THE DEVOPS HANDBOOK

How to Create World-Class Agility, Reliability, & Security in Technology Organizations

By Gene Kim, Jez Humble, Patrick Debois, and John Willis

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 foster a high-trust culture and a scientific approach to organizational improvement 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 DevOps,” shows an amazing progression of thinking and improbable connections. There are decades of lessons learned from manufacturing, high reliability organization, 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 major tenets include the deeply held belief that *manufacturing lead time* required to convert raw materials into finished goods was the best predictor of quality, customer satisfaction, and employee happiness, and that one of the best predictors of short lead times was 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 by seventeen of the leading thinkers in software development. They wanted to create a lightweight set of values and principles against heavyweight 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,” emphasizing the desire for small batch sizes, incremental releases instead of large, waterfall 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 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 Schafer held a “birds of a feather” session on applying Agile principles to infrastructure as opposed to application code. Although they were the only people who showed up, they rapidly gained a following of like-minded thinkers, including co-author John Willis.

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 first hand accounts, everyone attending the presentation immediately knew they were in the presence of something profound and of historic significance.

Patrick Debois was not there, but was so excited by Allspaw and Hammond’s idea that he created the first DevOpsDays in Ghent, Belgium, (where he lived) in 2009. There 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 in to 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*. 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 weekly target outcomes, and the continual improvement of daily work is what guided improvement at Toyota.

The above describes the history of DevOps and relevant movements that it draws upon. Throughout the rest of Part I, we 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).

1 Agile, Continuous Delivery, and the Three Ways

In this chapter, an introduction to the underpinning theory of Lean Manufacturing is presented, as well as the Three Ways, the principles from which all of 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 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.”

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 we don’t pass defects to downstream work centers, and constantly optimizing our system toward our global goals.

THE TECHNOLOGY VALUE STREAM

The same 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 that delivers value to the customer.

The input to our process is the formulation of a business objective, concept, idea, or hypothesis, and 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 in to 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, resulting in high variability of process times. In contrast, the second phase of work, which includes Testing and Operations, is akin to Lean Manufacturing. It requires creativity and expertise, and 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).

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 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

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 starts only when we begin work on the customer request—specifically, it omits the time that the work is in queue, waiting to be processed (figure 2).

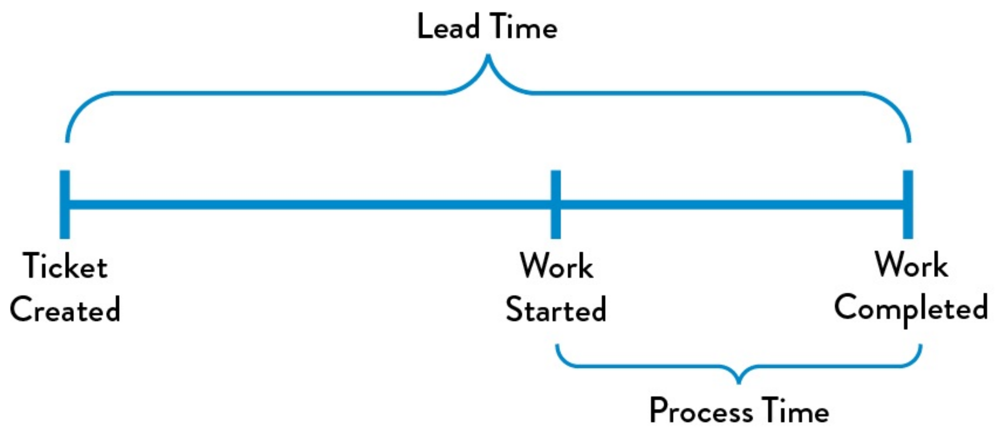

Figure 2. *Lead time vs. process time of a deployment operation*

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 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

In business as usual, we often find ourselves in situations where our deployment lead times require months. This is especially common in large, complex organizations that are working with tightly-coupled, monolithic applications, 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 3:

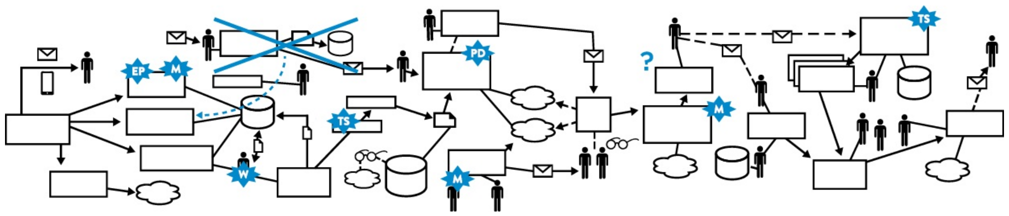

Figure 3: *A technology value stream with a deployment lead time of three months (Source: Damon Edwards, "DevOps Kaizen," 2015.)*

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, which enables 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 4:

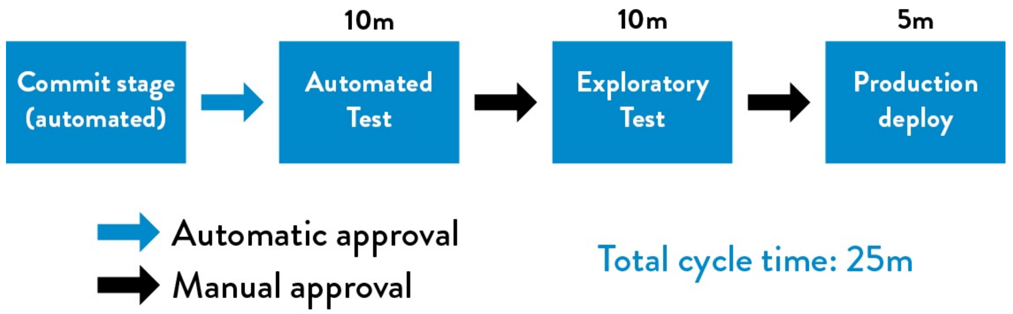

Figure 4: *A technology value stream with a lead time of minutes*

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."

THE THREE WAYS: THE PRINCIPLES UNDERPINNING DEVOPS

The Phoenix Project presents the Three Ways as the set of underpinning principles from which all the observed DevOps behaviors and patterns are derived (figure 5).

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 the global goals.

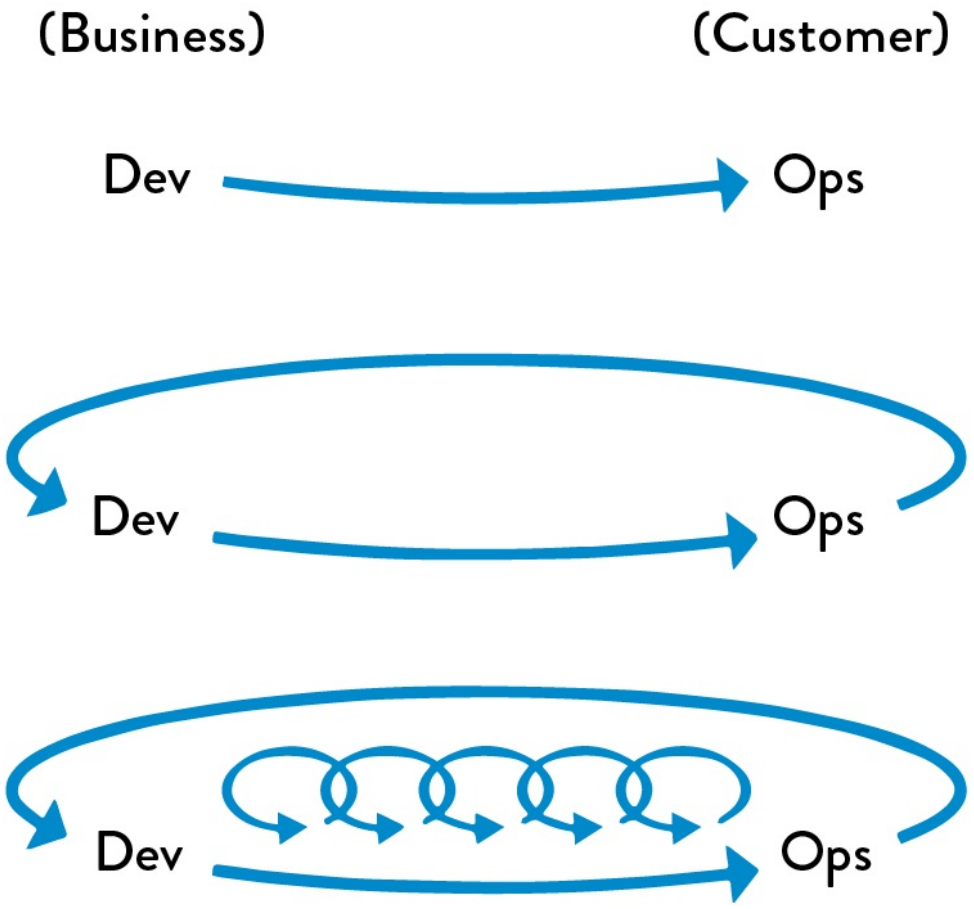

Figure 5: *The Three Ways (Source: Gene Kim, "The Three Ways: The Principles Underpinning DevOps," IT Revolution Press blog, accessed August 9, 2016, http://itrevolution.com/the-three-ways-principles-underpinning-devops/.)*

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 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 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.

CONCLUSION

In this chapter, we described the concepts of value streams, lead time as one of the key measures of the 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 is focused 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.

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

††† 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."