The DevOps Handbook

HOW TO CREATE WORLD-CLASS AGILITY, RELIABILITY, & SECURITY IN TECHNOLOGY ORGANIZATIONS

The DevOps Handbook

HOW TO CREATE WORLD-CLASS AGILITY, RELIABILITY, & SECURITY IN TECHNOLOGY ORGANIZATIONS

GENE KIM, JEZ HUMBLE, PATRICK DEBOIS, & JOHN WILLIS

FEATURING NEW FOREWORD AND UPDATED MATERIALS FROM NICOLE FORSGREN, PhD

ORIGINAL FOREWORD BY JOHN ALLSPAW

IT Revolution
Portland, Oregon

25 NW 23rd Pl, Suite 6314
Portland, OR 97210

Printed in the United States of America

27 26 25 24 23 22 2 3 4 5 6 7 8 9 10

Cover design by Devon Smith Creative
Cover illustration by eboy
Book design by Devon Smith Creative

ISBN: 9781950508402
eBook ISBN: 9781950508433
Web PDF ISBN: 9781942788867
Audio ISBN: 9781950508440

For information about special discounts for bulk purchases
or for information on booking authors for an event,
please visit ITRevolution.com.

CONTENTS

FIGURES & TABLES

NOTE FROM THE PUBLISHER ON THE SECOND EDITION

Impact of the First Edition

Since the original publication of *The DevOps Handbook*, data from the *State of DevOps Report*s and other research continue to show that DevOps improves time to value for businesses and increases productivity and worker well-being. It also helps create nimble, agile businesses that can adjust to overwhelming change, as witnessed in the COVID-19 pandemic of 2020 and beyond.

"I think 2020 has been illuminating in showing what technology can do in a time of incredible crisis," Gene Kim said in his "State of DevOps: 2020 and Beyond" article. "The crisis provided a catalyst for rapid change. And I'm thankful we were able to rise and meet it."[1]

One of the underpinnings of DevOps and *The DevOps Handbook* is that it shows—and is indeed written for—the horses not the unicorns of the business and technology world. DevOps was never, and still is not, only effective at technology giants—the FAANGs—or startups. This book and the DevOps community as a whole have shown time and time again that DevOps practices and processes can take even the most legacy-riddled, old "horse" enterprise organization and turn it into a nimble technology organization.

In 2021, it is clearer than ever before that every business is a technology business and every leader is a technology leader. Not only can technology no longer be ignored or relegated to the basements; it must also be considered a vital part of the entire strategic endeavor of the business.

Changes to the Second Edition

In this expanded edition of *The DevOps Handbook*, the authors have updated the main text where new research, learnings, and experiences have developed and shaped our understanding of DevOps and how it is used in the industry. Additionally we are pleased to include renowned researcher Dr. Nicole Forsgren as

co-author to help update and expand the text with new research and supporting metrics.

CONTINUOUS LEARNING

> We've added some additional insights and resources we've learned since the first edition came out. These "Continuous Learning" sections are highlighted throughout the book as you see here and include new supporting data and additional resources, tools, and techniques to use on your DevOps journey.

We've also expanded the book with additional case studies to illustrate how far DevOps has spread throughout all industries, especially how it has spread beyond the IT department and into the C-suite itself. In addition, at the end of each case study we have added a key takeaway or two that highlight the most important, though not exclusive, lessons learned. Finally, we've updated the conclusion to each part with new resources to continue your learning journey.

What's Next for DevOps and the Age of Software

If the past five years have taught us anything, it is how important technology is and how much we can achieve when IT and the business speak openly and honestly, as DevOps facilitates.

Perhaps nothing illustrates this more than the rapid changes that were necessary due to the COVID-19 pandemic of 2020 and beyond. Through the use of DevOps, organizations mobilized technology to get services to customers, internal and external, in a moment of rapid, unprecedented change. These large, complex organizations, known for their inability to pivot and adapt quickly, suddenly had no other choice.

American Airlines also was able to take advantage of their ongoing DevOps transformation to build big wins quickly, as you can read about in Chapters 1 and 5.

Dr. Chris Strear relates his experiences using the Theory of Constraints to optimize flow in hospitals, as you can read about in Chapter 2.

In 2020 Nationwide Building Society, the world's largest mutual financial institution, was able to respond in weeks to customer needs versus the typical years, thanks to their ongoing DevOps transformation. You can read more about their experience in Chapter 8.

But while technology is a piece of a successful transformation into future ways of working, business leadership must lead the charge. The bottleneck of today is no longer just technical practices (though they still exist); the biggest challenge and necessity is getting business leadership on board. Transformation must be co-created between the business and technology, and the theories presented here can lead that change.

The enterprise can no longer sustain a binary thought process: top down or tech only. We must achieve true collaboration. Ninety percent of that work involves getting the right people engaged, onboard, and aligned. Start there and we can maintain the resulting motivation into the future.

—IT Revolution
Portland, OR
June 2021

FOREWORD TO THE SECOND EDITION

It has been five years since the first edition of *The DevOps Handbook* was released. While so much has changed, so much has also remained the same. Some of our tools and technologies are no longer in vogue or might not exist, but that shouldn't detract any readers. Although the technology landscape has shifted, the underlying principles presented in this book remain as relevant as ever.

In fact, the need for DevOps is even greater today, as organizations need to deliver value quickly, safely, securely, and reliably to their customers and users. To do this, they need to transform their internal processes and leverage technology to deliver value—using DevOps practices. This is true for organizations around the world and across all industry verticals.

Over the past several years, I've led research in the annual *State of DevOps Report*s (first with Puppet and later with DORA and Google), with coauthors Jez Humble and Gene Kim. The research has confirmed that many of the practices described in this book lead to improved outcomes like speed and stability of software releases; reductions in deployment pain and burnout of our engineers; and contributions to organizational performance, including profitability, productivity, customer satisfaction, effectiveness, and efficiency.

For the second edition of *The DevOps Handbook*, we have refreshed the text with updated data based on the latest research and best practices, and included new case studies to share even more stories about what transformations look like "on the ground." Thanks for joining us on this journey of continuous improvement.

—Nicole Forsgren, PhD
Partner at Microsoft Research
2021

FOREWORD TO THE FIRST EDITION

In the past, many fields of engineering have experienced a sort of notable evolution, continually "leveling up" its understanding of its own work. While there are university curriculums and professional support organizations situated within specific disciplines of engineering (civil, mechanical, electrical, nuclear, etc.), the fact is, modern society needs all forms of engineering to recognize the benefits of and work in a multidisciplinary way.

Think about the design of a high-performance vehicle. Where does the work of a mechanical engineer end and the work of an electrical engineer begin? Where (and how, and when) should someone with domain knowledge of aerodynamics (who certainly would have well-formed opinions on the shape, size, and placement of windows) collaborate with an expert in passenger ergonomics? What about the chemical influences of fuel mixture and oil on the materials of the engine and transmission over the lifetime of the vehicle? There are other questions we can ask about the design of an automobile, but the end result is the same: success in modern technical endeavors absolutely requires multiple perspectives and expertise to collaborate.

In order for a field or discipline to progress and mature, it needs to reach a point where it can thoughtfully reflect on its origins, seek out a diverse set of perspectives on those reflections, and place that synthesis into a context that is useful for how the community pictures the future.

This book represents such a synthesis and should be seen as a seminal collection of perspectives on the (I will argue, still emerging and quickly evolving) field of software engineering and operations.

No matter what industry you are in, or what product or service your organization provides, this way of thinking is paramount and necessary for survival for every business and technology leader.

—John Allspaw, CTO, Etsy
Brooklyn, NY, August 2016

PREFACE

Aha!

The journey to complete *The DevOps Handbook* has been a long one—it started with weekly working Skype calls between the co-authors in February of 2011, with the vision of creating a prescriptive guide that would serve as a companion to the as-yet-unfinished book *The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win*.

More than five years later, with over two thousand hours of work, *The DevOps Handbook* is finally here. Completing this book has been an extremely long process, although one that has been highly rewarding and full of incredible learning, with a scope that is much broader than we originally envisioned. Throughout the project, all the co-authors shared a belief that DevOps is genuinely important, formed in a personal "aha" moment much earlier in each of our professional careers, which I suspect many of our readers will resonate with.

> I've had the privilege of studying high-performing technology organizations since 1999, and one of the earliest findings was that boundary-spanning between the different functional groups of IT Operations, Information Security, and Development was critical to success. But I still remember the first time I saw the magnitude of the downward spiral that would result when these functions worked toward opposing goals.
>
> It was 2006, and I had the opportunity to spend a week with the group that managed the outsourced IT Operations of a large airline reservation service. They described the downstream consequences of their large, annual software releases: each release would cause immense chaos and disruption for the outsourcer as well as customers; there would be SLA (service level agreement) penalties because of the customer-impacting

outages; there would be layoffs of the most talented and experienced staff because of the resulting profit shortfalls; there would be much unplanned work and firefighting so that the remaining staff couldn't work on the ever-growing service request backlogs coming from customers; the contract would be held together by the heroics of middle management; and everyone felt that the contract would be doomed to be put out for re-bid in three years.

The sense of hopelessness and futility that resulted created, for me, the beginnings of a moral crusade. Development seemed to always be viewed as strategic, but IT Operations was viewed as tactical, often delegated away or outsourced entirely, only to return in five years in worse shape than it was first handed over.

For many years, many of us knew that there must be a better way. I remember seeing the talks coming out of the 2009 Velocity Conference, describing amazing outcomes enabled by architecture, technical practices, and cultural norms that we now know as DevOps. I was so excited because it clearly pointed to the better way that we had all been searching for. And helping spread that word was one of my personal motivations to co-author *The Phoenix Project*. You can imagine how incredibly rewarding it was to see the broader community react to that book, describing how it helped them achieve their own "aha" moments.

My DevOps "aha" moment was at a startup in 2000—my first job after graduating. For some time, I was one of two technical staff. I did everything: networking, programming, support, systems administration. We deployed software to production by FTP directly from our workstations.

Then in 2004 I got a job at ThoughtWorks, a consultancy where my first gig was working on a project involving about seventy people. I was on a team of eight engineers whose full-time job was to deploy our software into a production-like environment. In the beginning, it was really stressful. But over a few months, we went from manual deployments that took two weeks to an automated deployment that took one hour, where we could roll forward and back in milliseconds using the blue-green deployment pattern during normal business hours.

That project inspired a lot of the ideas in both the *Continuous Delivery* book and this one. A lot of what drives me and others working in this

space is the knowledge that, whatever your constraints, we can always do better, and the desire to help people on their journey.

For me, it was a collection of moments. In 2007 I was working on a data center migration project with some Agile teams. I was jealous that they had such high productivity—able to get so much done in so little time.

For my next assignment, I started experimenting with Kanban in Operations and saw how the dynamic of the team changed. Later, at the Agile Toronto 2008 conference, I presented my IEEE paper on this, but I felt it didn't resonate widely in the Agile community. We started an Agile system administration group, but I overlooked the human side of things.

After seeing the 2009 Velocity Conference presentation "10 Deploys per Day" by John Allspaw and Paul Hammond, I was convinced others were thinking in a similar way. So I decided to organize the first DevOpsDays, accidentally coining the term DevOps.

The energy at the event was unique and contagious. When people started to thank me because it changed their life for the better, I understood the impact. I haven't stopped promoting DevOps since.

In 2008, I had just sold a consulting business that focused on large-scale, legacy IT operations practices around configuration management and monitoring (Tivoli) when I first met Luke Kanies (the founder of Puppet Labs). Luke was giving a presentation on Puppet at an O'Reilly open-source conference on configuration management (CM).

At first I was just hanging out at the back of the room, killing time and thinking, "What could this twenty-year-old tell me about configuration management?" After all, I had literally been working my entire life at some of the largest enterprises in the world, helping them architect CM and other operations management solutions. However, about five minutes into his session, I moved up to the first row and realized everything I had been doing for the last twenty years was wrong. Luke was describing what I now call second-generation CM.

After his session, I had an opportunity to sit down and have coffee with him. I was totally sold on what we now call infrastructure as code. However, while we met for coffee, Luke started going even further, explaining

his ideas. He started telling me he believed that Operations was going to have to start behaving like software developers. They were going to have to keep their configurations in source control and adopt CI/CD delivery patterns for their workflow. Being the old IT Operations person at the time, I think I replied to him with something like, "That idea is going to sink like Led Zeppelin with Ops folk." (I was clearly wrong.)

Then about a year later, in 2009, at another O'Reilly conference, Velocity, I saw Andrew Clay Shafer give a presentation on Agile infrastructure. In his presentation, Andrew showed this iconic picture of a wall between developers and Operations with a metaphorical depiction of work being thrown over the wall. He coined this "the wall of confusion." The ideas he expressed in that presentation codified what Luke was trying to tell me a year earlier. That was the light bulb for me. Later that year, I was the only American invited to the original DevOpsDays in Ghent. By the time that event was over, this thing we call DevOps was clearly in my blood.

DevOps Myths

Clearly, the co-authors of this book all came to a similar epiphany, even if they came there from very different directions. But there is now an overwhelming weight of evidence that the problems described above happen almost everywhere, and that the solutions associated with DevOps are nearly universally applicable.

The goal of writing this book is to describe how to replicate the DevOps transformations we've been a part of or have observed, as well as dispel many of the myths of why DevOps won't work in certain situations. Below are some of the most common myths we hear about DevOps.

Myth—DevOps Is Only for Startups: While DevOps practices have been pioneered by the web-scale, internet "unicorn" companies such as Google, Amazon, Netflix, and Etsy, each of these organizations has, at some point in their history, risked going out of business because of the problems associated with more traditional "horse" organizations: highly dangerous code releases that were prone to catastrophic failure, inability to release features fast enough to beat the competition, compliance concerns, an inability to scale, high levels of distrust between Development and Operations, and so forth.

However, each of these organizations was able to transform their architecture, technical practices, and culture to create the amazing outcomes that we

associate with DevOps. As Dr. Branden Williams, an information security executive, said, "Let there be no more talk of DevOps unicorns or horses but only thoroughbreds and horses heading to the glue factory."[1]

Myth—DevOps Replaces Agile: DevOps principles and practices are compatible with Agile, with many observing that DevOps is a logical continuation of the Agile journey that started in 2001. Agile often serves as an effective enabler of DevOps because of its focus on small teams continually delivering high-quality code to customers.

Many DevOps practices emerge if we continue to manage our work beyond the goal of "potentially shippable code" at the end of each iteration, extending it to having our code always in a deployable state, with developers checking into trunk daily, and if we demonstrate our features in production-like environments.

Myth—DevOps Is Incompatible with ITIL: Many view DevOps as a backlash to ITIL or ITSM (IT Service Management), which was originally published in 1989. ITIL has broadly influenced multiple generations of Ops practitioners, including one of the co-authors, and is an ever-evolving library of practices intended to codify the processes and practices that underpin world-class IT Operations, spanning service strategy, design, and support.

DevOps practices can be made compatible with ITIL processes. To support the shorter lead times and higher deployment frequencies associated with DevOps, many areas of ITIL become fully automated, solving many problems associated with the configuration and release management processes (e.g., keeping the configuration management database and definitive software libraries up to date). And because DevOps requires fast detection and recovery when service incidents occur, the ITIL disciplines of service design, incident, and problem management remain as relevant as ever.

Myth—DevOps Is Incompatible with Information Security and Compliance: The absence of traditional controls (e.g., segregation of duty, change approval processes, manual security reviews at the end of the project) may dismay information security and compliance professionals.

However, that doesn't mean that DevOps organizations don't have effective controls. Instead of security and compliance activities only being performed at the end of the project, controls are integrated into every stage of daily work in the software development life cycle, resulting in better quality, security, and compliance outcomes.

Myth—DevOps Means Eliminating IT Operations, or "NoOps": Many misinterpret DevOps as the complete elimination of the IT Operations function. However, this is rarely the case. While the nature of IT Operations work may change, it remains as important as ever. IT Operations collaborates far earlier in the software life cycle with Development, who continues to work with IT Operations long after the code has been deployed into production.

Instead of IT Operations doing manual work that comes from work tickets, it enables developer productivity through APIs and self-serviced platforms that create environments, test and deploy code, monitor and display production telemetry, and so forth. By doing this, IT Operations becomes more like Development (as do QA and Infosec), engaged in product development, where the product is the platform that developers use to safely, quickly, and securely test, deploy, and run their IT services in production.

Myth—DevOps Is Just "Infrastructure as Code" or Automation: While many of the DevOps patterns shown in this book require automation, DevOps also requires cultural norms and an architecture that allows for the shared goals to be achieved throughout the IT value stream. This goes far beyond just automation. As Christopher Little, a technology executive and one of the earliest chroniclers of DevOps, wrote, "DevOps isn't about automation, just as astronomy isn't about telescopes."[2]

Myth—DevOps Is Only for Open-Source Software: Although many DevOps success stories take place in organizations using software such as the LAMP stack (Linux, Apache, MySQL, PHP), achieving DevOps outcomes is independent of the technology being used. Successes have been achieved with applications written in Microsoft.NET, COBOL, and mainframe assembly code, as well as with SAP and even embedded systems (e.g., HP LaserJet firmware).

Spreading the Aha! Moment

Each of the authors has been inspired by the amazing innovations happening in the DevOps community and the outcomes they are creating, including safe systems of work and enabling small teams to quickly and independently develop and validate code that can be safely deployed to customers. Given our belief that DevOps is a manifestation of creating dynamic, learning organizations that continually reinforce high trust cultural norms, it is inevitable that these organizations will continue to innovate and win in the marketplace.

It is our sincere hope that *The DevOps Handbook* will serve as a valuable resource for many people in different ways:

- a guide for planning and executing DevOps transformations
- a set of case studies to research and learn from
- a chronicle of the history of DevOps
- a means to create a coalition that spans Product Owners, Architecture, Development, QA, IT Operations, and Information Security to achieve common goals
- a way to get the highest levels of leadership support for DevOps initiatives, as well as a moral imperative to change the way we manage technology organizations to enable better effectiveness and efficiency, as well as enabling a happier and more humane work environment, helping everyone become lifelong learners—this not only helps everyone achieve their highest goals as human beings, but also helps their organizations win

INTRODUCTION

Imagine a World Where Dev and Ops Become DevOps

Imagine a world where product owners, Development, QA, IT Operations, and Infosec work together, not only to help each other, but also to ensure that the overall organization succeeds. By working toward a common goal, they enable the fast flow of planned work into production (e.g., performing tens, hundreds, or even thousands of code deploys per day), while achieving world-class stability, reliability, availability, and security.

In this world, cross-functional teams rigorously test their hypotheses of which features will most delight users and advance the organizational goals. They care not just about implementing user features, but also about actively ensuring their work flows smoothly and frequently through the entire value stream without causing chaos and disruption to IT Operations or any other internal or external customer.

Simultaneously, QA, IT Operations, and Infosec are always working on ways to reduce friction for the team, creating the work systems that enable developers to be more productive and get better outcomes. By adding the expertise of QA, IT Operations, and Infosec into delivery teams and automated self-service tools and platforms, teams are able to use that expertise in their daily work without being dependent on other teams.

This enables organizations to create a safe system of work, where small teams are able to quickly and independently develop, test, and deploy code and value quickly, safely, securely, and reliably to customers. This allows organizations to maximize developer productivity, enable organizational learning, create high employee satisfaction, and win in the marketplace.

These are the outcomes that result from DevOps. For most of us, this is not the world we live in. More often than not, the system we work in is broken, resulting in extremely poor outcomes that fall well short of our true potential. In

our world, Development and IT Operations are adversaries; testing and Infosec activities happen only at the end of a project, too late to correct any problems found; and almost any critical activity requires too much manual effort and too many handoffs, leaving us always waiting. Not only does this contribute to extremely long lead times to get anything done, but the quality of our work, especially production deployments, is also problematic and chaotic, resulting in negative impacts to our customers and our business.

As a result, we fall far short of our goals, and the whole organization is dissatisfied with the performance of IT, resulting in budget reductions and frustrated, unhappy employees who feel powerless to change the process and its outcomes.* The solution? We need to change how we work; DevOps shows us the best way forward.

To better understand the potential of the DevOps revolution, let us look at the manufacturing revolution of the 1980s. By adopting Lean principles and practices, manufacturing organizations dramatically improved plant productivity, customer lead times, product quality, and customer satisfaction, enabling them to win in the marketplace.

Before the revolution, average manufacturing plant order lead times were six weeks, with fewer than 70% of orders shipped on time. By 2005, with the widespread implementation of Lean practices, average product lead times had dropped to less than three weeks, and more than 95% of orders were shipped on time.[1] Organizations that did not implement Lean practices lost market share, and many went out of business entirely.

Similarly, the bar has been raised for delivering technology products and services—what was good enough in previous decades is not good enough now. For each of the last four decades, the cost and time required to develop and deploy strategic business capabilities and features have dropped by orders of magnitude. During the 1970s and 1980s, most new features required one to five years to develop and deploy, often costing tens of millions of dollars.

By the 2000s, because of advances in technology and the adoption of Agile principles and practices, the time required to develop new functionality had dropped to weeks or months but deploying into production still required weeks or months, often with catastrophic outcomes.

And by 2010, with the introduction of DevOps and the never-ending commoditization of hardware, software, and now the cloud, features (even entire startup companies) could be created in weeks, quickly being deployed

* This is just a small sample of the problems found in typical IT organizations.

into production in just hours or minutes—for these organizations, deployment finally became routine and low risk. These organizations are able to perform experiments to test business ideas, discovering which ideas create the most value for customers and the organization as a whole, and which are then further developed into features that can be rapidly and safely deployed into production.

	1970s–1980s	1990s	2000s–Present
Era	Mainframes	Client/Server	Commoditization and Cloud
Representative technology of era	COBOL, DB2 on MVS, etc.	C++, Oracle, Solaris, etc.	Java, MySQL, Red Hat, Ruby on Rails, PHP, etc.
Cycle time	1–5 years	3–12 months	2–12 weeks
Cost	$1M–$100M	$100k–$10M	$10k–$1M
At risk	The whole company	A product line or division	A product feature
Cost of failure	Bankruptcy, sell the company, massive layoffs	Revenue miss, CIO's job	Negligible

Source: Adrian Cockcroft, "Velocity and Volume (or Speed Wins)," presentation at FlowCon, San Francisco, CA, November 2013.

Today, organizations adopting DevOps principles and practices often deploy changes hundreds or even thousands of times per day. In an age where competitive advantage requires fast time to market and relentless experimentation, organizations that are unable to replicate these outcomes are destined to lose in the marketplace to more nimble competitors and could potentially go out of business entirely, much like the manufacturing organizations that did not adopt Lean principles.

These days, regardless of what industry we are competing in, the way we acquire customers and deliver value to them is dependent on the technology value stream. Put even more succinctly, as Jeffrey Immelt, CEO of General

Electric, stated, "Every industry and company that is not bringing software to the core of their business will be disrupted."[2] Or as Jeffrey Snover, Technical Fellow at Microsoft, said, "In previous economic eras, businesses created value by moving atoms. Now they create value by moving bits."[3]

It's difficult to overstate the enormity of this problem—it affects every organization, independent of the industry we operate in, the size of our organization, whether we are profit or nonprofit. Now more than ever, how technology work is managed and performed predicts whether our organizations will win in the marketplace or even survive. In many cases, we will need to adopt principles and practices that look very different from those that have successfully guided us over the past decades. (See Appendix 1.)

Now that we have established the urgency of the problem that DevOps solves, let us take some time to explore in more detail the symptomatology of the problem, why it occurs, and why, without dramatic intervention, the problem worsens over time.

The Problem: Something in Your Organization Must Need Improvement (Or You Wouldn't Be Reading This Book)

Most organizations are not able to deploy production changes in minutes or hours, instead requiring weeks or months. Nor are they able to deploy hundreds or thousands of changes into production per day; instead, they struggle to deploy monthly or even quarterly. Nor are production deployments routine, instead involving outages and chronic firefighting and heroics.

In an age where competitive advantage requires fast time to market, high service levels, and relentless experimentation, these organizations are at a significant competitive disadvantage. This is in large part due to their inability to resolve a core, chronic conflict within their technology organization.

The Core, Chronic Conflict

In almost every IT organization, there is an inherent conflict between Development and IT Operations that creates a downward spiral, resulting in ever-slower time to market for new products and features, reduced quality, increased outages, and, worst of all, an ever-increasing amount of technical debt.

The term "technical debt" was first coined by Ward Cunningham. Analogous to financial debt, technical debt describes how decisions we make lead to problems that get increasingly more difficult to fix over time, continually reducing

our available options in the future—even when taken on judiciously, we still incur interest.

One factor that contributes to this is the often competing goals of Development and IT Operations. IT organizations are responsible for many things. Among them are the two following goals, which must be pursued simultaneously:

- Respond to the rapidly changing competitive landscape.
- Provide stable, reliable, and secure service to the customer.

Frequently, Development will take responsibility for responding to changes in the market and for deploying features and changes into production as quickly as possible. IT Operations will take responsibility for providing customers with IT service that is stable, reliable, and secure, making it difficult or even impossible for anyone to introduce production changes that could jeopardize production. Configured this way, Development and IT Operations have diametrically opposed goals and incentives.

Dr. Eliyahu M. Goldratt, one of the founders of the manufacturing management movement, called these types of configuration "the core, chronic conflict"—when organizational measurements and incentives across different silos prevent the achievement of global, organizational goals.[4*]

This conflict creates a downward spiral so powerful it prevents the achievement of desired business outcomes, both inside and outside the IT organization. These chronic conflicts often put technology workers into situations that lead to poor software and service quality and bad customer outcomes, as well as a daily need for workarounds, firefighting, and heroics, whether in Product Management, Development, QA, IT Operations, or Information Security. (See Appendix 2.)

Downward Spiral in Three Acts

The downward spiral in IT has three acts that are likely familiar to most IT practitioners. The first act begins in IT Operations, where our goal is to keep applications and infrastructure running so that our organization can deliver value to customers. In our daily work, many of our problems are due to applications and infrastructure that are complex, poorly documented, and incredibly

* In the manufacturing realm, a similar core, chronic conflict existed: the need to simultaneously ensure on-time shipments to customers and to control costs. How this core, chronic conflict was broken is described in Appendix 2.

fragile. This is the technical debt and daily workarounds that we live with constantly, always promising that we'll fix the mess when we have a little more time. But that time never comes.

Alarmingly, our most fragile artifacts support either our most important revenue-generating systems or our most critical projects. In other words, the systems most prone to failure are also our most important and are at the epicenter of our most urgent changes. When these changes fail, they jeopardize our most important organizational promises, such as availability to customers, revenue goals, security of customer data, accurate financial reporting, and so forth.

The second act begins when somebody has to compensate for the latest broken promise—it could be a product manager promising a bigger, bolder feature to dazzle customers with or a business executive setting an even larger revenue target. Then, oblivious to what technology can or can't do, or what factors led to missing our earlier commitment, they commit the technology organization to deliver upon this new promise.

As a result, Development is tasked with another urgent project that inevitably requires solving new technical challenges and cutting corners to meet the promised release date, further adding to our technical debt—made, of course, with the promise that we'll fix any resulting problems when we have a little more time.

This sets the stage for the third and final act, where everything becomes just a little more difficult, bit by bit—everybody gets a little busier, work takes a little more time, communications become a little slower, and work queues get a little longer. Our work becomes more tightly coupled, smaller actions cause bigger failures, and we become more fearful and less tolerant of making changes. Work requires more communication, coordination, and approvals; teams must wait just a little longer for their dependent work to get done; and our quality keeps getting worse. The wheels begin grinding slower and require more effort to keep turning. (See Appendix 3.)

Although it's difficult to see in the moment, the downward spiral is obvious when one takes a step back. We notice that production code deployments are taking ever-longer to complete, moving from minutes to hours to days to weeks. And worse, the deployment outcomes have become even more problematic, resulting in an ever-increasing number of customer-impacting outages that require more heroics and firefighting in Operations, further depriving them of their ability to pay down technical debt.

As a result, our product delivery cycles continue to move slower and slower, fewer projects are undertaken, and those that are, are less ambitious. Further-

more, the feedback on everyone's work becomes slower and weaker, especially the feedback signals from our customers. And, regardless of what we try, things seem to get worse—we are no longer able to respond quickly to our changing competitive landscape, nor are we able to provide stable, reliable service to our customers. As a result, we ultimately lose in the marketplace.

Time and time again, we learn that when IT fails, the entire organization fails. As Steven J. Spear noted in his book *The High-Velocity Edge*, whether the damages "unfold slowly like a wasting disease" or rapidly "like a fiery crash . . . the destruction can be just as complete."[5]

Why Does This Downward Spiral Happen Everywhere?

For over a decade, the authors of this book have observed this destructive spiral occur in countless organizations of all types and sizes. We understand better than ever why this downward spiral occurs and why it requires DevOps principles to mitigate. First, as described earlier, every IT organization has two opposing goals, and second, every company is a technology company, whether they know it or not.

As Christopher Little, a software executive and one of the earliest chroniclers of DevOps, said, "Every company is a technology company, regardless of what business they think they're in. A bank is just an IT company with a banking license."[6*] To convince ourselves that this is the case, consider that the vast majority of capital projects have some reliance on IT. As the saying goes, "It is virtually impossible to make any business decision that doesn't result in at least one IT change."

In the business and finance context, projects are critical because they serve as the primary mechanism for change inside organizations. Projects are typically what management needs to approve, budget for, and be held accountable for; therefore, they are the mechanism that achieves the goals and aspirations of the organization, whether it is to grow or even shrink.[†]

Projects are typically funded through capital spending (e.g., factories, equipment, and major projects, and expenditures are capitalized when payback is expected to take years), of which 50% is now technology related. This is even true in "low tech" industry verticals with the lowest historical spending on technology, such as energy, metal, resource extraction, automotive, and construction.[8]

* In 2013, the European bank HSBC employed more software developers than Google.[7]

† For now, let us suspend the discussion of whether software should be funded as a "project" or a "product." This is discussed later in the book.

In other words, business leaders are far more reliant upon the effective management of IT in order to achieve their goals than they think.*

The Costs: Human and Economic

When people are trapped in this downward spiral for years, especially those who are downstream of Development, they often feel stuck in a system that preordains failure and leaves them powerless to change the outcomes. This powerlessness is often followed by burnout, with the associated feelings of fatigue, cynicism, and even hopelessness and despair.

Many psychologists assert that creating systems that cause feelings of powerlessness is one of the most damaging things we can do to fellow human beings—we deprive other people of their ability to control their own outcomes and even create a culture where people are afraid to do the right thing because of fear of punishment, failure, or jeopardizing their livelihood. This type of culture can create the conditions for learned helplessness, where people become unwilling or unable to act in a way that avoids the same problem in the future.

For our employees, it means long hours, working on weekends, and a decreased quality of life, not just for the employee, but for everyone who depends on them, including family and friends. It is not surprising that when this occurs, we lose our best people (except for those who feel like they can't leave because of a sense of duty or obligation).

In addition to the human suffering that comes with the current way of working, the opportunity cost of the value that we could be creating is staggering—the authors believe that we are missing out on approximately $2.6 trillion of value creation per year, which is, at the time of this writing, equivalent to the annual economic output of France, the sixth-largest economy in the world.

Consider the following calculation—both IDC and Gartner estimated that in 2011 approximately 5% of the worldwide gross domestic product ($3.1 trillion) was spent on IT (hardware, services, and telecom).[10] If we estimate that 50% of that $3.1 trillion was spent on operating costs and maintaining existing systems, and that one-third of that 50% was spent on urgent and unplanned work or rework, approximately $520 billion was wasted.

* For instance, Dr. Vernon Richardson and his colleagues published this astonishing finding. They studied the 10-K SEC filings of 184 public corporations and divided them into three groups: A) firms with material weaknesses with IT-related deficiencies, B) firms with material weaknesses with no IT-related deficiencies, and C) "clean firms" with no material weaknesses. Firms in Group A saw eight times higher CEO turnover than Group C, and there was four times higher CFO turnover in Group A than in Group C. Clearly, IT may matter far more than we typically think.[9]

If adopting DevOps could enable us—through better management and increased operational excellence—to halve that waste and redeploy that human potential into something that's five times the value (a modest proposal), we could create $2.6 trillion of value per year.

The Ethics of DevOps: There Is a Better Way

In the previous sections, we described the problems and the negative consequences of the status quo due to the core, chronic conflict, from the inability to achieve organizational goals to the damage we inflict on fellow human beings. By solving these problems, DevOps astonishingly enables us to simultaneously improve organizational performance, achieve the goals of all the various functional technology roles (e.g., Development, QA, IT Operations, Infosec), and improve the human condition.

This exciting and rare combination may explain why DevOps has generated so much excitement and enthusiasm with so many and in such a short time, including technology leaders, engineers, and much of the software ecosystem we reside in.

Breaking the Downward Spiral with DevOps

Ideally, small teams of developers independently implement their features, validate their correctness in production-like environments, and have their code deployed into production quickly, safely, and securely. Code deployments are routine and predictable. Instead of starting deployments at midnight on Friday and spending all weekend working to complete them, deployments occur throughout the business day when everyone is already in the office and without our customers even noticing—except when they see new features and bug fixes that delight them. And, by deploying code in the middle of the workday, IT Operations is working during normal business hours like everyone else for the first time in decades.

By creating fast feedback loops at every step of the process, everyone can immediately see the effects of their actions. Whenever changes are committed into version control, fast automated tests are run in production-like environments, giving continual assurance that the code and environments operate as designed and are always in a secure and deployable state.

Automated testing helps developers discover their mistakes quickly (usually within minutes), which enables faster fixes as well as genuine learning—

learning that is impossible when mistakes are discovered six months later during integration testing, when memories and the link between cause and effect have long faded. Instead of accruing technical debt, problems are fixed as they are found, mobilizing the entire organization if needed because global goals outweigh local goals.

Pervasive production telemetry in our code and production environments ensures that problems are detected and corrected quickly, confirming that everything is working as intended and that customers are getting value from the software we create.

In this scenario, everyone feels productive—the architecture allows small teams to work safely and architecturally decoupled from the work of other teams who use self-service platforms that leverage the collective experience of Operations and Information Security. Instead of everyone waiting all the time, with large amounts of late, urgent rework, teams work independently and productively in small batches, quickly and frequently delivering new value to customers.

Even high-profile product and feature releases become routine by using dark launch techniques. Long before the launch date, we put all the required code for the feature into production, invisible to everyone except internal employees and small cohorts of real users, allowing us to test and evolve the feature until it achieves the desired business goal.

And, instead of firefighting for days or weeks to make the new functionality work, we merely change a feature toggle or configuration setting. This small change makes the new feature visible to ever-larger segments of customers, automatically rolling back if something goes wrong. As a result, our releases are controlled, predictable, reversible, and low stress.

It's not just feature releases that are calmer—all sorts of problems are being found and fixed early, when they are smaller and when they are cheaper and easier to correct. With every fix, we also generate organizational learnings, enabling us to prevent the problem from recurring and enabling us to detect and correct similar problems faster in the future.

Furthermore, everyone is constantly learning, fostering a hypothesis-driven culture where the scientific method is used to ensure nothing is taken for granted—we do nothing without measuring and treating product development and process improvement as experiments.

Because we value everyone's time, we don't spend years building features that our customers don't want, deploying code that doesn't work, or fixing something that isn't actually the cause of our problem.

Because we care about achieving goals, we create long-term teams that are responsible for meeting them. Instead of project teams where developers are reassigned and shuffled around after each release, never receiving feedback on their work, we keep teams intact so they can keep iterating and improving, using those learnings to better achieve their goals. This is equally true for the product teams who are solving problems for our external customers, as well as our internal platform teams who are helping other teams be more productive, safe, and secure.

Instead of a culture of fear, we have a high-trust, collaborative culture, where people are rewarded for taking risks. They are able to fearlessly talk about problems as opposed to hiding them or putting them on the back burner—after all, we must see problems in order to solve them.

And, because everyone fully owns the quality of their work, everyone builds automated testing into their daily work and uses peer reviews to gain confidence that problems are addressed long before they can impact a customer. These processes mitigate risk, as opposed to approvals from distant authorities, allowing us to deliver value quickly, reliably, and securely—even proving to skeptical auditors that we have an effective system of internal controls.

When something does go wrong, we conduct blameless post-mortems, not to punish anyone but to better understand what caused the accident and how to prevent it. This ritual reinforces our culture of learning. We also hold internal technology conferences to elevate our skills and ensure that everyone is always teaching and learning.

Because we care about quality, we even inject faults into our production environment so we can learn how our system fails in a planned manner. We conduct planned exercises to practice large-scale failures, randomly kill processes and compute servers in production, and inject network latencies and other nefarious acts to ensure we grow ever more resilient. By doing this, we enable better resilience, as well as organizational learning and improvement.

In this world, everyone has ownership in their work, regardless of their role in the technology organization. They have confidence that their work matters and is meaningfully contributing to organizational goals, proven by their low-stress work environment and their organization's success in the marketplace. Their proof is that the organization is indeed winning in the marketplace.

The Business Value of DevOps

We have decisive evidence of the business value of DevOps. From 2013 through 2016, as part of Puppet Labs' *State Of DevOps Report*, to which authors Nicole

Forsgren, Jez Humble, and Gene Kim contributed, we collected data from over twenty-five thousand technology professionals with the goal of better understanding the health and habits of organizations at all stages of DevOps adoption.*

The first surprise this data revealed was how much high-performing organizations using DevOps practices were outperforming their non–high-performing peers in the following areas:[11]

- Throughput metrics
  - code and change deployments (thirty times more frequent)
  - code and change deployment lead time (two hundred times faster)
- Reliability metrics
  - production deployments (sixty times higher change success rate)
  - mean time to restore service (168 times faster)
- Organizational performance metrics
  - productivity, market share, and profitability goals (two times more likely to exceed)
  - market capitalization growth (50% higher over three years)

In other words, high performers were both more agile and more reliable, providing empirical evidence that DevOps enables us to break the core, chronic conflict. High performers deployed code thirty times more frequently, and the time required to go from "code committed" to "successfully running in production" was two hundred times faster—high performers had lead times measured in minutes or hours, while low performers had lead times measured in weeks, months, or even quarters.

Furthermore, high performers were twice as likely to exceed profitability, market share, and productivity goals. And, for those organizations that provided a stock ticker symbol, we found that high performers had 50% higher market capitalization growth over three years. They also had higher employee job satisfaction, lower rates of employee burnout, and their employees were 2.2 times more likely to recommend their organization to friends as a great place to work.† High performers also had better information security outcomes. By integrating

* The *State of DevOps Report* has since been repeated every year. Additionally, the key findings from the 2013–2018 reports were collected into the book *Accelerate: The Science of Lean Software and DevOps: Building and Scaling High Performing Technology Organizations*.

† As measured by employee Net Promoter Score (eNPS). This is a significant finding, as research has shown that "companies with highly engaged workers grew revenues two and a half times as much as those with low engagement levels. And [publicly traded] stocks of companies with a high-trust work environment outperformed market indexes by a factor of three from 1997 through 2011."[12]

security objectives into all stages of the development and operations processes, they spent 50% less time remediating security issues.

DevOps Helps Scale Developer Productivity

When we increase the number of developers, individual developer productivity often significantly decreases due to communication, integration, and testing overhead.

This is highlighted in the famous book by Frederick Brook, *The Mythical Man-Month*, where he explains that when projects are late, adding more developers not only decreases individual developer productivity but also decreases overall productivity.[13]

On the other hand, DevOps shows us that when we have the right architecture, the right technical practices, and the right cultural norms, small teams of developers are able to quickly, safely, and independently develop, integrate, test, and deploy changes into production.

As Randy Shoup, formerly a director of engineering at Google and now VP Engineering at eBay, observed, large organizations using DevOps "have thousands of developers, but their architecture and practices enable small teams to still be incredibly productive, as if they were a startup."[14]

Only organizations that are deploying at least once per day are shown.
Source: Puppet Labs, *2015 State Of DevOps Report*.

The *2015 State of DevOps Report* examined not only "deploys per day" but also "deploys per day per developer." They hypothesized that high performers would be able to scale their number of deployments as team sizes grew.[15]

Indeed, this is what they found. Figure 0.1 shows that in low performers, deploys per day per developer go down as team size increases, stays constant for medium performers, and increases linearly for high performers. In other words, organizations adopting DevOps are able to linearly increase the number of deploys per day as they increase their number of developers, just as Google, Amazon, and Netflix have done.*

The Universality of the Solution

One of the most influential books in the Lean manufacturing movement is *The Goal: A Process of Ongoing Improvement*, written by Dr. Eliyahu M. Goldratt in 1984. It influenced an entire generation of professional plant managers around the world. It was a novel about a plant manager who had to fix his cost and product due date issues in ninety days, otherwise his plant would be shut down.

Later in his career, Dr. Goldratt described the letters he received in response to *The Goal*. These letters would typically read, "You have obviously been hiding in our factory, because you've described my life [as a plant manager] exactly . . ."[17] Most importantly, these letters showed that people were able to replicate the breakthroughs in performance that were described in the book in their own work environments.

The Phoenix Project: A Novel About IT, DevOps, and Helping Your Business Win, written by Gene Kim, Kevin Behr, and George Spafford in 2013, was closely modeled after *The Goal*. It is a novel that follows an IT leader who faces all the typical problems that are endemic in IT organizations: an over-budget, behind-schedule project that must get to market in order for the company to survive. He experiences catastrophic deployments; problems with availability, security, and compliance; and so forth.

Ultimately, he and his team use DevOps principles and practices to overcome those challenges, helping their organization win in the marketplace. In addition, the novel shows how DevOps practices improved the workplace environment for the team, creating lower stress and higher satisfaction because of greater practitioner involvement throughout the process.

---

* Another more extreme example is Amazon. In 2011, Amazon was performing approximately seven thousand deploys per day. By 2015, they were performing 130,000 deploys per day.[16]

As with *The Goal*, there is tremendous evidence of the universality of the problems and solutions described in *The Phoenix Project*. Consider some of the statements found in the Amazon reviews: "I find myself relating to the characters in *The Phoenix Project* . . . I've probably met most of them over the course of my career," "If you have ever worked in any aspect of IT, DevOps, or Infosec you will definitely be able to relate to this book," or "There's not a character in *The Phoenix Project* that I don't identify with myself or someone I know in real life . . . not to mention the problems faced and overcome by those characters."[18]

The DevOps Handbook: An Essential Guide

In the remainder of this book, we will describe how to replicate the transformation described in *The Phoenix Project*, as well as provide many case studies of how other organizations have used DevOps principles and practices to replicate those outcomes.

The purpose of *The DevOps Handbook* is to give you the theory, principles, and practices you need to successfully start your DevOps initiative and achieve your desired outcomes. This guidance is based on decades of sound management theory, the study of high-performing technology organizations, work we have done helping organizations transform, and research that validates the effectiveness of the prescribed DevOps practices, as well as interviews with relevant subject matter experts and analyses of nearly one hundred case studies presented at the DevOps Enterprise Summit.

Broken into six parts, this book covers DevOps theories and principles using the Three Ways, a specific view of the underpinning theory originally introduced in *The Phoenix Project*. *The DevOps Handbook* is for everyone who performs or influences work in the technology value stream (which typically includes Product Management, Development, QA, IT Operations, and Information Security), as well as for business and marketing leadership, where most technology initiatives originate.

The reader is not expected to have extensive knowledge of any of these domains, or of DevOps, Agile, ITIL, Lean, or process improvement. Each of these topics is introduced and explained in the book as it becomes necessary.

Our intent is to create a working knowledge of the critical concepts in each of these domains, to serve as a primer and to introduce the language necessary to help practitioners work with all their peers across the entire IT value stream, and to frame shared goals.

This book will be of value to business leaders and stakeholders who are increasingly reliant upon the technology organization to achieve their goals.

Furthermore, this book is intended for readers whose organizations might not be experiencing all the problems described in the book (e.g., long deployment lead times or painful deployments). Even readers in this fortunate position will benefit from understanding DevOps principles, especially those relating to shared goals, feedback, and continual learning.

In Part I, we present a brief history of DevOps and introduce the underpinning theory and key themes from relevant bodies of knowledge that span decades. We then present the high-level principles of the Three Ways: Flow, Feedback, and Continual Learning and Experimentation.

Part II describes how and where to start and presents concepts such as value streams, organizational design principles and patterns, organizational adoption patterns, and case studies.

Part III describes how to accelerate flow by building the foundations of our deployment pipeline: enabling fast and effective automated testing, continuous integration, continuous delivery, and architecting for low-risk releases.

Part IV discusses how to accelerate and amplify feedback by creating effective production telemetry to see and solve problems, better anticipate problems and achieve goals, enable feedback so that Development and Operations can safely deploy changes, integrate A/B testing into our daily work, and create review and coordination processes to increase the quality of our work.

Part V describes how we accelerate continual learning and experimentation by establishing a just culture, converting local discoveries into global improvements, and properly reserving time to create organizational learning and improvements.

Finally, in Part VI we describe how to properly integrate security and compliance into our daily work by integrating preventative security controls into shared source code repositories and services, integrating security into our deployment pipeline, enhancing telemetry to better enable detection and recovery, protecting the deployment pipeline, and achieving change management objectives.

By codifying these practices, we hope to accelerate the adoption of DevOps practices, increase the success of DevOps initiatives, and lower the activation energy required for DevOps transformations.

Part I

THE THREE WAYS

PART I: INTRODUCTION

In Part I of *The DevOps Handbook*, we will explore how the convergence of several important movements in management and technology set the stage for the DevOps movement. We describe value streams, how DevOps is the result of applying Lean principles to the technology value stream, and the Three Ways: Flow, Feedback, and Continual Learning and Experimentation.

Primary focuses within these chapters include:

- The principles of Flow, which accelerate the delivery of work from Development to Operations to our customers.
- The principles of Feedback, which enable us to create ever-safer systems of work.
- The principles of Continual Learning and Experimentation, which foster a high-trust culture and a scientific approach to organizational improvement and risk-taking as part of our daily work.

A Brief History

DevOps and its resulting technical, architectural, and cultural practices represent a convergence of many philosophical and management movements. While many organizations have developed these principles independently, understanding that DevOps resulted from a broad stroke of movements, a phenomenon described by John Willis (one of the co-authors of this book) as the "convergence of Dev and Ops," shows an amazing progression of thinking and improbable connections. There are decades of lessons learned from manufacturing, high-reliability organizations, high-trust management models, and others that have brought us to the DevOps practices we know today.

DevOps is the outcome of applying the most trusted principles from the domain of physical manufacturing and leadership to the IT value stream. DevOps

relies on bodies of knowledge from Lean, Theory of Constraints, the Toyota Production System, resilience engineering, learning organizations, safety culture, human factors, and many others. Other valuable contexts that DevOps draws from include high-trust management cultures, servant leadership, and organizational change management.

The result is world-class quality, reliability, stability, and security at ever-lower cost and effort and accelerated flow and reliability throughout the technology value stream, including Product Management, Development, QA, IT Operations, and Infosec.

While the foundation of DevOps can be seen as being derived from Lean, the Theory of Constraints, and the Toyota Kata movement, many also view DevOps as the logical continuation of the Agile software journey that began in 2001.

The Lean Movement

Techniques such as value stream mapping, kanban boards, and total productive maintenance were codified for the Toyota Production System in the 1980s. In 1997, the Lean Enterprise Institute started researching applications of Lean to other value streams, such as the service industry and healthcare.

Two of Lean's central tenets include the deeply held belief that the manufacturing lead time required to convert raw materials into finished goods is the best predictor of quality, customer satisfaction, and employee happiness, and that one of the best predictors of short lead times is small batch sizes of work.

Lean principles focus on how to create value for the customer through systems thinking by creating constancy of purpose, embracing scientific thinking, creating flow and pull (versus push), assuring quality at the source, leading with humility, and respecting every individual.

The Agile Manifesto

The Agile Manifesto was created in 2001 at an invite-only event by seventeen experts in what was then known as "lightweight methods" in software development. They wanted to create a set of values and principles that captured the advantage of these more adaptive methods, compared to the prevailing software development processes such as waterfall development and methodologies such as the Rational Unified Process.

One key principle was to "deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale,"[1]

emphasizing the desire for small batch sizes—incremental releases instead of large, big-bang releases. Other principles emphasized the need for small, self-motivated teams working in a high-trust management model.

Agile is credited for dramatically increasing the productivity and responsiveness of many development organizations. And interestingly, many of the key moments in DevOps history also occurred within the Agile community or at Agile conferences, as described below.

Agile Infrastructure and Velocity Movement

At the 2008 Agile conference in Toronto, Canada, Patrick Debois and Andrew Shafer held a "birds of a feather" session on applying Agile principles to infrastructure as opposed to application code. (In its early days, this was referred to as "Agile system administration.") Although they were the only people who showed up, they rapidly gained a following of like-minded thinkers, including co-author John Willis.

CONTINUOUS LEARNING

> Around the same time, a few academics started studying system administrators, how they applied engineering principles to their work, and how it impacted performance. The leading experts included a group from IBM Research, with ethnographies led by Dr. Eben Haber, Dr. Eser Kandogan, and Dr. Paul Maglio. This was extended to include behavioral quantitative studies led by co-author Dr. Nicole Forsgren in 2007–2009. Nicole went on to lead the research in the 2014–2019 *State of DevOps Reports*, the industry-standard research into practices and capabilities that drive software delivery and performance; these were published by Puppet and DORA.

Later, at the 2009 Velocity conference, John Allspaw and Paul Hammond gave the seminal "10 Deploys per Day: Dev and Ops Cooperation at Flickr" presentation, where they described how they created shared goals between Dev and Ops and used continuous integration practices to make deployment part of everyone's daily work. According to firsthand accounts, everyone attending the presentation immediately knew they were in the presence of something profound and of historic significance.

Patrick Debois was so excited by Allspaw and Hammond's idea that he created the first DevOpsDays in Ghent, Belgium, in 2009, where the term "DevOps" was coined.

The Continuous Delivery Movement

Building upon the development discipline of continuous build, test, and integration, Jez Humble and David Farley extended the concept to continuous delivery, which defined the role of a "deployment pipeline" to ensure that code and infrastructure are always in a deployable state and that all code checked into trunk can be safely deployed into production. This idea was first presented at the 2006 Agile conference and was also independently developed in 2009 by Tim Fitz in a blog post on his website titled "Continuous Deployment."*

Toyota Kata

In 2009, Mike Rother wrote *Toyota Kata: Managing People for Improvement, Adaptiveness, and Superior Results*, which framed his twenty-year journey to understand and codify the Toyota Production System. He had been one of the graduate students who flew with GM executives to visit Toyota plants and helped develop the Lean toolkit, but he was puzzled when none of the companies adopting these practices replicated the level of performance observed at the Toyota plants.

He concluded that the Lean community missed the most important practice of all, which he called the *improvement kata*.[2] He explains that every organization has work routines, and the improvement kata requires creating structure for the daily, habitual practice of improvement work because daily practice is what improves outcomes. The constant cycle of establishing desired future states, setting target outcomes on a cadence, and the continual improvement of daily work is what guided improvement at Toyota.

Throughout the rest of Part I, we will look at value streams, how Lean principles can be applied to the technology value stream, and the Three Ways of Flow, Feedback, and Continual Learning and Experimentation.

* DevOps also extends and builds upon the practices of infrastructure as code, which was pioneered by Dr. Mark Burgess, Luke Kanies, and Adam Jacob. In *infrastructure as code*, the work of Operations is automated and treated like application code, so that modern development practices can be applied to the entire development stream. This further enabled fast deployment flow, including continuous integration (pioneered by Grady Booch and integrated as one of the key 12 practices of Extreme Programming), continuous delivery (pioneered by Jez Humble and David Farley), and continuous deployment (pioneered by Etsy, Wealthfront, and Eric Ries's work at IMVU).

AGILE, CONTINUOUS DELIVERY, AND THE THREE WAYS

In this chapter, we present an introduction to the underpinning theory of Lean Manufacturing, as well as the Three Ways—the principles from which the observed DevOps behaviors can be derived.

Our focus here is primarily on theory and principles, describing many decades of lessons learned from manufacturing, high-reliability organizations, high-trust management models, and others, from which DevOps practices have been derived. The resulting concrete principles and patterns, and their practical application to the technology value stream, are presented in the remaining chapters of the book.

The Manufacturing Value Stream

One of the fundamental concepts in Lean is the value stream. We will define it first in the context of manufacturing and then extrapolate how it applies to DevOps and the technology value stream.

Karen Martin and Mike Osterling define a value stream in their book *Value Stream Mapping: How to Visualize Work and Align Leadership for Organizational Transformation* as "the sequence of activities an organization undertakes to deliver upon a customer request," or "the sequence of activities required to design, produce, and deliver a good or service to a customer, including the dual flows of information and material."[1]

In manufacturing operations, the value stream is often easy to see and observe: it starts when a customer order is received and the raw materials are released onto the plant floor. To enable fast and predictable lead times in any value stream, there is usually a relentless focus on creating a smooth and even flow of work, using techniques such as small batch sizes, reducing work in process (WIP), preventing rework to ensure defects are not passed to downstream work centers, and constantly optimizing systems toward global goals.

The Technology Value Stream

Many principles and patterns that enable the fast flow of work in physical processes are equally applicable to technology work (and, for that matter, for all knowledge work). In DevOps, we typically define our technology value stream as the process required to convert a business hypothesis into a technology-enabled service or feature that delivers value to the customer.

The input to our process is the formulation of a business objective, concept, idea, or hypothesis, and it starts when we accept the work in Development, adding it to our committed backlog of work.

From there, Development teams that follow a typical Agile or iterative process will likely transform that idea into user stories and some sort of feature specification, which is then implemented in code into the application or service being built. The code is then checked into the version control repository, where each change is integrated and tested with the rest of the software system.

Because value is created only when our services are running in production, we must ensure that we are not only delivering fast flow, but that our deployments can also be performed without causing chaos and disruptions, such as service outages, service impairments, or security or compliance failures.

Focus on Deployment Lead Time

For the remainder of this book, our attention will be on deployment lead time, a subset of the value stream described above. This value stream begins when any engineer* in our value stream (which includes Development, QA, IT Operations, and Infosec) checks a change into version control and ends when that change is successfully running in production, providing value to the customer and generating useful feedback and telemetry.

The first phase of work that includes design and development is akin to Lean Product Development and is highly variable and highly uncertain, often requiring high degrees of creativity and work that may never be performed again. Because of this, we expect high variability of process times. In contrast, the second phase of work, which includes testing, deployment, and operations, is akin to Lean Manufacturing. It strives to be predictable and mechanistic, with the goal of achieving work outputs with minimized variability (e.g., short and predictable lead times, near zero defects).

* Going forward, *engineer* refers to anyone working in our value stream, not just developers.

Instead of large batches of work being processed sequentially through the design/development value stream and then through the test/operations value stream (such as when we have a large-batch waterfall process or long-lived feature branches), our goal is to have testing, deployment, and operations happening simultaneously with design/development, enabling fast flow and high quality. This method succeeds when we work in small batches and build quality into every part of our value stream.*

Defining Lead Time vs. Processing Time

In the Lean community, lead time is one of two measures commonly used to measure performance in value streams, with the other being processing time (sometimes known as touch time or task time).†

Whereas the lead time clock starts when the request is made and ends when it is fulfilled, the process time clock only starts when we begin work on the customer request—specifically, it omits the time that the work is in queue, waiting to be processed (Figure 1.1).

Because lead time is what the customer experiences, we typically focus our process improvement attention there instead of on process time. However, the proportion of process time to lead time serves as an important measure

* In fact, with techniques such as test-driven development, testing occurs even before the first line of code is written.

† In this book, the term *process time* will be favored for the same reason Karen Martin and Mike Osterling cite: "To minimize confusion, we avoid using the term cycle time as it has several definitions synonymous with processing time and pace or frequency of output, to name a few."[2]

of efficiency—achieving fast flow and short lead times almost always requires reducing the time our work is waiting in queues.

The Common Scenario: Deployment Lead Times Requiring Months

Many teams and organizations find themselves in situations where deployment lead times require months. This is especially common in large, complex organizations that are working with tightly coupled, monolithic systems, often with scarce integration test environments, long test and production environment lead times, high reliance on manual testing, and multiple required approval processes. When this occurs, our value stream may look like Figure 1.2:

Source: Damon Edwards, "DevOps Kaizen," 2015.

When we have long deployment lead times, heroics are required at almost every stage of the value stream. We may discover that nothing works at the end of the project when we merge all the Development team's changes together, resulting in code that no longer builds correctly or passes any of our tests. Fixing each problem requires days or weeks of investigation to determine who broke the code and how it can be fixed, and still results in poor customer outcomes.

Our DevOps Ideal: Deployment Lead Times of Minutes

In the DevOps ideal, developers receive fast, constant feedback on their work, enabling them to quickly and independently implement, integrate, and validate their code and have the code deployed into the production environment (either by deploying the code themselves or by others).

We achieve this by continually checking small code changes into our version control repository, performing automated and exploratory testing against it and deploying it into production. This enables us to have a high degree of confidence that our changes will operate as designed in production and that any problems can be quickly detected and corrected.

This is most easily achieved when we have architecture that is modular, well encapsulated, and loosely coupled so that small teams are able to work with high degrees of autonomy, with failures being small and contained, and without causing global disruptions.

In this scenario, our deployment lead time is measured in minutes, or, in the worst case, hours. Our resulting value stream map should look something like Figure 1.3.

Observing "%C/A" as a Measure of Rework

In addition to lead times and process times, the third key metric in the technology value stream is percent complete and accurate (%C/A). This metric reflects the quality of the output of each step in our value stream.

Karen Martin and Mike Osterling state that "the %C/A can be obtained by asking downstream customers what percentage of the time they receive work that is 'usable as is,' meaning that they can do their work without having to correct the information that was provided, add missing information that should have been supplied, or clarify information that should have and could have been clearer."[3]

CONTINUOUS LEARNING

Flow Metrics to Measure Delivery of Business Value

When measuring the end-to-end value of any value stream it is important to stay away from proxy metrics (counting the number of lines of code committed or solely the frequency of deployments). While these metrics can reveal local optimiza-

tions, they don't directly link to business outcomes such as revenue.

Using flow metrics provides a look into the end-to-end value of your software delivery, making software products and value streams as visible as widgets on a production line. In his book *Project to Product: How to Survive and Thrive in the Age of Digital Disruption with the Flow Framework*, Dr. Mik Kersten describes flow metrics as: flow velocity, flow efficiency, flow time, flow load, and flow distribution:[4]

- **Flow velocity**: number of flow items (e.g., work items) that are completed in a set time period. Helps to answer whether value delivery is accelerating.
- **Flow efficiency**: the proportion of flow items actively worked on to the total time that has elapsed. Identifies inefficiencies like long wait times and helps teams see if work upstream is in a wait state or not.
- **Flow time**: a unit of business value pulled by a stakeholder through a product's value stream (i.e., features, defects, risks, and debts).Helps teams see if time to value is getting shorter.
- **Flow load**: number of active or waiting flow items in a value stream. This is similar to a measure of work in progress (WIP) based on flow items. High flow load leads to inefficiencies and to reduced flow velocity or increased flow time. Helps teams see if demand is outweighing capacity.
- **Flow distribution**: the proportion of each flow item type in a value stream. Each value stream can track and adjust these depending on their needs in order to maximize the business value being delivered.

The Three Ways: The Principles Underpinning DevOps

The Phoenix Project: A Novel about IT, DevOps, and Helping Your Business Win presents the Three Ways as the set of underpinning principles from which all the observed DevOps behaviors and patterns are derived (Figure 1.4).

The First Way enables fast left-to-right flow of work from Development to Operations to the customer. In order to maximize flow, we need to make work visible, reduce our batch sizes and intervals of work, build in quality by preventing defects from being passed to downstream work centers, and constantly optimize for global goals.

Source: Gene Kim, "The Three Ways: The Principles Underpinning DevOps," ITRevolution.com (blog), August 22, 2012, http://itrevolution.com/the-three-ways-principles-underpinning-devops/.

By speeding up flow through the technology value stream, we reduce the lead time required to fulfill internal or customer requests, especially the time required to deploy code into the production environment. By doing this, we increase the quality of work as well as our throughput and boost our ability to innovate and out-experiment the competition.

The resulting practices include continuous build, integration, test, and deployment processes, creating environments on demand, limiting work in process (WIP), and building systems and organizations that are safe to change.

The Second Way enables the fast and constant flow of feedback from right to left at all stages of our value stream. It requires that we amplify feedback to prevent problems from happening again, or that we enable faster detection and recovery. By doing this, we create quality at the source and generate or embed

knowledge where it is needed—this allows us to create ever-safer systems of work where problems are found and fixed long before a catastrophic failure occurs.

By seeing problems as they occur and swarming them until effective countermeasures are in place, we continually shorten and amplify our feedback loops, a core tenet of virtually all modern process improvement methodologies. This maximizes the opportunities for our organization to learn and improve.

The Third Way enables the creation of a generative, high-trust culture that supports a dynamic, disciplined, and scientific approach to experimentation and risk-taking, facilitating the creation of organizational learning, both from our successes and failures. Furthermore, by continually shortening and amplifying our feedback loops, we create ever-safer systems of work and are better able to take risks and perform experiments that help us learn faster than our competition and win in the marketplace.

As part of the Third Way, we also design our system of work so that we can multiply the effects of new knowledge, transforming local discoveries into global improvements. Regardless of where someone performs work, they do so with the cumulative and collective experience of everyone in the organization and throughout the organization's history.

CONTINUOUS LEARNING

Research Supported: The Three Ways

The Three Ways aren't just a good idea: research has shown that adopting these strategies leads to superior outcomes for both organizations and people.

In a six-year study led by co-author Dr. Nicole Forsgren in the 2014–2019 *State of DevOps Report*s, with Puppet and then DORA and published in the book *Accelerate: The Science of Lean and DevOps*, data shows there are better outcomes by combining capabilities and practices like continuous integration, test, deployment, and working in small batches (the First Way), fast feedback and monitoring (the Second Way), and a generative culture (the Third Way).[5]

The Three Ways help teams become elite performers by shipping software faster and more reliably, helping contribute to their organization's revenue, market share, and customer satisfaction. Elite performers are twice as likely to meet or exceed their organizational performance goals. The Three Ways also improve the well-being of those doing the work. By

adopting these practices, the research from the *State of DevOps Reports* shows decreased burnout and deployment pain.[6]

Approaching Cruising Altitude: American Airlines' DevOps Journey (Part 1) (2020)

American Airlines' DevOps journey grew out of a series of questions, the first being simply "What is DevOps?"

"We were really starting at the very bottom, at the very beginning," Maya Leibman, Executive Vice President and Chief Information Officer of American Airlines related at the DevOps Enterprise Summit-London 2020.[7]

To get started, the team did their research but, most importantly, they stopped making excuses. In the beginning of DevOps, most examples were coming from digital-native companies like Netflix and Spotify. It was easy for the team to discount their accomplishments—after all, they were born in the cloud. But as more traditional enterprises, companies like Target, Nordstrom, and Starbucks, got on board, American Airlines knew they didn't have any excuses left.

The team started by,

1. setting concrete goals
2. formalizing their toolchain
3. bringin in coaches and mentors from outside the company
4. experimenting and automating
5. conducting immersive practical training (to learn while they were doing)

All of this was tied to their ultimate goal, which was to deliver value faster. As Leibman said:

> There were so many times when a business counterpart would bring something to the table, a new idea, and they'd say, "Oh this is what we want to do but it's going to take IT six months or a year to get it done." And those experiences just killed me. So the impetus behind this was really "how do we not be the long tent pole." We knew there was a better way of working that would help us achieve that.[8]

Next, they decided what outputs they were going to measure:[9]

- deployment frequency
- deployment cycle time
- change failure rate
- development cycle time
- number of incidents
- mean time to recover (MTTR)

Early successes in value stream mapping helped team members better understand the end-to-end processes of the system and inspired motivation. From these successes, they built energy around how to attack issues and improve them. They also conducted immersive learning opportunities across IT.

These initial successes, learning about DevOps and starting to actually practice some elements of it, led them to the second big question on their DevOps journey: Finance, friend or foe?

The current finance approval process was cumbersome and lengthy, with months of approval cycles. "I used to describe it as a process that's designed to make you give up," said Leibman.[10]

The process looked like this:[11]

- No projects approved without Finance's involvement.
- Projects were approved but no headcount added to do them (and no other priorities were stopped).
- Requests were given equal scrutiny regardless of size or risk.
- Requests were given equal scrutiny, even if the request was a top corporate priority and there was no question that it was going to be done.
- Projects were often completed before they were approved.

Even Finance knew that the process needed to change, but a lack of trust between Finance and IT caused a block. To help shed light on where the money was being spent and to build trust with Finance, the team undertook a cost mapping exercise and assigned all the costs to their products, including the costs to run them.

After this exercise, the IT team was able to better see where money was actually being invested and question whether that was the best use of it. And Finance was able to gain the visibility they needed to trust there weren't large amounts of waste.

This visibility built the trust needed for experimentation. Finance took four product teams and gave them a set budget for the year. The teams defined the OKRs and used the budget for the top priorities they felt met those OKRs. This allowed the team to test before rollout and focus on accountability and outcomes, and Finance was able to gain even more visibility.

Source: With permission of Ross Clanton

This success allowed them to scale the new model against all of their products and define a new funding process. "This was a huge accelerator in our journey," said Leibman.[12]

With Finance on board and new processes in place, American Airlines discovered the third question in their DevOps journey: How do we know what the score is? With each small success, the team wanted to better understand how they were doing overall. In other words, they wanted to know what the score was.

For the American Airlines team, year one of their DevOps journey was really focused on inputs: learning about Agile/DevOps, focusing on products, cloud, and security, etc. Year two of their journey focused more on outputs, including the metrics they began measuring, like deployment frequency and mean time to recover. Finally in year three they started to focus not just on inputs and outputs but on outcomes. "At the end of the day, what do we really want to do?" said Leibman.

They came up with the following outcomes: make money, improve Ops, increase LTR, and reduce cost.[13]

> In year one, one of our objectives was X% of people are going to go to Agile training. That really represents an input. In year two, as we started focusing more on outputs, the objectives sort of changed to X% of teams are going to up their agile maturity from this level to this level. And by the time we got to year three, agile wasn't even an objective anymore. We realized the inputs and outputs are great, we have to measure them, but ultimately we have to be focused on the outcome.[14]

This finally led to the fourth question in their DevOps journey: What's a product? It was clear that it was time to flesh out their taxonomy. This proved to be one of the most challenging moments of their journey. There were lots of opinions and no single right answer. In the end, they decided to just get started, put something on paper, organize around it, and fix it as they learned. And ultimately, this all led to their fifth question: Does this feel way bigger than DevOps? To answer that and to show some specific product success examples, we'll continue the American Airlines journey later in the book.

Conclusion

In this chapter, we described the concepts of value streams, lead time as one of the key measures of effectiveness for both manufacturing and technology value streams, and the high-level concepts behind each of the Three Ways, the principles that underpin DevOps.

In the following chapters, the principles for each of the Three Ways are described in greater detail. The first of these principles is Flow, which focuses on how we create the fast flow of work in any value stream, whether it's in manufacturing or technology work. The practices that enable fast flow are described in Part III.

THE FIRST WAY: THE PRINCIPLES OF FLOW

In the technology value stream, work typically flows from Development to Operations, the functional areas between our business and our customers. The First Way requires the fast and smooth flow of work from Development to Operations in order to deliver value to customers quickly. We optimize for this global goal instead of local goals, such as Development feature completion rates, test find/fix ratios, or Operations availability measures.

We increase flow by making work visible, by reducing batch sizes, and by building quality in, preventing defects from being passed to downstream work centers. By speeding up the flow through the technology value stream, we reduce the lead time required to fulfill internal and external customer requests, further increasing the quality of our work while making us more responsive to customer and market needs and able to out-experiment the competition.

Our goal is to decrease the amount of time required for changes to be deployed into production and to increase the reliability and quality of those services. Clues on how we do this in the technology value stream can be gleaned from how Lean principles were applied to the manufacturing value stream.

Make Our Work Visible

A significant difference between technology and manufacturing value streams is that our work is invisible. Unlike physical processes, in the technology value stream we cannot easily see where flow is being impeded or when work is piling up in front of constrained work centers. Transferring work between work centers in manufacturing is usually highly visible and slow because inventory must be physically moved.

However, in technology work the move can be done with a click of a button, such as by reassigning a work ticket to another team. Because it is so easy to move, work can bounce between teams endlessly due to incomplete informa-

tion, or work can be passed onto downstream work centers with problems that remain completely invisible until we are late delivering what we promised to the customer or our application fails in the production environment.

To help us see where work is flowing well and where work is queued or stalled, we need to make our work as visible as possible. One of the best methods of doing this is using visual work boards, such as kanban boards or sprint planning boards, where work can be represented on physical or electronic cards. Work originates on the left (often being pulled from a backlog), is pulled from work center to work center (represented in columns), and finishes when it reaches the right side of the board, usually in a column labeled "done" or "in production."

Source: David J. Andersen and Dominica DeGrandis, *Kanban for IT Ops*, training materials for workshop, 2012.

Not only does our work become visible, but we can also manage our work so that it flows from left to right as quickly as possible. This also helps surface unnecessary handoffs in our work, which can introduce errors and additional delays. Furthermore, we can measure lead time from when a card is placed on the board to when it is moved into the "done" column.

Ideally, our kanban board will span the entire value stream, defining work as completed only when it reaches the right side of the board (Figure 2.1). Work is not done when Development completes the implementation of a feature. Rather, it is only done when our application is running successfully in production, delivering value to the customer.

By putting all work for each work center in queues and making it visible, all stakeholders can more easily prioritize work in the context of global goals. Doing this enables each work center to single-task on the highest priority work until it is completed, increasing throughput.

Limit Work in Process (WIP)

In manufacturing, daily work is typically dictated by a production schedule that is generated regularly (e.g., daily, weekly), establishing which jobs must be run based on customer orders, order due dates, parts available, and so forth.

In technology, our work is usually far more dynamic—this is especially the case in shared services, where teams must satisfy the demands of many different stakeholders. As a result, daily work becomes dominated by the priority du jour, often with requests for urgent work coming in through every communication mechanism possible, including ticketing systems, outage calls, emails, phone calls, chat rooms, and management escalations.

Disruptions in manufacturing are also highly visible and costly. They often require breaking the current job and scrapping any incomplete work in process in order to start the new job. This high level of effort discourages frequent disruptions.

However, interrupting technology workers is easy because the consequences are invisible to almost everyone, even though the negative impact to productivity may be far greater than in manufacturing. For instance, an engineer assigned to multiple projects must switch between tasks, incurring all the costs of having to reestablish context, as well as cognitive rules and goals.

Studies have shown that the time to complete even simple tasks, such as sorting geometric shapes, significantly degrades when multitasking. Of course, because our work in the technology value stream is far more cognitively complex than sorting geometric shapes, the effects of multitasking on process time is much worse.[1]

We can limit multitasking when we use a kanban board to manage our work, such as by codifying and enforcing WIP (work in process) limits for each column

or work center, that puts an upper limit on the number of cards that can be in a column.

For example, we may set a WIP limit of three cards for testing. When there are already three cards in the test lane, no new cards can be added to the lane unless a card is completed or removed from the "in work" column and put back into queue (i.e., putting the card back to the column to the left). Nothing can be worked on until it is first represented in a work card, reinforcing that all work must be made visible.

Dominica DeGrandis, one of the leading experts on using kanban in DevOps value streams and author of *Making Work Visible*, notes that "controlling queue size [WIP] is an extremely powerful management tool, as it is one of the few leading indicators of lead time—with most work items, we don't know how long it will take until it's actually completed."[2]

Limiting WIP also makes it easier to see problems that prevent the completion of work.* For instance, when we limit WIP, we find that we may have nothing to do because we are waiting on someone else. Although it may be tempting to start new work (i.e., "It's better to be doing something than nothing"), a far better action would be to find out what is causing the delay and help fix that problem. Bad multitasking often occurs when people are assigned to multiple projects, resulting in prioritization problems. In other words, as David J. Anderson, author of *Kanban: Successful Evolutionary Change for Your Technology Business*, said, "Stop starting. Start finishing."[4]

Reduce Batch Sizes

Another key component to creating smooth and fast flow is performing work in small batch sizes. Prior to the Lean manufacturing revolution, it was common practice to manufacture in large batch sizes (or lot sizes), especially for operations where job setup or switching between jobs was time-consuming or costly. For example, producing large car body panels requires setting large and heavy dies onto metal stamping machines, a process that can take days. When changeover cost is so expensive, we often stamp as many panels at a time as possible, creating large batches in order to reduce the number of changeovers.

However, large batch sizes result in skyrocketing levels of WIP and high levels of variability in flow that cascade through the entire manufacturing plant. The results are long lead times and poor quality—if a problem is found in one body panel, the entire batch has to be scrapped.

* Taiichi Ohno compared enforcing WIP limits to draining water from the river of inventory in order to reveal all the problems that obstruct fast flow.[3]

One of the key lessons in Lean is that in order to shrink lead times and increase quality, we must strive to continually shrink batch sizes. The theoretical lower limit for batch size is single-piece flow, where each operation is performed one unit at a time.*

The dramatic differences between large and small batch sizes can be seen in the simple newsletter mailing simulation described in *Lean Thinking: Banish Waste and Create Wealth in Your Corporation* by James P. Womack and Daniel T. Jones.[5]

Suppose in our own example we have ten brochures to send, and mailing each brochure requires four steps: (1) fold the paper, (2) insert the paper into the envelope, (3) seal the envelope, and (4) stamp the envelope.

The large batch strategy (i.e., "mass production") would be to sequentially perform one operation on each of the ten brochures. In other words, we would first fold all ten sheets of paper, then insert each of them into envelopes, then seal all ten envelopes, and then stamp them.

On the other hand, in the small batch strategy (i.e., "single-piece flow"), all the steps required to complete each brochure are performed sequentially before starting on the next brochure. In other words, we fold one sheet of paper, insert it into the envelope, seal it, and stamp it—only then do we start the process over with the next sheet of paper.

The difference between using large and small batch sizes is dramatic (see Figure 2.2 on page 24). Suppose each of the four operations takes ten seconds for each of the ten envelopes. With the large batch size strategy, the first completed and stamped envelope is produced only after 310 seconds.

Worse, suppose we discover during the envelope sealing operation that we made an error in the first step of folding—in this case, the earliest we would discover the error is at two hundred seconds, and we have to refold and reinsert all ten brochures in our batch again.

In contrast, in the small batch strategy the first completed stamped envelope is produced in only forty seconds, eight times faster than the large batch strategy. And, if we made an error in the first step, we only have to redo the one brochure in our batch. Small batch sizes result in less WIP, faster lead times, faster detection of errors, and less rework.

The negative outcomes associated with large batch sizes are just as relevant to the technology value stream as in manufacturing. Consider when we have an annual schedule for software releases, where an entire year's worth of code that Development has worked on is released to production deployment.

* Also known as "batch size of one" or "1x1 flow," terms that refer to batch size and a WIP limit of one.

(Fold, insert, seal, and stamp the envelope.)
Source: Stefan Luyten, "Single Piece Flow," Medium.com, August 8, 2014, https://medium.com/@stefanluyten/single-piece-flow-5d2c2bec845b.

Like in manufacturing, this large batch release creates sudden, high levels of WIP and massive disruptions to all downstream work centers, resulting in poor flow and poor quality outcomes. This validates our common experience that the larger the change going into production, the more difficult the production errors are to diagnose and fix, and the longer they take to remediate.

In a post on *Startup Lessons Learned*, Eric Ries states,

> The batch size is the unit at which work-products move between stages in a development [or DevOps] process. For software, the easiest batch to see is code. Every time an engineer checks in code, they are batching up a certain amount of work. There are many techniques for controlling these batches, ranging from the tiny batches needed for continuous deployment to more traditional branch-based development, where all of the code from multiple developers working for weeks or months is batched up and integrated together.[6]

The equivalent to single piece flow in the technology value stream is realized with continuous deployment, where each change committed to version control is integrated, tested, and deployed into production. The practices that enable this are described in Part IV of this book.

Reduce the Number of Handoffs

In the technology value stream, whenever we have long deployment lead times measured in months, it is often because there are hundreds (or even thou-

sands) of operations required to move our code from version control into the production environment. To transmit code through the value stream requires multiple departments to work on a variety of tasks, including functional testing, integration testing, environment creation, server administration, storage administration, networking, load balancing, and information security.

Each time the work passes from team to team, we require all sorts of communication: requesting, specifying, signaling, coordinating, and often prioritizing, scheduling, deconflicting, testing, and verifying. This may require using different ticketing or project management systems; writing technical specification documents; communicating via meetings, emails, or phone calls; and using file system shares, FTP servers, and Wiki pages.

Each of these steps is a potential queue where work will wait when we rely on resources that are shared between different value streams (e.g., centralized operations). The lead times for these requests are often so long that there is constant escalation to have work performed within the needed timelines.

Even under the best circumstances, some knowledge is inevitably lost with each handoff. With enough handoffs, the work can completely lose the context of the problem being solved or the organizational goal being supported. For instance, a server administrator may see a newly created ticket requesting that user accounts be created, without knowing what application or service the accounts are for, why they need to be created, what all the dependencies are, or whether the user account creations are actually recurring work.

To mitigate these types of problems, we strive to reduce the number of handoffs, either by automating significant portions of the work, or by building platforms and reorganizing teams so they can self-service builds, testing, and deployments to deliver value to the customer themselves instead of having to be constantly dependent on others. As a result, we increase flow by reducing the amount of time that our work spends waiting in queue, as well as the amount of non–value-added time. (See Appendix 4.)

Continually Identify and Elevate Our Constraints

To reduce lead times and increase throughput, we need to continually identify our system's constraints and improve its work capacity. In *Beyond the Goal*, Dr. Goldratt states, "In any value stream, there is always a direction of flow, and there is always one and only constraint; any improvement not made at that constraint is an illusion."[7] If we improve a work center that is positioned before the constraint, work will merely pile up at the bottleneck even faster, waiting for work to be performed by the bottlenecked work center.

On the other hand, if we improve a work center positioned *after* the bottleneck, it remains starved, waiting for work to clear the bottleneck. As a solution, Dr. Goldratt defined the "five focusing steps":[8]

- Identify the system's constraint.
- Decide how to exploit the system's constraint.
- Subordinate everything else to the above decisions.
- Elevate the system's constraint.
- If a constraint has been broken in the previous steps, go back to step one but do not allow inertia to cause a system constraint.

In typical DevOps transformations, as we progress from deployment lead times measured in months or quarters to lead times measured in minutes, the constraint usually follows this progression:

- **Environment creation:** We cannot achieve deployments on demand if we always have to wait weeks or months for production or test environments. The countermeasure is to create environments that are on-demand and completely self-serviced, so that they are always available when we need them.
- **Code deployment:** We cannot achieve deployments on demand if each of our production code deployments takes weeks or months to perform (e.g., each deployment requires 1,300 manual, error-prone steps, involving up to three hundred engineers). The countermeasure is to automate our deployments as much as possible, with the goal of being completely automated so deployments can be done self-service by any developer.
- **Test setup and run:** We cannot achieve deployments on demand if every code deployment requires two weeks to set up our test environments and data sets and another four weeks to manually execute all our regression tests. The countermeasure is to automate our tests so we can execute deployments safely and to parallelize them so the test rate can keep up with our code development rate.
- **Overly tight architecture:** We cannot achieve deployments on demand if overly tight architecture means that every time we want to make a code change we have to send our engineers to scores of committee meetings in order to get permission to make our changes. Our counter-

measure is to create more loosely coupled architecture so that changes can be made safely and with more autonomy, increasing developer productivity.

After all these constraints have been broken, our constraint will likely be Development or the product owners. Because our goal is to enable small teams of developers to independently develop, test, and deploy value to customers quickly and reliably, this is where we want our constraint to be. High performers, regardless of whether an engineer is in Development, QA, Operations, or Infosec, state that their goal is to help maximize developer productivity.

When the constraint is here, we are limited only by the number of good business hypotheses we create and our ability to develop the code necessary to test these hypotheses with real customers.

The progression of constraints listed above are generalizations of typical transformations—techniques to identify the constraint in actual value streams, such as through value stream mapping and measurements, are described later in this book.

Eliminate Hardships and Waste in the Value Stream

Shigeo Shingo, one of the pioneers of the Toyota Production System, believed that waste constituted the largest threat to business viability—the commonly used definition in Lean is "the use of any material or resource beyond what the customer requires and is willing to pay for."[9] He defined seven major types of manufacturing waste: inventory, overproduction, extra processing, transportation, waiting, motion, and defects.

More modern interpretations of Lean have noted that "eliminating waste" can have a demeaning and dehumanizing context; instead, the goal is reframed to reduce hardship and drudgery in our daily work through continual learning in order to achieve the organization's goals. For the remainder of this book, the term *waste* will imply this more modern definition, as it more closely matches the DevOps ideals and desired outcomes.

In the book *Implementing Lean Software Development: From Concept to Cash*, Mary and Tom Poppendieck describe waste and hardship in the software development stream as anything that causes delay for the customer, such as activities that can be bypassed without affecting the result.[10] Mary and Tom Poppendieck listed the following seven categories of waste and hardship:[11]

- **Partially done work:** This includes any work in the value stream that has not been completed (e.g., requirement documents or change orders not yet reviewed) and work that is sitting in queue (e.g., waiting for QA review or server admin ticket). Partially done work becomes obsolete and loses value as time progresses.
- **Extra processes:** Any additional work being performed in a process that does not add value to the customer. This may include documentation not used in a downstream work center, or reviews or approvals that do not add value to the output. Extra processes add effort and increase lead times.
- **Extra features:** Features built into the service that are not needed by the organization or the customer (e.g., "gold plating"). Extra features add complexity and effort to testing and managing functionality.
- **Task switching:** When people are assigned to multiple projects and value streams, requiring them to context switch and manage dependencies between work, adding additional effort and time into the value stream.
- **Waiting:** Any delays between work requiring resources to wait until they can complete the current work. Delays increase cycle time and prevent the customer from getting value.
- **Motion:** The amount of effort to move information or materials from one work center to another. Motion waste can be created when people who need to communicate frequently are not colocated. Handoffs also create motion waste and often require additional communication to resolve ambiguities.
- **Defects:** Incorrect, missing, or unclear information, materials, or products create waste, as effort is needed to resolve these issues. The longer the time between defect creation and defect detection, the more difficult it is to resolve the defect.

We also add the following two categories of waste from Damon Edwards:[12]

- **Nonstandard or manual work:** Reliance on nonstandard or manual work from others, such as using non-rebuilding servers, test environments, and configurations. Ideally, any manual work that can be automated should be automated, self-serviced, and available on demand. However, some types of manual work will likely always be essential.

- **Heroics:** In order for an organization to achieve goals, individuals and teams are put in a position where they must perform unreasonable acts, which may even become a part of their daily work (e.g., nightly 2:00 AM problems in production, creating hundreds of work tickets as part of every software release).

Our goal is to make these wastes and hardships—anywhere heroics become necessary—visible, and to systematically do what is needed to alleviate or eliminate these burdens and hardships to achieve our goal of fast flow.

Flow and Constraint Management in Healthcare (2021)

DevOps and constraint management theories aren't just for software development or physical manufacturing. They can be applied to nearly any situation. Just look at this case study from the healthcare industry. At the DevOps Enterprise Summit 2021, Dr. Chris Strear, an emergency physician for more than nineteen years, related his experience improving patient outcomes by working with flow.[13]

> Around 2007, our hospital was struggling. We had unbelievable problems with flow. We were boarding patients in the emergency department for hours and hours, and sometimes days, while they waited for an inpatient bed to become available.
>
> Our hospital was so crowded and flow was so backed up that our emergency department was on ambulance diversion for sixty hours a month on average. Now that means that for sixty hours a month, our emergency department was closed to the sickest patients in our community. One month we hit over two hundred hours of diversion.
>
> It was horrible. We couldn't keep nurses. It was such a hard place to work that nurses would quit. And we relied on temporary nurses, on agencies for placing nurses, or traveler nurses to fill in the gaps in staffing. For the most part, these nurses weren't experienced enough to work in the kind of emergency setting where we practiced. It felt dangerous to come to work every day. It felt dangerous to take care of patients. We were just waiting around for something bad to happen.

> The president of our hospital recognized how bad things were, and she put together a committee for flow, and I was lucky enough to be on that committee. . . .
>
> [The change] was transformative. Within a year, we had basically eliminated ambulance diversion. We went from sixty hours a month [of ambulance diversion] to forty-five minutes a month. We improved the length of stay of all of our admitted patients in the hospital. We shortened the time patients spent in the emergency department. We virtually eliminated the patients who left the department without being seen because the waits were too long. And we did all of this in a time when we had record volumes, record ambulance traffic, and record admissions.
>
> [The transformation] was amazing. We took better care of patients. It was safer. And it felt so much easier to take care of patients. It was such an amazing turnaround, in fact, that we were able to stop hiring temporary nurses. We were able to fill our staff completely with dedicated emergency nurses who were qualified to work there. In fact, our department became the number one place for emergency nurses to want to work in the Portland/Vancouver area.
>
> Honestly, I'd never been a part of anything that amazing before, and I haven't been since. We made patient care better for tens of thousands of patients, and we made life better for hundreds of healthcare workers in our hospital.[14]

So how did they manage this turnaround? Sometime before, Chris had been introduced to the book *The Goal*. Constraint management had a profound influence on him and the way he tackled the problem of flow at his hospital.

> So I get asked a bunch of times, what's the difference? I don't have all the answers, but I've seen some trends. I've seen some recurrent themes. Flow needs to be important to leaders, not just in words, but in deeds. They need to walk the walk and not just talk the talk. And a lot of them don't do that.
>
> Part of that is they need to create the bandwidth. The hospital leaders aren't going to be the ones who are actually going to be making the changes day to day. What they have to do is, they have to allow the people who are going to be making those changes to have

enough room on their plate to put in the work. If a nurse manager, for instance, has fifteen projects, fifteen committee meetings that they have to go to day in and day out, and the leader comes along and says, "Flow's important," but now flow is their sixteenth task and the sixteenth meeting that they have to go to, really, it doesn't say that it's important. All it says is it's sixteenth most important.

And then there's managers that aren't going to have time to put in for the sixteenth project. Leaders need to figure out what really is important and what can wait, what can take a back burner, and then take an active role in clearing some of that work off of people's plates so that they can do a job. It doesn't just make those people who have to do the work more effective; it conveys to them in a very real, tangible, palpable sense that this new project, flow, is the most important task.

You have to break down silos. You're looking at flow through a system. You're not looking at flow on an inpatient unit or flow just in the emergency department, because each of these departments, when taken individually, has competing interests. When you move a patient out of the emergency department and onto an inpatient unit, you're creating work for the inpatient unit. You incentivize people differently throughout the hospital.

When you're discussing how to make flow better, and somebody says no, it can't just stop at that. No can't be the final word. I heard time and time again, "We can't do that because that's not how we've done things." And that's ridiculous. No is okay, as long as it's followed up with another idea to try. Because if I have a lousy idea, but it's the only idea out there, then you know what? My lousy idea is the best idea we got going, and so that's the one we try.

Leaders need to make sure that they're measuring things correctly and that they're rewarding things thoughtfully. And what do I mean by that? Well, part of silos in a hospital setting is that a manager for a particular department is often measured on how things go just in that department. And they're rewarded accordingly. People behave based on how they're measured and how they're rewarded. So if improving flow in the emergency department is what's right for patients and what's right for the hospital system, but it may shift burden onto another unit, and that other unit then falls off in their metrics, that should be okay because flow through the hospital is improved. Who cares about flow through an individual unit?

> Make sure that what you're measuring is commensurate with what your overall goals are. Make sure people are rewarded appropriately, and they're not unfairly penalized for improving flow through the system. You need to think about the system, not about the department.
>
> And finally, how we've set things up, that's all artificial, that's a constraint. It's not a natural law of physics. Keep that in mind because so much resistance comes from the uncertainty of doing something differently.
>
> There's often this mindset that because we haven't done something a certain way before, it can't be done. But we've made all of this stuff up. How a body responds to a treatment, that's not artificial, that is a natural law. But where you put a patient, who's in charge of them, how you move a patient from one unit to another, we all just made that up and then perpetuated it. That's all negotiable.[15]

Conclusion

Improving flow through the technology value stream is essential to achieving DevOps outcomes. We do this by making work visible, limiting WIP, reducing batch sizes and the number of handoffs, continually identifying and evaluating our constraints, and eliminating hardships in our daily work.

The specific practices that enable fast flow in the DevOps value stream are presented in Part IV of this book. In the next chapter, we present The Second Way: The Principles of Feedback.