shown in Figure 9-1.

Gojko Adzic presents a technique called *impact mapping* to break down high-level business goals at the program level into testable hypotheses. Adzic describes an impact map as “a visualization of scope and underlying assumptions, created collaboratively by a cross-functional group of stakeholders. It is a mind-map grown during a discussion facilitated by answering the following questions: 1. Why? 2. Who? 3. How? 4. What?”⁵ An example of an impact map is shown in Figure 9-1.

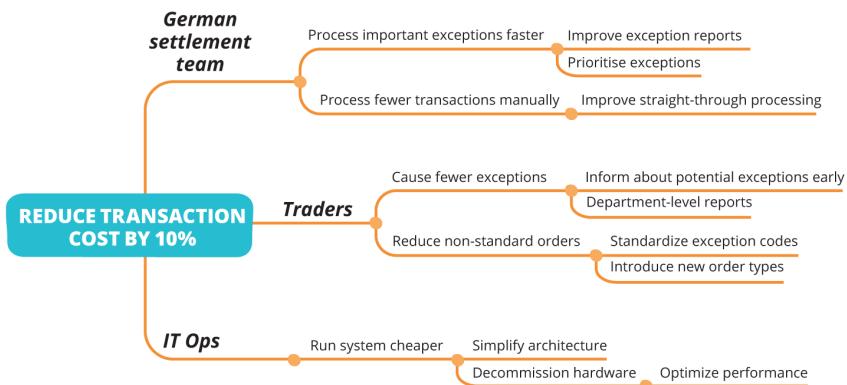


Figure 9-1. An example of an impact map

³ See [yu], [lapouchnian], and [gilb-05] for more on goal-oriented requirements engineering.

⁴ This is an old trick used by Taiichi Ohno, called “the five whys.”

⁵ [adzic], l. 146.

We begin an impact map with a program-level target condition. By stating a target condition, including the *intent* of the condition (why we care about it from a business perspective), we make sure everyone working towards the goal understands the purpose of what they are doing, following the Principle of Mission. We also provide clear acceptance criteria so we can determine when we have reached the target condition.

The first level of an impact map enumerates all the stakeholders with an interest in that target condition. This includes not only the end users who will be affected by the work, but also people within the organization who will be involved or impacted, or can influence the progress of the work—either positively or negatively.

The second level of an impact map describes possible ways the stakeholders can help—or hinder—achieving the target condition. These changes of behavior are the *impacts* we aim to create.

So far, we should have said nothing about possible solutions to move us towards our target condition. It is only at the third level of the impact map that we propose options to achieve the target condition. At first, we should propose solutions that don't involve writing code—such as marketing activities or simplifying business processes. Software development should always be a last resort, because of the cost and complexity of building and maintaining software.

The possible solutions proposed in the impact map are *not* the key deliverable. Coming up with possible solutions simply helps us refine our thinking about the goal and stakeholders. The solutions we come up with at this stage are unlikely to be the best—we expect, rather, that the people working to deliver the outcomes will come up with better options and evaluate them to determine which ones will best achieve our target condition. The impact map can be considered a set of assumptions—for example, in [Figure 9-1](#), we assume that standardizing exception codes will reduce nonstandard orders, which will reduce the cost of processing nonstandard transactions.

For this tool to work effectively, it's critical to have the right people involved in the impact-mapping exercise. It might be a small, cross-functional team including business stakeholders, technical staff, designers, QA (where applicable), IT operations, and support. If the exercise is conducted purely by business stakeholders, they will miss the opportunity to examine the assumptions behind the target conditions and to get ideas from the designers and engineers who are closest to the problem. One of the most important goals of impact mapping is to create a shared understanding between stakeholders, so not involving them dooms it to irrelevance.

Once we have a prioritized list of target conditions and impact maps created collaboratively by technical and business people, it is up to the teams to determine the shortest possible path to the target condition.

This tool differs in important ways from many standard approaches to thinking about requirements. Here are some of the important differences and the motivations behind them:

There are no lists of features at the program level

Features are simply a mechanism for achieving the goal. To paraphrase Adzic, *if achieving the target condition with a completely different set of features than we envisaged won't count as success, we have chosen the wrong target condition.* Specifying target conditions rather than features allows us to rapidly respond to changes in our environment and to the information we gather from stakeholders as we work towards the target condition. It prevents “feature churn” during the iteration. Most importantly, it is the most effective way to make use of the talents of those who work for us; this motivates them by giving them an opportunity to pursue mastery, autonomy, and purpose.

There is no detailed estimation

We aim for a list of target conditions that is a stretch goal—in other words, if *all* our assumptions are good and *all* our bets pay off, we think it would be possible to achieve them. However, this rarely happens, which means we may not achieve some of the lower-priority target conditions. If we are regularly achieving much less, we need to rebalance our target conditions in favor of process improvement goals. Keeping the iterations short—2–4 weeks initially—enables us to adjust the target conditions in response to what we discover during the iteration. This allows us to quickly detect if we are on a wrong path and try a different approach before we overinvest in the wrong things.

There are no “architectural epics”

The people doing the work should have complete freedom to do whatever improvement work they like (including architectural changes, automation, and refactoring) to best achieve the target conditions. If we want to drive out particular *goals* which will require architectural work, such as compliance or improved performance, we specify these in our target conditions.

Performing User Research

Impact mapping provides us with a number of possible solutions and a set of assumptions for each candidate solution. Our task is to find the shortest path to the target condition. We select the one that seems shortest, and validate the solution—along with the assumptions it makes—to see if it really is capable of

delivering the expected value (as we have seen, features often fail to deliver the expected value). There are multiple ways to validate our assumptions.

First, we create a *hypothesis* based on our assumption. In *Lean UX*, Josh Seiden and Jeff Gothelf suggest the template shown in [Figure 9-2](#) to use as a starting point for capturing hypotheses.⁶

We believe that

[building this feature]

[for these people]

will achieve [this outcome].

We will know we are successful when we see

[this signal from the market].

Figure 9-2. Jeff Gothelf's template for hypothesis-driven development

In this format, we describe the parameters of the experiment we will perform to test the value of the proposed feature. The *outcome* describes the target condition we aim to achieve.

As with the agile story format, we summarize the *work* (for example, the feature we want to build or the business process change we want to make) in a few words to allow us to recall the conversation we had about it as a team. We also specify the *persona* whose behavior we will measure when running the experiment. Finally, we specify the signal we will measure in the experiment. In online controlled experiments, discussed in the next section, this is known as the *overall evaluation criterion* for the experiment.

Once we have a hypothesis, we can start to design an experiment. This is a cross-functional activity that requires collaboration between design, development, testing, techops, and analysis specialists, supported by subject matter experts where applicable. Our goal is to *minimize* the amount of work we must perform to gather a sufficient amount of data to validate or falsify the assumptions of our hypothesis. There are multiple types of user research we can perform to test our hypothesis, as shown in [Figure 9-3](#).⁷ For more on different types of user research, read *UX for Lean Startups* (O'Reilly) by Laura Klein.

⁶ [\[gothelf\]](#), p. 23.

⁷ This diagram was developed by Janice Fraser; see <http://slidesha.re/1v715bL>.



Figure 9-3. Different types of user research, courtesy of Janice Fraser

The key outcome of an experiment is *information*: we aim to reduce the uncertainty as to whether the proposed work will achieve the target condition. There are many different ways we can run experiments to gather information. Bear in mind that experiments will often have a negative or inconclusive result, especially in conditions of uncertainty; this means we'll often need to tune, refine, and evolve our hypotheses or come up with a new experiment to test them.

The key to the experimental approach to product development is that *we do no major new development work without first creating a hypothesis* so we can determine if our work will deliver the expected value.⁸

Online Controlled Experiments

In the case of an internet-based service, we can use a powerful method called an *online controlled experiment*, or A/B test, to test a hypothesis. An A/B test is a randomized, controlled experiment to discover which of two possible versions of a web page produces better outcome. When running an A/B test, we

⁸ In many ways, this approach is just an extension of test-driven development. Chris Matts came up with a similar idea he calls *feature injection*.

prepare two versions of a page: a control (typically the existing version of the page) and a new treatment we want to test. When a user first visits our website, the system decides which experiments that user will be a subject for, and for each experiment chooses at random whether they will view the control (A) or the treatment (B). We instrument as much of the user's interaction with the system as possible to detect any differences in behavior between the control and the treatment.

Most Good Ideas Actually Deliver Zero or Negative Value

Perhaps the most eye-opening result of A/B testing is how many apparently great ideas do not improve value, and how utterly impossible it is to distinguish the lemons in advance. As discussed in [Chapter 2](#), data gathered from A/B tests by Ronny Kohavi, who directed Amazon's Data Mining and Personalization group before joining Microsoft as General Manager of its Experimentation Platform, reveal that 60%–90% of ideas *do not improve the metric they were intended to improve*.

Thus if we're not running experiments to test the value of new ideas before completely developing them, the chances are that about 2/3 of the work we are doing is of either *zero* or *negative* value to our customers—and certainly of negative value to our organization, since this work costs us in three ways. In addition to the cost of developing the features, there is an opportunity cost associated with more valuable work we could have done instead, and the cost of the new complexity they add to our systems (which manifests itself as the cost of maintaining the code, a drag on the rate at which we can develop new functionality, and often, reduced operational stability and performance).

Despite these terrible odds, many organizations have found it hard to embrace running experiments to measure the value of new features or products. Some designers and editors feel that it challenges their expertise. Executives worry that it threatens their job as decision makers and that they may lose control over the decisions.

Kohavi, who coined the term "HiPPO," says his job is "to tell clients that their new baby is ugly," and carries around toy rubber hippos to give to these people to help lighten the mood and remind them that most "good" ideas aren't, and that it's impossible to tell in the absence of data which ones will be lemons.

By running the experiment with a large enough number of users, we aim to gather enough data to demonstrate a statistically significant difference between A and B for the business metric we care about, known as the overall evaluation criterion, or OEC (compare the One Metric That Matters from [Chapter 4](#)). Kohavi suggests optimizing for and measuring *customer lifetime value* rather than short-term revenue. For a site such as Bing, he recommends using a weighted sum of factors such as time on site per month and visit frequency per user, with the aim being to improve the overall customer experience and get them to return.

Unlike data mining, which can only discover correlations, A/B testing has the power to show a *causal relationship* between a change on a web page and a corresponding change in the metric we care about. Companies such as Amazon and Microsoft typically run hundreds of experiments in production at any one time and test every new feature using this method before rolling it out. Every visitor to Bing, Microsoft's web search service, will be participating in about 15 experiments at a time.⁹

Using A/B Testing to Calculate the Cost of Delay for Performance Improvements

At Microsoft, Ronny Kohavi's team wanted to calculate the impact of improving the performance of Bing searches. They did it by running an A/B test in which they introduced an artificial server delay for users who saw the "B" version. They were able to calculate a dollar amount for the revenue impact of performance improvements, discovering that "an engineer that improves server performance by 10 msec more than pays for his fully-loaded annual costs." This calculation can be used to determine the cost of delay for performance improvements.

When we create an experiment to use as part of A/B testing, we aim to do much less work than it would take to fully implement the feature under consideration. We can calculate the maximum amount we should spend on an experiment by determining the expected value of the information we will gain from running it, as discussed in [Chapter 3](#) (although we will typically spend much less than this).

In the context of a website, here are some ways to reduce the cost of an experiment:

Use the 80/20 rule and don't worry about corner cases

Build the 20% of functionality that will deliver 80% of the expected benefit.

Don't build for scale

Experiments on a busy website are usually only seen by a tiny percentage of users.

Don't bother with cross-browser compatibility

With some simple filtering code, you can ensure that only users with the correct browser get to see the experiment.

⁹ <http://www.infoq.com/presentations/controlled-experiments>

Don't bother with significant test coverage

You can add test coverage later if the feature is validated. Good monitoring is much more important when developing an experimentation platform.

An A/B Test Example

Etsy is a website where people can sell handcrafted goods. Etsy uses A/B testing to validate all major new product ideas. In one example, a product owner noticed that searching for a particular type of item on somebody's storefront comes up with zero results, and wanted to find out if a feature that shows similar items from somebody else's storefront would increase revenue. To test the hypothesis, the team created a very simple implementation of the feature. They used a configuration file to determine what percentage of users will see the experiment.

Users hitting the page on which the experiment is running will be randomly allocated either to a control group or to the group that sees the experiment, based on the weighting in the configuration file. Risky experiments will only be seen by a very small percentage of users. Once a user is allocated to a bucket, they stay there across visits so the site has a consistent appearance to them.

Making It Safe to Fail

A/B testing allows teams to define the constraints, limits, or thresholds to create a safe-to-fail experiment. The team can define the control limit of a key metric before testing so they can roll back or abort the test if this limit is reached (e.g., conversion drops below a set figure). Determining, sharing, and agreeing upon these limits with all stakeholders before conducting the experiment will establish the boundaries within which the team can experiment safely.

Users' subsequent behavior is then tracked and measured as a cohort—for example, we might want to see how many then make it to the payment page. Etsy has a tool, shown in [Figure 9-4](#), which measures the difference in behavior for various endpoints and indicates when it has reached statistical significance at a 95% confidence interval. For example, for “site—page count,” the bolded “+0.26%” indicates the experiment produces a statistically significant 0.26% improvement over the control. Experiments typically have to run for a few days to produce statistically significant data.

Generating a change of more than a few percent in a business metric is rare, and can usually be ascribed to Twyman’s Law: “If a statistic looks interesting or unusual it is probably wrong.”

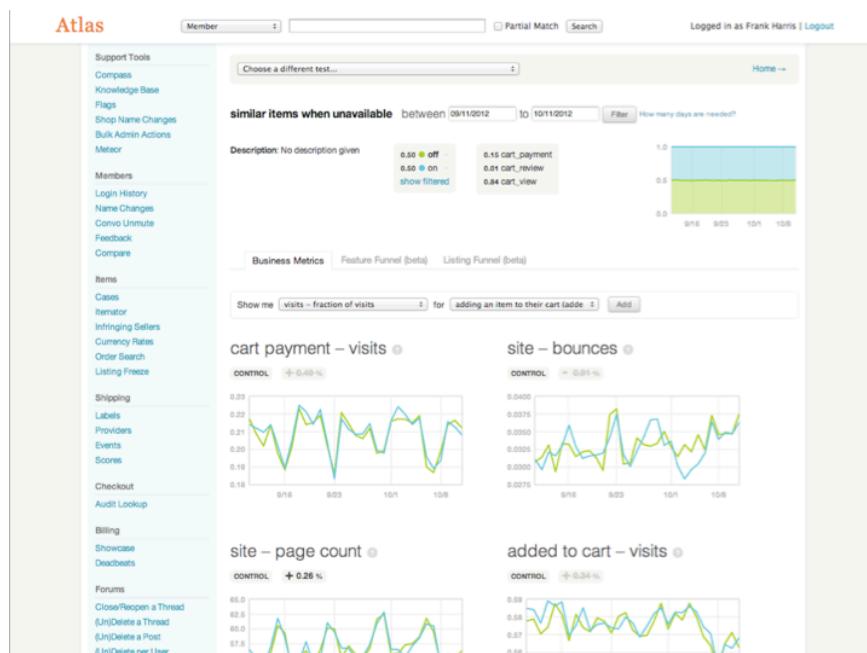


Figure 9-4. Measuring changes in user behavior using A/B testing

If the hypothesis is validated, more work can be done to build out the feature and make it scale, until ultimately the feature is made available to all users of the site. Turning the visibility to 100% of users is equivalent to publicly releasing the feature—an important illustration of the difference between deployment and release which we discussed in [Chapter 8](#). Etsy always has a number of experiments running in production at any time. From a dashboard, you can see which experiments are planned, which are running, and which are completed, which allows people to dive into the current metrics for each experiment, as shown in [Figure 9-5](#).

Etsy Launch Calendar			
Home By Team Add an Item Last Updated More			
Have an upcoming launch? Add an Item			
Date	Name	Team	Notes
Nov 7	Gift Ideas browse pages 🎁	Buyer Experience	This is a gift guide browse destination. Subsections will focus on recipient (for him, for her, for kids, etc.) and price (under \$25, under \$100, etc.). It will work just like all other browse pages. There will be NO HAND ...
Nov 7	Etsy for iPhone (v2.1.1)	Mobile	Example — We submitted the app on Friday. We will be pushing it out when it's approved by Apple; our hope is that it's approved by Wednesday. There will be no coordination with PR or blog post. We may send ...
Nov 2	Winter Holidays browse pages 🎅	Buyer Experience	Example — These are browse pages for the Winter Holidays and will feature subsections for holiday decor, cards, etc. They'll be similar to our holiday merch hub from last year, but much deeper in terms of browsing opportunities. Those in UK ...
Nov 1	Updated treatment of homepage browse links 🎁	Buyer Experience	Example — Over a two week period we observed 4%-5% increases in browse landing page and subsection page views. There were also slight increases in add to cart and listings viewed events. Visits with a search and search events were down ...
Oct 24	Next day availability of DC funds 💰	Payments	We plan to allow established sellers to be able to deposit their funds prior the next day after a sale. Non established sellers will still need to ship items to have available funds.
Oct 23	Reduce one-time hold from 10 days to 5 days	Payments	Whenever a new seller signs up for direct checkout, a 10 day hold is placed on deposits. This also occurs anytime a bank account is updated. We have decided to reduce this standard hold period to 5 days. The main ...
Oct 23	Etsy for iPhone (v2.1) ❤️	Mobile	Example — Update: We have been approved by Apple and will be launching Tuesday, 10/23 at 8am ET. ... Our target submit date to Apple is Wednesday 10/10. Depending on Apple's turnaround time, we expect the app to be ...
Oct 22	Recipient Query Rewriting	Search & Destroy	Example — This didn't move metrics positively or negatively. However we decided to keep it because this is the first step towards using recipient in search, and encouraging users to properly associate their listing w/ a recipient. We will reevaluate how ...
Oct 19	Parcel Insurance for Shipping Labels 💰	Seller Team	Example 1, Example 2 — Rampup started 10/9. Scheduled to finish 10/19.
Oct 18	Search Ads respecting filters	Search & Destroy	This experiment didn't hurt inventory: https://plink.etsycorp.com/en-US/app/search/fastimeline?sid=1350940765.163366&vs=h8m3sk4b Also it looks like CTR might have improved.

Figure 9-5. Experiments currently running at Etsy

Alternatives to A/B Testing

Although we spend a lot of time on A/B testing in this chapter, it is just one of a wide range of experimental techniques for gathering data. User experience designers have a variety of tools to get feedback from users, from lo-fi prototypes to ethnographic research methods such as contextual enquiry, as shown in [Figure 9-3](#). *Lean UX: Applying Lean Principles to Improve User Experience* discusses a number of these tools and how to apply them in the context of hypothesis-driven development.¹⁰

Prerequisites for an Experimental Approach to Product Development

Convincing people to gather—and then pay attention to—real data from experimentation, such as A/B testing, is hard enough. But an experimental, scientific approach to creating customer value has implications for the way we do work, as well as for the way we think about value. As Dan McKinley of Etsy

10 [\[gothelf\]](#)

points out,¹¹ experimentation can't be bolted on to a waterfall product development process. If we get to the end of several weeks (or months) of work and attempt an experiment, there's a very good chance we'll find the huge batch of work we did either has zero effect or makes things worse. At that point we'll have to throw it all away because there is no way to accurately identify the effect of each specific change introduced.

This is an extremely painful decision, and in practice many teams succumb to the sunk cost fallacy by giving undue weight to the investment made to date when taking this decision. They ignore the data and deploy the product as is because shelving the work is considered a total failure, whereas successful deployment of anything into production is perceived as success—so long as it is on time and on budget.

If we're going to adopt a thorough experimental approach, we need to change what we consider to be the outcome of our work: not just validated ideas but the information we gain in the course of running the experiments. We also need to change the way we think about developing new ideas; in particular, it's essential to work in small batches and test every assumption behind the idea we are validating. This, in turn, requires that we implement continuous delivery, as described in [Chapter 8](#).

Working in small batches creates *flow*—a key element of Lean Thinking. But small batches are hard to achieve, for both philosophical and technical reasons. Some people have a problem with taking an incremental approach to creating products. A common objection to an experimental approach is that it leads to locally optimal but globally suboptimal decisions, and that it compromises the overall integrity of the product, murdering a beautiful holistic vision by a thousand A/B tests.

While it is certainly possible to end up with an ugly, overcomplex product when teams fail to take a holistic approach to user experience, this is not an inevitable outcome of A/B testing. Experimentation isn't supposed to replace having a vision for your product. Rather, it enables you to evolve your strategy and vision rapidly in response to real data from customers using your products in their environment. A/B testing will not be effective in the absence of a vision and strategy. Product managers, designers, and engineers need to collaborate and apply the lessons of design thinking in order to take a long-term view of the needs of users and establish a direction for the product.

11 <http://slidesha.re/1v71gUs>

TIP

What Is Design Thinking?

Tim Brown, CEO and President of IDEO and one of the key figures in design thinking, says, "As a style of thinking, it is generally considered the ability to combine empathy for the context of a problem, creativity in the generation of insights and solutions, and rationality to analyze and fit solutions to the context." We discuss design thinking and Lean UX further in [Chapter 4](#).

There are two further obstacles to taking an experimental approach to product development. First, designing experiments is tricky: we have to prevent them from interfering with each other, apply alerts to detect anomalies, and design them to produce valid results. At the same time, we want to minimize the amount of work we must do to gather statistically significant data.

Finally, taking a scientific approach to customer and product development requires intensive collaboration between product, design, and technical people throughout the lifecycle of every product. This is a big cultural change for many enterprises where technical staff do not generally contribute to the overall design process.

These obstacles are the reason why we strongly discourage people from adopting the tools discussed in this chapter without first putting in place the foundations described in the earlier chapters in [Part III](#).

Innovation Requires a Culture of Experimentation

Greg Linden, who developed Amazon's first recommendations engine, came up with a hypothesis that showing personalized recommendations at checkout time might convince people to make impulse buys—similar to the rack at the checkout lane in a grocery store but compiled personally for each customer by an algorithm. However, a senior vice-president who saw Greg's demo was convinced it would distract people from checking out. Greg was forbidden to do any further work on the feature.¹²

Linden disobeyed the SVP and put an A/B test into production. The A/B test demonstrated such a clear increase in revenue when people received personalized recommendations at check-out that the feature was built out and launched with some urgency.

Is it even conceivable that an engineer at your company could push an A/B test into production in the face of censure by a senior executive? If the experiment's data proved the executive wrong, how likely is it that the feature would be picked up rather than buried? As Linden writes, "Creativity must flow from everywhere. Whether you are a summer intern or the CTO, any good idea must be able to seek an objective test,

12 <http://bit.ly/1v71kmW>

preferably a test that exposes the idea to real customers. Everyone must be able to experiment, learn, and iterate. Position, obedience, and tradition should hold no power. For innovation to flourish, measurement must rule.”

A culture based on measurement and experimentation is not antithetical to crazy ideas, divergent thinking, and abductive reasoning. Rather, it gives people license to pursue their crazy ideas—by making it easy to gather real data to back up the good crazy and reject the bad crazy. Without the ability to run cheap, safe-to-fail experiments, such ideas are typically trampled by a passing HiPPO or by the mediocrity of decision-by-committee.

One of the most common challenges encountered in software development is the focus of teams, product managers, and organizations on managing cost rather than value. This typically manifests itself in undue effort spent on zero-value-add activities such as detailed upfront analysis, estimation, scope management, and backlog grooming. These symptoms are the result of focusing on maximizing utilization (keeping our expensive people busy) and output (measuring their work product)—instead of focusing on outcomes, minimizing the output required to achieve them, and reducing lead times to get fast feedback on our decisions.

Conclusion

Most ideas—even apparently good ones—deliver zero or negative value to users. By focusing on the *outcomes* we wish to achieve, rather than solutions and features, we can separate *what* we are trying to do from the possible ways to do it. Then, following the Principle of Mission, teams can perform user research (including low-risk, safe-to-fail online experiments) to determine what will actually provide value to customers—and to our organization.

By combining impact mapping and user research with the Improvement Kata framework presented in [Chapter 6](#), we can scale agile software delivery and combine it with design thinking and an experimental approach to product development. This allows us to rapidly discover, develop, and deliver high-value, high-quality solutions to users at scale, harnessing the skill and ingenuity of everybody in the organization.

Questions for readers:

- What happens at your organization when a substantial amount of effort has been invested in an idea that turns out to provide little value to users or the organization, or even to make things worse?
- Have the expected customer outcomes for the features you are working on been quantified? Do you have a way to measure the actual outcomes?

- What kind of user research do you perform on prototypes before releasing them more widely? How might you get that feedback more quickly and cheaply?
- When was the last time you personally observed your product used or discussed in real life?
- Can you think of a cheap way to test the value of the next piece of work in your backlog?

Implement Mission Command

The more alignment you have, the more autonomy you can grant. The one enables the other.

— Stephen Bungay

The best managers figure out how to get great outcomes by setting the appropriate context, rather than by trying to control their people.

— Reed Hastings

In his 2009 presentation on Netflix culture, *Freedom and Responsibility*,¹ CEO Reed Hastings describes a dynamic common to many growing organizations. As organizations get larger, they become more complex in terms of the systems they are evolving and running, the business environment in which they operate, and their ability to “get things done.” Eventually the business becomes too complex to run informally, and formal processes are put in place to prevent it from descending into chaos. Processes provide a certain level of predictability, but they slow us down and do little to prevent bad outcomes from events that cannot be managed through process (for example, work that goes according to plan but does not deliver customer value).

Management through process control is acceptable in certain contexts within manufacturing processes (the kind of systems for which Six Sigma makes sense), but not in product development—where its result is optimizing for efficiency and predictability at the expense of innovation and ability to adapt to changing conditions. Geoff Nicholson, the father of the Post-It Note, claims

¹ <http://slidesha.re/1v71niI>

that 3M’s adoption of Six Sigma at the behest of CEO James McNerney (formerly of GE and now of Boeing) “killed innovation.”² Prescriptive, rule-based processes also act as a brake on continuous improvement unless people operating the process are allowed to modify them. Finally, an overreliance on process tends to drive out people who tinker, take risks, and run safe-to-fail experiments. These kind of people tend to feel suffocated in a process-heavy environment—but they are essential drivers of an innovation culture.

Similarly, as organizations grow, the systems they build and operate increase in complexity. To get new features to market quickly, we often trade off quality for higher velocity. This is a sensible and rational decision. But at some point, the complexity of our systems becomes a limiting factor on our ability to deliver new work, and we hit a brick wall. Many enterprises have thousands of services in production, including mission-critical systems running on legacy platforms. These systems are often interconnected in ways that make it very hard to change any part of the system without also changing others, which acts as a significant drag on their ability to innovate at scale.

These organizational and architectural concerns are often the biggest barriers to executing the strategy for moving fast at scale based on the principles of Mission Command described in [Chapter 1](#). We will start by presenting a virtuoso execution of a strategy to manage organizational and systems complexity in the web age: Amazon. We’ll then present organizational, architectural, and leadership principles that enable organizations to grow successfully.

Amazon’s Approach to Growth

In 2001, Amazon had a problem: the huge, monolithic “big ball of mud” that ran their website, a system called Obidos, was unable to scale. The limiting factor was the databases. CEO Jeff Bezos turned this problem into an opportunity. He wanted Amazon to become a platform that other businesses could leverage, with the ultimate goal of better meeting customer needs. With this in mind, he sent a memo to technical staff directing them to create a service-oriented architecture, which Steve Yegge summarizes thus:³

1. All teams will henceforth expose their data and functionality through service interfaces.
2. Teams must communicate with each other through these interfaces.

² <http://zd.net/1v71quY>

³ Steve Yegge’s legendary “platform rant” is required reading for technical leaders: <https://plus.google.com/+RipRowan/posts/eVeouesvaVX>.

3. There will be no other form of interprocess communication allowed: no direct linking, no direct reads of another team's data store, no shared-memory model, no back-doors whatsoever. The only communication allowed is via service interface calls over the network.
4. It doesn't matter what technology they use. HTTP, Corba, Pubsub, custom protocols—doesn't matter. Bezos doesn't care.
5. All service interfaces, without exception, must be designed from the ground up to be externalizable. That is to say, the team must plan and design to be able to expose the interface to developers in the outside world. No exceptions.
6. Anyone who doesn't do this will be fired.

Bezos hired West Point Academy graduate and ex-Army Ranger Rick Dalzell to enforce these rules. Bezos mandated another important change along with these rules: each service would be owned by a cross-functional team that would build and run the service throughout its lifecycle. As Werner Vogels, CTO of Amazon, says, “You build it, you run it.”⁴ This, along with the rule that all service interfaces are designed to be externalizable, has some important consequences. As Vogels points out, this way of organizing teams “brings developers into contact with the day-to-day operation of their software. It also brings them into day-to-day contact with the customer. This customer feedback loop is essential for improving the quality of the service.”

Each team is thus effectively engaged in *product development*—even the people working on the infrastructural components that comprise Amazon Web Services, such as EC2. It's hard to overemphasize the importance of this transition from a project-based funding and delivery paradigm to one based on product development.

One of the biggest problems as organizations grow is maintaining effective communication between people and between teams. Once you move people to a different floor, a different building, or a different timezone, communication bandwidth becomes drastically limited and it becomes very hard to maintain shared understanding, trust, and effective collaboration. To control this problem, Amazon stipulated that all teams must conform to the “two pizza” rule: they should be small enough that two pizzas can feed the whole team—usually about 5 to 10 people.

⁴ Werner Vogel's article on Amazon's move to an SOA is also required reading: <http://queue.acm.org/detail.cfm?id=1142065>.

This limit on size has four important effects:

1. It ensures the team has a clear, shared understanding of the system they are working on. As teams get larger, the amount of communication required for everybody to know what's going on scales in a combinatorial fashion.
2. It limits the growth rate of the product or service being worked on. By limiting the size of the team, we limit the rate at which their system can evolve. This also helps to ensure the team maintains a shared understanding of the system.
3. Perhaps most importantly, it decentralizes power and creates autonomy, following the Principle of Mission. Each two-pizza team (2PT) is as autonomous as possible. The team's lead, working with the executive team, would decide upon the key business metric that the team is responsible for, known as the *fitness function*, that becomes the overall evaluation criteria for the team's experiments. The team is then able to act autonomously to maximize that metric, using the techniques we describe in [Chapter 9](#).
4. Leading a 2PT is a way for employees to gain some leadership experience in an environment where failure does not have catastrophic consequences—which “helped the company attract and retain entrepreneurial talent.”⁵

An essential element of Amazon's strategy was the link between the organizational structure of a 2PT and the architectural approach of a service-oriented architecture.

A Brief Introduction to Service-Oriented Architectures

A key principle of a service-oriented architecture (SOA) is decomposing systems into components or services. Each component or service provides an *interface* (also known as an Application Programming Interface, or API) so that other components can communicate with it. Other parts of the system—and the teams that create them—don't need to know the details of how the components or services they consume are built. Instead, they simply need to know the interface. This also means that there doesn't need to be a lot of communication between the teams that use a service or component and the team that builds and maintains it. Indeed, if the API is sufficiently well designed and documented, no communication is required.

Any system can be decomposed in multiple ways. Understanding how to decompose a system is an art—and, as the system evolves, the ideal decomposition is likely to change. There are two rules of thumb architects follow when decomposing systems. First, ensure that adding a new feature tends to change only one service or a

⁵ <http://blog.jasoncrawford.org/two-pizza-teams>

component at a time. This reduces interface churn.⁶ Second, avoid “chatty” or fine-grained communication between services. Chatty services scale poorly and are harder to impersonate for testing purposes.

All well-designed systems are split into components. What differentiates a service-oriented architecture is that its components can be deployed to production independently of each other. No more “big bang” releases of all the components of the system together: each service has its own independent release schedule. This architectural approach is essential to continuous delivery of large-scale systems. The most important rule that must be followed is this: the team managing a service has to ensure that its consumers don’t break when a new version is released.

To avoid the communication overhead that can kill productivity as we scale software development, Amazon leveraged one of the most important laws of software development—Conway’s Law: “Organizations which design systems...are constrained to produce designs which are copies of the communication structures of these organizations.” One way to apply Conway’s Law is to align API boundaries with team boundaries. In this way we can distribute teams all across the world. So long as we have each service developed and run by a single, co-located, autonomous cross-functional team, rich communication between teams is no longer necessary.

Organizations often try to fight Conway’s Law. A common example is splitting teams by function, e.g., by putting engineers and testers in different locations (or, even worse, by outsourcing testers). Another example is when the front end for a product is developed by one team, the business logic by a second, and the database by a third. Since any new feature requires changes to all three, we require a great deal of communication between these teams, which is severely impacted if they are in separate locations. Splitting teams by function or architectural layer typically leads to a great deal of rework, disagreements over specifications, poor handoffs, and people sitting idle waiting for somebody else.

Amazon’s approach is certainly not the only way to create velocity at scale, but it illustrates the important connection between communication structures, leadership, and systems architecture.

Create Velocity at Scale Through Mission Command

As organizations grow, informal processes and communication channels become increasingly ineffective at achieving the system-level outcomes we desire. Indeed it is easy for people to lose sight of system-level outcomes in the

⁶ See [parnas] for the original work on this subject.

face of rapid growth. As organizations grow, they move into the complex domain. In particular, two characteristics of complex adaptive systems begin to matter. First, there is no privileged perspective from which the system as a whole can be understood—not even the CEO’s office. Second, nobody can hope to understand more than a small part of the whole, depending on the information and context available to them.

Thus if we are not careful in the way we grow our organization, we will end up with a system where people optimize for what is visible to them and for the feedback they get, which is more or less determined by which people they interact with on a day-to-day basis. Thus each department or division optimizes for its own benefit—not because people are stupid or evil but because they simply have insufficient visibility into the effects of their actions on the wider organization. A simplified diagram of a traditionally structured enterprise is shown in [Figure 10-1](#).

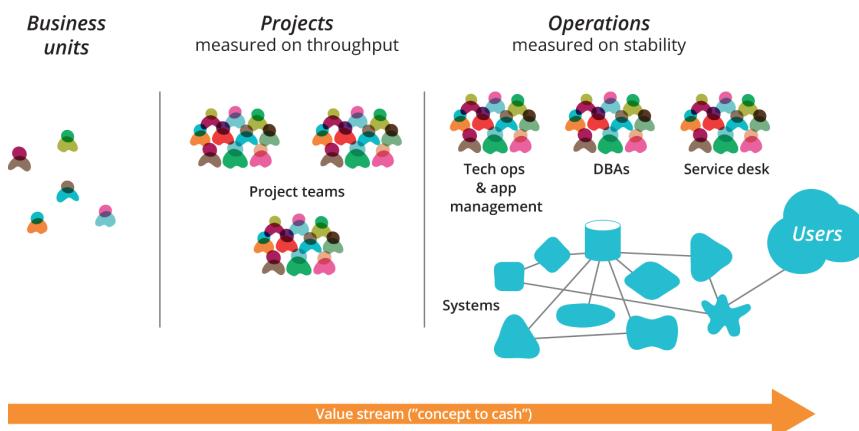


Figure 10-1. An example of a traditional enterprise organization

The key to moving fast at scale is to create many small, decentralized, autonomous teams, based on the model of Mission Command described in [Chapter 1](#). In truly decentralized organizations, we follow the principle of *subsidiarity*: by default, decisions should be made by the people who are directly affected by those decisions. Higher levels of bureaucracy should only perform tasks that cannot be performed effectively at the local level—that is, the authority of higher levels of bureaucracy should be *subsidiary* to that of the local levels.⁷

⁷ If we take this idea to its logical conclusion, we end up with what is known as *holacracy* (see <http://holacracy.org/constitution>). The Brazilian company Semco is an example of an enterprise that follows a radically decentralized model, see [\[semler\]](#).

Several successful large organizations have followed this principle for many years—for example, the Gore Company, Southwest Airlines, and the Swedish bank Handelsbanken, all of which have consistently demonstrated better than average performance in their markets.

Our starting point is to define the basic organizational unit—a team of up to 10 people (following Amazon’s two-pizza rule). Once you get beyond 10 people, group dynamics and coordination become hard to manage, and it becomes difficult to make consensus decisions and achieve a shared understanding of the context for everybody in the team.

In an enterprise context, teams usually collaborate to achieve program-level goals, and larger products and services will require multiple teams, perhaps including dedicated marketing and support people. As Reed Hastings says, our goal is to create teams that are highly aligned but loosely coupled. We ensure teams are *aligned* by using the Improvement Kata as described in [Chapter 6](#) and [Chapter 9](#)—that is, by having iterations at the program level with defined target conditions and having teams collaborate to work out how to achieve them.

Here are some strategies enterprises have successfully applied to create autonomy for individual teams:

Give teams the tools and authority to push changes to production

In companies such as Amazon, Netflix, and Etsy, teams, in many cases, do not need to raise tickets and have changes reviewed by an advisory board to get them deployed to production. In fact, in Etsy this authority is devolved not just to teams but to individual engineers. Engineers are expected to consult with each other before pushing changes, and certain types of high-risk changes (such as database changes or changes to a PCI-DSS cardholder data environment) are managed out of band. But in general, engineers are expected to run automated tests and consult with other people on their team to determine the risk of each change—and are trusted to act appropriately based on this information. ITIL supports this concept in the form of *standard changes*. All changes that launch dark (and which thus form the basis of A/B tests) should be considered standard changes. In return, it’s essential that teams are responsible for supporting their changes; for more on this, see [Chapter 14](#).

Ensure that teams have the people they need to design, run, and evolve experiments

Each team should have the authority and necessary skills to come up with a hypothesis, design an experiment, put an A/B test into production, and gather the resulting data. Since the teams are small, this usually means they are cross-functional with a mix of people: some generalists with one or

two deep specialisms (sometimes known as “T-shaped” people⁸), along with specialist staff such as a database administrator, a UX expert, and a domain expert. This does not preclude having centralized teams of specialists who can provide support to product teams on demand.

Ensure that teams have the authority to choose their own toolchain

Mandating a toolchain for a team to use is an example of optimizing for the needs of procurement and finance rather than for the people doing the work. Teams must be free to choose their own tools. One exception to this is the technology stack used to run services in production. Ideally, the team will use a platform or infrastructure service (PaaS or IaaS) provided by internal IT or an external provider, enabling teams to self-service deployments to testing and (where applicable) production environments on demand through an API (not through a ticketing system or email). If no such system exists, or it is unsuitable, the team should be allowed to choose their own stack—but must be prepared to meet any applicable regulatory constraints and bear the costs of supporting the system in production. We cover this thorny topic in more detail in [Chapter 14](#).

Ensure teams do not require funding approval to run experiments

The techniques described in this book make it cheap to run experiments, so funding should not be a barrier to test out new ideas. Teams should not require approval to spend money up to a certain limit (for example, a per-transaction and per-month limit).

Ensure leaders focus on implementing Mission Command

In a growing organization, leaders must continuously work to simplify processes and business complexity, to increase the effectiveness, autonomy, and capabilities of the smallest organizational units, and to grow new leaders within these units.

An example of how this might look is shown in [Figure 10-2](#). In the case of user-installed products, mobile apps, and embedded systems, PaaS/IaaS is used for testing purposes, but the release process happens on demand rather than continuously. Note that this structure does *not* require a change in who people report to. People can still report up the traditional functional lines (for example, testers to a Director of Testing) even if they work on cross-functional teams on a day-to-day basis. Enterprises often waste a great deal of time on unnecessary, disruptive reorganizations—when they would do better simply by having people who work on the same product or service sit all in the same room (or, for larger products, on the same floor).

⁸ David Guest, “The hunt is on for the Renaissance Man of computing,” *The Independent* (London), September 17, 1991.

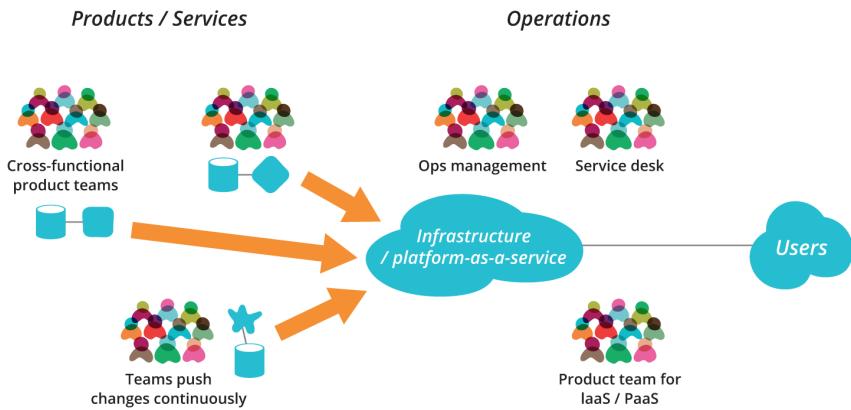


Figure 10-2. Product teams working together, with a service layer for performing deployments

WARNING

Ensure Rewards Are Aligned with Desired Behavior

Although it's not necessary for reporting structures to reflect team organization, poor management can easily destroy collaboration by rewarding people for behavior that optimizes for their function at the expense of customer outcomes or wider organizational goals. Examples of this include rewarding developers for features that are "dev complete" but not production ready, or rewarding testers for the number of bugs they find. In general, rewarding people for output rather than system-level outcomes leads to dysfunction, and in any case monetary rewards or bonuses have been demonstrated to *reduce* performance in the context of knowledge work. We cover the topic of incentives and culture in more detail in [Chapter 1](#) and [Chapter 11](#).

Creating small, autonomous teams makes it economic for them to work in small batches. When done correctly, this combination has several important benefits:

Faster learning, improved customer service, less time spent on work that does not add value

Autonomy—combined with an enterprise architecture that supports it—reduces dependencies between teams so they can get changes out faster. This is a key component in enabling teams to create prototypes of new products and features to gather customer feedback, run A/B tests, and improve customer service by responding quickly to user requests for improvements and bug fixes. If we can quickly learn what users actually value, we can stop wasting time building things that don't add value. The most important metric is: how fast can we learn? Change lead time is a

useful proxy variable for this metric, and autonomous teams are essential to improving it.

Better understanding of user needs

In organizations where work moves through functional silos, the feedback loop from users to the designers and engineers who create the product is often slow and has low fidelity. When everybody on the team can build small experiments, push them into production, and analyze the metrics, the entire team comes into contact with users on a day-to-day basis.

Highly motivated people

When we can design an experiment or push a bug fix or enhancement to users and see the results almost immediately, it's an incredibly empowering experience—a proof of your autonomy, mastery, and purpose. Nobody we know who has ever worked in this way wants to go back to the old way of doing things.

Easier to calculate profit and loss

Cross-functional customer-facing teams that own a service over its lifecycle make it much easier to calculate profit and loss (P&L) for the service. The cost of the service is simply the cost of the resources consumed by the team, plus their salaries. This allows us to use simple dollar numbers to identify teams generating the highest margins for the company. Note that this is independent of the idea of internal chargeback, which if implemented dogmatically often requires high levels of business complexity to determine costs with unnecessary precision.

It's one thing to adopt the principles of Mission Command in a growing startup—but another thing entirely in an enterprise with a more traditional, centralized approach to management and decision making. Mission Command drastically changes the way we think about management—in particular, management of risk, cost, and other system-level outcomes. Many organizations adopt a one-size-fits-all approach to risk and cost management, with centralized processes for software release management (by the IT department) and budgeting (by the finance department). In Mission Command, teams have the authority and responsibility to manage cost and risk appropriately in their particular context. The role of finance, the project management office, enterprise architects, GRC teams, and other centralized groups changes: they specify target outcomes, help to make the current state transparent, and provide support and tools where requested, but do not dictate *how* cost, processes, and risk are managed. We discuss lean approaches to governance and finance in [Part IV](#).

Evolving Your Architecture Using the Strangler Application Pattern

Autonomous teams will make little difference to customer outcomes if the enterprise architecture prevents teams from running experiments and responding quickly to customer needs. To enable both continuous delivery and decentralization, teams must be able to get changes out quickly and safely. Unfortunately, the reality is that in many enterprises there are thousands of tightly coupled systems, and it is very hard to make changes to any of them without navigating a web of dependencies. Too often, one of the dependencies is a system of record maintained by a team which releases updates every few months at the cost of significant heroics.

TIP

Architecting for Continuous Delivery and Service Orientation

Architecting for continuous delivery and service orientation means evolving systems that are *testable* and *deployable*. Testable systems are those for which we can quickly gain a high level of confidence in the correctness of the system without relying on extensive manual testing in expensive integrated environments. Deployable systems are those that are designed to be quickly, safely, and independently deployed to testing and (in the case of web-based systems) production environments. These “cross-functional” requirements are just as important as performance, security, scalability, and reliability, but they are often ignored or given second-class status.

A common response to getting stuck in a big ball of mud is to fund a large systems replacement project. Such projects typically take months or years before they deliver any value to users, and the switchover from the old to the new system is often performed in “big bang” fashion. These projects also run an unusually high risk of running late and over budget and being cancelled. Systems rearchitecture should not be done as a large program of work funded from the capital budget. It should be a continuous activity that happens as part of the product development process.

Amazon did not replace their monolithic Obidos architecture in a “big bang” replacement program. Instead, they moved to a service-oriented architecture incrementally, while continuing to deliver new functionality, using a pattern known as the “strangler application.” As described by Martin Fowler, the pattern involves gradual replacement of a system by implementing new features in a new application that is loosely coupled to the existing system, porting

existing functionality from the original application only where necessary.⁹ Over time, the old application is “strangled”—just like a tree enveloped by a tropical strangler fig (Figure 10-3).

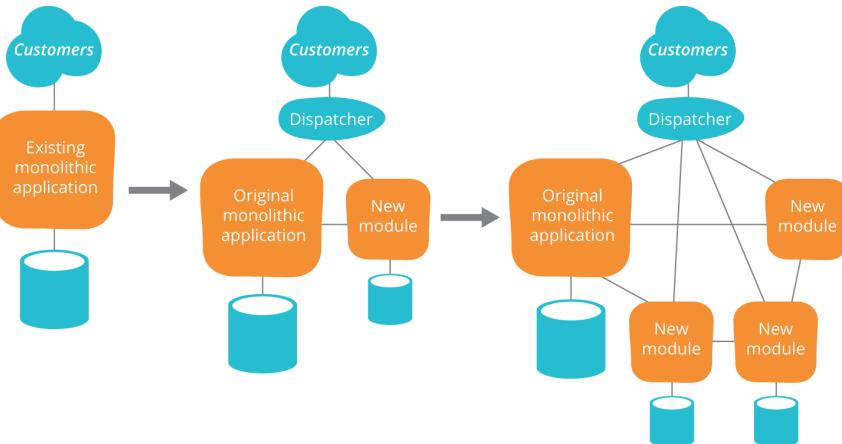


Figure 10-3. The evolution of stranglers

Strangling applications should use the methods described earlier in this book. There are some important rules to follow when implementing the strangler pattern:

Start by delivering new functionality—at least at first

Always find ways to satisfy a need that is not served by the existing software, and prioritize features using the cost of delay divided by duration (CD3), as described in Chapter 7, to ensure you deliver the largest amount of value in the shortest possible time.

Do not attempt to port existing functionality unless it is to support a business process change

The biggest mistake people make is porting existing features over as-is. This generally means reproducing complexity created to serve business processes as they looked years ago, which is enormously wasteful. Whenever you are asked to add a feature which represents a change to a business process, go and observe the process from scratch and look for ways to simplify it before implementing the code to support it. You will find that much accidental complexity in business processes actually comes from being forced to use the old software you are replacing!

⁹ <http://bit.ly/1v71DOH>, following Chris Stevenson and Andy Pols' paper, <http://bit.ly/1v71GtR>.

Deliver something fast

Make the initial release of your new application small enough that you can get it deployed and providing value in a few weeks to a few months. When building the first module, it's hard—but essential—to resist feature creep. The measure of success for the first release is how quickly you can do it, not how much functionality is in it. Typically, this is achieved by using the “vertical slice” approach in which we build small increments of functionality end-to-end across the whole technology stack.

Design for testability and deployability

Functionality in the new application must always be built using good software development practices: test-driven development, continuous integration, a well-encapsulated, loosely coupled modular design. Strangler applications are an opportunity to test out such practices, so make sure the team working on it is enthusiastic about these methods and has enough experience to have a good chance at succeeding.

Architect the new software to run on a PaaS

Work with operations to drive the design of the software hand in hand with the platform as a service, as we describe in [Chapter 14](#). If the operations team is not ready to do this, work with them to ensure that the system doesn't drive up the complexity of the existing operational environment.

There is of course a trade-off to migrating in an incremental way. *Overall*, it takes longer to do a replacement incrementally compared to a hypothetical “big bang” rearchitecture delivering the same functionality. However, since a strangler application delivers customer value from early on, evolves in response to changing customer needs, and can advance at its own rate, it is almost always to be preferred.

Enterprise architecture is usually driven by expensive, top-down plans to “rationalize” the architecture, move from legacy systems to a modern platform, and remove duplication to create a single source of truth. Often, the end state is represented by a good-looking diagram that fits on a single (large) sheet of paper. However, the end state is rarely achieved because the ecosystem which the architecture serves always changes too fast, and it is even rarer for the promised benefits to materialize. Typically, what happens is that new structures are added, but the systems that were supposed to be replaced never actually get turned off, leading to ever-increasing complexity which makes it even harder to change things in future.

TIP

Create Architectural Alignment Through Specifying Target Conditions, Not Standardization and Architectural Epics

Our experience is that standardization on a particular toolchain or technology stack is neither necessary nor sufficient for achieving enterprise architecture goals such as enabling teams to respond rapidly to changing requirements, creating high-performance systems at scale, or reducing the risk of intrusion or data theft. Just like we drive product and process innovation through the Improvement Kata, we can drive architectural alignment through it too. Architectural goals—for example, desired performance, availability, and security—should be approached by iteratively specifying target conditions at the program level. Following the Principle of Mission, set out a clear vision of the *goals* of your enterprise architecture without specifying *how* the goals are to be achieved, and create a context in which teams can determine how to achieve them through experimentation and collaboration. We cover alternatives to standardization and related issues in more detail in [Chapter 14](#).

We do much better by accepting that we will always be in a state of change, and working slowly and incrementally to reduce complexity through the strangler application pattern. Find a way to measure the surface area of the systems that you aim to retire, and make it visible so that teams can work to reduce it—and ultimately eliminate such systems—as they continue to deliver value to customers. Accept that evolving enterprise architecture—and reducing unnecessary complexity—is a continuing, unending process.

Conclusion

Moving quickly at scale requires implementing the Mission Command. One commonly used approach is to create small teams that are highly aligned but loosely coupled. However, given the strong coupling between systems architecture and communication flows observed by Melvin Conway and codified in his eponymous law, we also need to evolve a systems architecture that supports this kind of decentralized organization.

Moving from a more traditional centralized model to the kind of structure described in this chapter is hard. We must proceed slowly and incrementally. It requires changes to the existing centralized processes—in particular, budgeting, procurement, risk management, governance, and release management. These are discussed in the last part of this book, [Part IV](#).

Even though change is hard and takes time, we should not be dissuaded. The key is to find ways to make small, incremental changes that deliver improved customer outcomes—and then keep going. Just as we apply the strangler pattern to enterprise architecture, we can also apply it to our organizational

culture and processes—this is the topic of the final chapter of this book, [Chapter 15](#).

Questions for readers:

- Can your teams run experiments and achieve customer outcomes independently, or are they dependent on other teams in order to get anything done?
- Can you deploy pieces of your system independently of each other, or must you release everything at once?
- What is the smallest possible amount of work you could do to enable either experimentation or independent deployment for a single team or component/service?
- How are people on your teams rewarded? Does this encourage or discourage them from collaborating with other people on your team or with other teams?

PART IV

TRANSFORM

Everyone thinks of changing the world, but no one thinks of changing himself.

— Leo Tolstoy

If you've made it from the beginning of this book to here, you should have a pretty good idea of how to apply lean concepts and principles to make great software products, and of the importance of strategy and culture in enabling the discovery and exploitation of new businesses. But to reap the maximum reward for all our efforts, lean principles and concepts need to be scaled throughout the entire organization. Only when this happens will we realize the full value of the work we have invested in, exploring new ideas and exploiting those that deliver value to customers.

We readily grasp that these concepts work well to address the needs of rapidly changing environments and fierce competition. However, it is hard to extend lean concepts to process improvement, COTS applications, and the evolution and support of internal systems, particularly systems of record. Supplier and vendor relationships present a further obstacle. The nature of our relationship with suppliers of proprietary, specialized, or customized solutions often inhibits collaboration, fast feedback, or small incremental change. We need to seek suppliers who are willing to treat us as we expect to treat our own customers. We must encourage suppliers to listen to us, understand what we need, and

experiment. They have to be willing to go on the journey of improvement with us.

To add further complexity to this problem, many of our traditional approaches to governance, risk, and compliance (GRC), financial management, procurement, vendor/supplier management, and human resources (recruiting, promotion, compensation) create additional waste and bottlenecks. These can only be eliminated when the entire organization embraces lean concepts and *everyone* works together in the same direction.

Making an enterprise lean is not a one-person or one-department show. It won't work through a special tactical task force. We can't mandate that from now on, everyone will work this way and expect them to adjust as per our implementation plan. Real lean transformation is the result of committed, fearless leaders who encourage and enable lean thinking to propagate throughout the entire fabric of the organization—not just customer-facing products. Those at the top need to walk the talk and be role models for everyone. They need to set aside egos, listen and respect contrary opinions, and build trusting relationships at all levels of the organization. This is essential for new leaders to emerge and for lean concepts and practices to become woven into the organization's culture.

People must feel empowered to make decisions that involve risk and try out new ideas, while recognizing their responsibilities to customers and maintaining alignment with the overall organization strategy. As leaders, we need to set limitations and context for everyone, but ensure they are not unduly restrictive. When everyone is united in pursuit of a common purpose, and we have empathy with our customers and put serving their needs first, most people can figure out what risks are acceptable and what are not.

Conflict arises when our espoused values do not match up with actual practice. This is where modeling the behavior we hope to see in everyone is most important. There are no formulas, instructions, or rituals that will work for everyone. Each of us needs to take time daily, maybe even several times a day, to reflect on our own actions and decide if they support our stated values and work to move us in the right direction.

A lean mindset cannot thrive in an organization with a centralized, command-and-control management style. Nevertheless, we still need to maintain visibility and transparency into what everyone is doing. It is not easy for large organizations to find this balance, and we must recognize that constant adjustment will be required. Many people within our organization will perceive this cultural shift as threatening and will push back on it. Command and control is easy; I follow the rules and if it doesn't work, it is not my fault. However, if you ask me to make decisions and take responsibility for them, someone might hold me

accountable for the mistakes I am very likely to make. Give me command and control!

If we are successful at creating a lean enterprise, we will have failures and setbacks, at all levels and by all people. If we don't, it means we haven't created a high-trust, high-performance culture and are continuing to judge our performance by vanity metrics, not real outcomes. We'll never have the culture of a learning organization if we can't be allowed to make mistakes and get worse at something before we get better.

In this part, we concentrate our discussion on how to pursue the never-ending work of transforming the enterprise. We will address some of the most common areas where we see a mismatch between lean concepts and the prevailing leadership and management principles, practices, and processes. These gaps are revealed when we face obstacles that prevent us doing better at delivering value to customers, or when we are robbed of satisfaction and fulfilment in our daily work. Our hope is that readers will be inspired to find ways to overcome these obstacles, and to share their successes—and their failures—with others.

Grow an Innovation Culture

The ability of your company to be competitive and survive lies not so much in solutions themselves, but in the capability of the people in your organization to understand a situation and develop solutions.

— Mike Rother

We now accept the fact that learning is a lifelong process of keeping abreast of change. And the most pressing task is to teach people how to learn.

— Peter Drucker

You know, I'm all for progress. It's change I object to.

— Mark Twain

Culture is the most critical factor in an organization's ability to adapt to its changing environment. However, being intangible, it is hard to analyze and even harder to change. Every organization has its own unique culture, and there are "as many successful cultures as there are successful companies."¹ In Chapter 1 we presented the characteristics of a high-performance, generative culture. In this chapter, we discuss how to understand your organization's culture and what you can do to change it.

Culture is constantly changing in every organization. New employees and leaders join, people quit, strategies and products evolve and die, and the market

¹ *The Economist*, 410, no. 8869, p. 72.

constantly shifts. The most important question is: can we mindfully evolve our organizational culture in response to these changes in environment?

To understand how to influence organization culture, we need to understand its foundations. We introduce a model of organizational culture and discuss how to measure it. We follow with strategies to kick-start organizational change, with the goal of making these strategies self-sustaining. Finally, we examine the relationship between individuals and organizations, and discuss how to hire and retain “good” people.

Model and Measure Your Culture

CEOs can talk and blab all day about culture, but the employees know who the jerks are.

— Jack Welch

In *The Corporate Culture Survival Guide*, Schein defines culture as “a pattern of shared tacit assumptions that was learned by a group as it solved its problems of external adaptation and internal integration, that has worked well enough to be considered valid and, therefore, to be taught to new members as the correct way to perceive, think, and feel in relation to those problems.”² The “tacit” part of this definition is important—and it is what makes culture so intangible. Shanley Kane, author of *Your Startup Is Broken: Inside the Toxic Heart of Tech Culture*, provides another perspective, commenting that “our true culture is made primarily of the things no one will say...Culture is about power dynamics, unspoken priorities and beliefs, mythologies, conflicts, enforcement of social norms, creation of in/out groups and distribution of wealth and control inside companies.”³

Even though culture is intangible, it is measurable, and there is a large body of work dedicated to precisely this task. Of course every methodology is based on an underlying model, and all models are limited to a different extent. Nevertheless, such measurements are important as a way of making culture visible and encouraging people to pay attention to it. Here are examples of work that has been done to measure culture:

- Karen E. Watkins and Victoria J. Marsick developed the *Dimensions of the Learning Organization Questionnaire* (DLOQ), which has been extensively studied in the academic literature. You can take the questionnaire for free at <http://www.partnersforlearning.com/instructions.html>.

2 [schein]

3 [kane], “Five Tools for Analyzing Dysfunction in Engineering Culture” and “Values Towards Ethical and Radical Management.”

- Gallup's Q12 survey asks what they believe are "the only 12 questions that matter" to measure employee engagement. You can find the questions, along with more information, at <http://q12.gallup.com/>.
- In [Chapter 1](#), we discussed how the *2014 State of DevOps Report* measured both job satisfaction and culture (using the Westrum model) and their impact on organizational performance. Analysis showed that Westrum's model predicted both job satisfaction and organizational performance in the context of knowledge work. Read more at <http://bit.ly/1v71SjL>.

TIP

Practicalities of Running Cultural Surveys

Whether you use a service or come up with your own survey, be careful about how much information is collected. To obtain honest answers, don't ask people to disclose identifying information. Present results only in aggregate. It may be useful to capture some demographic information so you can see, for example, how results vary between genders or roles, but only when you have numbers large enough to provide anonymity. Be mindful of how the information can work against the respondents. At one large enterprise, managers reacted to poor survey results in their department by ordering their own reports to paint them in a better light next time.

Disassociate culture surveys from pay and performance reviews. Make the aggregated results available to all employees and ensure executives set up meetings to discuss the findings and plan next steps. Run surveys annually or semiannually to provide a baseline for comparison and measurement of change over time.

Measuring organizational culture and making problems visible is the first step. Next, we must investigate *why* a culture is the way it is. For this inquiry, it is helpful to use Schein's model, which divides culture into three layers: artifacts, espoused values, and underlying assumptions ([Figure 11-1](#)).

Artifacts: visible, observable aspects of culture such as organizational structures and processes, and how people dress and behave. Hard to decipher.



Espoused values: Strategies, goals, philosophies, such as the “value statements” you find in the company lobby.



Underlying assumptions: Unconscious beliefs, perceptions, thoughts, feelings (which ultimately govern values and behavior).

Figure 11-1. Layers of organizational culture

Inconsistencies between espoused values and observed behaviors within an organization are common. Observed behaviors are better indicators of real values. Who gets rewarded for what behavior? Who gets hired, promoted, or fired? In order to understand the nature and source of the real values, we have to descend to the level of underlying assumptions. This level is hard to unpack, but it is the most important to understand.

Schein presents an exhaustive typology of tacit assumptions, of which the most important are the beliefs leaders and managers hold about workers. In his management classic *The Human Side of Enterprise*, Douglas McGregor describes two contrasting sets of beliefs held by managers he observed, which he calls Theory X and Theory Y. Managers who hold Theory X assumptions believe that people are inherently lazy and unambitious and value job security more than responsibility; extrinsic (carrot-and-stick) motivation techniques are

the most effective to deal with workers.⁴ In contrast, Theory Y managers believe “that employees could and would link their own goals to those of the organization, would delegate more, function more as teachers and coaches, and help employees develop incentives and controls that they themselves would monitor.”⁵

As we saw in [Chapter 1](#), while extrinsic motivators such as bonuses are effective in a Taylorist world of routine, mechanical work, they actually *reduce* performance in the context of knowledge work. People involved in nonroutine work are motivated by intrinsic factors summarized by Dan Pink as “1. *Autonomy*—the desire to direct our own lives. 2. *Mastery*—the urge to get better and better at something that matters. 3. *Purpose*—the yearning to do what we do in the service of something larger than ourselves.”⁶

What is especially problematic is that, by generating behavioral responses that align with their management style, both types of managers believe their style works best. People whose management strategy is consistent with Theory X end up with employees who are passive, resistant to change, unwilling to accept responsibility, and make “unreasonable demands for economic benefits.”⁷ This is a rational response by employees to not having their higher needs satisfied through work. Work becomes something to be endured in order to get a paycheck.

In an organization whose leaders share Theory Y assumptions, their job is “the creation of conditions such that the members of the organization can achieve their own goals *best* by directing their efforts towards the success of the enterprise,”⁸ delivering value to customers and the organization while growing their own capabilities. Until leaders and managers with Theory X attitudes work to adopt a Theory Y mindset and demonstrate it consistently over time through their actions, they will not be able to achieve a perceptible difference in people’s behavior. The story of NUMMI in [Chapter 1](#) is a good example of this shift in mindset and behavior.

Culture is hard to change *by design*. As Schein says, “Culture is so stable and difficult to change because it represents the accumulated learning of a group—

⁴ A sophisticated and entertaining exploration of a Theory X organization and its lifecycle, based on the work of Ricky Gervais, can be found at <http://bit.ly/1v71WJq>.

⁵ [\[schein\]](#), p. 64.

⁶ <http://www.danpink.com/drive-the-summaries>. Pink also references a number of studies which demonstrate conclusively that extrinsic motivation reduces performance in knowledge work.

⁷ [\[mcgregor\]](#), p. 42.

⁸ [\[mcgregor\]](#), p. 49.

the ways of thinking, feeling, and perceiving the world that have made the group successful.”⁹

Change Your Culture

In his revolutionary work *Pedagogy of the Oppressed*, published in 1970, Paulo Freire describes what is still the dominant model of teaching today. In this model, students are viewed as empty “bank accounts” to be filled with knowledge by teachers—not as participants who have a say in what and how they learn. This model is not designed to enable students to learn—especially not to learn to think for themselves—but rather to control the learning process, students’ access to information, and their ability to critically analyze it. In this way, the education system perpetuates existing social structures and power hierarchies.

Similarly, most companies seem to treat their employees as filled-up bank accounts to be drained of skills and knowledge in service of the company’s goals. This is the implication when we speak of employees as “resources” and wonder how to increase their utilization and productivity with little regard for their personal development. This kind of behavior indicates an environment in which employees exist primarily as providers of labor, not as active participants in creating value.¹⁰ In contrast, high-performance organizations are effective at both developing and harnessing the unique capabilities of their people.

Organizations with a “bank account” attitude to employees tend to treat change in a transactional way. This all too common and flawed approach involves funding a change program which is expected to “fix” the organization so it is fit for purpose. Organizational change is treated as a product—sold by consultants, paid for by leadership, and consumed by the rest of the organization as directed.

These change programs commonly focus on reorganizing teams and reporting structures, sending employees on short training courses, and rolling out tools and methodologies across the organization. These strategies usually don’t work because they are ineffective at changing people’s patterns of behavior. As Mike Rother points out in *Toyota Kata*, “what is decisive is not the form of the organization, but how people act and react.”¹¹ This is determined primarily by the actions of leadership and management. To pick some examples: are people given the autonomy to act and trusted to take risks? Is failure punished or does

⁹ [schein], pp. 27–28.

¹⁰ The key concept here is the idea that workers are fungible—that is, essentially interchangeable—resources. Any time you hear people referred to as “resources,” this is what is implied.

¹¹ [rother-2010], p. 236.

it lead to enquiry and improvements of our systems? Is cross-functional communication rewarded or discouraged?

We began this book by discussing the case of NUMMI, in which a broken organization was reformed under a new leadership and management paradigm. Despite rehiring the same people, NUMMI achieved extraordinary levels of quality and productivity and reduced costs. In an article for *MIT Sloan Management Review*, John Shook, Toyota City's first US employee, reflected on how that cultural change was achieved:¹²

What my NUMMI experience taught me that was so powerful was that the way to change culture is not to first change how people think, but instead to start by changing how people behave—what they do. Those of us trying to change our organizations' culture need to define the things we want to do, the ways we want to behave and want each other to behave, to provide training and then to do what is necessary to reinforce those behaviors. The culture will change as a result... What changed the culture at NUMMI wasn't an abstract notion of "employee involvement" or "a learning organization" or even "culture" at all. What changed the culture was giving employees the means by which they could successfully do their jobs. It was communicating clearly to employees what their jobs were and providing the training and tools to enable them to perform those jobs successfully.

Shook offers his own interpretation of Schein's model, showing how people normally approach cultural change in contrast to the approach taken at NUMMI, in [Figure 11-2](#).

NUMMI had an advantage in achieving their cultural change. The entire workforce was newly hired—with many workers having been freshly fired from their jobs at Fremont Assembly. It's hard to achieve sustained, systemic change without any crisis. In *The Corporate Culture Survival Guide*, Schein asks if crisis is a necessary condition of successful transformations; his answer is, "Because humans avoid unpredictability and uncertainty, hence create cultures, the basic argument for adult learning is that indeed we do need some new stimulus to upset the equilibrium. The best way to think about such a stimulus is as *disconfirmation*: something is perceived or felt that is not expected and that upsets some of our beliefs or assumptions...disconfirmation creates *survival anxiety*—that something bad will happen if we don't change—or *guilt*—we realize that we are not achieving our own ideals or goals."¹³

12 <http://bit.ly/1v720ZH>

13 [schein], p. 106.

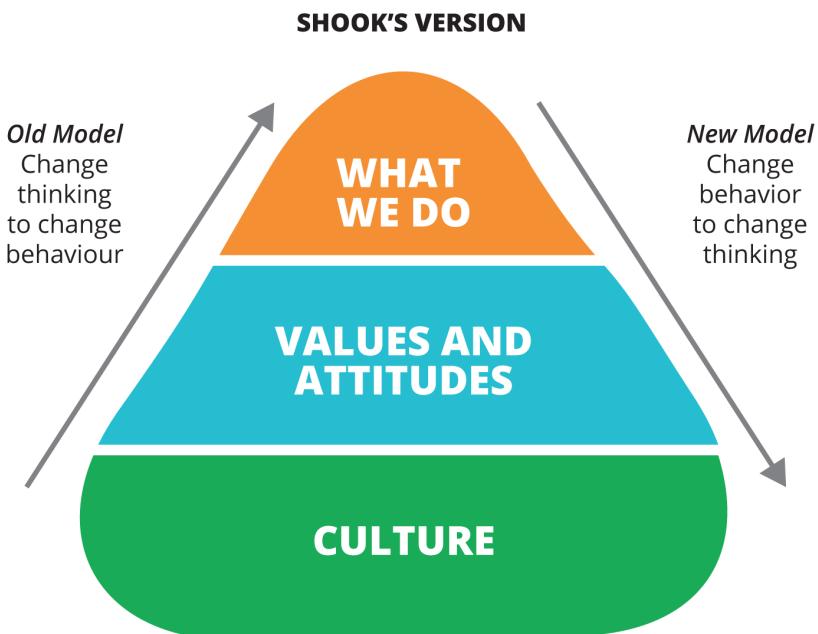


Figure 11-2. Old and new approaches to cultural change (2010 from MIT Sloan Management Review/Massachusetts Institute of Technology, all rights reserved, distributed by Tribune Content Agency, LLC)

Disconfirmation can come naturally from a number of sources that may threaten our survival: economic, political, technological, legal, moral, or simply a realization that we are not achieving our purpose. A common cause of unplanned disconfirmation is leaders acting in a way that contradicts their stated values. It is also possible to create disconfirmation in a controlled way through joint ventures, planned leadership activity, or by creating an artificial crisis.

Once people accept the need for change, they are confronted with the fear that they may fail at learning the new skills and behavior required of them, or that they may lose status or some significant part of their identity—a phenomenon Schein calls *learning anxiety*.

Schein postulates that for change to succeed, survival anxiety must be greater than learning anxiety, and to achieve this, “learning anxiety must be reduced rather than increasing survival anxiety.”¹⁴ Many leaders and managers make

14 [schein], p. 114.

the mistake of trying to achieve change by increasing survival anxiety. This creates an environment of fear which, in turn, results in significant amounts of energy spent on diverting blame, avoiding responsibility, or playing political games.

The most powerful systematic tool to reduce survival anxiety that we have encountered is the Improvement Kata, described in [Chapter 6](#). It is designed for people to safely learn new skills and experiment with new ideas in the pursuit of clearly defined, measurable organizational goals. Essential to creating a high-performance culture is an environment in which mistakes are accepted as learning opportunities to build systems and processes that reduce the impact of future mistakes.

Make It Safe to Fail

Your organization's attitude to failure—whether of a change effort or simply a decision—is critical in creating an adaptive, resilient organization. Organizational theorist Professor Russell L. Ackoff noted, “It’s our treatment of error that leads to a stability which prevents significant change.” If people are told that making mistakes is bad, and if people are punished for them, the inevitable outcome is that they will avoid taking *any* risky decisions.¹⁵

In a complex, adaptive system such as an enterprise, nobody has perfect information. Every decision will have unintended consequences whose causes may be clear looking back, but are almost impossible to predict looking forward. Whenever it appears that one person is responsible for a given outcome, we should be honest and ask ourselves, “If I had been in the same situation, is it possible I would have made the same decisions?” Usually, the answer is “yes.”

Rather than punishing mistakes, we must ensure that people have the necessary information to make effective decisions, find ways to limit the possible negative outcomes of decisions, and be disciplined about learning from mistakes. For example, how do managers and leaders in your organization respond to failures? Do they lead to scapegoating, justice, or enquiry?

One practice often used by organizations with high-performance cultures is a blameless postmortem run after every incident or accident. The goal of the postmortem is to improve the system so that, in similar situations in the future, people have better information and tools at their disposal and the negative impact is limited.

At the beginning of every postmortem, every participant should read aloud the following words, known as the *Retrospective Prime Directive*: “Regardless of

¹⁵ <http://youtu.be/MzSSV5-0VsA?t=6m>

what we discover, we understand and truly believe that everyone did the best job they could, given what they knew at the time, their skills and abilities, the resources available, and the situation at hand.”¹⁶ A postmortem should aim to provide:¹⁷

- A description and explanation of how the incident happened, from the perspective of those involved and affected, including a timeline of events and a list of contributing factors
- Artifacts (recommendations, remediations, checklists, runbook updates, etc.) for better prevention, detection, and response to improve the handling of similar events in the future

Postmortems should not attempt to identify a single root cause. The idea that a single event can be identified as the cause of a failure is a misunderstanding of the nature of complex adaptive systems. As safety experts Sidney Dekker, Erik Hollnagel, David Woods, and Richard Cook point out:¹⁸

Our understanding of how accidents happen has undergone a dramatic development over the last century. Accidents were initially viewed as the conclusion of a sequence of events (which involved “human errors” as causes or contributors). This is now being increasingly replaced by a systemic view in which accidents emerge from the complexity of people’s activities in an organizational and technical context. These activities are typically focused on preventing accidents, but also involve other goals (throughput, production, efficiency, cost control) which means that goal conflicts can arise, always under the pressure of limited resources (e.g., time, money, expertise). Accidents emerge from a confluence of conditions and occurrences that are usually associated with the pursuit of success, but in this combination—each necessary but only jointly sufficient—able to trigger failure instead.

Every failure is the result of multiple things going wrong—often *invisibly* (Dekker refers to complex adaptive systems “drifting into failure”).¹⁹ Every postmortem should result in multiple ideas for incremental improvement. We must also schedule a follow-up to test whether these improvements were effective,

16 [kerth]

17 These two points are John Allspaw’s: <http://bit.ly/1e9idko>. And if you’re interested in how Knight Capital lost \$460m in 30 minutes, this post is worth reading in full.

18 [dekker], p. 6.

19 You can find a short guide to failure in complex systems at <http://bit.ly/1F7O3Mg>.

ideally by running an exercise simulating a similar failure, as we describe in [Chapter 14](#).

There Is No Talent Shortage

In the tech industry it's common to hear about a "talent shortage" and the difficulty of finding "good people."²⁰ In this section we'll dismantle the assumptions behind these kinds of remarks. We will examine what we mean by "good people" by looking at one particular role—software engineers—and then progress to the general case.

It's a widely held belief that there is an order-of-magnitude difference between the best and the worst engineers.²¹ In reality, the 10x figure is (to put it mildly) "poorly supported by empirical evidence."²² However, once you get to the bottom of the debate over the claim, it is really about the validity, or usefulness, of individual productivity measurements in the context of an organization.

Individual productivity is most commonly measured by throughput—the time it takes to complete a standardized task under controlled conditions. This approach is premised upon a Taylorist view of work where managers define the tasks to be done and workers try to complete these tasks as rapidly as possible. Thus, old-school metrics such as lines of code per day and number of hours worked are used to measure individual productivity of software engineers. The flaws in these measures are obvious if we consider the ideal outcomes: the fewest lines of code possible in order to solve a problem, and the creation of simplified, common processes and customer interactions that reduce complexity in IT systems. Our most productive people are those that find ingenious ways to avoid writing any code at all.

In many organizations, worrying unduly about variations between individuals is futile. If there's one thing we should learn from the NUMMI case study in [Chapter 1](#), it's that organizational culture and leadership dwarf differences between individuals. As journalist and author Malcolm Gladwell writes, "The talent myth assumes that people make organizations smart. More often than not, it's the other way around...Our lives are so obviously enriched by individual brilliance. Groups don't write great novels, and a committee didn't come up with the theory of relativity. But companies work by different rules. They

20 The title of this section is taken from a presentation by Andrew Shafer: https://www.youtube.com/watch?v=P_sWGI7MzbU.

21 The original reference is from [\[sackman\]](#), and a robust discussion and defense of the claim can be found at <http://bit.ly/1v72hvu>.

22 [\[bossavit\]](#), in Chapters 5 and 6, does an effective job at demolishing the existing studies and data.

don't just create; they execute and compete and coordinate the efforts of many different people, and the organizations that are most successful at that task are the ones where the system is the star." As W. Edwards Deming noted, "A bad system will beat a good person every single time."

The rate at which we can understand and solve complex problems—the key skill for which we still need people, rather than machines—is determined as much by our environment as our own skills and abilities. We can hardly blame people for failing to learn and solve problems if we limit their opportunities by organizational silos that insulate workers from each other and from customers, by long cycle times that delay feedback, by focusing on completing assigned work rather than achieving customer outcomes, and by working long hours so we have no time to try out new ideas and technologies or even talk to each other!

Given that the culture of an organization has such a dominant effect on the performance of individuals, should we care at all about the particular skills and attitudes of individuals? Instead of taking a "bank account" view that focuses on people's existing capabilities, it's more important to consider their ability to *acquire* new skills—particularly in the field of technology where useful knowledge and skills change rapidly.

Carol Dweck, Professor of Psychology at Stanford, has spent years researching the psychology of learning, development, and motivation. Her research reveals there is a way to judge how good people will be at learning new skills. Dweck discovered that our ability to learn is determined by our beliefs concerning the question: is ability innate, or can it be learned? We can observe, based on people's behavior, where they fall on a continuum between two extremes:²³

In a fixed mindset, students believe their basic abilities, their intelligence, their talents are just fixed traits. They have a certain amount and that's that, and then their goal becomes to look smart all the time and never look dumb. In a growth mindset, students understand that their talents and abilities can be developed through effort, good teaching, and persistence. They don't necessarily think everyone's the same or anyone can be Einstein, but they believe everyone can get smarter if they work at it.

Dweck showed through a series of experiments that our mindset determines how we decide our goals, how we react to failure, what are our beliefs about effort and strategies, and what is our attitude towards the success of others (Figure 11-3). Our mindset is particularly important in terms of our attitude to

23 <http://bit.ly/1v72nmV>

failure. People with a fixed mindset fear failure as they believe it makes their innate limitations visible to others, whereas those with a growth mindset are less risk averse by seeing failure as an opportunity to learn and develop new skills.

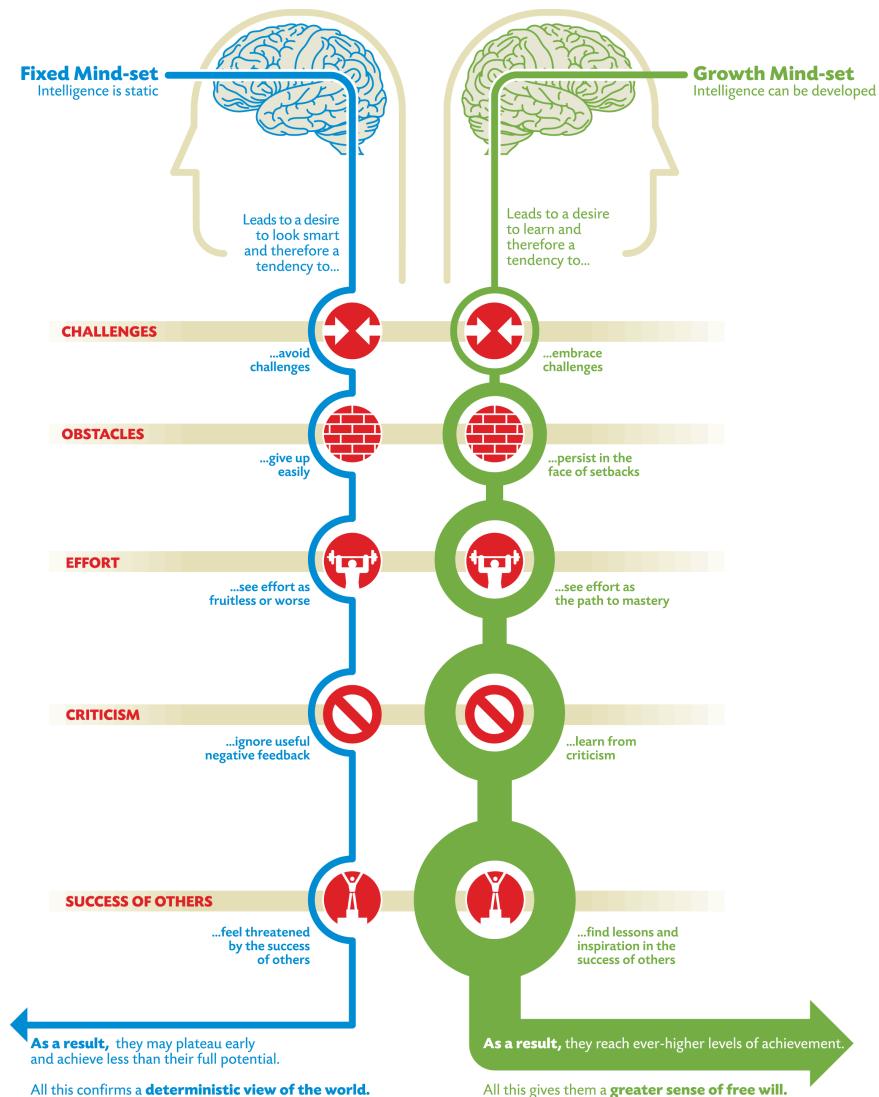


Figure 11-3. Dweck's two mindsets, courtesy of Nigel Holmes

The good news is that we can change our beliefs, as shown by one of Dweck's most interesting experiments.²⁴ Dweck's work shows that if we reward people for the effort they put into solving problems that they find challenging, it shifts them towards a growth mindset. If, in contrast, we praise and reward people for their ability to deploy their existing skills, we create a fixed mindset. This has important implications both for people managers and for HR departments, particularly in the context of performance reviews.

You can be sure that the behavior and attitudes of the people in your organization—your organization's culture—affect the mindsets of the individuals within it, and thus their ability to learn. Thus, organizational culture determines not just the productivity and the performance of the people working in it, but also their ability to gain new skills, their attitude to failure and new challenges, and their goals. Setting people stretch goals that require them to learn new skills, while providing them with support, training, and slack time to reduce learning anxiety, and creating a culture in which collaboration is rewarded and failure leads to reflection and improvement rather than blame—all this works to instill a growth mindset in employees and must be a key goal of organizational change.

Dweck's work tells us there are indeed “A-players” and “B-players.” A-players are simply people with a growth mindset who, upon joining a team, will try to discover how to make the team successful, working to acquire the necessary skills in the process. In contrast, people with a fixed mindset—the true B-players—are the biggest barrier to organizational change and continuous improvement. These are the kind of people who resist experimentation saying that others' approaches “can't work here.” They are also likely to hire people they perceive as worse than them so as to avoid challenges to their status and identity. While such people are capable of changing their mindset, they can also poison attempts to change culture, holding back high-performing teams. To reduce learning anxiety during change efforts, it must be widely publicized that support and resources will be available to help people acquire new skills, that no one will lose their job if they are willing to learn, and that those wishing to leave will receive a generous severance package.

Ultimately, the most important responsibility of an organization's leaders is for its culture, demonstrated by the way they treat others. For example, Dweck argues that while Steve Jobs possessed a growth mindset when it came to his own abilities, he had a fixed mindset attitude towards others: “He wanted

²⁴ <http://gladwell.com/the-talent-myth> is an article well worth reading in its entirety. Dweck's research also has important implications for how we bring up our children, particularly girls who are often praised for being “good” or “pretty,” creating a fixed mindset. This is just one way in which implicit biases reinforce each other. See <http://bit.ly/1zkrLOK>.

them to be perfect and they lived in fear of coming to him and getting his disapproval, instead of his approval.”²⁵

How Google Recruits

Dweck’s work demands that we rethink recruiting. We should not hire people solely on the basis of the skills they already possess. This is particularly short-sighted in the software industry where technology, and thus the needed skills, change so rapidly. Neither should we be using brainteasers or test scores, which Laszlo Bock, senior vice president of people operations at Google, describes as “worthless as a criteria for hiring...They don’t predict anything.”²⁶ Google has done a great deal of research into what makes for an effective recruiting process in the context of technology. The top three criteria are:²⁷

- *Learning ability*, including the ability to “process on the fly” and “pull together disparate bits of information.”
- *Leadership*, “in particular emergent leadership as opposed to traditional leadership. Traditional leadership is, were you president of the chess club? Were you vice president of sales? How quickly did you get there? We don’t care. What we care about is, when faced with a problem and you’re a member of a team, do you, at the appropriate time, step in and lead. And just as critically, do you step back and stop leading, do you let someone else? Because what’s critical to be an effective leader in this environment is you have to be willing to relinquish power.”
- *Mindset*. “Successful bright people rarely experience failure, and so they don’t learn how to learn from that failure...They, instead, commit the fundamental attribution error, which is if something good happens, it’s because I’m a genius. If something bad happens, it’s because someone’s an idiot or I didn’t get the resources or the market moved.”

Bock goes on to observe that the most successful people at Google “will have a fierce position. They’ll argue like hell. They’ll be zealots about their point of view. But then you say, *here’s a new fact*, and they’ll go, *Oh, well, that changes things; you’re right.*” This reflects the advice of Paul Saffo, Director of the Palo Alto Institute for the Future, who says that “to deal with an uncertain future and still move forward,” people should have “strong opinions, which are weakly held.”²⁸

Google’s recruiting strategy is liberating because it enormously expands the pool of qualified applicants. Instead of looking for “purple squirrels” with precisely the skills and experience required for a job, we should look for people who can rapidly acquire the necessary skills and then invest in an environment that enables them to do so.

25 <http://bit.ly/1v72nmV>

26 <http://nyti.ms/1v72xuz>

27 Quotations are from <http://nyti.ms/1v72sHl>.

28 <http://bit.ly/1v72zTg>

Growing Talent

The “talent shortage” problem is solved by creating an environment in which people can learn on the job, and hiring people with a growth mindset. Investing in employee development is one of the few opportunities enterprises have to create a competitive advantage over startups (the others being research and development, and the pursuit of optionality in Horizon 3, as described in [Chapter 2](#)). There are many ways in which enterprises can invest in people:

Help employees create and update personal development plans

To help employees take control of their own development and ensure that managers know how to help them, it’s essential that they, their managers, and people that give them feedback understand their career goals. Creating and regularly updating a simple personal development plan is the foundation of employee development.

Separate performance reviews from compensation reviews

The goal of performance reviews is to provide an opportunity for employees to get feedback on their progress towards their personal development goals, update their goals, and discuss them with their line manager. Coupling performance reviews with compensation reviews, and particularly the practice of “stack-ranking” employees, is based on outmoded ideas of extrinsic motivation which encourage employees to compete rather than cooperate with each other, and reduce employee engagement.

Facilitate regular feedback

Employees should share informal feedback on a regular basis to help each other move towards their personal goals. Good feedback is timely, designed for the benefit of the receiver, and given with permission. In a formal process (such as during a performance review, official reprimand, or exit interview), nobody should hear feedback that they have not already received informally.

Give employees access to training funds

Employees learn through different channels and should have easy access to funds that enable them to buy books, attend conferences and training, or engage in other activities that help them move towards their personal development goals. The conditions for spending should be as liberal as possible, within the limitations of applicable tax regulations.

Give employees time to pursue their own goals

Many innovative organizations reserve time for people to work on whatever they want. 3M has allowed employees to spend 15% of their time on their own projects since 1948. The Post-It Note is just one of the

innovations created as a result of this initiative.²⁹ In *2004 Founders' IPO Letter*, Google's Sergey Brin and Larry Page write, "We encourage our employees, in addition to their regular projects, to spend 20% of their time working on what they think will most benefit Google. This empowers them to be more creative and innovative. Many of our significant advances have happened in this manner. For example, AdSense for content and Google News were both prototyped in '20% time.' Most risky projects fizzle, often teaching us something. Others succeed and become attractive businesses."³⁰

Norman Bodek tells a story about Taiichi Ohno closing down a warehouse at a Toyota subsidiary: "Get rid of this warehouse and in one year I will come back and look! I want to see this warehouse made into a machine shop and I want to see everyone trained as machinists."³¹ Bodek reports that Ohno's orders were carried out, and within one year the warehouse had been replaced with a machine shop and the workers retrained. In line with the standard post-World War II Japanese corporate policy of providing people with jobs for life, Toyota expects to retrain people to do different types of work throughout their careers. Employees at Toyota understand that part of their job is to learn new skills. Toyota provides the necessary training and support for this, removing a great deal of the learning anxiety that is the most serious barrier to creating a learning organization and organizational change. Most importantly, when people are treated with respect and are given opportunities to pursue autonomy, mastery, and purpose, they become highly motivated to deliver value. Employee job satisfaction is the best predictor of organizational performance.

Eliminate Hidden Bias

Another major contributor to the "talent shortage" in technology is the large number of qualified people who decide not to enter the field or quit prematurely. Look at your technology teams and notice that women, in particular, are strongly underrepresented, as are non-white people in the US and the EU. Given that "biological sex differences in inherent aptitude for math and science are small or nonexistent,"³² and the same holds true for differences between races, what is the cause of this underrepresentation?

A number of studies done on recruitment processes aiming to hire on merit universally show that our implicit gender bias plays a strong role in rejecting

29 http://solutions.3m.com/innovation/en_US/stories/time-to-think

30 <http://investor.google.com/corporate/2004/ipo-founders-letter.html>

31 [bodek], p. 29.

32 See [ceci] for the most recent of a number of studies cited in [moss-racusin].

suitably qualified women. In a study performed in 2012, researchers took 127 biology, chemistry, and physics professors from across the USA and gave them job application materials for an undergraduate science student applying for a job as a science laboratory manager. The materials were all identical, but were randomly assigned either a male or female name. Participants were asked to rate the student's "competence and hire-ability, as well as the amount of salary and the amount of mentoring they would offer the student."³³ The results are reproduced in Figure 11-4. Perhaps most interestingly, both male *and* female professors demonstrated the same bias, showing that it is not intentional or explicit but rather "shaped by implicit or unintended biases, stemming from repeated exposure to pervasive cultural stereotypes that portray women as less competent." Other studies have shown the same effects in different domains, as well as a similar effect with regard to race.³⁴

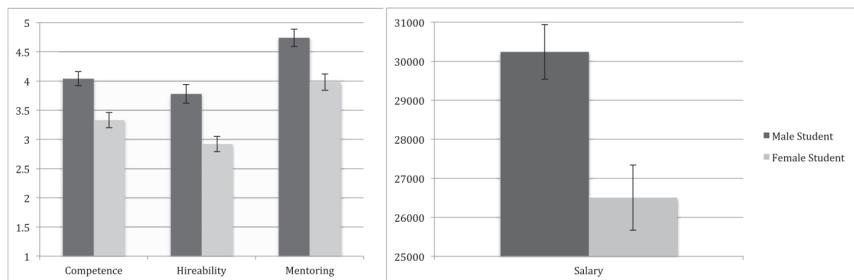


Figure 11-4. The effects of implicit gender bias on hiring

These implicit biases aren't limited to recruitment or gender. Implicit bias and unequal access to resources act at every stage of our educational³⁵ and professional lives, resulting in white male domination in science, technology, engineering, and mathematics (STEM) fields.³⁶ A representative survey of 19,000 people carried out in the USA by the Level Playing Field Institute between 2001 and 2006 found that the annual cost to US businesses attributable to voluntary turnover of managers and professionals due solely to unfairness was \$64 billion. Respondents cited the following behaviors: rudeness, having

³³ [moss-racusin]

³⁴ See, for example, [bertrand] and <http://www.eurofound.europa.eu/ewco/2008/02/FR0802019I.htm> which references [cediey] on implicit race bias and <http://bit.ly/1v72MG7> which references [goldin] on the effect of blind auditions in increasing the number of women in orchestras.

³⁵ University of California, Berkeley, recently redesigned its introductory computer science course, leading to the enrollment of 106 women and 104 men: <http://bit.ly/1v72O0L>.

³⁶ In general, highly compensated professions tend to be male dominated.

coworkers at a similar or higher level who are less educated or less experienced, others taking credit for your work, being given assignments that are usually considered below your job level, feeling excluded from the team, and being stereotyped.³⁷

What can we do about this? Here is a selection of strategies that have proven useful.³⁸ For further reading, consult Freada Kapor Klein's *Giving Notice: Why the Best and Brightest are Leaving the Workplace and How You Can Help them Stay*:

Ensure equitable pay

It's impossible to make exact comparisons between individuals, so instead examine salaries by role. Compare the average salary for white men within a particular role (such as UX analyst or senior engineer) with the average salary of people from underrepresented groups. Correct any discrepancies you find. Netflix' annual compensation review process follows a simple rule: every employee's salary is adjusted to "top of market" by ensuring they are paid "more than [any other company] likely would; as much as a replacement would cost; as much as we would pay to keep them if they had a higher offer for elsewhere."³⁹ If implemented comprehensively, this practice has the effect of redressing pay inequality for historically disadvantaged groups.

Create target conditions for recruitment and promotion

The Improvement Kata can and should be used as part of efforts to increase diversity. Target conditions for hiring and promotion of underrepresented groups are one example of an appropriate use of this tool. One large enterprise wanted to improve the number of women in senior management positions. To avoid accusations of positive discrimination, they didn't create a quota for the positions, but they did impose a target condition for the proportion of women on the list of candidates (for example, "50% of the candidates for the post must be women").⁴⁰ A similar approach can be used for recruiting and team mix.

Monitor tenure, rate of advancement, and job satisfaction

Gather data on the average tenure of white men compared to people from underrepresented groups. Look to see how long it takes different groups to receive promotions. Find out what proportion of each underrepresented

³⁷ [klein], pp. 7–8.

³⁸ An excellent summary of why women leave the tech industry and what to do about it can be found at <http://bit.ly/1toep4k>.

³⁹ <http://www.slideshare.net/reed2001/culture-1798664>

⁴⁰ <http://bit.ly/1v72Rtt>

group has at least one other person reporting to them. Analyze your job satisfaction survey to reveal differences between demographics. Higher employee churn from underrepresented groups, longer time to promotion, and lower job satisfaction are clear indicators of (at best) an implicit bias within your organization.

Regularly review policies, interactions, and HR processes

Implicit bias doesn't just play a role in recruiting—it pervades the corporate environment. To take just one example, women are much more likely to receive critical feedback in performance reviews (the word “abrasive” is almost exclusively used in feedback to women). Similar patterns are apparent in feedback to other underrepresented groups.⁴¹ It's essential to create clear policies, have leaders publicly and regularly set expectations for acceptable behavior, and ensure they model appropriate behavior and are seen to take action in the event of inappropriate behavior. Hire an external expert to review interactions, policies, and HR processes, make recommendations, and return regularly to monitor implementation and review progress.

Conclusion

In a high-performance organization, employees enjoy and take pride in their daily work, and leaders and managers are dedicated to supporting employees in their pursuit of the organization's purpose. No organization does this perfectly, and those that do best are constantly working to get better.

To create this kind of environment, we must address the behavior of everyone in the organization, starting with executives. As John Kotter observes, “a majority of employees, perhaps 75 percent of management overall, and virtually all of the top executives, need to believe that considerable change is absolutely essential.”⁴² The essence of a lean mindset is understanding that this should be the case not just in a crisis but *all the time*. Change, improvement, and development are habitual in a truly lean organization.

Changing culture is achieved by the deliberate, repeated, mindful practice of everyone in the organization. Leaders and managers must facilitate this by investing in employees' development and creating conditions to support people working together to continuously improve processes, knowledge, and the value delivered to customers. Finally, it's essential that leaders model the behaviors they expect the rest of the organization to adopt. Leaders whose actions

41 See <http://bit.ly/1v72Q8Rs> and <http://bit.ly/1v72WNz>.

42 [kotter], p. 51.

contradict their words—particularly when their status is threatened or in times of stress—will lose the trust of their people.

Questions for readers:

- Does your organization send out an anonymous survey (at least annually) to measure job satisfaction and other indicators of culture? Are aggregated results published to estimate progress towards targets for job satisfaction, diversity, and real cultural change? Are the results discussed and acted upon?
- What happens when something goes wrong? Is there a systematic process to learn from accidents in order to improve the systems, or do managers focus on assigning blame?
- What does your organization do to invest in the long-term growth of employees?
- Does your company see culture change as continuous or event based? What practices could you start to move to a continuous model?
- Does your organization hire those with particular skills and experience, or people with the capability and attitude to learn the relevant skills to help their team succeed?
- Has your organization invested in reducing and eliminating the effects of systematic implicit biases? How are you measuring your progress?

Embrace Lean Thinking for Governance, Risk, and Compliance

All things are subject to interpretation. Whichever interpretation prevails at a given time is a function of power and not truth.

— Friedrich Nietzsche

Trust is not simply a matter of truthfulness, or even constancy. It is also a matter of amity and goodwill. We trust those who have our best interest at heart, and mistrust those who seem deaf to our concerns.

— Gary Hammel

We often hear that Lean Startup principles and the techniques and practices we suggest in this book would never work in large enterprises because of governance. “This won’t meet regulatory requirements.” “That doesn’t fit in our change management process.” “Our team can’t have access to servers or production.” These are just a few examples of the many reasons people have given for dismissing the possibility of changing the way they work.

When we hear these objections, we recognize that people aren’t really talking about governance; they are referring to processes that have been put in place to manage risk and compliance and conflating them with governance. Like any other processes within an organization, those established for managing

governance, risk, and compliance (GRC)¹ must be targets for continuous improvement to ensure they contribute to overall value.

There are many large enterprise organizations that have been able to apply lean engineering practices and develop a culture of experimentation as we have described earlier. They are subject to the same level of regulatory compliance and review as others. So we know it can be done.

In this chapter, we aim to guide you through the maze that is GRC, particularly as it relates to managing the concepts and practices required to be a lean enterprise. This area is sometimes poorly understood by those who have not made GRC their career focus, so we present some background to help you reach a common understanding with GRC teams. With that, it should be easier to discuss how GRC processes and controls can be improved to allow product teams to continuously explore and improve their work. We provide some examples of how lean concepts and principles can be applied to improve GRC processes, resulting in better governance and reduced overall risk, while still meeting compliance.

Throughout this chapter, we refer to “GRC teams.” For clarity, our discussion and examples focus on teams that strongly influence how technology can be used within organizations; the more common ones are the PMO, technical architecture, information security, risk and compliance, and internal audit teams.

Understanding Governance, Risk, and Compliance

In the introduction to [Part I](#), we stated that the primary responsibility of leaders is to steer the larger organization towards its goals, adjusting course as necessary. This is governance. Unfortunately, within organizations the term *governance* is often misused and conflated with management theories, models, and processes designed to meet the needs of a bygone era.

Governance is about keeping our organization on course. It is the primary responsibility of the board of directors, but it applies to all people and other entities working for the organization. It requires the following concepts and principles to be applied at all levels:

Responsibility

Each individual is responsible for the activities, tasks, and decisions they make in their day-to-day work and for how those decisions affect the overall ability to deliver value to stakeholders.

¹ Typical GRC processes include access control, solution delivery (project management), change management, and related activities to reduce risks with the use of IT.

Authority or accountability

There is an understanding of who has the power and responsibility to influence behaviors within the organization and of how it works.

Visibility

Everyone at all times can view the outcomes achieved by the organization and its components, based on current and real data. This, in turn, can be mapped to the organization's strategic goals and objectives.

Empowerment

The authority to act to improve the delivery of value to stakeholders is granted at the right level—to the people who will deal with the results of the decision.

Risk is the exposure we run for the possibility of something unpleasant occurring. We all manage risks daily, at work, home, and play. As it is impossible to eliminate every risk, the question to be answered in managing risk is, “Which risks are you willing to live with?” As you take steps to mitigate risk in one area, you inevitably introduce more risk in another area. A classic example of this is restricting development team access to hardware and forcing them to rely on a separate centralized infrastructure team to set up access and environments for testing or experiments. This may be effective for the server support team’s goal of reducing the risk of instability within systems, but it increases the risk of delayed delivery as teams have to submit requests to other teams and wait for them to be fulfilled.

Compliance is obedience to laws, industry regulations, legally binding contracts, and even cultural norms. The intention of mandated compliance is usually to protect the interest of stakeholders with regard to privacy of information, physical safety, and financial investments. When bound by law, regulation, or contract, compliance is not optional. If we choose not to comply, we increase our risk of fines, operational shutdowns, or damage to our reputation. In extreme cases, jail terms can be the outcome of knowingly and systematically misrepresenting an organization’s compliance.

Management Is Not Governance

COBIT 5² clearly explains the difference between governance and management.

Governance ensures that stakeholder needs, conditions, and options are evaluated to determine balanced agreed-on enterprise objectives to be achieved; sets direction through prioritization and decision making; and monitors performance and compliance against agreed-on direction and objectives.

Management plans, builds, runs, and monitors activities in alignment with the direction set by the governance body to achieve the enterprise objectives.

For example, governance involves creating the vision and goals for implementing technology changes at a rate that will allow the business to succeed. It defines what should be measured to determine if we are headed in the right direction to achieve our goals. Management determines how the organization will achieve that vision. In the case of technology changes, that includes structuring of the delivery teams, their boundaries, and what level of decision they are empowered to exercise. Will it be a single, one-size-fits-all, top-down driven process, or will teams be granted autonomy and empowered to make decisions without having to wait for high-level approvals? Good GRC management maintains a balance between implementing enough control to prevent bad things from happening and allowing creativity and experimentation to continuously improve the value delivered to stakeholders.

Take an Evolutionary Approach to Risk Management

A struggle we often experience when implementing GRC structures and processes for compliance is thinking of them as something carved in stone, rather than something that should be changed, modified, and improved. To enable good governance, changes to GRC processes must happen over time in response to the changing needs of the organization and the market environment within which it exists.

When done well, GRC management processes improve value delivery through effective risk management. The intent is to improve communication, visibility, and understanding of who is doing what, when, how, and why, as well as the outcomes of the work that is done. This is strongly aligned with what product

2 As set out in [COBIT5], COBIT formally stands for Control Objectives for Information and Related Technology. It strives to provide an end-to-end business view of the governance of enterprise IT. Auditors as well as risk and compliance teams use the framework and related tools to create and assess governance over the use of technology in delivering value. For more information, see <http://www.isaca.org/cobit/pages>.

delivery teams are trying to achieve. The question then becomes: why are GRC processes viewed as blockers when looking for ways to improve our productivity and the value we deliver to customers and our organization?

Unfortunately, many GRC management processes within enterprises are designed and implemented within a command-and-control paradigm. They are highly centralized and are viewed as the purview of specialized GRC teams, who are not held accountable for the outcomes of the processes they mandate. The processes and controls these teams decree are often derived from popular frameworks without regard to the context in which they will be applied and without considering their impact on the entire value stream of the work they affect. They often fail to keep pace with technology changes and capabilities that would allow the desired outcomes to be achieved by more lightweight and responsive means. This forces delivery teams to complete activities adding no overall value, create bottlenecks, and increase the overall risk of failure to deliver in a timely manner.

Apply Lean Principles to GRC Processes

As with everything else we address in this book, the journey to apply lean principles to GRC processes—and the ensuing results—will look different in every organization, depending on the nature of our business and where we operate. There is no cookbook recipe that fits all circumstances (as reputable frameworks like ITIL³ and COBIT explain). However, lean principles and concepts can be applied to any GRC management process: visualizing the value stream, increasing feedback, amplifying learning, empowering teams, reducing waste and delays, limiting work in process, making small incremental changes, and continuously improving to achieve better outcomes.

A natural tension exists between GRC teams—charged with recommending and advising on how to reduce risks and meet compliance for applicable laws and regulations—and the rest of the organization who just want to get work done, the sooner the better. Tension can be good, though. It sparks creativity, but that creativity is only good if all parties involved know and strive to meet common objectives and are ultimately measured by the same standard. When tension is bad, the result is less collaboration, visibility, and compliance as individuals and teams develop secret ways to circumvent GRC processes. This leads to decisions based on inadequate or inaccurate information, which weakens overall governance.

³ ITIL (Information Technology Infrastructure Library, see <http://www.itil-officialsite.com>) is a framework, evolving over 20 years, providing recommended sets of practices for managing IT based on experience from both public and private sectors. It is largely used by IT management and practitioners.

The GRC teams' goals and objectives usually result in more work for all teams. Some of this is good. Upfront attention to risks, threats, and controls can save a lot of pain during the final steps towards production. Being able to prove we have adequate control measures in place is important during audits and helps keep us in compliance. The challenge is to find the correct balance of control that allows teams to move forward quickly and keeps risks related to compliance down to an acceptable level.

Define the Value of GRC Processes from the Customer Perspective

To get value out of GRC processes such as access control, technical change management, and solution delivery lifecycle, we must always start with a shared understanding of our organization's goals, values, and the intended outcomes of the process. We need a common view of how our daily work contributes to these at the organizational level, no matter with which team we associate ourselves. This means our GRC teams need to take responsibility for the outcomes (good and bad) of compliance and risk management activities and their impact on the ability of teams to deliver in a timely manner. As well, product delivery teams need to understand the language, intent, and purpose of the processes and controls established for compliance and governance. Only then will these teams, who are usually viewed as working at cross-purposes, be able to "stop fighting stupid and make more awesome."⁴

Thus, GRC teams must view themselves as members of the product delivery team, learn about the capabilities of the technology and techniques used in lean engineering, and help teams leverage them to provide evidence of being in compliance without creating waste and bottlenecks. At the same time, the entire delivery team needs to start paying attention to the language and frameworks used by GRC teams to understand what exactly it is that the GRC teams are trying to achieve.

We have seen a lot of waste and destructive tension between GRC and delivery teams because many GRC processes and management practices are disconnected from how teams work. Typically, GRC teams focus on performing and measuring compliance (for example, by asking, "Did everyone follow the activity as described in our framework?"), not on improving the outcomes ("Are we doing what will allow us to meet compliance *and* continue to deliver value in a timely fashion?").

⁴ Jesse Robbins, <http://www.infoq/presentations/Hacking-Culture>

TIP

Avoid the “Wouldn’t It Be Horrible If” Approach to Risk Management

In *How to Measure Anything*, Douglas Hubbard reports Peter Tippet of Cybertrust discussing “what he finds to be a predominant mode of thinking about [IT security]. He calls it the ‘wouldn’t it be horrible if...’ approach. In this framework, IT security specialists imagine a particularly catastrophic event occurring. Regardless of its likelihood, it must be avoided at all costs. Tippet observes: ‘since every area has a “wouldn’t it be horrible if...” all things need to be done. There is no sense of prioritization.’⁵

When prioritizing work across our portfolio, there must be no free pass for work mitigating “bad things” to jump to the front of the line. Instead, quantify risks by considering their impacts and probabilities using impact mapping (see [Chapter 9](#)), and then use Cost of Delay (see [Chapter 7](#)) to balance the mitigation work against other priorities. In this way we can manage security and compliance risks using an economic framework instead of fear, uncertainty, and doubt.

GRC teams are measured by “Are we compliant?”; product teams are measured by “How fast can we deliver value through use of technology?” Both of these are wrong because they measure a team’s performance from an isolated functional perspective and not as the net value for the organization. It is easy to be compliant with laws when GRC teams are allowed to mandate processes and force all boxes to be ticked. However, when team performance measures are not aligned at the organizational level, we can be compliant and still make remarkably bad decisions about delivering value to stakeholders. This is truly ironic, as most related laws and regulations have been established with the intent to protect and improve value to stakeholders.

Rules-based Approaches Lead to Risk Management Theater

When GRC teams do not take a principles-based approach and instead prescribe the rules that teams must blindly follow, the familiar result is *risk management theater*: an expensive performance that is designed to give the appearance of managing risk but actually increases the chances of unintended negative consequences.

At one large European enterprise we worked at, the change approval process involved developers filling in a spreadsheet with seven tabs, which was emailed to a change manager in another country who then decided whether or not to approve it. The change could not proceed without this approval, and if the form was not filled out completely it got sent back. The change manager did not really understand the contents of the spreadsheet; before approving, he relied on conversations with the developers to determine what were the risks and whether the planned mitigation activities were appropriate. The developers knew this and did the minimum possible amount of

⁵ [\[hubbard\]](#), p. 188.

work to fill in the spreadsheet, often just changing the date and title on a previous submission and sending it back as a new request. The change manager knew the developers were doing this, but it made no difference to him so long as the documented process was followed to the letter. It added zero value in terms of risk management, while making it unnecessarily painful for the team to get their changes live. However, compliance was being met through the “evidence” documented on the change request. The real value was realized in the conversations and completing mitigation activities before the change proceeded.

When product teams push back on risk management theater, a common response is that it is required by some popular framework such as ITIL or COBIT, or by a law or regulation such as Sarbanes-Oxley. However, with a few exceptions, neither frameworks nor laws prescribe particular processes. For example, many people think that segregation of duties⁶ is required by Sarbanes-Oxley section 404, so organizations set up elaborate controls over access to IT systems and environments to meet their interpretation of what this means. In fact, nowhere in the act—nor in the SEC rules that were created through the act—is segregation of duties mentioned.

If you find that you are expected to follow a process that compromises your ability to do a good job, it’s worth actually reaching out to the people who created the process to discuss its intent. Return to the Principle of Mission discussed in [Chapter 1](#) and use it as an opportunity to collaborate, build relationships, and develop a shared understanding. You may be surprised to discover that you are able to have a productive conversation about how to meet their goals in a different way, or indeed to see if your work is even in scope for the law or regulation in question. If you are told that a particular process is “required” by some regulation, politely ask where you can find more information about that requirement. In many cases, onerous rules and GRC processes that are put in place are simply somebody’s *interpretation* of what is required, not mandated by the regulation in question.

Map the Value Stream, Create Flow, and Establish a Pull System

With a shared understanding of GRC processes and product delivery team goals and methods, the collaboration to achieve organization-level goals can really begin. As discussed in [Chapter 7](#), value stream mapping is a powerful tool that can be used to provide us with a view of the current state and identify areas for improvement. In the context of GRC processes, it is important to layer these on top of the delivery team activities and understand how they influence the ability of the team to get their work done.

⁶ *Segregation of duties* is a concept that seeks to prevent errors and malicious activities by an individual by requiring at least two people to complete any end-to-end transaction. Another way to approach it is to ensure no one person can complete a transaction without it being detected or controlled by at least one other person.

Most GRC processes are designed in isolation to apply controls such as required approvals, limited access, segregation of duties, monitoring, and review of activity. These are meant to provide visibility and transparency into who does what, when, and with what authority. More importantly, the frameworks commonly used by GRC teams to create the processes emphasize improving overall efficiency and effectiveness for the organization. Unfortunately, many of the processes and controls do the exact opposite when considered in the larger end-to-end value chain.

The Wrong Control Interrupts Flow

Controls can be preventive in nature by the application of a barrier. Alternatively, they can be detective—monitoring and reviewing events after they occur, and eliciting an appropriate response to the discovery of potential exceptions such as errors, omissions, or malicious actions.

Many of us make the mistake of thinking that preventive controls are more effective: if we can create barriers or take away people's ability to do things, it won't happen. The reality is, people need to get things done. If you try to stop them, many will get creative and figure out ways to work around whatever barriers have been put in place. The reactive response is then to lock everything down even more, which emboldens further creative underground solutions to get the work done, fomenting a subversive culture of risky behavior. A good example is teams who will share an elevated user ID and password to access different environments. It would be far better to give each team member access under their own IDs and then monitor their use of those privileges.

An even more tragic outcome of too many preventive controls is when teams just stop caring and assume an automaton mode of operation, abandoning all efforts to make things better.

Preventive controls, when executed on the wrong level, often lead to unnecessarily high costs, forcing teams to:

- Wait for another team to complete menial tasks that can be easily automated and run when needed
- Obtain approvals from busy people who do not have a good understanding of the risks involved in the decision and thus become bottlenecks
- Create large volumes of documentation of questionable accuracy which becomes obsolete shortly after it is finished
- Push large batches of work to teams and special committees for approval and processing and then wait for responses

If preventive controls are not executed properly and consistently, they are no longer effective. They must be continuously monitored to ensure they have

been applied correctly and are still relevant. Without monitoring and resulting corrective actions, preventive controls are less effective than well-executed detective controls such as ongoing monitoring, early and frequent testing and review, and highly visible measurement of outcomes.

Although relying on preventive controls may contribute to a false sense of security, they are extremely valuable when applied at the right level, and are the best solution in certain circumstances. However, they should never be applied unilaterally but only in conjunction with other controls and to the correct level of granularity, and we must always consider their effect on the ability of teams to get their work done.

Therefore, when we perform value mapping of governance processes on top of delivery team processes, we need to look carefully at all of the controls and ask two questions:

- Is the intent of the control being met?
- Is it truly contributing to overall effectiveness and efficiency of the organization?

We need to look carefully at the level of authority granted to our teams. The goal is to bring the approval decisions to the right level and give teams as much authority as possible to enable them to keep moving. This involves defining boundaries and making sure the team knows how and when to escalate decisions that fall outside their authority. We also need to make sure documentation is kept to a sane level and, when done, make sure it is accessible, easy to understand, and updated as required, preferably automatically.

“Trust, but verify”⁷ is a concept that is gaining acceptance in GRC circles. Instead of preventing teams from accessing environments and hardware so they can’t do anything bad, we trust people to do the right thing and give the team access and control on the systems and hardware they need to use daily. We then verify the team is not abusing their authority by developing good monitoring and frequent review processes to ensure the established boundaries are observed and there is complete visibility and transparency built into the team’s work.

⁷ This saying, popularized by Ronald Reagan, is originally a Russian proverb.

Reducing Feedback Loops on Compliance Activities

Meeting compliance for Information Security has been a thorn in the side of many delivery teams. In the spirit of the big bang project delivery methodology, the security team is brought in at the latest possible moment—days before we go live—to run a final code review for security vulnerabilities and required compliance.

The Information Security community now realizes this approach doesn't work. On most products, there is just too much complexity and volume to complete a meaningful review. When vulnerabilities or other breaches in compliance are discovered this way, it is generally too late to do much about it. It becomes more risky to fix the vulnerabilities in a fragile system, or wait for the changes, than it is to allow the vulnerabilities to go to production with a promise to fix them later.

To meet compliance and reduce security risks, many organizations now include information security specialists as members of cross-functional product teams. Their role is to help the team identify what are the possible security threats and what level of controls will be required to reduce them to an acceptable level. They are consulted from the beginning and are engaged in all aspects of product delivery:

- Contributing to design for privacy and security
- Developing automated security tests that can be included in the deployment pipeline
- Pairing with developers and testers to help them understand how to prevent adding common vulnerabilities to the code base
- Automating the process of testing security patches to systems

They also create their own environments for performing mandatory code reviews and security testing so they don't block the team from performing other work while this is done.

As working members of the team, information security specialists help shorten feedback loops related to security, reduce overall security risks in the solution, improve collaboration and the knowledge of information security issues in other team members, and themselves learn more about the context of the code and the delivery practices. Everybody wins.

As we become better at creating flow for teams by changing governance processes, GRC teams benefit as well. Using controls designed in collaboration with GRC teams, product delivery teams are able to embed evidence of true compliance into daily work and tools, and do away with risk management theater. As we do with functional and performance quality, we build evidence of compliance into our daily work so we don't have to resort to large batch inspections after most of the work has been done.

The net effect for GRC teams is that they can now pull information related to compliance from product delivery teams at any time without interrupting the team's overall workflow, unless something untoward or unaccountable seems to be happening. Annual audits are less painful because the delivery teams understand the intent of the controls the auditors are asking for and can give evidence of meeting that intent through their processes.

By using an economic framework (such as Cost of Delay, discussed in [Chapter 7](#)) we can quantify the economic trade-offs we make when we implement controls to mitigate risk. This allows us to prioritize GRC work against the other kinds of work we do—and thus pull additional work required for compliance at the right time for the business.

Case Study: PCI-DSS Implementation at Etsy

Etsy is an online handmade and vintage marketplace with over \$1bn in gross merchandise sales in 2013. In Etsy's high-trust culture, developers normally push their own changes live—indeed, as part of onboarding new engineers, developers use the automated deployment system to update their profile on the live site within their first few days. Engineers are also allowed to work on—and have access to—all parts of the system.

However, since Etsy processes credit-card transactions, it is subject to PCI-DSS, an industry standard that is quite prescriptive in how to manage systems that store or transmit payment cardholder data (these systems are known as the cardholder data environment, or CDE). For example, the CDE *must* be physically segregated, and there *must* be segregation of duties for people who work on systems within the CDE.

Segregation of duties is usually interpreted to mean (among other things) that developers should not have access to the production database and should not be able to push their own changes live. Both of these requirements conflict with the way Etsy typically operates. Here's how they approached PCI-DSS compliance.

1. Minimize the fallout of the required compliance. Understand there is no one-size-fits-all compliance solution, and architect systems to separate the concerns related to different compliance demands.

Etsy's mainstream engineering culture is optimized for speed of innovation. However, credit card processing is an area where user data security is paramount. Etsy recognizes that different parts of their system have different concerns and need to be treated differently.

Etsy's most important architectural decision was to decouple the CDE environment from the rest of the system, limiting the scope of the PCI-DSS regulations to one segregated area and preventing them from "leaking" through to all their production systems. The systems that form the CDE are separated (and managed differently) from the rest of Etsy's environments at the physical, network, source code, and logical infrastructure levels.

Furthermore, the CDE is built and operated by a cross-functional team that is solely responsible for the CDE. Again, this limits the scope of the PCI-DSS regulations to just this team.

2. Establish and limit the blast radius of frameworks and regulations.

Always start by asking, “What’s the smallest possible set of changes we can make to our ideal architecture and culture while still achieving compliance with regulations we are subject to?” Then take an incremental, iterative approach to implementing and validating those changes.

For example, while PCI-DSS mandates segregation of duties, that doesn’t prevent the cross-functional CDE team from working together in a single space. When members of the CDE team want to push a change, they create a ticket to be approved by the tech lead; otherwise, the code commit and deployment process is fully automated as with the main Etsy environment. There are no bottlenecks and delays, as the segregation of duties is kept local: a change is approved by a different person than the one doing it.

3. Use compensating controls.

It’s essential to respect the outcomes the regulations are trying to achieve, while recognizing there are many ways to achieve those outcomes. For example, PCI-DSS allows organizations to implement “compensating controls”—a workaround designed to create the same outcome—where there is a legitimate technical or business constraint preventing implementation of a particular control.⁸

In the case of PCI-DSS, you should talk to your qualified security auditor (QSA) and acquiring bank to discuss possible alternatives to controls that have an unacceptable technical or business impact. For example, the deployment pipeline described in [Chapter 8](#) and used by Etsy provides a powerful set of compensating controls that can provide an alternative to segregation of duties in their other systems.

The advantage of using lean principles and continuous delivery in product development is that it enables a fine-grained, adaptive approach to risk management. As we work in small batches and are able to trace each change to our systems from check-in to deployment, we can quantify the risk of each change and manage it appropriately.

The best way to achieve the objectives of good GRC is by embedding compliance and risk management into the daily activities of product teams, including systems and UX design and testing. As organizations move away from the command and control paradigm and GRC teams adopt a collaborative approach to risk management, we begin to value them as trusted advisors and experts in their knowledge domain. For many GRC teams, this requires a major shift in their roles, responsibilities, and behavior within an enterprise

⁸ <http://bit.ly/1v732EU>

organization. This is the move from a policing role to that of a contributing team member who is measured on the same outcomes as the product team, not solely a compliance perspective.

Conclusion

Good governance requires everyone to focus on discovering ways to improve value and provide accurate information on which to base our decisions. We start with leadership and direction from the Board and Executives, and rely on the ability of employees to embrace their responsibility to make good decisions at work. A culture of openness, trust, and transparency is required for good governance.

GRG structures and processes must be developed collaboratively by both GRG teams and the product teams that work day to day to deliver value to customers. By identifying the intent of the laws and regulations we must comply with, our GRG teams can collaborate with product teams to determine local approaches that fit best with improving value delivery. We start by exploring, with GRG teams, how we can minimize the negative effects of relying on restrictive controls through creative use of system architecture, process improvement, containment of scope, applying compensating controls, and leveraging new technologies. We can then exploit our learning to continuously improve our processes to provide both better governance and better outcomes for all stakeholders.

Questions for readers:

- How do your product teams view your current GRG processes? To what extent is your organization engaged in risk management theater?
- What actions do leaders take to develop a shared understanding of GRG language and frameworks throughout the organization?
- Do your GRG structures (policies, organization, and processes) prevent product teams from performing process improvement or require them to seek approval for any process change? If so, how might you support teams improving their processes while maintaining compliance?
- How might you enable GRG teams to collaborate with your product delivery teams as trusted team members throughout the value creation process?

Evolve Financial Management to Drive Product Innovation

Adhering to budgeting rules shouldn't trump good decision-making.
— Emily Oster

Right now, your company has 21st-century Internet-enabled business processes, mid-20th century management processes, all built atop 19th-century management principles.

— Gary Hamel

Introduction

In many large enterprises, financial management processes (FMPs) are designed around the project paradigm. This presents an obstacle to taking a product-based approach to innovation. It is relatively easy for small teams to work and collaborate amongst themselves. However, on an enterprise scale, we eventually reach a point where evolution is blocked by rigid, centralized FMPs that drive the delivery and procurement processes that limit the options for innovating at scale. We will address some of the problems created by these FMPs, with particular emphasis on the budgeting process. We emphasize that to work through the issues your organization experiences as a result of financial management processes, you will need the help of your finance team. Start building good relationships with them and work collaboratively to improve outcomes for customers and the business.

Recognition of the interdependence of *all* management processes, the detrimental behaviors they can enforce, and the barriers they present to continuous improvement and innovation is essential to success at becoming lean. It is hard

to let go of the long-held belief that strong, centralized control provides valuable efficiencies. However well it may have served us in an era of lower complexity and slower technical advances, it now creates barriers that prevent us from adapting quickly to emerging opportunities. In this context, the resources and efforts required to gather information, communicate, and monitor rigid centralized processes outweigh any efficiencies gained. As well, a strongly controlled centralized budget process encourages competitive, rather than collaborative, internal behavior. This is counter-productive to innovation, which requires teamwork.

Many large multinational organizations have transformed themselves by dropping the long-held belief that command and control is the best way to manage their financial processes. As further reading on this topic, we recommend *Beyond Budgeting*¹ and *Implementing Beyond Budgeting*,² as well as the Beyond Budgeting Round Table website.³

Dancing to the Beat of the Financial Drum Slows Innovation

Planning, budgeting, forecasting, and monitoring are essential for defining our success, in particular our commitment to shareholders. Relatively new or revised regulations and standards, such as Sarbanes-Oxley and International Financial Reporting Standards, have intensified the perceived need to centralize and control these processes. However, the intent of these regulations is to improve transparency and visibility into financial reporting, as well as our ability to make better decisions. Centralized control and decision making through annual budgets can easily create the opposite outcomes.

In this chapter, we consider the organizational financial management practices within enterprises that are typically identified as deterrents to innovation:

- *Basing business decisions on a centralized annual budget cycle*, with exceptions considered only under extreme circumstances. This combines forecasting, planning, and monitoring into a single centralized process, performed once a year, which results in suboptimal output from each of these important activities.
- *Using the capability to hit budget targets as a key indicators of performance for individuals, teams, and the organization as a whole*, which

1 [hope]

2 [bogsnes]

3 <http://www.bbprt.org>

merely tells you how well people play the process but not the outcomes they have achieved over the past year.

- *Basing business decisions on the financial reporting structure of capital versus operating expense.* This limits the ability to innovate by starting with a minimal viable product that grows gradually or can be discarded at any time. The CapEx/OpEx model of reporting costs is largely based on physical assets and is project based; it does not translate well to the use of information to experiment, learn, and continually improve products over time.

Combined, these practices force us to time key business decisions and annual work plans for the optimization of the finance department and reporting cycles, which in turn restricts when and how business innovation within the organization occurs. They are out of step with our ability and need to continually deliver value to customers. Large, fully funded, bloated programs of work that deliver questionable value grind on whilst new, unanticipated opportunities drift by because there is no funding available for exploring and testing our hypotheses about them. Time that could be spent on innovation is instead spent on managing and reporting on “the budget.”

Liberating Ourselves from the Annual Budget Cycle

Centralized budgeting processes are typically used to plan, forecast, monitor, and report on the financial position and overall performance of an organization. They drive everything from revenue target reporting to tax planning and resource allocation. However, in the context of product development, the traditional annual budget cycle can easily:

- Reduce transparency into the actual costs of delivering value—costs are allocated by functional cost centers or by which bucket the money comes from, without an end-to-end product view.
- Remove decisions from the people doing the work—the upper management establishes and mandates detailed targets.
- Direct costs away from value creation by enforcing exhaustive processes for approving, tracking, and justifying costs.
- Measure performance by the ability to please the boss or produce output—not by actual customer outcomes—by rewarding those who meet budget targets, no matter what the overall and long-range cost may be.

However, many large enterprises have found alternatives to the traditional centralized budget process to achieve the goals of good financial management. [Figure 13-1](#) emphasizes the importance of separating out the goals of budgeting and suggests possible approaches.

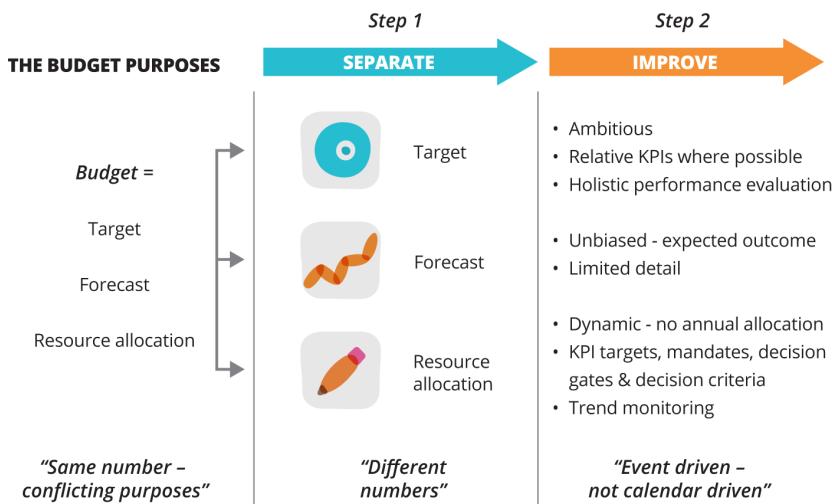


Figure 13-1. Approaches to achieving the goals of budgeting, courtesy of Bjarte Bogsnes, author of *Implementing Beyond Budgeting: Unlocking the Performance Potential*

Stop Conflating Good Financial Management with “The Budget”

I hate the yearly budget with a fire of a thousand suns.

— Anonymous

A budget should be viewed as the total sum of funds set aside, or needed, for a purpose: “What is the ceiling for how much we may spend on this activity?” It does not define what we are actually going to do—that is *strategy*. It is not a plan for how to achieve the strategy, nor does it forecast or measure our success in delivering value to customers. When we roll all of these essential activities into the budgeting process, we lose our focus.

Having a budget is a good thing, especially when we have set some stretch financial targets for ourselves. Financial constraints can be a strong catalyst for creativity, collaboration, and innovation. Particularly in the explore domain, we can spur innovation if we purposefully reduce funding to localized areas or products and allow teams to decide how they can best utilize available funds, as we describe in **Part II** of this book. However, this approach will not work if we simply reduce funding and tell teams what their targets are and how to achieve them.

Following the principle of subsidiarity described in **Chapter 10**, the responsibility to manage allocated funds should be pushed to the lowest appropriate level—generally, the people who are performing the work. We still need to provide teams with clear definitions of what is off-limits, but the teams need to be

trusted and given the chance to make decisions. As described in *Implementing Beyond Budgeting*, when European petrochemicals giant Borealis took this approach, they expected that costs would go up. Instead, they went down.⁴ Although Borealis was well positioned and prepared for the change with a culture that supported the move, CFO Bjarte Bogsnes attributes most of the outcome to better visibility into cost drivers through the use of activity-based accounting principles:⁵ those responsible for the activities that generate costs report on their finances, and teams assume responsibility for better management of costs.

The great planning fallacy, evident in the centralized budget process, is that if we develop a detailed upfront financial plan for the upcoming year, it will simply happen—if we stick to the plan. The effort to develop these kinds of plans is a waste of time and resources, because product development is as much about discovery as it is about execution. Costs will change, new opportunities will arise, and some planned work will turn out not to generate the desired outcomes. In today’s world of globalization, rapid technology growth, and increasing unpredictability it is foolish to think that accurate, precise plans are achievable or even desirable.

A better approach is to set high-level long-term goals, carefully manage the more predictable near future, and constantly adjust our shorter-range plans to get closer to our targets. We can adopt this approach by implementing *strategy deployment* described in [Chapter 15](#). Strategy deployment takes the Improvement Kata meta-method presented in [Chapter 6](#) and makes it cascade through the whole organization, following the Principle of Mission described in [Chapter 1](#). Bjarte Bogsnes presents a similar approach called “Ambition to Action” in *Implementing Beyond Budgeting*.⁶

⁴ [bogsnes], p. 90.

⁵ Activity-based accounting is an approach to costing and monitoring activities that involves tracing resource consumption and costing final outputs [[CIMA](#)].

⁶ Ambition to Action, presented in Chapter 4 of [bogsnes] from p. 114 onwards, is derived from Kaplan and Norton’s Balanced Scorecard approach.

TIP

Replace Annual Budgets with Rolling Forecasts

Rolling forecasts are one tool that can be useful to help improve financial planning and decrease dependency on the budget. As every period is completed, another is appended to the far end of the forecast so that it always covers the same length of time into the future. The far end doesn't provide great detail, but does include known cost line items with estimates on what they will be for the period in question. In rolling forecasts, attention is focused on the near future, based on current and accurate information. We don't spend as much time chasing details further out into the future that are likely to change in an unknown way.

In adopting this approach, remember that the forecasts are not meant to define targets or manage resources. Unless you use an approach such as strategy deployment or Ambition to Action to set targets and manage resources and performance, you will end up with a rolling budget instead of rolling forecasts, which Bogsnes describes as "a bit more dynamic but also four times more work."⁷

As we separate activities required to perform good financial management from the annual budget process, we improve our ability to understand our current condition. We focus on developing flow in decisions and adjustments required to meet the targets we have set for ourselves. The shift is from "Do I have the funding to do what I am told to do?" to "Is this really necessary?"

Disassociate Funding Decisions from the Annual Fiscal Cycle

The use of the traditional annual fiscal cycle to determine resource allocation encourages a culture that thwarts our ability to experiment and innovate. It perpetuates spending on wasteful activities and ideas that are unlikely to deliver value. We must recognize innovation has an ongoing cost that can't be defined and fully planned a year in advance. We need practical lightweight processes to fund innovation, and to be disciplined about stopping work on anything that's not generating the desired outcomes.

When an annual process is the only avenue for obtaining funds tied to specific line items or new initiatives, it is nearly impossible to change direction in response to new information. Instead, every year we must spend a great deal of effort to present the best business plan to get as much funding as we possibly can, instead of being honest about what we think we need. Of all the submissions made during this annual event, only those with the most compelling story make it through unchanged. The rest get cut, or put on the backlog for consideration next year, accumulating delay cost.

⁷ Personal communication.

Instead, some companies are taking an approach known as *dynamic resource allocation* shown in [Figure 13-2](#). This creates more frequent checkpoints for funding decisions, and each decision has less risk associated with it. All decisions are based on the empirical evidence, so they become easier to make. When done correctly, access to funding expands to more teams, gets more frequent, has less associated risks, and brings better results. We thus encourage more innovation and reduce financial risks associated with large initiatives.

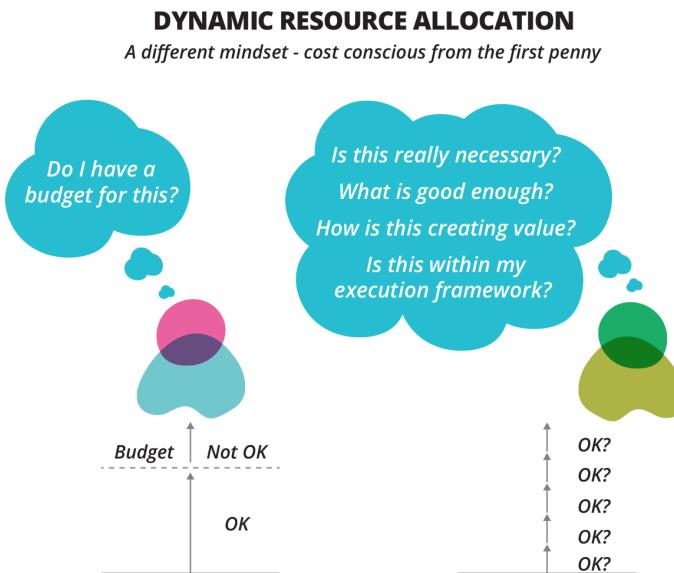


Figure 13-2. Dynamic resource allocation, courtesy of Bjarte Bogsnes, author of *Implementing Beyond Budgeting: Unlocking the Performance Potential*

The product development model we discuss in this book works well with dynamic resource allocation. When we have a new idea, we must begin with an explore phase. The cost of exploring the idea can be measured in terms of the product team's operating costs. Boundaries are defined: you can have a small team for a defined period, and the maximum amount to spend is X. Once the team has evidence the idea will deliver value, we can provide further funding to move into the exploit domain. Across Horizon 3 as a whole, we aim to use the Principle of Optionality to manage our investments (see [Chapter 2](#) for more on the three horizons and optionality). Our goal is to invest limited resources in a number of possible options, with the expectation that most will fail but a few will show a large upside.

Teams that successfully exit the explore domain and scale up will begin to practice continuous improvement, as described in [Chapter 6](#), to constantly

remove waste in the delivery process. It's essential to avoid "rewarding" teams that achieve performance improvements by reducing their operating costs, cutting team size, or breaking teams apart. This instantly demotivates teams and kills the innovation mindset. Instead, the team should get to spend more time on exploring new ideas without onerous documentation, reviews, and approvals—as long as they maintain their high performance and keep costs within established boundaries. By creating lightweight processes to approve small blocks of additional funding to support the exploitation of ideas, we keep the momentum going.

By using a product paradigm rather than a project paradigm, it becomes much easier to calculate profit and loss on a per-product or per-service basis. We can calculate the costs of delivering and running a product or service simply through the operating costs of the team building and running it. This makes it much easier to see when the costs associated with a product or service exceed the value it provides, or when we are not obtaining the expected margin. When we want to build features that cut across multiple products, we can use Cost of Delay to make an investment decision (see [Chapter 7](#)).

As the value proposition and development and support costs of a product change over its lifecycle, we can change the composition of the team running and enhancing it. Finally, when the product starts delivering a negative value, we should retire it sooner rather than later. Often, an investment is required in order to retire products and services—and, again, Cost of Delay can be used to make an investment decision. This can require buy-in from executives: we know of one Fortune 500 company that gave bonuses to its VPs based on the number of services retired during the year, aiming to reduce system complexity and encourage innovation.

Smaller, simpler, local initiatives involving less risk should go through less review and a lighter approval process than complex, enterprise-level initiatives. Hand-in-hand with this, we need an ongoing process and defined criteria for when funding will cease. Review and oversight can be decentralized by creating local teams responsible for reporting the outcomes of their funding decisions. This can be rolled up for enterprise-level reporting. We still want to maintain high-level centralized control over larger enterprise initiatives, but there should be very few of these at any point in time. See [Table 13-1](#) for some sample funding models.

Table 13-1. Sample funding models

Relationship complexity	Focus	Rate of change required	Funding model
Simple, one to one	Customer-facing	Fast—multiple times a day, daily, or weekly	Short duration—2 weeks. Small teams. Small funding blocks. Use temporary infrastructure.
Interdependence between two or three product teams	Middle value—orchestration of business value between product teams	Moderate—2 weeks to 3 months	Short duration—2 to 4 weeks. Small, mixed product teams. Small funding blocks. Use temporary infrastructure initially.
Enterprise-level	Core operations—e.g., ERPs, CRMs, Big Data, reporting	Slower—less than 4 times a year	Longer duration—3 to 6 months. Start with small teams and build up over time. Continued funding decision every 4 to 6 weeks. Larger funding blocks to support core infrastructure changes.

Getting rid of a highly centralized annual budget cycle does not mean we are shirking our responsibilities for good financial management. Many large global companies, including Handelsbanken, Maersk, and Southwest Airlines, have started journeys to escape large centralized budgets and manage costs through other means.⁸ They start by decentralizing financial responsibility for operations and moving it down to individual business units:

- Senior management doesn't set the targets for all costs and revenues for the upcoming fiscal year.
- Critical business decisions are not based on the budget.
- Teams and individuals are not measured by their ability to stay within budget.

Everyone still has targets and is held responsible for improving the value they deliver. However, these targets are not mandated from the top but set by teams themselves, aligned with the organization-level goals and targets.

Explore Activity-Based Accounting Principles

Resource consumption should be directly tied to activities that generate value. Traditionally, costs are tracked solely by functional cost centers, such as IT,

⁸ <http://www.slideshare.net/LESSConf/11-12-what-is-bb>

and there is little visibility into what drives these costs. IT departments and teams engaged in product development are often viewed as cost centers that can be managed and controlled independently from the business. Tradition thinking is that sourcing cheaper supply of IT services will reduce costs and provide equally good outcomes. If it were that easy, you probably wouldn't be reading this book. The reality is that our business drives our IT and product development costs, and we can't manage them independently.

Activity-based accounting (or costing) allows us to allocate the total costs of services and activities to the business activity or product that drives those costs. It provides us with a better picture of the true financial value being delivered by the product. However, like all models and approaches to business, activity accounting is not a panacea. We need to be careful that we do not pursue unnecessary precision, creating complicated models and processes that outweigh the value we aim to provide. The goal is simply to get better information for adjusting plans and activities to improve value—starting small and stopping when we have enough empirical evidence on which to base our decisions.

Making Better Decisions with Activity-Based Costing

Here is a story from work at a former employer that shows how activity-based accounting gives clarity into how technology supports value to customers.

In an attempt to cut costs, our executive finance committee had mandated unrealistic targets for the upcoming fiscal year. They had difficulties relating to the established budget line items, such as cost of servers, and didn't understand why we just couldn't reduce our staffing levels and support more people and systems at the same time. To give them a picture they could relate to, we turned to activity-based accounting principle of allocating costs to business activities (operations, revenue management, marketing, customer relations, supply chain management, etc.), rather than the line items in our budget (IT people, software, hardware, IPS, servers, etc.). Our financial management system was not set up to provide this view, and we had a tight deadline, so we used what we had at hand: spreadsheets, current operational numbers, two people, two days, full access to IT senior management for information, and plenty of food and beverages.

We didn't focus on being 100% accurate or precise: we figured 90–95% accuracy was good enough. Our goal was to give the executives the big picture of how IT costs were related to servicing our customers and the growth of our organization.

Fortunately, we already had a clear understanding of our IT services and costs related to the business activities. We were able to link many costs directly to a business product or service. For example, our customer support center got all of the costs associated with interactive voice recognition software. Other costs, such as email services, had to be divided between business units, so we took their number of people as a measure and divided those costs proportionately. We also allocated specific costs to our own

department—those related to department service management and our own consumption of common services.

The output from this effort was a series of graphs and charts that showed how business activities drove the IT costs. When presented with this information, rather than the traditional budget expense line items, executives were more comfortable making decisions on what to support (or not) in our budget submission. Most importantly, everyone got a better picture of the true cost of ownership of business products and services and we were able to better calculate the cost of delay for retirement and replacement of systems.

Avoid Using Budgets as the Basis for Performance Measurement

Perhaps the biggest mistake we can make with budgets is to use them as a key indicator of performance—to base reward and recognition on the capability of an individual, a team, or the organization as a whole to adhere to a budget. Staying within an assigned budget only tells us if we spent or earned as much as we said we would. If we tell teams they are going to spend more or less than they need to do their work, they will find a way to make it happen or spend a great deal of time justifying why they couldn't. However, this prevents us from paying attention to the most important questions: did we plan at the right level, set good targets, get more efficient, or improve customer satisfaction? Are our products improving or dying? Are we in a better financial position than we were before?

Bonuses and rewards for good bottom-line financial results work better when they are shared equally—not just with upper management and executives but with every employee within the organization. Working teams will eventually cripple the organization by inertia and subterfuge when their contributions are not acknowledged and rewards are based on a process perceived as unfair. Conversely, people tend follow good leaders and make the organization great when recognition and incentives are shared equally with all.

Financial Incentives with Positive Impact: The Case of WestJet

WestJet has been one of the most financially successful airlines in North America over the past 15 years. WestJet's founders knew that, if they were to succeed, it was essential to create a culture of responsibility and ownership for all employees. To create this culture, they clearly stated their strategy and goals, screened and trained employees for a good culture and values fit, and established financial incentives that benefit all employees.

Twice a year, a portion of the company profits are distributed to all employees, prorated on their base salary. All employees are invited to attend the profit share party where physical cheques are handed to team members by their managers—face-to-face whenever possible, so managers can personally recognize every employee for their contribution.

In addition, WestJet allows employees to voluntarily purchase up to 20% of their base gross pay in WestJet shares, and then matches the employee's contribution in shares under the employee's name. In 2012, over 85% of employees participated in this program, becoming part owners of WestJet.

These financial incentives have helped to establish a true sense of responsibility and ownership for all employees, from call center agents to executives. Everyone knows that the decisions they make in their day-to-day work and the way they treat their guests will have a direct effect on the overall earnings of WestJet; they will personally share the rewards, or sorrows, resulting from those decisions.

This approach has helped WestJet remain profitable in a tough, highly regulated industry for close to two decades. As of the end of 2013, WestJet has reported an annual profit in 17 of the 18 years of its operation.⁹

Stop Basing Business Decisions on Capital Versus Operational Expense

Concern over capital (CapEx) versus operating (OpEx) expense reporting is important for organizations. There are tax advantages and positive financial impacts from reporting organization expenditures appropriately in these different buckets, so a lot of attention is paid to it. The basic premise of capitalizing software systems is they are viewed as an asset that creates future benefits for our organization. This can have significant impact on balance sheets and, in turn, on the market value of an organization.

⁹ <http://www.westjet.com/pdf/greatWestJetJobs.pdf>, <https://www.westjet.com/pdf/global-reporting.pdf>, WestJet Management Discussion and Analysis of Financial Results 2013, <http://bit.ly/1v73i6N>.

Unfortunately, this distinction is often used as the foundation on which critical business decisions are made. It injects another element of complexity into decision making and funding for innovation. All costs associated with any work must be categorized into one of the two buckets, and the traditional process for managing the buckets assumes that a team's work is one or the other but can't be both at the same time.

The traditional process also serves to obscure the true cost of ownership and escalates operating costs. A *project* will be fully capitalized, allowing us to spread out the reporting of that cost over an extended period, so it has less short-term impact on our profit. However, many of the items that are being capitalized during the initial project have an immediate negative impact on our OpEx, starting before or immediately after the project dissipates. The long-term operating costs required to support the increasing complexity of systems created by projects are not calculated when capitalized projects are approved (because they don't come out of the same bucket). Ongoing support and retirement of products and services is an OpEx problem. In the end, OpEx teams are stuck with justifying their ever growing costs caused by the bloat and complexity created by CapEx decisions.

If we are serious about innovation, it shouldn't really matter which bucket funding comes from. Open, frank discussion, based on real evidence of the total end-to-end cost of the product, is what we should use as the basis of business decisions. Funding allocation of a product's development into CapEx or OpEx should be performed by accountants *after* the business decisions are made.

Let's look at the explore domain first. Most ideas turn out to deliver zero or negative value—but CapEx applies to assets that deliver long-term value. It therefore makes sense to consider all explore activities to be OpEx. We still need teams to define the amount of resources they intend to consume on exploring, but they shouldn't need to obtain further funding approvals every time they want to try something new within their defined boundaries.

Moving to the exploit domain, finance and product teams need to talk to each other frequently to determine how to allocate funding. We want to avoid adding complexity and unnecessary waste when we decide on the tracking methods to determine CapEx versus OpEx allocation. It should be easy enough for everyone to understand how it works and provide a fairly accurate representation of the long-term value proposition. In the end, allocating costs into CapEx could be as simple as defining that a fixed percentage of this teams' resources (or time) are spent on developing assets that have a lifespan long enough to be capitalized. Here are some topics to discuss with Finance regarding CapEx funding decisions:

- How can we build a flexible model for allocating funds based on CapEx and OpEx principles but avoid using rules and rigid processes that require everyone to compete for all funding at the same time?
- What elements, other than projected project costs, should affect the amount of scrutiny and rigor we apply to funding decisions: complexity, time estimates, team size, net effect of operating costs, anything else?
- How can we manage local initiatives versus enterprise-level initiatives to reduce delays and overall response time for local opportunities?
- How can we structure funding decisions to accommodate shorter time frames and improve availability to more teams?
- How do we base further funding decisions on demonstrated delivery of working products, as opposed to the amount of activity?

In these discussions, it is important to consider the projected lifespan of the product. Technically, software assets should be capitalized only if the projected lifespan of the product matches or exceeds the current depreciation term for software products—most businesses currently use three years. However, it seems unrealistic to believe that all of these systems have a valuable lifespan of three years *if left unchanged*. This presents an interesting question. Which is less risky and more responsible:

- Categorize software product development entirely in OpEx, or
- Capitalize costs and claim a write-down in the future if the product is retired before it is totally depreciated?

There's probably no single definitive answer; you will need to consider many variables that are different for each organization.

Modify Your IT Procurement Processes to Gain Greater Control over Value Delivery

In his 1982 book *Out of the Crisis*,¹⁰ W. Edwards Deming proposed 14 principles of management required for American companies to improve the effectiveness of their businesses. Number 4 on the list reads, “End the practice of awarding business on the basis of price tag. Instead, minimize total cost. Move toward a single supplier for any one item, on a long-term relationship of loyalty and trust.”

10 [deming]

More than 30 years later, we see many organizations that have not grasped the true meaning of this principle. We fail to quantify the total cost, and we treat products and services as fungible commodities that can be easily produced by any supplier. While procurement processes used to award contracts may result in long-term relationships, they are seldom built on collaboration and trust.¹¹ The net result is that procurement policies, processes, and practices conspire to prevent us from improving the value we provide through software delivery.

The first mistake we make is thinking that with large amounts of upfront planning, we can manage the risk of getting something that doesn't deliver the expected value. In many large enterprises, the manager engaging any third party for software delivery services must define all expected outputs up front, in the form of a request for proposal (RFP). This is followed by detailed responses from suppliers as to how they would deliver the exact expected outputs and at what cost, often derived from hourly rates and expenses plus a desired margin. Award decisions are then based on the responses received and presentations made by the groups that make it to the short list.

This painful, highly detailed contractual process has several negative side effects:

It is a poor way to manage the risks of product development

It is based on the misconception that we can know exactly what we need up front—in other words, that while building the system we won't make any significant discoveries about what users find valuable nor encounter any significant unexpected complexities.

It favors incumbents

Suppliers who already have a presence in the organization have easier access to its people so can better understand its budgets and targets that need to be matched. As the procurement process for new suppliers is so painful, it's easier to automatically renew established contracts even with mediocre suppliers. The perceived costs and risks associated with finding a new supplier are often thought to be greater than renewing existing contracts.

¹¹ In the NUMMI story in [Chapter 1](#), some of the problems GM faced in adopting the TPS related to the fact that suppliers were not used to the idea of working collaboratively to improve the quality and specification of parts in response to results in the field. This is a direct analog to the problems we face when we outsource development and, at the end of the contract, find that the product delivered is not fit for its purpose.

It favors large service providers

Large companies employ people who specialize in responding to RFPs, which can be filled with minutiae and check boxes but have little or no bearing on outcomes.

It inhibits transparency

Success is usually awarded on cost of delivery, with little regard for the related costs of integration, business process change, or ongoing operational costs. Rarely do we see consideration for how the products provided will affect end-to-end value delivery. Offshore delivery, for example, usually looks inexpensive in terms of unit cost. However, when we take into account increased communication and travel costs, as well as delays in response and rework due to time differences, it can easily take longer than co-located delivery—at similar overall costs.

It is inaccurate

Due to the planning fallacy (see [Chapter 2](#)), both suppliers and managers tend to be overly optimistic in their estimates of the number of people and the amount of work required because they know that contract price has the most weight in the decision. Suppliers know they can make up the money with change requests.

It ignores outcomes

Contract performance is measured on the ability to supply services as contracted, at the rate contracted, for the period of time contracted. There is usually no mention of how well the delivered services work. Regardless of outcomes, the suppliers are usually rewarded with further contracts for supporting, fixing, and improving the service.

TIP

If you are stuck with a long term, project-based contract that specifies delivery of a solution at the end of the term, mitigate your risks by offering to pay the supplier in advance, based on the delivery of incremental working software. This incentivizes them to provide you with working software that you can put into production so you can get feedback from your customers on the value of the solution. In turn, you can adjust the direction of the product's development, based on your test outcomes.

The second mistake in the typical procurement process is that it assumes all services are equal in both the quality of the people working on the delivery and the quality of the software delivered. Through painful experience, many of us know this is not the case. There are many factors that determine how successful the outcome of engaging a supplier will be, the least of which is cost. What is their error occurrence and fix rate? What is the maintenance effort

associated with their solutions? How many lines of code are in the solution (less is better)? How closely can we work with them? Can we trust them? Although we may gain some insights by talking to other customers of the supplier, the acid test is experimentation. We need to test out the relationship and measure the result—which requires modifying the processes we use to engage third-party service providers.

UK Government Changes Its IT Procurement Process to Encourage Innovation

Early in 2014, the UK government announced dramatic changes to the rules applied to awarding and managing contracts for IT services.¹² These changes aimed to encourage competition in the IT service sector and help the government become a more intelligent customer of the service providers. They believe they will achieve better outcomes in their IT services by limiting big IT contracts and by broadening their source of potential suppliers.

To make it easier for small- to medium-size enterprises to bid on government IT contracts, four major changes were announced in the UK government's IT services procurement process:

- No contracts over £100m would be rewarded except under exceptional circumstances.
- Companies with a contract for service provision will not be allowed to provide system integration in the same part of the government.
- New hosting contracts will be for a maximum period of two years.
- No contracts will be automatically renewed.

The UK government hopes to create more efficient and responsive services meeting public demands by getting better access to innovative and cost-effective digital solutions. They are expanding the range of suppliers and creating more opportunities to assess and negotiate contracts that may be based on outdated and expensive technologies or are just not delivering value.

12 <http://bit.ly/1v73rXY>

The agile manifesto says we should prefer customer collaboration over contract negotiation. We need to continuously work with our suppliers to produce high-quality results. The best relationships and results are achieved when we don't throw requirements over the wall and then expect a product or service to magically appear months later. We have to be engaged, manage contracts and relationships, encourage flexibility, and seek opportunities to experiment with different providers so we can assess their competence and capability to deliver value.

Conclusion

Organizations that continue to structure funding decisions around financial cycles will face serious obstacles to improving their innovation capabilities. We need to move beyond the centralized budget paradigm and introduce flow into the processes of financial forecasting, planning, and reporting. This is essential if we want to see more than incremental benefits from applying lean principles.

We should disassociate funding decisions from the annual budget cycle, and stop worrying if it is capital or operational expense. This is how we can make better decisions on what and when should be funded to create the outcomes we want. We still need to manage our costs carefully, but better results can be achieved through shorter cycles of planning, forecasting, and monitoring, along with relating costs back to the business activities that drive them. Restricting resources and time frames for testing ideas helps us to gracefully cut investing in ideas that don't pan out. Resources can then be channeled into exploring new ideas for products and services.

Finally, to support innovation and experimentation, we need to modify our procurement processes and rules. Without long-term trust relationships based on a mutual desire to get better at delivering value, large organizations will forever be hampered by legacy systems and products that are often outdated before they are even deployed.

Questions for readers:

- Can your product teams openly experiment with new ideas and technology without spending large amounts of time on seeking approval and funding? Can you easily obtain funds for new technology-related work at any time of the year, or are you restricted to an annual cycle?
- What are your criteria for a successful investment? Is it enough for projects to come in on time and on budget, or do you attempt to measure customer and organizational outcomes?
- How much time is spent managing your team's budget throughout the year, including reviewing reports and justifying variances?

- How far out and how often are you expected to plan for detailed costs? Is there an easy process for continuous adjustments and reporting within the plan?
- Does the process for allocating capital expenditure versus operational expenditure prevent people from making responsible investment decisions? If so, is there a simple, low-risk way to experiment with a different approach?

Turn IT into a Competitive Advantage

The cost-center pattern fills the vacuum of our inability to define, model, and measure the value most workers create for their organization.

— Ken H. Judy

Enterprise IT departments face powerful and conflicting forces. Their first priority is to keep the existing business-critical systems running, even as they age and grow in complexity. There is also an increasing pressure to up the speed at which new products and features can be delivered. Finally, IT has traditionally been seen as a cost center, so there is constant pressure to increase efficiency (which normally plays out as cost-cutting).

These apparently conflicting goals often lead to a downward spiral. Reducing complexity and replacing legacy systems requires investment. However, investment often comes in the form of large, multiyear projects that often get abandoned or deployed uncompleted due to spiraling costs and/or personnel changes at the executive level. This increasing complexity, along with the need for further efficiency gains, reduces the capacity of IT to manage planned work effectively. The increasing demand for changes, when combined with a brittle IT environment, leads to proliferation of unplanned work that further reduces IT capacity.

In this chapter, we discuss some strategies to increase the responsiveness of IT to changing business needs, improve the stability of IT services, and reduce the complexity of our IT systems and infrastructure. Many of these strategies come from the DevOps movement whose goal is to enable us to work safely at scale in a high-tempo and high-consequence environment.

Rethinking the IT Mindset

IT has historically been seen as a cost center and an internal enabler of the business, not a creator of competitive advantage. For years the orthodoxy has been that, as Nicholas Carr infamously said, “IT doesn’t matter.”¹ Even amongst lean practitioners, IT is sometimes seen as “just a department.” This has created what Marty Cagan, author of *Inspired: How to Create Products Customers Love*,² calls an “IT mindset” in which IT is simply a service provider to “the business” (Figure 14-1).

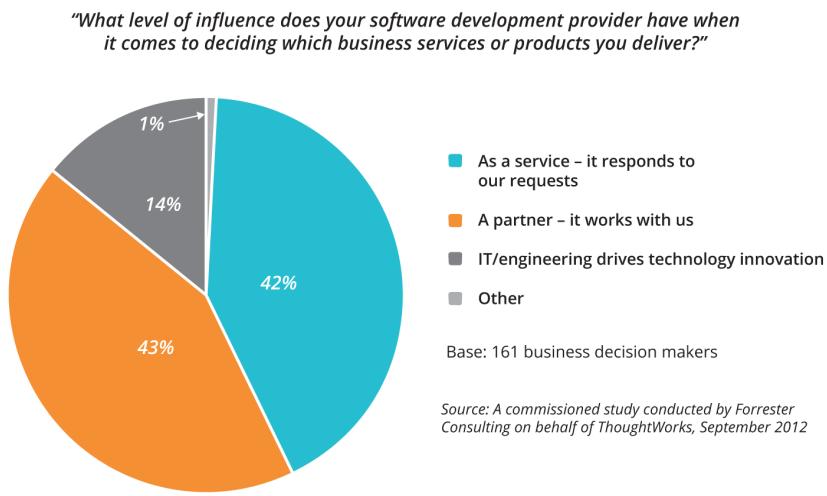


Figure 14-1. What business leaders think about the business-IT relationship

This problem is exacerbated by the typical project model through which IT projects are funded and managed. The work created in IT projects is typically handed over (or thrown over) to IT operations to run, so the people managing the projects have little incentive to think about the long-term consequences of their design decisions—and large incentives to ship as much functionality as possible in what is typically an extremely tight timeframe. This leads to software that is hard to operate, change, deploy, maintain, and monitor, and which adds complexity to operational environments which, in turn, makes further

¹ The original article is at <https://hbr.org/2003/05/it-doesnt-matter> with further commentary and discussion by Nicholas Carr at http://www.nicholascarr.com/?page_id=99.

² [cagan]

projects harder to deliver.³ As Charles Betz, author of *Architecture and Patterns for IT Service Management, Resource Planning, and Governance: Making Shoes for the Cobbler's Children*, says:⁴

Because it is the best-understood area of IT activity, the project phase is often optimized at the expense of the other process areas, and therefore at the expense of the entire value chain. The challenge of IT project management is that broader value-chain objectives are often deemed “not in scope” for a particular project, and projects are not held accountable for their contributions to overall system entropy.

IT operations—a department within the IT department and perhaps the ultimate cost center—experiences the consequences of these decisions on a daily basis. In particular, the integrated systems they must keep running are incredibly complex and crusty, built up over years, and often fragile, so they tend to avoid changing them. Since stability is their first priority, IT operations has developed a reputation as the department that says “no”—an entirely rational response to the problems they face.

IT operations departments have two primary mechanisms they use to stem the tide: the change management process and standardization. The change management process is used to mitigate the risk of changes to production environments and meet regulatory requirements, and it usually requires every change to production to be reviewed by a team (known as the Change Advisory Board in ITIL terminology) before it can be deployed. Standardization is used to manage the heterogeneity of production environments, reduce cost, and prevent security breaches; it also requires that all software used in production (and often in development environments as well) is approved for usage.

The result of these processes is that the rate of change slows down enormously in production environments and teams cannot use the tools they choose. Under certain circumstances, this might be an acceptable trade-off if these limitations could actually improve the stability of production environments. However, the data shows that they do not. In fact, many of the assumptions that underlie IT departments’ operations and their relationships to other parts of the organization are no longer valid.

In the 2014 *State of DevOps Report*, over 9,000 people worldwide were polled about what creates high-performance organizations, whether IT does in fact matter to the business, and what factors impact the performance of IT

³ These problems are described in more detail in Evan Bottcher’s poetically named blog post “Projects Are Evil and Must Be Destroyed,” <http://bit.ly/1v73umC>.

⁴ [betz], p. 300.

departments.⁵ The first major result from the survey was a statistically valid way to measure IT performance. High-performing IT organizations are able to achieve both high *throughput*, measured in terms of change lead time and deployment frequency, and high *stability*, measured as the time to restore service after an outage or an event that caused degraded quality of service. High-performing IT organizations also have 50% lower change fail rates than medium- and low-performing IT organizations.

The data shows that organizations with high-performing IT are able to achieve higher levels of both throughput *and* stability. Furthermore, firms with high-performing IT organizations are also twice as likely to exceed their profitability, market share, and productivity goals as those with low IT performance.

The practices most highly correlated with high IT performance (increasing both throughput and stability) are:

- Keeping systems configuration, application configuration, and application code in version control
- Logging and monitoring systems that produce failure alerts
- Developers breaking up large features into small, incremental changes that are merged into trunk daily (as discussed in Chapter 8)
- Developers and operations regularly achieving win/win outcomes when they interact

There are two other factors that strongly predict high performance in IT. The first is a high-trust organizational culture as described in Chapter 1. The second is a lightweight peer-reviewed change approval process. Many organizations have an independent team to approve changes that go to production. However, the data shows that while such external processes significantly decrease throughput, they have negligible positive impact on stability. Peer-reviewed change approval mechanisms (such as pair programming or code review by other developers) are as effective at creating stable systems as change advisory boards—but have a drastically better throughput.

While this data supports the existing practices of high-performing companies such as Amazon and Google, it directly contradicts the received wisdom that segregation of duties is an effective way to manage risk. However, Westrum's work on safety culture shows that *no* process or control can compensate for an environment in which people do not care about customer and organizational outcomes. Instead of creating controls to compensate for pathological cultures,

⁵ [forsgren]; the report can be downloaded from <http://bit.ly/2014-devops-report>.

the solution is to create a culture in which people take responsibility for the consequences of their actions—in particular, customer outcomes.

There is a simple but far-reaching prescription to enable this behavior:

1. *You build it, you run it.* Teams that build new products and services must take responsibility for the operation and support of those services, at least until they are stable and the operation and support burden becomes predictable. By doing this, we also ensure that it is easy to measure the cost of running the service and the value it delivers.
2. *Turn central IT into a product development organization.* The product development lifecycle and strategies described in this book should be used to deliver internal products and services as well as customer-facing ones.
3. *Invest in reducing the complexity of existing systems.* Use the capacity gained from step 1 to invest in ongoing improvement work with the goal of reducing the cost and risk of making changes to existing services.

Freedom and Responsibility

In order to reduce the burden on IT operations, it's essential that we shift supporting new products, services, and features to the teams that build them. To do this, we need to give them both the autonomy to release and operate new products and features and the responsibility for supporting them.

In Google, teams working on a new product must pass a “production readiness review” before they can send any services live. The product team is then responsible for its service when it initially goes live (similarly to ITIL's concept of early life support). After a few months, when the service has stabilized, the product team can ask operations—called Google's Site Reliability Engineers, or SREs—to take over the day-to-day running of the service, but not before it passes a “handover readiness review” to ensure the system is ready for handover. If the service encounters a serious problem after the handover, responsibility for supporting it is transferred back to the product team until they can pass another handover readiness review.⁶

As discussed in [Chapter 12](#), this model requires that product teams work with other parts of the organization responsible for compliance, information security, and IT operations throughout the development process. In particular, centralized IT departments are responsible for:

⁶ Tom Limoncelli, <https://www.youtube.com/watch?v=iIuTrhdTzK0>. See also [\[limoncelli\]](#).

- Providing clear and up-to-date documentation on which processes and approvals are necessary for new services to go live and on how teams can access them
- Monitoring lead time and other SLAs for these services, such as approving software packages, provisioning infrastructure (such as testing environments), and working to constantly reduce them

For live services under active development, developers share equal responsibility with operations for:⁷

- Responding to outages and being on call
- Designing and evolving monitoring and alerting systems, and the metrics they rely on
- Application configuration
- Architecture design and review

Engineers building new features should be able to push code changes live themselves, following peer review, except in the case of high-risk changes. However, they *must* be available when their changes go live so they can support them. Many new code changes (particularly high-risk ones) should be launched “dark” (as described in [Chapter 8](#)) and either switched off in production or made part of an A/B test.

Some people describe this model as “no-ops,”⁸ since (if successful) we drastically reduce the amount of reactive support work that operations staff must perform. Indeed teams running all their services in the public cloud can take this model to its logical conclusion where product teams have complete control over—and responsibility for—building, deploying, and running services over their entire lifecycle (a model pioneered at scale by Netflix). This has led to a great deal of resistance from operations folks who are concerned about losing their jobs. The “no-ops” label is clearly provocative, and we find it problematic; in the model we describe, demand for operations skills is in fact *increased*, because delivery teams must take responsibility for operating their own services. Many IT staff will move into the teams that build, evolve, operate, and support the organization’s products and services. It is true that traditional operations people will have to go through a period of intense learning and

⁷ Adapted from a post by John Allspaw: <https://gist.github.com/jallspaw/2140086>.

⁸ This term was coined by Forrester’s Mike Gualtieri: <http://bit.ly/1v73wLd>; responses from John Allspaw of Etsy and Adrian Cockcroft of Netflix can be found at <https://gist.github.com/jallspaw/2140086>.

cultural change to succeed in this model—but that is true for all roles within adaptive organizations.

It must be recognized and accepted that this will be scary for many people. Support and training must be provided to help those who wish to make the transition. It must be made clear that the model we describe is not intended to make people redundant—but everyone needs to be willing to learn and change (see [Chapter 11](#)). Generous severance packages should be offered to those who are not interested in learning new skills and taking on new roles within the organization.

Removing the burden of creating and supporting new products and services frees up central IT organizations so they can focus on operating and evolving existing services and building tools and platforms to support product teams.

Creating and Evolving Platforms

The most important role of central IT is supporting the rest of the organization, including management of assets such as computers and software licenses, and the provision of services such as telephony, user management, and infrastructure. This is as true for high-performing organizations as it is for the low performers. The difference lies in how these services are managed and provided.

Traditionally, companies have relied on packages supplied by external vendors (such as Oracle, IBM, and Microsoft) to provide infrastructure components such as databases, storage, and computing power. Nobody could have missed the move to the utility computing paradigm known as “cloud.” However, while few companies can avoid the move, many are failing to execute it correctly.

To succeed, IT organizations must take one of the two paths: either outsource to external suppliers of infrastructure or platform as a service (IaaS or PaaS), or build and evolve their own.

While moving to external cloud suppliers carries different risks compared to managing infrastructure in-house, many of the reasons commonly provided for creating a “private cloud” do not stand up to scrutiny. Leaders should treat objections citing cost and data security with skepticism: is it reasonable to suppose your company’s information security team will do a better job than Amazon, Microsoft, or Google, or that your organization will be able to procure cheaper hardware?

Given that break-ins into corporate networks are now routine (and sometimes state-sponsored), the idea that data is somehow safer behind the corporate firewall is absurd. The only way to effectively secure data is strong encryption combined with rigorous hygiene around key management and access controls.

This can be done as effectively in the cloud as within a corporate network. Many organizations have been outsourcing IT operations for years; even the CIA has outsourced the building and running of some of its data centers to Amazon.⁹ Many countries are now updating their regulations to explicitly allow for data to be stored in infrastructure that is externally managed.

There are two good reasons to be cautious about public clouds. The first risk is vendor lock-in, which can be mitigated through careful architectural choices. The second is the issue of data sovereignty. Any company storing its data in the cloud “is subject both to the laws of the nation hosting the server and to their own local laws regarding how that data should be protected, leading to a potential conflict of laws over data sovereignty. The implications of these overlapping legal obligations depend on the specific laws of the nation and the relationship and agreements between governments.”¹⁰

Nevertheless, there are compelling reasons to move to public cloud vendors, such as lower costs and faster development. In particular, public clouds enable engineering teams to self-service their own infrastructure instantly on demand. This significantly reduces the time and cost of developing new services and evolving existing ones. Meanwhile, many companies that claim to have implemented “private clouds” still require engineers to raise tickets to request test and production environments, and take days or weeks to provision them.

Any cloud implementation project not resulting in engineers being able to self-service environments or deployments instantly on demand using an API must be considered a failure. The only criterion for the success of a private cloud implementation should be a substantial increase in overall IT performance using the throughput and stability metrics presented above: change lead time, deployment frequency, time to restore service, and change fail rate. This, in turn, results in higher quality and lower costs, as well as freeing up capital to invest in new product development and improving of the existing services and infrastructure.

The alternative to using an external vendor is developing your own service delivery platform in-house. A service delivery platform (SDP) lets you automate all routine activity associated with building, testing, and deploying services, including the provisioning and ongoing management of infrastructure services. It is also the foundation on which deployment pipelines for building, testing, and deploying individual services run. *The Practice of Cloud System*

⁹ <http://theatlantic.com/technology/2013/07/the-cloud-is-not-a-new-world/217342>

¹⁰ <http://bit.ly/1v73C5K>

Administration: Designing and Operating Large Distributed Systems is an excellent guide to designing and running a service delivery platform.¹¹

However, companies who have succeeded at creating their own SDP (per the criteria above) have not typically done so through the traditional IT route of buying, integrating, and operating commercial packages.¹² Instead, they have used the product development paradigm described in this book to create and evolve an SDP, preferring to use open source components as a foundation. This approach requires a substantial retooling and realignment of IT to focus on *exploring* new platforms by testing them with a subset of internal customers (as we discuss in [Part II](#)) with the goal of delivering early value and providing performance superior to that of the external vendors. Validated products should be evolved using the principles described in [Part III](#), using cross-functional product teams measuring their success by the IT performance metrics above.

Preparing for Disasters

Organizations that do choose to manage their own SDP must take business continuity extremely seriously. Amazon, Google, and Facebook inject faults into their production systems on a regular basis to test their disaster recovery processes. In these exercises, called Game Days at Amazon and Disaster Recovery Testing (DiRT) at Google, a dedicated team is put together to plan and execute a disaster scenario.

Typically, this includes physically powering down data centers and disconnecting the fiber connections to offices or data centers. This has real consequences but is reversible in the event of an uncontrollable failure. People running affected services are expected to meet their service-level agreements (SLAs), and the disruptions are carefully planned to not exceed the limits of what is necessary to run the service. Crucially, a blameless postmortem is held after every exercise (see [Chapter 11](#)), and the proposed improvements are tested some time later.

Kripa Krishnan, Google's program manager for DiRT exercises, comments that "for DiRT-style events to be successful, an organization first needs to accept system and process failures as a means of learning. Things will go wrong. When they do, the focus needs to be on fixing the error instead of reprimanding an individual or team for a failure of complex systems...we design tests

11 [\[limoncelli\]](#)

12 This reflects the way Toyota approaches buying machinery. Norman Bodek reports that "Toyota and the major suppliers, instead of buying machines, which would do 'everything possible needed in the future,' would build over 90% of their own machines themselves to do the specific job needed at the time" [\[bodek\]](#), p. 37.

that require engineers from several groups who might not normally work together to interact with each other. That way, should a real large-scale disaster ever strike, these people will already have strong working relationships established.”¹³

Netflix takes this idea to its logical extreme by running a set of services known as the Simian Army, led by Chaos Monkey, a service that shuts down production servers at regular intervals to test the resilience of the production environment. Like many Netflix systems, the software behind the Simian Army is open source and available on Github. Organizations that do not have the intestinal fortitude to perform real failure injection exercises on at least an annual basis should not be in the business of developing their own infrastructure services—at least, not for mission-critical systems.

Finally, organizations that develop their own infrastructure services must give their internal customers the choice of whether or not to use them. Enterprises rely on standardization of the services and assets provided by IT operations to manage support costs, for example by maintaining a list of approved tools and infrastructure components from which teams may choose. However, trends such as employees bringing their own devices to work (BYOD) and product development teams using nonstandard open source components such as NoSQL databases, present a challenge to this model. We have seen cases in which open source components were *necessary* to achieve the levels of performance, maintainability, and security required by their customers, but were resisted by IT operations departments—resulting in a great deal of wasted time and money trying to force the products to run on existing packages.

The correct way to address this problem is to allow product teams to use the tools and components they want, but to require them to take on the risks and costs of managing and operating the products and services they build—to repeat Amazon CTO Werner Vogels’ dictum, “You build it, you run it.” Recall the Lean definition of optimal performance from Chapter 7: “Delivering customer value in a way in which the organization incurs no unnecessary expense; the work flows without delays; the organization is 100 percent compliant with all local, state and federal laws; the organization meets all customer-defined requirements; and employees are safe and treated with respect. In other words, the work should be designed to eliminate delays, improve quality, and reduce unnecessary cost, effort, and frustration.”¹⁴ Processes inhibiting optimal performance should be a target for improvement.

13 <http://queue.acm.org/detail.cfm?id=2371297>

14 [martin], p. 101.

Managing Existing Systems

A service delivery platform, whether created in-house or provided by a vendor, must ensure standardization and reduced cost to run *new* systems. However, it will not help reduce the complexity of existing ones. The large number of existing systems is one of the biggest factors limiting the ability of enterprise IT departments to move fast.

In operations departments that must maintain hundreds or thousands of existing services, delivering even an apparently simple new feature can involve touching multiple systems, and any kind of change to production is fraught with risk. Obtaining integrated test environments for such changes is expensive—even a part of the production environment cannot be reproduced without a lot of work (and it's usually hard to tell how much we need to reproduce, and at what level of detail, for testing purposes). Combine this with functional silos, outsourcing, and distributed teams juggling multiple priorities, and we swiftly find that our feet are encased in concrete.

In this section we present three strategies for mitigating this problem. The short-term strategy is to create transparency of priorities and improve communication between the teams working on these systems. The medium-term solution is to build abstraction layers over systems that are hard to change, and create test doubles for systems that have to integrate with them. The long-term solution is to incrementally rearchitect systems with the ability to move fast at scale as an architectural goal.

The short-term solution—creating transparency of priorities and improving communication—is important and can be extremely effective. IT has to serve multiple stakeholders with often conflicting priorities. Who wins often depends on who is shouting loudest or has the best political connections, not on an economic model such as Cost of Delay (discussed in [Chapter 7](#)). It's important to have a shared understanding at all levels of the organization on what the current priorities are. This can be as simple as a weekly or monthly meeting of the key stakeholders, including all customers of IT, to issue a one-page prioritized list. Regular communication between those responsible for systems that are coupled is also essential.

Coupling Requires Frequent Communication

A major travel company wanted to continuously deliver new features to their website. However, the website needed to talk to a legacy booking system. Often new features were delayed due to dependencies on changes to the booking system, which was updated every six months. This was costing the company large amounts of money in lost opportunity costs.

One simple way they eased the problem was by improving communication between the teams. The product manager for the website would regularly meet up with the program manager for the booking system, and they would compare notes on upcoming releases, noting dependencies. They'd find ways to shift their schedules around to help each other deliver features on time, or to push back features that couldn't be delivered.

The medium-term solution is to find ways to simulate the infrequently changing systems we must integrate with. One technique is to use virtualized versions of these systems. Another is to create a test double that simulates the remote system for testing purposes (Figure 14-2).¹⁵ The important thing to bear in mind is that we're not aiming to faithfully reproduce the real production environment. We're attempting to discover and fix most of the big integration problems early on, before we go to a full staging environment.

By faking out remote systems or running them in a virtual environment, we can integrate and run system-level tests to validate our changes on a regular basis (say, once per day). This reduces the amount of work we have to do in a properly integrated environment.

The long-term solution is to architect our systems in such a way that we can move fast. In particular, this means being able to independently deploy parts of our system at will, without having to go through complex orchestrated deployments. However, this requires careful rearrangement using the strangler application pattern described in Chapter 10.

¹⁵ For more on this, see <http://martinfowler.com/bliki/SelfInitializingFake.html>.

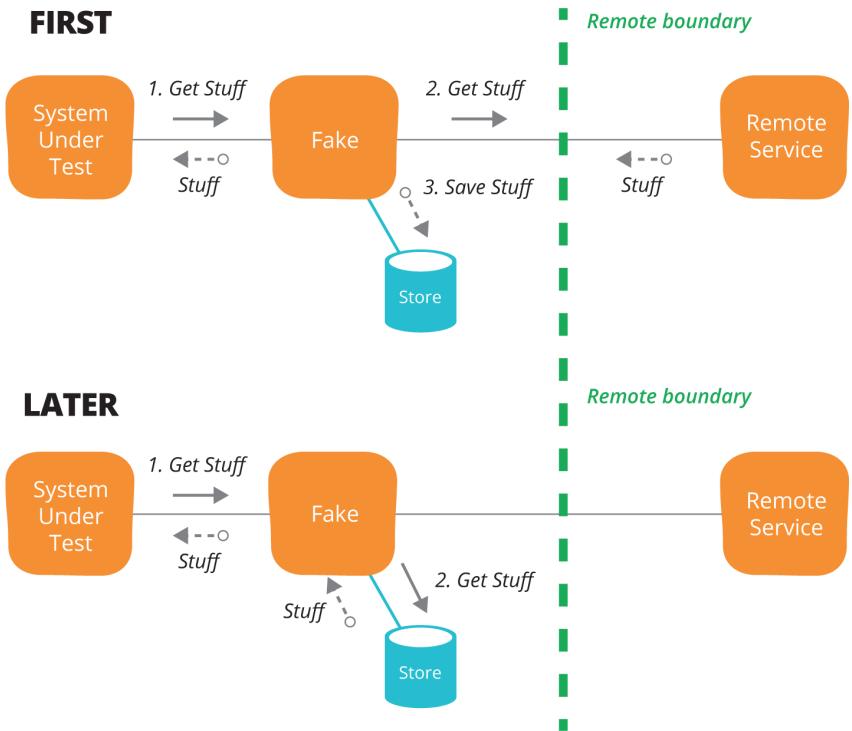


Figure 14-2. Simulating remote systems for test purposes

When starting on this process, an important first step is to map your services and the connections between them. Based on the lifecycle of innovations (see [Chapter 2](#)) and the *value* each service provides to our organization, we can draw value and lifecycle on two axes, and then create a *value chain map* to visualize each product and its dependencies ([Figure 14-3](#)). To create a value chain map, take a product and put it in the appropriate position at the top of a new diagram. Then map the services it depends on and the connections between them. This exercise can be performed quickly and cheaply on a whiteboard using sticky notes.¹⁶

¹⁶ <http://blog.gardeviance.org/2013/09/why-map.html>

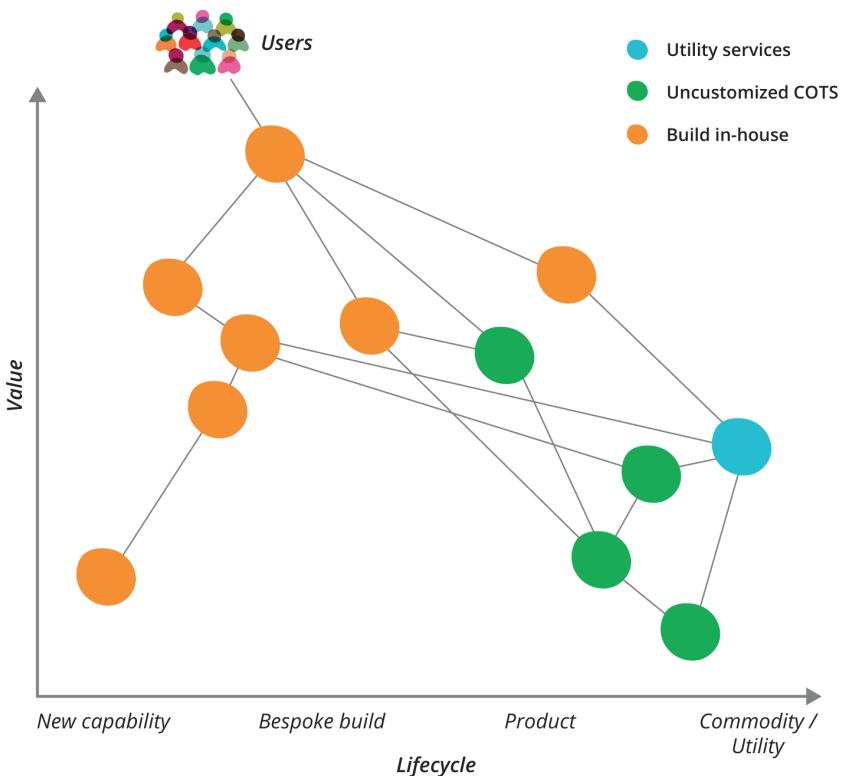


Figure 14-3. Value chain map, courtesy of Simon Wardley

The next step is to create a “to-be” version of this diagram, following these principles:

- Use software-as-a-service providers for all “utility” services, such as payroll, vendor management, email, version control, and so forth. If there are systems we can’t move to the cloud, we should use commercial off-the-shelf software packages (COTS).
- For strategic services and applications that provide a competitive advantage, we should be doing custom software development as described in the rest of the book. Avoid at all costs the temptation to use packages for these capabilities.
- Systems of record will typically be the hardest to change, and will be a combination of COTS and older systems including mainframes. These often require consolidation and some amount of abstraction so as to

reduce the cost of maintaining and integrating with them. Over time, they can be strangled if necessary.

When using COTS, it is crucial not to customize the packages. We can't emphasize strongly enough the problems and risks associated with customizing COTS. When organizations begin customizing, it's hard to stop—but customizations of COTS packages are extremely expensive to build and maintain over time. Once you get beyond a certain amount of customization, the original vendor will often no longer support the package. Upgrading customized packages is incredibly painful, and it's hard to make changes quickly and safely to a customized COTS system.

Instead, make changes to your business processes to match what the COTS packages can do out of the box. Any time you choose a solution based on COTS, you'll have a list of features to implement or bugs to fix. These should always be treated as the input to a business process change management activity. Business process changes are much cheaper and more effective than changing COTS packages to match an existing process. If you have gone down the road of customization, take the next major release of your COTS package as an opportunity to migrate to a new, uncustomized version of the package, as Telstra (Australia's biggest telco) did when it moved from a heavily customized install of Remedy to a completely vanilla one.¹⁷

Moving from your current state to your to-be state will likely take years. As with all large-scale changes, the correct way to proceed is incrementally, breaking down large programs of work into small steps that provide the biggest bang for the buck in terms of improving customer and business outcomes.

17 <http://bit.ly/1v73C5K>

The Suncorp Simplification Program

by Scott Buckley and John Kordyback

Australia's Suncorp Group has ambitious plans to decommission their legacy general insurance policy systems, improve their core banking platform, and start an operational excellence program. "By decommissioning duplicate or dated systems, Suncorp aims to reduce operating costs and reinvest those savings in new digital channels," says Matt Pancino, now CEO of Suncorp Business Systems.

Lean practices and continuous improvement are necessary strategies to deliver the simplification program. Suncorp is investing successfully in automated testing frameworks to support developing, configuring, maintaining, and upgrading systems quickly. These techniques are familiar to people using new technology platforms, especially in the digital space, but Suncorp is successfully applying agile and lean approaches to the "big iron" world of mainframe systems.

Delivery Practices

In their insurance business, Suncorp is combining large and complex insurance policy mainframe systems into a system to support common business processes across the organization and drive more insurance sales through direct channels. Some of the key pieces were in place from the "building blocks" program which provided a functional testing framework for the core mainframe policy system, agile delivery practices, and a common approach to system integration based on web services.

During the first year of the simplification program, testing was extended to support integration of the mainframe policy system with the new digital channels and the pricing systems. Automated acceptance criteria were developed while different systems were in development. This greatly reduced the testing time for integrating the newer pricing and risk assessment system with multiple policy types. Automated testing also supported management and verification of customer policies through different channels, such as online or call center.

Nightly regression testing of core functionality kept pace with development and supported both functional testing and system-to-system integration. As defects were found in end-to-end business scenarios, responsive resolutions were managed in hours or days, not weeks typical for larger enterprise systems.

Outcomes

In the process, Suncorp has reduced 15 complex personal and life insurance systems to 2 and decommissioned 12 legacy systems. Technical upgrades are done once and rolled out across all brands. They have a single code base for customer-facing websites for all their different brands and products. This enables faster response to customer needs and makes redundant separate teams each responsible for one website.

From a business point of view, the simpler system has allowed 580 business processes to be redesigned and streamlined. Teams can now provide new or improved services according to demand, instead of improving each brand in isolation. It has reduced the

time to roll out new products and services, such as health cover for APIA customers or roadside assistance for AAMI customers.

The investment in simplification and management of Suncorp's core systems means they can increase their investment in all their touch points with customers. In both technology and business practices, Suncorp increased their pace of simplification, with most brands now using common infrastructure, services, and processes.

Suncorp's 2014 annual report notes that "simplification has enabled the Group to operate a more variable cost base, with the ability to scale resources and services according to market and business demand. Simplification activity is anticipated to achieve savings of \$225 million in 2015 and \$265 million in 2016."¹⁸

Conclusion

If we want to compete in a world of ever shorter product cycles, central IT needs to be business units' trusted partner, not an order-taking cost center. In turn, IT needs to achieve higher levels of throughput while improving stability and quality and reducing costs. The complexity of existing enterprise IT environments, combined with the amount of planned and unplanned work that must be done to keep them running, are the chief barriers to achieving these outcomes.

We can only begin to address these problems when we consider the effects of new work on IT operations and treat it as an integral part of the product development lifecycle. To manage the additional complexity introduced by new products, services, and features, we must start by moving from a project-based model to a product-centric model, as described in [Chapter 10](#). Product teams must own the costs and service-level agreements of the systems they build; in return for this responsibility, they have the freedom to choose the technologies to use, and to manage their own changes. In this way, we free up people and resources within central IT to focus on reducing the complexity of their systems and infrastructure and building a toolchain and platform that enables the product development lifecycle we describe in this book.

We often treat throughput and stability as opposing forces—increase throughput and you will reduce quality and stability. However, these goals can be complementary if the correct strategy is in place. As with any improvement effort, we must start by clearly articulating our goal and identifying the key performance indicators we care about. Then, we use the Improvement Kata to work towards our goal.

18 <http://bit.ly/1v73OC3>

Questions for readers:

- Does IT consider itself to be a service provider, a partner to business units, or a driver of innovation? What do other leaders in the organization think?
- Are you measuring change lead time, release frequency, time to restore service, and change fail rate across all your products and services? Are you making them visible to all teams?
- How many services did you retire in the last year? And how many did you add? How long would it take you to find out how many products and services you are managing? How certain would you be of the answer? How many of them are running on systems that are no longer officially supported by the vendor?
- How long does it take to approve a change request? How long does it take to get a new open source component approved for use in a production environment?
- How often do you perform a realistic disaster recovery exercise on your production systems? What is your process for following up on recommendations for improvement that come out of these exercises?
- Do all developers, development leads, and architects rotate through pager duty and support on a regular basis for the systems they build?

Start Where You Are

If you do something and it turns out pretty good, then you should go do something else wonderful, not dwell on it for too long. Just figure out what's next.

— Steve Jobs

A year from now you will wish you had started today.

— Karen Lamb

Our goal with this book is to inspire you to envision an alternative future for large organizations. A future that puts employees, customers, and products at the heart of its strategy. A future where a renewed culture and environment enable the organization to adapt rapidly to changing market demands.

We have shared stories and lessons learned from a diverse set of organizations with varied backgrounds and circumstances to highlight that even in complex environments, you can thrive and address the most challenging problems. However, the path to success is not likely to be linear, with defined instructions, milestones, and KPIs. Organizations needed to get comfortable moving forward with uncertainty and imperfect information, while learning, adjusting, and developing their people along the way.

The biggest barrier to success in changing the way you work is a conviction that your organization is too big or bureaucratic to change, or that your special context prevents adopting the particular practices we discuss. Always remember that each person, team, and business that started this journey was unsure of what paths to take and how it would end. The only accepted truth was that if they failed to take action, a more certain, negative ending lay ahead.

Principles of Organizational Change

All change is risky, particularly organizational change which inherently involves cultural change—the hardest change of all, since you are playing with the forces that give the organization its identity. We are still amazed when leaders plan “organizational change” programs that they expect to complete in months. Such programs fail to recognize that turning innovation or change into an *event* rather than part of our daily work can never produce significant or lasting results. Periodically funding a new change program in response to current issues, leadership changes, or market trends without instilling a culture of experimentation will only achieve short-term incremental change, if any at all (Figure 15-1). Organizations will quickly slip back to their previous state. Instead, we must create a culture of continuous improvement through the deliberate, ongoing practice of everyone in the organization.

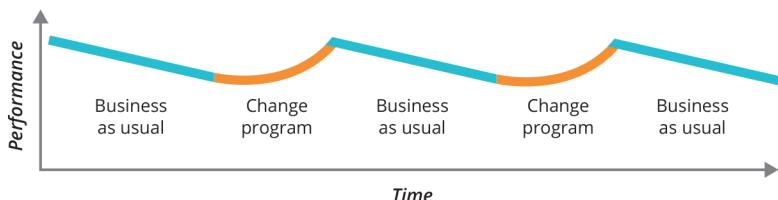


Figure 15-1. The reality of “event-based” change programs

If your organization is waiting for an event to stimulate change, you’re already in trouble. In the current environment and competitive economy, a sense of urgency should be a permanent state. Survival anxiety always exists in leading organizations, as we describe in Chapter 11. However, as Schein noted, using it as a motivator for ongoing change is ineffective. The only path to a culture of continuous improvement is to create an environment where learning new skills and getting better at what we do is considered valuable in its own right and is supported by management and leadership, thus reducing learning anxiety. We can use the Improvement Kata presented in Chapter 6 to create this culture and drive continuous improvement (Figure 15-2).

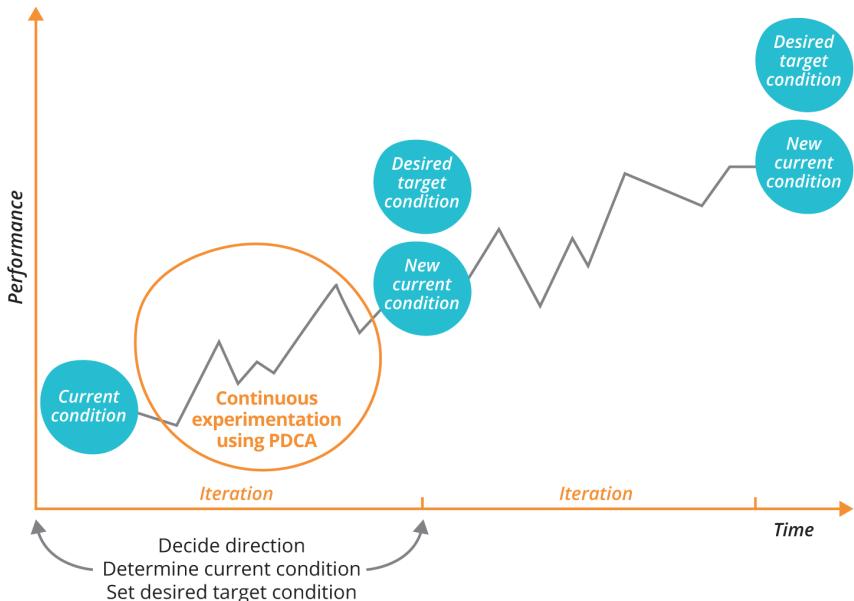


Figure 15-2. Continuous evolution and adaption to change

In order to propagate the Improvement Kata through organizations, managers must learn and deploy a complementary practice known as the Coaching Kata¹. To start the journey, an advance team including an executive sponsor—ideally, the CEO—should pilot the Coaching Kata and the Improvement Kata. As this team will guide wider adoption within the organization, it is imperative that they understand how it works.

Watch out for the following obstacles:

- Adopting the Improvement Kata requires substantial changes in behavior at all levels of the organization. The Coaching Kata is used to teach people the Improvement Kata, but the problem of how to deploy the Coaching Kata within an organization remains significant.
- Running experiments is hard and requires great discipline. Coming up with good experiments requires ingenuity and thought. By nature, people tend to jump straight to solutions instead of first agreeing on measurable target objectives (outcomes) and then working in rapid cycles—and by rapid, we mean hours or days—to create hypotheses, test, and learn from

¹ For free materials on the Improvement Kata and Coaching Kata, see <http://bit.ly/1v73SSg>.

the results. The body of knowledge on how to design and run experiments in the context of product development is still in its early stages, and the necessary skills and techniques are not widely known or understood.

- Ensure there is capacity to run the Improvement Kata. One of the biggest obstacles teams face when trying to schedule improvement work is that it is often seen as a distraction from delivery work. This is a fallacy, and the point must be made early and forcefully. In the HP FutureSmart case, the reason delivery work was progressing so slowly was that no-value-add work was driving 95% of their costs. It is vital for executives at the director or VP level to ensure that teams limit their work in process as described in [Chapter 7](#) to create time for improvement work.
- As with all methods, progress is likely to be bumpy at the beginning as people learn how to work in new ways. Things will get worse before they get better. Resistance is likely as people learn the new skills, and some will become frustrated when it conflicts with their existing habits and behaviors.

Aim Towards Strategy Deployment

Although we discussed the Improvement Kata as a way to drive continuous improvement at the program level, it can be used at every level from individual teams up to strategic planning. To apply the Improvement Kata at the strategic planning level, start by agreeing on the *purpose* of the organization. What is it that we aim to do for our customers? Then, those participating in the strategic planning exercise must define and agree upon the overall direction of the company—identify our “true north.”

The next step is to understand and clarify our organization’s current situation. Participants in the strategic planning exercise should identify which problems need to be addressed and gather data to better understand each problem. Typically, even large organizations have limited capacity and can manage only a handful of initiatives at any one time; choosing what *not* to focus on and making sure the team sticks to its decision is critical. An economic framework such as Cost of Delay (see [Chapter 7](#)) is useful to stimulate discussion about prioritizing work.

Once we have decided what problems to focus on, we need to define our target conditions. These target conditions should clearly communicate what success looks like; they must also include KPIs so we can measure our progress towards the goal. The traditional balanced scorecard approach to KPIs has four standard perspectives: finance, market, operations, and people and organization. Statoil, borrowing from the balanced scorecard approach in their Ambition to Action framework ([Chapter 13](#)), added HSE (health, safety, and

environment). The lean movement teaches us to focus on reducing cost and improving quality, delivery, morale, and safety (these five “lean metrics” are sometimes abbreviated as QCDMS). Bjarte Bogsnes, vice president of Performance Management Development at Statoil, recommends choosing 10–15 KPIs and preferring *relative* targets that connect input with outcomes (for example, unit cost rather than absolute cost) and are based on comparison with a baseline (for example, “10% higher return on capital investment than our leading competitor”).²

The target objectives at the strategic level form the *direction* for the next organizational level, which then goes through its own Improvement Kata process. The target objectives at this level then form the direction for the next organizational level down, as shown in Figure 15-3. This process, allowing us to set targets and manage resources and performance by creating alignment between levels in the organization, is called *strategy deployment* (otherwise known as *Hoshin* or *Hoshin Kanri*; Ambition to Action is a variation of strategy deployment).³

The process of creating alignment and consensus between levels is critical. In strategy deployment, this process is described as *catchball*, a word chosen to evoke a collaborative exercise. The target conditions from one level should not be transcribed directly into the direction for teams working at the level below; catchball is more about *translation* of strategy, with “each layer interpreting and translating what objectives from the level above mean for it.”⁴ We should expect that feedback from teams will cause the higher-level plan to be updated. Don’t subvert Hoshin by using it to simply cascade targets down through the organization: the key to Hoshin is that it is a mechanism for creating alignment based on collaboration and feedback loops at multiple levels.

The time horizons for each level should be clearly defined, and regular review meetings scheduled, with target objectives updated based on the progress of the next-level teams. To be truly effective, this conversation must also be cross-functional, promoting cooperation along value streams, within and between business units. It’s not easy, as it requires honest listening to the ideas and concerns of the people responsible for results—and responding by adjusting the plans based on feedback.⁵

2 [bogsnes], pp. 125–126.

3 For a detailed description of Ambition to Action, see [bogsnes], pp. 114–169.

4 [bogsnes], p. 124.

5 Find out more about strategy deployment in Chapter 3 of Karen Martin’s *The Outstanding Organization* [martin-12], and read a case study at <http://www.lean.org/Search/Documents/54.pdf>.

HOSHIN PLANNING PROCESS USING PDCA

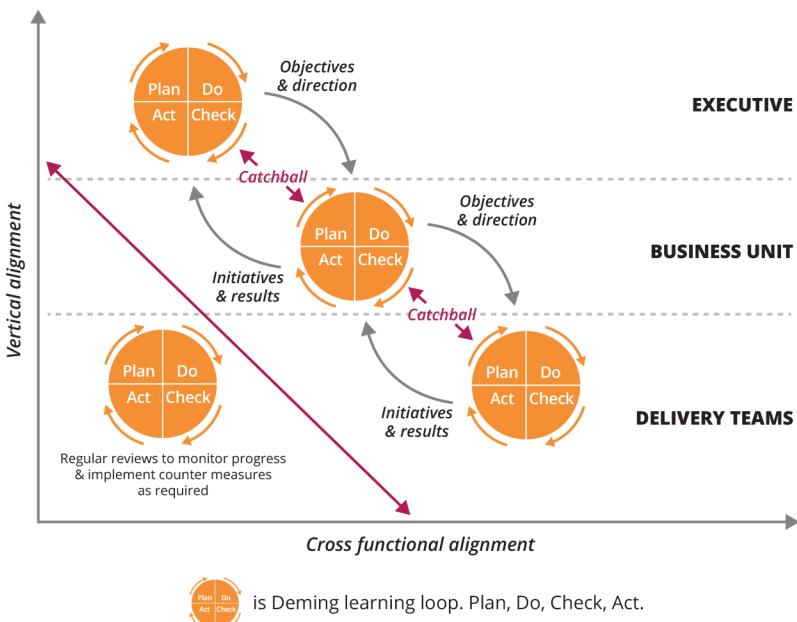


Figure 15-3. Using catchball to drive strategic alignment of objectives and initiatives

A top-level strategy planning exercise can have a horizon of six months to a few years, depending on what is appropriate to your business. Review meetings should be held at least monthly where the team, along with the leaders of all teams that report to them, gather to monitor progress and update target conditions in response to what they discover. Teams at lower levels will typically work to a shorter horizon, with more frequent review meetings.

Strategy deployment is an advanced tool that depends on the aligned culture and behaviors, as we describe in [Chapter 11](#). The main goal is to create consensus and alignment and enable autonomy across the organization, following the Mission Command paradigm presented in [Chapter 1](#). Let's look at how the UK government applied a version of strategy deployment to transform its use of digital platforms to provide services to citizens, starting small and growing iteratively and incrementally.

The UK Government Digital Service

The UK government, like many others, has recognized the potential of the Internet “both to communicate and interact better with citizens and to deliver significant efficiency savings.”⁶ However, government projects involving software development have a checkered past. The UK government had several large IT projects go enormously over budget while failing to deliver the expected benefits, culminating in the “National Programme for IT” debacle. The world’s biggest civil information technology program, supposed to deliver a completely new IT infrastructure for the British National Health Service and a computerized patient record system, was projected to cost £2.3bn at its inception in 2002. Its delivery was outsourced to multiple private sector providers including Accenture, Computer Sciences Corporation, Fujitsu, and British Telecom. Despite the cancellation of the programme in 2011, it is expected to end up costing over £10bn.

The government procurement process for large IT projects involved writing a complete specification for the product, creating several business cases at increasing levels of detail, and then putting the contract out for bidding—a process that required one to two years before work could even start on the product. “By which time,” comments Francis Maude, Minister for the UK’s Cabinet Office, “it will almost certainly be out of date. You’re locked into a supplier, it’s really expensive to make changes.”⁷

As a result of the outsourcing of IT projects, every government department had their own independently designed and operated web presence, with dissimilar user experiences that reflected each department’s internal organization. It was complicated and extremely painful for citizens to use, so they preferred to use more expensive channels of service such as walk-in, mail, and phone services.

In 2010, Martha Lane Fox, co-founder of UK startup *lastminute.com*, was commissioned to advise the UK government on its strategy for online delivery of public services. Her report recommended creating a central team of civil servants responsible for designing and delivering the government’s online presence, implementing an open data policy whereby all government data was made available through public APIs, and appointing a CEO “with absolute authority over the user experience across all government online services (websites and APIs) and the power to direct all government online spending.”⁸ Thus the UK’s Government Digital Service (GDS) was born. Martha Lane Fox

⁶ [lane-fox]

⁷ <http://bit.ly/1F7yvbs>

⁸ [lane-fox]

described her goals for the GDS as follows: “For me, the acid test...is whether it can *empower, and make life simpler for, citizens* and at the same time allow government to *turn other things off*. A focus on vastly increasing the range, usage, and quality of online transactions will deliver the greatest impact: less hassle for citizens & businesses, and greater efficiency.”

Government Digital Service Case Study, by Gareth Rushgrove

GOV.UK is the new single domain for all central government services in the UK. It was launched in October 2012 and replaced two of the largest existing government websites on day one, going on to replace all central government department sites over the next few months. By 2014 we will have closed thousands of websites and built a single service that is simpler, clearer, and faster, covering everything from information about benefits you may be eligible for to how to apply for a passport.

One aspect of GOV.UK that sets it apart from a typical government project is that it was developed nearly completely in-house, by civil servants working for the newly formed Government Digital Service (GDS), part of the UK Cabinet Office. It was also built iteratively, cheaply, and using agile methods and technologies more commonly associated with startups than large organizations. Here is a description of how that was done.

Alpha and Beta

By late 2013 the team running GOV.UK had over 100 people—but it didn’t start that way. In fact, the first version wasn’t even called GOV.UK. An Alpha version was built by 14 people working from a small back room in a large government building. Its aim wasn’t to be a finished product but to provide a snapshot of what a single government website could be, and how it could be built quickly and cheaply. In total, the Alpha took 12 weeks and cost £261,000.

The feedback from users of the Alpha led directly to work on a Beta, which scaled up the Alpha proposition and involved more people from across government. The first release of the Beta was six months after the Alpha project shipped, but this included time to build up the team. The first public Beta release was a real government website, but at the time it lacked all the content and features needed to replace the existing main government sites. Eight months of constant iterations later, with the team up to 140 people and with new content and features added daily, traffic was redirected from two of the largest government websites to the new GOV.UK.

All this work paid off. During the financial year 2012–2013, GDS saved £42 million by replacing the Directgov and BusinessLink websites with GOV.UK. In 2013–2014, it is estimated GDS will save £50 million by closing more websites and bringing them onto the single domain.

Multidisciplinary Teams

The Government Digital Service is made up of specialists in software development, product management, design, user research, web operations, content design, and more, as well as specialists in government policy and other domain-specific areas. From this group of specialists, teams were formed to build and run GOV.UK. Those teams did not have a narrow focus, however; most of them were multidisciplinary, made up of people with the right mix of skills for the tasks at hand.

As an example, the team that worked on the initial stages of the Beta of GOV.UK consisted of seven developers, two designers, a product owner, two delivery managers, and five content designers. Even within these disciplines, a wide range of skills existed. The developers had skills ranging from frontend engineering to systems administration.

By employing multidisciplinary teams, the end-to-end responsibility for entire products or individual tasks could be pushed down to the team, removing the need for large-scale command and control. Such small self-contained teams had few dependencies on other teams so could move much more quickly.

This multidisciplinary model also helped to minimize problems typical in large organizations with siloed organizational structures. For instance, the Government Digital Service has grown over time, adding experts in government information assurance, procurement, and IT governance to avoid bottlenecks and improve the prioritization of resources.

Continuous Delivery

An important aspect of the success of GOV.UK has been constant improvement based on user feedback, testing, and web analytics data. The GOV.UK team releases new software on average about six times a day—with all kinds of improvements, from small bug fixes to completely new features, to the site and supporting platforms.

After the launch of the Beta of GOV.UK, one of the product managers, with bad memories of releasing software at other organizations, asked whether the software deployment mechanism was really going to work. The answer was “yes”: at that point, GDS had done more than 1,000 deployments, so there was a high level of confidence. Practiced automation makes perfect.

This rate of releases is not typical for large organizations where existing processes sometimes appear designed to resist all change. The development teams working on GOV.UK worked extensively on automation, and they had in-depth conversations with people concerned about such rapid change. The key term when discussing this approach was *risk*—specifically, how regular releases can manage and minimize the risks of change.

Most people are bad at undertaking repetitive tasks, but computers are perfect for automating these tasks away. Deployment of software, especially if you are going to do it regularly, is a great candidate for automation. With the development and operation of GOV.UK, this was taken even further: provisioning of virtual machines, network configuration, firewall rules, and the infrastructure configuration were all automated. By describing large parts of the entire system in code, developers used tools like version control and unit testing to build trust in their changes, and focused on a smaller set of

well-practiced processes rather than a separate process (and requisite specialist skills) for each type of change.

Other techniques helped too. A relentless focus on users and a culture of trust from the very top of the organization have put GDS in a position to take much of what it learned building and running GOV.UK and use that to transform the rest of the UK government.

The GDS approach has been adopted by all arms of the government, with transformative results for citizens. To take just one example, the UK Ministry of Justice Digital Team recently worked with the National Offender Management Service and HM Prison Service to change the way people book prison visits. Previously, visitors had to request paper forms to be mailed out and then got on the phone to book a visit. Requests were often rejected because the date was unavailable, forcing people to start over. Now, prison visits can be booked online in 5 minutes, selecting from up to three dates.⁹

Not everybody is thrilled with the idea of governments growing their own IT capabilities. Tim Gregory, the UK president of CGI, the biggest contractor for the US HealthCare.gov website that received a contract valued at \$292 million through 2013 before being replaced by Accenture in January 2014,¹⁰ argues that the GDS approach will make it unprofitable for large outsourcing vendors to bid for government projects. GDS Executive Director Mike Bracken describes Gregory's view as "beyond parody."¹¹

There are several observations to be made from the GDS case study.

First, starting small with a cross-functional team and gradually growing the capability of the product, while delivering value iteratively and incrementally, is an extremely effective way to mitigate the risks of replacing high-visibility systems, while simultaneously growing a high-performance culture. It provides a faster return on investment, substantial cost savings, and happier employees and users. This is possible even in a complex, highly regulated environment such as the government.

Second, instead of trying to replace existing systems and processes in a "big bang," the GDS replaced them incrementally, choosing to start where they could most quickly deliver value. They took the "strangler application" pattern presented in [Chapter 10](#) and used it to effect both architectural *and* organizational change.

⁹ <http://bit.ly/1v73X8w>

10 Reuters: "As Obamacare tech woes mounted, contractor payments soared," <http://reut.rs/1v741oj>.

11 <http://bit.ly/1v742ZT>

Third, the GDS pursued principle-based governance. The leadership team at GDS does not tell every person what to do but provides a set of guiding principles for people to make decisions aligned to the objectives of the organization. The GDS governance principles state:¹²

1. Don't slow down delivery.
2. Decide, when needed, at the right level.
3. Do it with the right people.
4. Go see for yourself.
5. Only do it if it adds value.
6. Trust and verify.

People are trusted to make the best decisions in their context, but are accountable for those decisions—in terms of both the achieved outcomes and knowing when it is appropriate to involve others.

Finally, the GDS shows that extraordinary levels of compensation and using a private sector model are not decisive for creating an innovation culture. GDS is staffed by civil servants, not Silicon Valley entrepreneurs with stock options.¹³ An innovation culture is created by harnessing people's need for mastery, autonomy, and purpose—and making sure people are deeply committed to the organization's purpose and the users they serve.

Begin Your Journey

Use the following principles for getting started:¹⁴

Ensure you have a clearly defined direction

The direction should succinctly express the business or describe organizational outcomes you wish to achieve in measurable terms, even if they look like an unachievable ideal. Most importantly, it should inspire everyone in the organization. Think of HP FutureSmart's goal of 10x productivity improvement.

¹² GDS Governance principles, <http://bit.ly/1v747fT>.

¹³ In fact, the relatively low probability of a startup "exiting" successfully means that, for purely financial reasons, you'd be crazy to prefer a job at a startup over a solid position at (say) Google, as shown on slides 6–15 at <http://slidesha.re/1v6ZQZZ>.

¹⁴ These principles are partly inspired by John Kotter's eight-step process described in [kotter]: establish a sense of urgency, create the guiding coalition, develop a vision and strategy, communicate the change vision, empower broad-based action, generate short-term wins, consolidate gains and produce more change, anchor new approaches in the culture.

Define and limit your initial scope

Don't try to change the whole organization. Choose a small part of the organization—people who share your vision and have the capability to pursue it. As with the GDS, start with a single, cross-functional slice, perhaps a single product or service. Make sure you have support at all levels from executives down and from shop floor up. Create target objectives, but don't overthink them or plan how to achieve them. Ensure the team has what they need to experiment, follow the Improvement Kata, and iterate.

Pursue a high-performance culture of continuous improvement

Perhaps the most important outcome of deploying the Improvement Kata is to create an organization in which continuous improvement is a habit.

Start with the right people

New ways of working diffuse through organizations in the same way other innovations do, as we describe at the beginning of [Chapter 2](#). The key is to find people who have a growth mindset (see [Chapter 11](#)) and are comfortable with trying out new ideas. Once you have achieved positive results, move on to the early adopters, followed by the early majority. The rest is relatively easy, because there's nothing the late majority hates more than being in the minority. This approach can be applied for each of the three horizons described in [Chapter 2](#).

Find a way to deliver valuable, measurable results from early on

Although lasting change takes time and is never completed, it is essential to demonstrate real results quickly, as the GDS team did. Then, keep doing so to build momentum and credibility. In fact, the Improvement Kata strategy is designed to achieve this goal, which we hope will make it attractive to executives who typically have to demonstrate results quickly and consistently on a tight budget.

As you experiment and learn, share what works and what doesn't. Run regular showcases inviting key stakeholders in the organization and your next adoption segment. Hold retrospectives to reflect on what you have achieved and use them to update and refine your vision. Always, keep moving forward. Fear, uncertainty, and discomfort are your compasses toward growth. You can start right now by filling out the simple one-page form shown in [Figure 15-4](#) (see [Chapter 11](#) for more details on target conditions). For more on how to create sustainable change, particularly in the absence of executive support, we recommend *Fearless Change: Patterns for Introducing New Ideas* by Mary Lynn Manns and Linda Rising.¹⁵

¹⁵ [\[manns\]](#)

Your Draft Transformation Plan

Business Objective		
Change Management Plan		
Iteration 1 Objectives (target date: 2-6 weeks from now)		
Rank	Theme	Target Condition

Figure 15-4. Draft transformation plan

Conclusion

Creating a resilient, lean enterprise that can adapt rapidly to changing conditions relies on a culture of learning through experimentation. For this culture to thrive, the whole organization must be aware of its purpose and work continuously to understand the current conditions, set short-term target conditions, and enable people to experiment to achieve them. We then reassess our current conditions, update our target conditions based on what we learned, and keep going. This behavior must become habitual and pervasive. That is how we create a mindset of continuous improvement focused on ever higher levels of customer service and quality at ever lower costs.

These principles are the threads that link all scientific patterns together. Whether you're seeking a repeatable business model through the Lean Startup learning loop, working to improve your product through user research and continuous delivery, or driving process innovation and organizational change using PDCA cycles of the Improvement Kata—all that is based on a disciplined, rigorous pursuit of innovation in conditions of uncertainty. That the same principles are at the heart of both lean product development and effective process and cultural change was an epiphany for the authors of this book, but perhaps it should not be a surprise—in both cases we face uncertainty and have to deal with a complex adaptive system whose response to change is

unpredictable. Both these situations call for iterative, incremental progress, achieved through human creativity harnessed by the scientific method.

Organizations must continually revisit the question: “What is our purpose, and how can we organize to increase our long-term potential and that of our customers and employees?” The most important work for leaders is to pursue the high-performance culture described in this book. In this way, we can prosper in an environment of constant advances in design and technology and wider social and economic change.

Bibliography

- [adzic] Adzic, G. (2012). *Impact Mapping: Making a Big Impact with Software Products and Projects*. Provoking Thoughts.
- [anderson] Anderson, D. (2010). *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.
- [argyris] Argyris, C. and Schön, D. (1978). *Organizational Learning: A Theory of Action Perspective*. Addison Wesley.
- [arnold] Arnold, J. and Yüce, Ö. (2013). “Black Swan Farming Using Cost of Delay: Discover, Nurture and Speed Up Delivery of Value.” *Agile Conference, 2013*: 101–116.
- [baghai] Baghai, M., Coley, S., and White, D. (1999). *The Alchemy of Growth*. Texere.
- [bell] Bell, S. C. and Orzen, M. A. (2011). *Lean IT*. Productivity Press.
- [bertrand] Bertrand, M. and Mullainathan, S. (2004). “Are Emily and Greg More Employable Than Lakisha and Jamal? A Field Experiment on Labor Market Discrimination?” *American Economic Review*, vol. 94, no. 4: 991–1013.
- [betz] Betz, C. (2006). *Architecture and Patterns for IT Service Management, Resource Planning, and Governance: Making Shoes for the Cobbler’s Children*. Morgan Kaufmann.
- [blank] Blank, S. (2005). *The Four Steps to the Epiphany: Successful Strategies for Products That Win*. K&S Ranch Press.
- [bodek] Bodek, N. (2004). *Kaikaku: The Power and Magic of Lean*. PCS Press.

- [bognes] Bognes, B. (2009). *Implementing Beyond Budgeting*. John Wiley & Sons.
- [bossavit] Bossavit, L. (2013). *The Leprechauns of Software Engineering: How Folklore Turns into Fact, and What to Do About It*. Leanpub. <https://leanpub.com/leprechauns>, 2013-11-20 edition.
- [bungay] Bungay, S. (2010). *The Art of Action: How Leaders Close the Gaps Between Plans, Actions, and Results*. Nicholas Brealey Publishing.
- [cagan] Cagan, M. (2008). *Inspired: How to Create Products Customers Love*. SVPG Press.
- [ceci] Ceci, S. J. and Williams, W. M. (2010). “Gender Differences in Math-Intensive Fields.” *Current Directions in Psychological Science*, 19: 275–279.
- [cediey] Cédiey, E., Foroni, F., and Garner, H. (2008). “Discrimination à l'embauche fondée sur l'origine à l'encontre des jeunes français(e)s peu qualifié(e)s.” *Premières Infos Premières Synthèses*, 06.3.
- [creveld] Creveld, M. van, Brower, K., and Canby, S. (1994). *Air Power and Maneuver Warfare*. Air University Press. Freely available at <https://archive.org/details/airpowermaneuver00mart>.
- [crispin] Crispin, L. and Gregory, J. (2009). *Agile Testing: A Practical Guide for Testers and Agile Teams*. Addison-Wesley.
- [croll] Croll, A. and Yoskovitz, B. (2012). *Lean Analytics: Use Data to Build a Better Startup Faster*. O'Reilly.
- [dekker] Dekker, S., Hollnagel, E., Woods, D., and Cook, R. (2008). *Resilience Engineering: New Directions for Measuring and Maintaining Safety in Complex Systems*. Lund University School of Aviation.
- [deming] Deming, W. E. (2000). *Out of the Crisis*. MIT Press.
- [CIMA] Edwards, S. (2008). *Activity Based Costing: Topic Gateway Series No 1*. CIMA.
- [farris] Farris, P. W., Bendle, N. T., Pfeifer, P. E., and Reibstein, D. J. (2010). *Marketing Metrics: The Definitive Guide to Measuring Marketing Performance*. 2nd ed. Pearson.
- [forrester] Forrester Consulting (2013). “Continuous Delivery: A Maturity Assessment Model.” Can be accessed at <http://thght.works/1zkLGlx>.
- [forsgren] Forsgren, N., Kim, G., Kersten, N., and Humble, J. (2014). *2014 State of DevOps Report*. PuppetLabs.
- [freeman] Freeman, S. and Price N. (2009). *Growing Object-Oriented Software, Guided by Tests*. Addison-Wesley.

- [gilb-88] Gilb, T. (1988). *Principles of Software Engineering Management*. Addison-Wesley.
- [gilb-05] Gilb, T. (2005). *Competitive Engineering: A Handbook for Systems Engineering, Requirements Engineering, and Software Engineering Using Planguage*. Butterworth-Heinemann.
- [goldin] Goldin C. and Rouse C. (2000). “Orchestrating Impartiality: The Impact of ‘Blind’ Auditions on Female Musicians.” *American Economic Review*, 90, no. 4: 715–741.
- [gothelf] Gothelf, J. and Seiden, J. (2013). *Lean UX: Applying Lean Principles to Improve User Experience*. O'Reilly.
- [gray] Gray, D., Brown, S., and Macanufo, J. (2010). *Gametstorming*. O'Reilly.
- [groysberg] Groysberg, B. (2010). *Chasing Stars: The Myth of Talent and the Portability of Performance*. Princeton University Press.
- [gruver] Gruver, G. (2012). *A Practical Approach to Large-Scale Agile Development: How HP Transformed LaserJet FutureSmart Firmware*. Addison-Wesley.
- [hope] Hope, J. and Fraser, R. (2003). *Beyond Budgeting: How Managers Can Break Free from the Annual Performance Trap*. Harvard Business School Press.
- [hubbard] Hubbard, D. (2010). *How to Measure Anything: Finding the Value of “Intangibles” in Business*. 2nd ed. Wiley.
- [humble] Humble, J. and Farley, D. (2010). *Continuous Delivery: Reliable Software Releases through Build, Test and Deployment Automation*. Addison-Wesley.
- [isaac] Isaac, C., Lee, B., and Carnes, M. (2009). “Interventions That Affect Gender Bias in Hiring: A Systematic Review.” *Academic Medicine*, 84: 1440–1446.
- [COBIT5] ISACA (2012). *COBIT 5 Framework*. ISACA and ITGI.
- [kahneman] Kahneman, D. (2011). *Thinking, Fast and Slow*. Farrar, Straus and Giroux.
- [kane] Kane, S. (2014). *Your Startup Is Broken: Inside the Toxic Heart of Tech Culture*. Model, View, Culture.
- [kerth] Kerth, N. (2001). *Project Retrospectives: A Handbook for Team Reviews*. Dorset House.
- [kim] Kim, G., Behr, K., and Spafford, G. (2013). *The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win*. IT Revolution Press.

- [klein] Klein, F. K. (2007). *Giving Notice: Why the Best and Brightest Are Leaving the Workplace and How You Can Help Them Stay*. Jossey-Bass.
- [kohavi] Kohavi, R. (2009). “Online Experimentation at Microsoft.” <http://stanford.io/130uW6X>.
- [kotter] Kotter, J. (2012). *Leading Change*. Harvard Business Review Press.
- [lane-fox] Lane-Fox, M. (2010). “DirectGov 2010 and Beyond: Revolution Not Evolution.” Letter to Francis Maude. <http://bit.ly/11iByi9>.
- [lapouchnian] Lapouchnian, A. (2005). “Goal-Oriented Requirements Engineering: An Overview of the Current Research.” University of Toronto Department of Computer Science.
- [liker] Liker, J. (2003). *The Toyota Way: 14 Management Principles from the World’s Greatest Manufacturer*. McGraw-Hill.
- [limoncelli] Limoncelli, T. A., Chalup, S. R., and Hogan, C. J. (2014). *The Practice of Cloud System Administration: Designing and Operating Large Distributed Systems*, vol. 2. Addison-Wesley.
- [manns] Manns, M. L. and Rising, L. (2004). *Fearless Change: Patterns for Introducing New Ideas*. Addison-Wesley.
- [march] March, J. (1991). “Exploration and Exploitation in Organizational Learning.” *Organizational Science*, 2: 71–87.
- [martin-12] Martin, K. (2012). *The Outstanding Organization: Generate Business Results by Eliminating Chaos and Building the Foundation for Everyday Excellence*. McGraw-Hill.
- [martin] Martin, K. and Osterling, M. (2014). *Value Stream Mapping: How to Visualize Work and Align Leadership for Organizational Transformation*. McGraw-Hill.
- [mcgregor] McGregor, D. (1985). *The Human Side Of Enterprise: 25th Anniversary Printing*. McGraw-Hill.
- [michaels] Michaels, E., Handfield-Jones, H., and Axelrod, B. (2001). *The War for Talent*. Harvard Business School Press.
- [moore] Moore, G. A. (2011). *Escape Velocity: Free Your Company’s Future from the Pull of the Past*. HarperCollins.
- [moss-racusin] Moss-Racusin, C. A., Dovidio, J. F., Brescoll, V. L., Graham, M. J., and Handelsman, J. (2012). “Science Faculty’s Subtle Gender Biases Favor Male Students.” *Proceedings of the National Academy of Sciences of the United States of America*, 109, no. 41: 16474–16479.
- [OCG] OCG (2007). *ITIL V3, Service Design*. TSO.

- [ohno12] Ohno, T. (2012). *Taiichi Ohnos Workplace Management: Special 100th Birthday Edition*. McGraw-Hill Professional.
- [osterwalder] Osterwalder, A., Pigneur, Y., Smith, A., and 470 practitioners from 45 countries (2010). *Business Model Generation: A Handbook for Visionaries, Game Changers, and Challengers*. Wiley.
- [parnas] Parnas, D. L. (1972). “On the Criteria to Be Used in Decomposing Systems into Modules.” *Communications of the ACM*, 15, no. 12: 1053–1058.
- [patton] Patton, J. (2014). *User Story Mapping: Discover the Whole Story, Build the Right Product*. O'Reilly.
- [pink] Pink, D. H. (2009). *Drive: The Surprising Truth About What Motivates Us*. Penguin Group.
- [poppendieck-06] Poppendieck, M. and Poppendieck, T. (2006). *Implementing Lean Software Development: From Concept to Cash*. Addison-Wesley.
- [poppendieck-09] Poppendieck, M. and Poppendieck, T. (2009). *Leading Lean Software Development: Results Are Not the Point*. Addison-Wesley.
- [poppendieck-14] Poppendieck, M. and Poppendieck, T. (2014). *The Lean Mindset: Ask the Right Questions*. Addison-Wesley.
- [raynor] Raynor, M. and Ahmed, M. (2013). *The Three Rules: How Exceptional Companies Think*. Portfolio.
- [reinertsen] Reinertsen, D. (2009). *The Principles of Product Development Flow: Second Generation Lean Product Development*. Celeritas.
- [rogers] Rogers, E. (2003). *Diffusion of Innovations, 5th Edition*. Free Press.
- [rother-2010] Rother, M. (2010). *Toyota Kata: Managing People for Improvement, Adaptiveness, and Superior Results*. McGraw-Hill.
- [rother] Rother, M. (2014). *Improvement Kata Handbook*. Available from <http://bit.ly/11iBzIY>.
- [rother-2009] Rother, M. and Shook, J. (2009). *Learning to See: Value-Stream Mapping to Create Value and Eliminate Muda*. Lean Enterprise Institute.
- [sackman] Sackman, H., Erikson W. J., and Grant E. E. (1968). “Exploratory Experimental Studies Comparing Online and Offline Programming Performance.” *Communications of the ACM*, 11, no. 1: 3–11.
- [schein] Schein, E. H. (2009). *The Corporate Culture Survival Guide: New and Revised Edition*. Jossey-Bass.

- [schpilberg] Schpilberg, D., Berez, S., Puryear, R., and Shah, S. (2007). “Avoiding the Alignment Trap in Information Technology.” *MIT Sloan Management Review* Fall 2007.
- [seddon] Seddon, J. (1992). *I Want You to Cheat!: The Unreasonable Guide to Service and Quality in Organisations*. Vanguard Consulting.
- [semler] Semler, R. (1995). *Maverick: The Success Story Behind the World’s Most Unusual Workplace*. Grand Central Publishing.
- [senge] Senge, P. (2010). *The Fifth Discipline: The Art & Practice of the Learning Organization*. Doubleday.
- [sobek] Sobek, D. K., II and Smalley, A. (2008). *Understanding A3 Thinking: A Critical Component of Toyota’s PDCA Management System*. Productivity Press.
- [stanovich] Stanovich, K. and West, R. (2000). “Individual Differences in Reasoning: Implications for the Rationality Debate?” *Behavioral and Brain Sciences*, 23: 645–726.
- [stewart] Stewart, P., Murphy, K., Danford, A., Richardson, T., Richardson, M., and Wass, V. (2009). *We Sell Our Time No More: Workers’ Struggles Against Lean Production in the British Car Industry*. Pluto Press.
- [taleb] Taleb, N. N. (2012). *Antifragile: Things That Gain From Disorder*. Random House.
- [westrum] Westrum, R. (2004). “A Typology of Organizational Cultures.” *Quality & Safety in Health Care*, 13: ii22–ii27.
- [westrum-2014] Westrum, R. (2014). “The Study of Information Flow: A Personal Journey.” *Safety Science*, 67: 58–63.
- [widener] Widener, S. K. (2007). “An Empirical Analysis of the Levers of Control Framework.” *Accounting, Organizations and Society*, 32, no. 7–8: 757–788.
- [williams] Williams, B. and Chuvakin, A. (2012). *PCI Compliance, Third Edition: Understand and Implement Effective PCI Data Security Standard Compliance*. Syngress.
- [womack] Womack, J. P. and Jones, D. T. (2010). *Lean Thinking: Banish Waste and Create Wealth in Your Corporation*. 2nd ed. Simon and Schuster.
- [yu] Yu, E., Giorgini, P., Maiden, N., and Mylopoulos, J. (2010). *Social Modeling for Requirements Engineering*. MIT Press.

Index

Symbols

3D printing, [xiv](#)
3M, [40](#), [190](#), [224](#)

A

A/B testing, [32](#), [178-186](#), [270](#)
A3 Thinking, [83](#), [102](#), [122](#)
AAMI, [281](#)
Accenture, [289](#), [292](#)
access control, [232](#), [236](#)
accountability, [89](#), [233](#)
Ackoff, Russell L., [217](#)
acqui-hiring, [39](#)
acquisition, [100](#)
actionable metrics, [90](#)
activation, [92](#)
activity accounting, [113](#), [129](#), [249](#),
[253-255](#)
Adzic, Gojko, [174](#), [176](#)
Aetna, [41](#)
affiliates, [100](#)
Affordable Care Act, [41](#)
Alcoa, [117](#)
alignment gap, [14](#)
Allspaw, John, [218](#), [270](#)
Amazon
 A/B testing at, [32](#), [180](#)
 Data Mining and Personalization
 group, [179](#)

deployments at, [155-156](#), [195](#)
Game Days, [273](#)
growth strategy of, [70](#), [100](#), [104](#),
[190-193](#)
innovations at, [39](#)
Marketplace, [40](#), [100](#), [104](#)
performance of, [19](#), [268](#)
portfolio management at, [37](#)
recommendations engine, [185](#)
security at, [271](#)
systems replacement at, [199](#)
 Web Services, [100](#), [191](#)
Ambition to Action, [249](#), [286](#)
Anderson, David J., [143](#)
andon cord, [6](#), [9](#)
AOL Instant Messenger (AIM), [100](#)
APIA (Australian Pipeline Industry Association), [281](#)
Apple, [xiv](#)
 AppStore, [100](#)
 creating mouse for, [73](#)
 growth strategy of, [70](#), [100](#)
 Macintosh, [125](#), [155](#)
 portfolio management at, [37](#)
Application Programming Interfaces
 (APIs), [192](#)
architectural epics, [176](#)
Argyris, Chris, [120](#)
ARM CPU, [29](#)

Arnold, Joshua J., 134-136
artifacts, 211
auditing, 165
Auftragstaktik, 12-15
authority, 195, 198, 233
autonomy, 213-214

B

backlog grooming, 186
Balanced Scorecard, 249
banner ads, 100
batches
 releasing, 157
 size of, 144, 156, 159, 184, 197
Bell, Steve, 49
Bertolini, Mark, 41
Betz, Charles, 267
Bezos, Jeff, 190-191
big bang approach, 32, 104, 170, 193, 199, 201, 241
Big Rewrite, 99
Bing, 179
Blank, Steve, 24, 26, 68, 73
Blockbuster, 68
Bock, Laszlo, 223
Bodek, Norman, 225, 273
Boeing, 190
Bogsnes, Bjarte, 287
Borealis, 249
Bottcher, Evan, 267
Boyd, John, 54-57
Bracken, Mike, 292
branch-based development, 114-115
Brin, Sergey, 225
bring your own device (BYOD), 274
British National Health Service, 289
British Telecom, 289
Brodzinski, Pawel, 147
Brown, Tim, 185
Buckley, Scott, 280
budgeting
 annual cycles for, 31, 246-250
 approaches to, 17, 247
 basing business decisions on, 246, 253
 measuring performance against, 247, 253-256
 replacing with rolling forecasts, 250

build-measure-learn loop, 57
building the right thing, 48, 63, 81, 88, 171
Bungay, Stephen, 13-15, 17
bureaucratic organizations, 9
 acqui-hiring in, 39
 experimental approach in, 60
 hiding information in, 16, 56
 innovations in, 94
 not achieving target conditions in, 124
business cases, 45-53
Business Model Canvas, 68-71
business models
 disrupting, 67, 70
 evaluating, 28-32
 exploiting, 26, 32-39
 measuring changes in, 92
 new, experimenting with, 38, 68, 88
 simplified, 49
 testing, 49-50, 78
business plans, 17, 68, 250
business process changes, 279

C

Cagan, Marty, 77, 80, 266
CapEx/OpEx model, 247, 256-258
cardholder data environment (CDE), 242-243
Carr, Nicholas, 266
catchball, 287
CCS Insight, xiv
Change Advisory Board, 267
change approval processes, 165, 237, 268
change control systems, 165-166
change management process, 232, 267
Chaos Monkey, 274
chasm, 22, 104
chatty services, 193
Christensen, Clayton, 38
churn rate, 100
CIA (Central Intelligence Agency), 272
Clausewitz, Carl von, 12-14
clouds, 271-272
Coaching Kata, 121, 285
COBIT (Control Objectives for Information and Related Technology), 234-235, 238

Cockcroft, Adrian, 270
cohort, 91
collaboration, 104
 and financial constraints, 248
 effective, 191
 improving, 241
 reduced, 197, 235
 rewarding for, 35, 222
command and control, 11, 17, 235
commercial off-the-shelf (COTS), 278
 customizing, 279
 evaluating, 53
 implementing, 30
compensating controls, 243
compensation reviews, 224
competitive advantages, 8, 10, 102
 and IT, 266-281
 for enterprises, 19, 73
 from innovations, 22, 66
 limited timeframe for, 68
competitors
 acquiring, 23
 disrupting, 57
 opening information to, 8, 18
complex adaptive systems, 14-16, 194, 218
compliance, 233
 different demands for, 242
 feedback on, 241-242
 measuring, 236-237
 monitoring, 234
 reduced, 235
 smallest possible changes for, 243
Computer Sciences Corporation, 289
confirmation bias, 56
conscious decisions, 55
continuous delivery, 34, 118, 156-166, 291
 and change control, 165-166
 and service-oriented architecture, 193, 199
 goal of, 156
continuous improvement, 6, 9, 17, 34, 109, 111-130
 and financing, 251
 and process control, 190
 for GRC processes, 232
 for organizational changes, 284
continuous integration (CI), 99, 114, 127, 159-162, 201
 budgeting for, 130
 objections to, 161
 scaling effectively, 160
contract performance, 260
convergent thinking, 67
Conway, Melvin, 202
Conway's Law, 193
Cook, Richard, 218
Cook, Scott, 82
Cost of Delay, 135-136, 147-152, 237, 242, 252, 286
 assumptions for, 151
 consequences of, 150
 for performance improvements, 180
 time sensitivity of, 151
cost of delay divided by duration (CD3), 135-136, 150, 200
costs
 approving, 247
 managing, 186
 of delivering value, 247, 260
 operating, 251-252, 256-258
 reducing, 272
 visibility of, 249
creativity, 235
crisis, 215
Croll, Alistair, 90
culture, 17-19, 209-228
 and differences between individuals, 219
 constantly changing, 209
 definition of, 210
 evolving, 2
 high-performance, 10-11, 19, 73, 111-112, 217
 high-trust, 6, 30, 240, 242, 249, 268
 layers of, 211
 measuring, 11, 210-214
 of continuous improvement, experimentation, and innovation, 9, 35, 104, 116
 of fear, 9, 56, 217
 of responsibility and ownership, 256
 transforming, 39, 98, 213-219
cumulative flow diagram, 144
customer acquisition cost, 93

customer churn, 82
customer lifetime value (CLV), 93, 179
customer success metrics, 93
customers
 access to, 73
 acquiring new, 23, 26, 95, 100
 addicted to the product, 100
 delivering value to, 30, 32, 91
 empathy for, 72, 98
 engaged, 103
 feedback from, 73, 82, 95, 184, 191
 in a mature market, 24
 personal habits of, 74-75
 satisfaction of, 136
 understanding, 72-74
 vs. users, 64
Cybertrust, 237
Cynefin framework, 14

D

Dalzell, Rick, 191
dark launching, 168, 270
data sovereignty, 272
Dekker, Sidney, 218
delivery teams
 and compliance, 236-238, 241-242
 cross-functional, 156
 managing, 234
 no-value-add activities of, 235
Dell, 70
demand, managing, 128
Deming cycle, 58, 119
Deming, W. Edwards, 56, 220, 258
deployment pipeline, 156, 162-166
 developed in-house, 272
 security tests in, 241
deployments
 blue-green, 167
 by teams, autonomous, 195
 decoupled from releases, 167-168, 182
 frequency of, 268
 of a failed product, 184
 push-button, 156-158, 162, 165
design thinking, xiv, 185
 combined with Lean Startup, 64
dev complete, 161, 197
development costs

estimating, 46
reducing, 126-127
DevOps movement, 265, 267
digital cameras, inventing, 39
Dimensions of the Learning Organization Questionnaire (DLOQ), 210
directed opportunism, 15
direction, 117, 293
disasters, testing for, 273-274
Discovery, 64-76
disruption, 54
 avoiding with new repertoire, 57
 of business models, 67
divergent thinking, 67
double-loop learning, 120
Duhigg, Charles, 74
Dweck, Carol, 220-223
Dynamic Priority List, 135
dynamic resource allocation, 251-251

E

eBay
 competitors for, 104
 growth strategy of, 100
EC2, 191
effects gap, 14
elevator pitch, 83
email, software for, 278
empathy
 for coworkers, 141
 for customers and users, 72-74, 98
empowerment, 233
enterprises
 acquiring startups, 19, 39, 102
 agile, 129
 and big bang approach, 31
 architecture of, 199-202
 as human systems, 9
 balancing portfolio in, 35-41
 change control in, 165-166
 decentralizing decision making in, 152
 definition of, 1
 disrupting, 12, 38-41
 growth strategies for, 26, 100
 interactions between components in, 14
 investing in people, 224-225

kaikaku in, 48
long-term survival of, xiv
performance of, 141
reorganizations in, 196
structuring, 194-198
successful, 1, 26
vs. startups, 73, 224
environment, impact on, 2
espoused values, 206, 211
estimation, 186
Etsy, 181-184, 270
 deployments at, 195
PCI-DSS compliance in, 242-243
executives
 controlling business decisions, 179,
 185
 delusional optimism of, 33
 responsibilities of, 2
exit criteria, 122
expand growth strategy, 100
expected opportunity loss (EOL), 53
expected value of information (EVI), 52
experimental approach, 50-60
experiments
 by autonomous teams, 195
 cheap and quick, 27, 29, 74, 98, 156,
 180, 196
 designing, 177, 185, 285
 funding approval for, 196
 key outcome for, 178
 on real users, 158
 online controlled, 177-181
 overall evaluation criteria for, 177,
 179
 reducing uncertainty with, 50
 running series of, 49-50
 safe to fail, 28, 181, 186
 testing, 161, 233
exploit phase, 25, 32-39, 109-202
 growth strategies for, 103
exploratory testing, 156
explore phase, 24-32, 43-105, 251
 costs of, 251, 257
extrinsic motivation, 7, 212

F

F-16 fighter jet, 54

Facebook
 release process at, 168
 testing at, 273
 viral growth of, 100
failure demand, 113, 126
failures, 217-219
 attitude to, 220
 identifying causes of, 217-219
 learning from, 102, 223, 273
 punishing for, 11, 35, 130, 214
 responsibility for, 206-207
 testing for, 273-274
fear
 in pathological culture, 9, 56
 increasing, 217
feature branches, 160
feature churn, 176
feature injections, 178
features
 batched up into projects, 33, 135, 147
 breaking into incremental changes,
 135, 159, 268
 estimating value of, 135
 granting access to, 168
 new, vs. legacy systems, 276
 prioritizing, 135, 145, 148-150
 releasing, 182, 270
 rewarding for, 197
 testing, 179-181, 275
FedEx, xiv
feedback loops
 and organizational culture, 9
 delays in, 16
 effective, 31, 198
 on compliance activities, 241-242
 to improve quality of service, 191
 via MVPs, 80
 with continuous delivery, 34
financial management processes (FMPs),
 17, 245-262
 decentralizing, 253
fitness function, 192
Five Ws and One H, 83
fixed mindset, 222
flow, 184
Forbes, xiv
Force.com, 100
Ford, xiv

forecasting, 246
Forrester, 270
Fortune 500 companies, xiii, 252
Foster, Richard, xiii
Fowler, Martin, 99, 199
Fox, Martha Lane, 289
Fraser, Janice, 177
Freire, Paulo, 214
Fremont Assembly plant, 5, 18, 215
friction, 14
Fujitsu, 289
Furber, Steve, 29
future-state value streams, 138, 142-143
fuzzy front end, 47-48, 134

G

Gallup, 211
Gamestorming, 66
GE (General Electric), xv, 190
gembá, 73, 102, 130
genchi genbutsu, 73
gender bias, 225-228
generative organizations, 10, 59
 managing demand in, 128
 not achieving target conditions in, 124
 observing IGT in, 56
 using metrics in, 130
Gervais, Ricky, 213
getting out of the building, 73
Gilb, Tom, 124
Github, 274
Gladwell, Malcolm, 219
GM (General Motors), xv, 5-9, 259
goal-oriented requirements engineering, 174
Google, 293
 AdSense, 225
 continuous integration at, 159
 Disaster Recovery Testing (DiRT), 273
 innovations at, 40, 225
 News, 225
 performance of, 19, 268
 portfolio management at, 37
 product teams at, 269
 recruiting at, 223
 reverting bad changes at, 162
 security at, 271

Site Reliability Engineers (SREs), 269
Gore Company, 195
Gothelf, Jeff, 64, 177
GOV.UK, 290-292
governance, risk, and compliance (GRC), 232-244
 applying lean principles to, 235-238
 changes to, 234-235
 measuring, 237
 responsibility of outcomes for, 236
 vs. management, 234
Government Digital Service (GDS), 289-294
Gray, David, 66
Gregory, Tim, 292
growth hypothesis, 26, 102
growth metrics, 93
growth mindset, 220-224, 294
growth strategies, 100
 in mature market, 23, 103
growth/materiality matrix, 36
Gruver, Gary, 112, 130
Gualtieri, Mike, 270
Guest, David, 196

H

Hammant, Paul, 169
Handelsbanken, 195, 253
handover readiness review, 269
Hastings, Reed, 189, 195
Hauser, Herman, 30
health, safety, and environment (HSE), 287
Healthagen, 41
HealthCare.gov, 292
Hertzfeld, Andy, 155
Hewlett, Bill, 130
high utilization, 34
highest paid person's opinion (HiPPO), 35, 135, 179
HM Prison Service, 292
HMV, 68
holacracy, 194
Hollnagel, Erik, 218
horizon model, 37-41
 and optionality, 251
 growth metrics for, 93

- transitioning between horizons in, 101-104
 Hoshin Kanri, 287
 Hotmail, 100
 HP (Hewlett-Packard), 36
 HP FutureSmart platform, 113, 122, 125-130, 156, 158, 286
 continuous integration in, 159
 deployment pipeline in, 164
 goal of, 293
 team autonomy in, 130
 HP LaserJet Firmware team, 112-129, 122-127
 Hubbard, Douglas, 46, 53, 91, 237
 human errors, 9, 218
 hypotheses, 49
 capturing, 177
 pivoting, 49, 82
 refining, 178
 testing, 27, 76, 97, 178, 285
 validating, 68
 hypothesis-driven development, 177-178
- I**
- IAG (Insurance Australia Group Limited), 2
 IBM
 as package supplier, 271
 mature market for, 23
 portfolio management at, 36
 ICQ Messenger, 100
 ideas
 crazy, 186
 generating, 67
 testing, 179, 185
 validating, 184
 IDEO, 73, 185
 impact mapping, 174-176, 237
 implicit guidance and control (IGT), 55
 Improvement Kata, 17, 59, 112-128, 217, 284-294
 and recruitment, 227
 deploying, 121
 for team alignment, 195
 length of iterations in, 118
 planning, 118
 stages of, 116
 target conditions for, 118-119, 128-129, 172
 with value stream mapping, 138, 142
 impulse buys, 185
 incentives, 197
 for reducing system complexity, 252
 shared equally, 255
 individual productivity, 219
 industry trends, 75
 information security teams, 232, 241, 271
 infrastructure as a service (IaaS), 196, 271
 infrastructure components
 approved list of, 274
 outsourcing, 271
 innovation accounting, 88
 innovation labs, 19, 67
 innovations
 and financial constraints, 248, 250, 271
 and process control, xv, 190
 and simplicity, 95-98
 creating new customers with, 18
 disruptive, 23, 41
 early-stage, 89
 encouraging, 252, 261
 in bureaucratic organizations, 94
 lifecycle of, 21
 responding to, 22, 32
 risks of, 2, 28-29
 scientific approach to, 59
 time spent on, 113
 virtuous cycle of, 102
 Institute for the Future (IFTF), 223
 integration hell, 159, 161
 Intel, xiv, 30
 interfaces, 190
 internal audit teams, 232
 internal tools
 chosen by teams, 196, 274
 Lean Startup approach for, 30, 53
 mandating, 53, 64
 reorganizing, 214
 International Financial Reporting Standards, 246
 Internet, xiv
 intrinsic motivation, 7, 11, 213

Intuit, 82
investments
 managing, 251
 risks of, 46-60
 versus profits, 2
IT mindset, 266
IT operations, 267-272
 performance of, 268-274
IT Revolution Press, 10
iterative development, 125
ITIL (Information Technology Infrastructure Library, 235, 238, 267, 269
iTunes, 68

J

Jenkins, Jon, 156
Jensen, Michael C., 2
jidoka, 158
job satisfaction, 11, 211, 215, 228
job security, 212
Jobs, Steve, 222
JustGiving, 78

K

kaikaku, 32, 48, 142
kaizen, 6, 32
Kanban board, 9, 144
Kanban Method, 118, 128, 143-146
Kane, Shanley, 210
Kaplan, Robert, 249
kata, 116
Kay, John, 2
Kellogg, 2
Kenny, Graham, 1
Kettering Town Football Club, 78
key performance indicators (KPIs), 286
Kindle, 39
Klein, Freada Kapor, 227
Klein, Laura, 177
Knight Capital, 218
knowledge gap, 14
Kodak, 39
Kohavi, Ronny, 32, 179-180
Kordyback, John, 280
Kotter, John, 228, 293
Krishnan, Kripa, 273

L

last responsible moment, 55
lastminute.com, 66, 289
lead time
 and WIP, 144
 in value stream mapping, 139
 monitoring, 157, 270
 reducing, 34, 134-136, 142, 156-157
 with peer review, 165
leadership, 223
Lean Canvas, 71
lean development, 102
lean operations, 102
Lean Startup, 26-38
 combined with design thinking, 64
 executing, 49
 lifecycle of, 51
Lean Thinking, 184
learning anxiety, 216, 222, 225, 284
legacy systems, 275-281
 rearchitecting, 275
 testing, 275-276
Level Playing Field Institute, 226
Limoncelli, Tom, 269
Linden, Greg, 185
LinkedIn, 84
Little's Law, 144
love metrics, 40, 82, 92

M

Madrid, Rick, 7
Maersk, 134-136, 144, 147
 budgeting in, 253
Management by Wandering Around, 130
managers
 beliefs about workers, of, 212
 cooperating with workers, 6
 in experimental approach, 59
 privileges of, 7
 responding to failures, 217
 training for, 121
 under conditions of uncertainty, 116
maneuver warfare, 12, 54
Manns, Mary Lynn, 294
markets
 mature, 23
 researching, 47

- selecting, 103
- Marsick, Victoria J., 210
- Martin, Karen, 137, 139, 142, 287
- mastery, 213
- Matts, Chris, 178
- Maude, Francis, 289
- Maurya, Ash, 49, 91
- McClure, Dan, 91
- McGregor, Douglas, 212
- McKinley, Dan, 183
- McNerney, James, 190
- measurable customer outcome, 53
- measurements
- as probability distribution, 50
 - cost of performing, 52
 - from deployment pipeline, 165
 - that matter most, 91
 - with MVP, 27, 30
- Meckling, William H., 2
- Microsoft
- A/B testing at, 32, 180
 - acquisition of Nokia, xv
 - as package supplier, 271
 - Excel, 47
 - growth strategy of, 100
 - Money, 100
 - Office, 100
 - portfolio management at, 36
 - security at, 271
 - Windows, 100
- military training, 12-13
- minimum viable product (MVP), 76-82
- designing, 49
 - executing, 54
 - key metric for, 81
 - measurements with, 27, 30, 51
 - shortcuts in, 52
 - types of, 78
- mission, 1
- Mission Command, 12-15, 190, 288
- implementing, 194-202
 - in business, 16-17
- mistakes, tolerance for, 18
- mobile apps, dark launching for, 168
- mobile technologies, 75
- monitoring, 240, 246
- monitoring systems, 268, 270
- Monte Carlo simulations, 47-48
- monthly burn rate, 93
- Moore, Geoffrey, 22, 36
- Moore, Gordon, xiv
- Motorola, 30
- Mott, Randy, xv
- Musk, Elon, 2
- MySpace, 100
- ## N
- Napoleon Bonaparte, 12
- narrative fallacy, 115
- National Broadband Network, 165
- National Offender Management Service, 292
- National Programme for IT, 289
- NATO, 13, 108
- Netflix, 270
- annual compensation reviews in, 227
 - culture of, 189
 - deployments at, 195
 - growth strategy of, 100
 - performance of, 19, 68
 - Simian Army, 274
- Nicholson, Geoff, 189
- no-ops model, 270
- no-value-add activities, 112-113, 125, 186, 235, 286
- Nokia, xiv
- Norton, David, 249
- NoSQL databases, 274
- NUMMI (New United Motor Manufacturing, Inc.), 5-9, 18, 213-215, 219, 259
- ## O
- Obamacare, 292
- Obidos, 190, 199
- observe, orient, decide, act (OODA) loop, 54, 56
- Ohno, Taiichi, xvi, 174, 225
- One Metric That Matters (OMTM), 81-84, 94, 105, 148
- operating costs, 251-252, 256-258
- Opportunity Canvas, 71
- options, 27
- Oracle, 271
- organizational change programs, 284-288

- organizations
acqui-hiring in, 39
adaptive, 217
changing structure of, 7, 143
creating value for, 32
decentralized, 194-195
evolving, 109
existing repertoire of, 57
growing, 189-198
infrastructure components for, 271
interactions between components in, 14
observed behaviors within, 212
purpose of, 1-2, 286, 293
reorganizing, 19
strategy of, 2
under conditions of uncertainty, 6, 13, 26-29
winning, 68
- orientation, 55-56
- Orzen, Mike, 49
- Osterling, Mike, 137, 139, 142
- Osterwalder, Alex, 68, 70
- outages, responding to, 268, 270
- outcomes, 172, 177, 285
focusing on, 186, 236
highly visible measurement of, 240
rewarding for, 197, 247
- outsourcing, xv, 289
in banks, 113
of IT operations, 271
of software engineering, 143
of testers, 193
profitability of, 292
- overall evaluation criterion (OEC), 177, 179
- O'Neill, Paul, 117
- P**
- Packard, Dave, 130
- Page, Larry, 225
- pair programming, 9, 165, 268
- Pancino, Matt, 280
- patents, 18
- pathological organizations, 9
acqui-hiring in, 39
control in, 268
- experimental approach in, 60
hiding information in, 16, 56
not achieving target conditions in, 124
- Patton, Jeff, 97
- pay growth strategy, 100
- Paypal, 100
- payroll, software for, 278
- PCI-DSS, 156, 242-243
- percent complete and accurate (%C/A), 139-142
- performance
and bureaucracy, 195
and rewards, 197, 255-256
and size of organization, 19
improving, 111-112, 136-146, 180
in enterprises, 141
measuring, 10, 211, 237, 247, 253-256
monitoring, 234
of IT departments, 268-274
optimal, 142, 274
reducing, 7, 213
- performance reviews, 211, 222, 224
gender bias in, 228
- personal development plans, 224
- personas, 72-74
- Peters, Tom, 57
- phase-gate paradigm, 108-109, 143
- Pigneur, Yves, 68
- Pink, Dan, 65, 213
- pirate metrics, 91-92
- planning, 246
performed annually, 246
strategic, 286
upfront, 17, 31-32, 113, 249, 259
- planning fallacy, 33, 249, 260
- platform as a service (PaaS), 196, 201, 271
- platform growth strategy, 100
- platform, developing vs. outsourcing, 271-272
- Pols, Andy, 200
- Poppendieck, Mary and Tom, 117, 124, 157
- Post-It Note, 189, 224
- postmortem, 217-218, 273
- Prasad, Venkatesh, xiv
- pregnant women, 74

- preventive controls, 239-240
 Principle of Mission, 16, 34, 56, 117, 150, 175, 192, 202, 238
 Principle of Optionality, 27, 251
 principle-based governance, 293
 prison visits, booking, 292
 probability distribution, 50
 problem/solution fit, 49, 53
 metrics for, 92
 problems
 articulating, 65
 discovering and resolving, 6
 exploring solution first, 43, 273
 identifying potential solutions for, 67
 process control, 189
 product development
 creating hypotheses first, 178
 engaging all teams in, 191
 experimental approach to, 171-186
 long cycles of, 47
 outsourcing, 259
 risks of, 259
 target conditions for, 173
 product paradigm, 252
 product/market fit, 49, 88-105
 metrics for, 92-94
 productivity
 improving, 125, 127, 158, 235
 measuring, 33, 129
 products
 complementary, 100
 creating new, 38, 59, 88
 customized on demand, xiv
 declining, 23
 disruptive, xiv, 2, 38
 evaluating, 28
 feedback on, 31
 fully functioning, 96
 in a mature market, 24
 inspiration for, 75
 launching, 77, 104
 lifecycle of, 21, 51
 requirements for, 49
 retiring, 252, 257, 258
 successful, 23
 support for, 257, 269-271
 testing, 49-50, 99
 validated, 99
 virality of, 93
 vulnerabilities in, 241
 profit and loss (P&L), 198, 252
 profit-seeking paradox, 2
 profits
 distributed to all employees, 256
 maximizing, 2
 using for continuous innovation, 102
 program management, 17
 program-level backlogs, 173
 programmable logic chips (PALs), 125
 project management office (PMO) teams, 232
 projects
 and productivity, 33
 batching features up into, 33, 135, 147
 capitalized costs of, 257
 discovering new information during, 33
 in enterprises, 32
 planning, 33, 107-108
 successful, 32
 promotions, 227
 Prussian Army, 12-14
 public relations, 100
 pull requests, 165
 PuppetLabs, 10
 purpose, 213
 push-button deployments, 156-158, 162, 165
- ## Q
- Q12 survey, 211
 qi, 56
 qualified security auditor (QSA), 243
 quality, cost, delivery, morale, safety (QCDMS), 287
 Queue Theory, 144
- ## R
- Reagan, Ronald, 240
 recommendations engine, 185
 recruiting, 223
 referral, 92, 100
 Reinertsen, Donald, 16, 33, 47, 49, 147
 release trains, 160

releases
big bang, 104, 170, 193, 241
decoupled from deployment, 167-168, 182
frequency of, 157, 161, 165
regular, 125
Remedy, 279
repertoire, 57
request for proposal (RFP), 259
resources, allocating dynamically, 17
responsibility, 212, 232
retention, 92
Retrospective Prime Directive, 217
return on investment (ROI)
and long development cycles, 47
and profits, 2
for IT programs, 47
revenue, 92
optimizing for, 179
rewards, for desired behavior, 197, 255
rework time, 141
Richards, Chet, 56
Ries, Eric, 25, 57, 76, 88, 90, 100, 157
Rising, Linda, 294
risk and compliance teams, 232-234
risks, 46-60
associated with larger initiatives, 251
managing, 28-29, 233-238, 259
measuring, 46, 237
mitigating, 292
modelling, 46-48
Roberts, Mike, 58
Rogers, Everett, 21
rolling forecasts, 250
Rossi, Chuck, 168
Rother, Mike, 115-118, 121, 214
Royal Pharmaceutical Society, 96
runway, 27, 38, 97
Rushgrove, Gareth, 290

S

safety hazards, 73
Saffo, Paul, 223
Salesforce, 100
SAP accounting system, 136
Sarasvathy, Saras, 29
Sarbanes-Oxley, 156, 238, 246

Sasson, Steve, 39
Schaefer, Ernie, 8
Scharnhorst, David, 12
Schein, Edgar, 210-217, 284
scientific management, 6, 59
and friction, 16
scope management, 186
Scrum framework, 109, 129-130, 147
search engine marketing, 100
security, preventing breaches of, 267
Seddon, John, 34, 113
segregation of duties, 238, 242-243, 268
Seiden, Josh, 64, 177
Semco, 194
service delivery platform (SDP), 272
service-level agreements (SLAs), 273
service-oriented architecture (SOA), 192, 199-199
services
chatty, 193
complementary, 100
costs of, 272
disruptive, xiv, 2
documentation on, 270
handover readiness review for, 269
location-based, 75
retiring, 252, 257
stability of, 165, 265
support for, 257, 269-271
shadow IT, 30
Shafer, Andrew, 219
shared understanding, 65
Shook, John, 6, 215
Simian Army, 274
single-loop learning, 120
Six Sigma, 189
Smith, Burrell, 125
Snowden, Dave, 14, 28
software development, 108-109
agile, 109, 172
as a last resort, 175
in-house, xv, 290
scaling, 193
solution delivery, 232, 236
Southwest Airlines, 195, 253
SpaceX, 2
Spiegel, Larry, 8
Spotify, 68

- stability, 267-268, 272
stand-up meetings, 9
standardization, 267
startups
 A/B testing in, 32
 acquiring, 19, 39, 102
 exploring new opportunities in, 24
 growth strategies for, 100
 survival rate of, 26
 vs. enterprises, 12, 73, 224
Statoil, 286
Stevenson, Chris, 200
sticky growth strategy, 100
story maps, 97
strangler application pattern, 199-201, 276-279, 292
Strategic Factors, 1
strategic planning, 286
strategy deployment, 249, 287-288
subsidiarity, 194, 248
Sun Tzu, 56
Suncorp Group, 280-281
sunk cost fallacy, 184
suppliers, 258-262
 incremental delivery of working software by, 260
 locking into, 289
 working collaboratively, 7, 259
survival anxiety, 215, 284
systems replacement, 47
 big bang, 199, 201
 incremental, 292
 Lean Startup approach for, 53
- T**
- T-shaped people, 196
Taleb, Nassim, 28, 115
talent, 224-228
Target, 74
target conditions, 171-178
 acceptance criteria for, 175
 defining, 286
 focusing on, 142
 for architectural alignment, 202
 for each iteration, 122
 for recruitment, 227
 in value stream mapping, 138
intent of, 175
not achieved, 124
people affected by, 175
program-level, 129-130, 174-175
relative, 287
translated to different levels, 287
updating, 288
Taylor, Frederick Winslow, 6
Taylorism, 6, 59, 147, 213, 219
 and friction, 16
teams
 agile, 120
 articulating problems to, 65, 175
 authority of, 240
 autonomous, 130, 195-199, 234
 capturing information for, 83
 communication between, 191-193, 275-276
 comparing productivity of, 129
 coordinating work across, 128
 cross-functional, 27, 32, 74, 175, 191, 193, 195, 291
 engagement of, 104
 motivating, 65, 252
 practices used by, 9
 reorganizing, 214
 running experiments, 195
 security specialists in, 241
 setting targets themselves, 253
 size of, 31, 52, 64
 two-pizza (2PT), 191-192
 velocity of, 129, 172
 work scheduling for, 135
technical architecture teams, 232
technical change management, 236
technical debt, 99
technologies
 adopting, 23
 lifecycle of, 21
Telstra, 279
tenure, 227
Tesla Motors, 2, 18
test automation, 114, 127, 156, 159-161
 budgeting for, 130
 for experiments, 161
 in parallel, 160
test doubles, 275-276

- test-driven development (TDD), 99, 119, 201
- testing
- automated acceptance criteria for, 280
 - for security, 241
 - frequent, 240
 - regression, 280
 - with a subset of internal customers, 273
- ThoughtWorks, 10
- throughput, 268, 272
- and change approval processes, 268
- Thulin, Inge G., 40
- tinkering, 28
- Tippet, Peter, 237
- Tower Records, 68
- Toyoda, Kiichiro, 18
- Toyota, 5-9
- and patents, 18
 - business model of, 70
 - competitive advantages of, 102
 - genchi genbutsu in, 73
 - Improvement Kata in, 116
 - innovations in, 115
 - long-term vision of, 117
 - managing at, 121
 - retraining workers in, 225
 - suppliers for, 273
- Toyota Production System (TPS), xvi, 5-9, 102
- A3 Thinking in, 122
 - engineering practices in, 99
 - jidoka in, 158
 - organizational culture for, 8, 59
- traditional project management, 59
- training courses, 214, 271
- trunk-based development, 114, 157-162
- two-pizza team (2PT), 191-192
- Twyman's Law, 181
- U**
- UK government, xv, 261, 289-293
- UK Ministry of Justice Digital Team, 292
- uncertainty, 26-29
- early in the design process, 102
 - exploring, 63-84
 - project management for, 59
- reducing, 50, 178
- working under conditions of, 115
- underlying assumptions, 211
- University of California, 226
- urgency profiles, 151
- US Air Force, 54
- US government, 8, 108
- US Marine Corps, 13
- usability testing, 156
- user experience (UX) design, xiv, 183
- user-journey tests, 99
- users
- empathy for, 72, 98
 - feedback from, 156
 - interviewing, 74
 - understanding, 72-74
 - vs. customers, 64
- V**
- validated learning, 27, 52
- value chain mapping, 277
- value demand, 113
- value hypothesis, 26
- Value Proposition Canvas, 71
- value stream loops, 139
- value stream mapping, 102, 129, 137-148
- for governance processes, 238-240
 - future-state, 117
 - in enterprises, 141
 - process blocks in, 139, 143
 - process time in, 139
 - state of processes in, 140
 - visualizing, 143-145
 - with Improvement Kata, 138
- Van Nuys plant, 8
- vanity metrics, 90, 207
- vendor management, software for, 278
- version control, 162, 164
- and high performance, 268
 - changes to environment configuration via, 166
 - running builds from, 166
 - software for, 278
- viral coefficient, 93, 100
- viral growth strategy, 100
- visibility, 233

reduced, 235
vision statement, 1, 117, 234
 evolving rapidly, 184
Visual Management, 102
Vogels, Werner, 191, 274
Voice of the Customer, 102
Votizen, 92

W

waiting time, 134-135, 141
Walmart, 70
Wardley, Simon, 278
water-scrum-fall, 108
Watkins, Karen E., 210
websites
 A/B testing for, 178-180
 delivering new features for, 276
 replacing, 290-292
Welch, John, 3
WestJet, 256
Westrum, Ron, 9, 211, 268
Wilson, Sophie, 29
Wood, Ben, xiv
Woods, David, 218
word of mouth, 40, 101, 104
work in process (WIP), 118, 128
 limiting, 144-147, 150, 286
workers
 acquiring new skills, 142, 220-225,
 286

cooperating, 6, 224
feedback from, 224
frustrated, 30, 286
motivating, 7, 11, 224-225, 255
responsibility of, 18
rewarding, 34
rotating through jobs, 7, 225
unambitious, 212

workshops

generating new ideas with, 67
 immersing in context, 73
wouldn't it be horrible if..., 237

X

Xerox PARC, 38

Y

Yale University, xiii
Yegge, Steve, 190
YIMBY, 78
Yoskovitz, Benjamin, 90
YouTube, 68
Yüce, Özlem, 134-136

Z

Zanca, David, xiv
zheng, 56
zShops, 104