ADAPTED FROM *THE DEVOPS HANDBOOK*

How to Get Started with DevOps

GENE KIM, JEZ HUMBLE, PATRICK DEBOIS & JOHN WILLIS

How to Get Started with DevOps

Part 1: Selecting which Value Stream to Start With 3

Determing Whether to Start with a Greenfield or Brownfield Project 3

Systems of Record vs. Systems of Engagment 4

Expanding DevOps Across the Organization 5

Part 2: Understanding the Work in the Value Stream 8

Identify the Teams 8

Creating a Value Stream Map 9

DevOps Transformation Team 11

Part 3: How to Design with Conway's Law in Mind 17

Conway's Law 17

Organizational Archetypes 18

Developing the Right Habits and Capabilities in Your Team 21

Part 4: Integrating Ops into the Daily Work of Dev 25

Use Shared Services to Create Internal Marketplace 25

Create Self-Sufficient Teams by Embedding Ops into Dev 27

Proactive Integration of the DevOps Team 28

PART 1: SELECTING WHICH VALUE STREAM TO START WITH

Choosing the best value stream for your DevOps transformation deserves careful consideration. Not only does the value stream you choose dictate the difficulty of your transformation, but it also dictates who will be involved in the transformation, how you organize teams, and how you can best enable those teams and the individuals within them.

Select the best value stream for beginning your DevOps transformation by looking at three key components:

1

Whether they are a greenfield or brownfield service.

2

Whether they are systems of engagement or systems of record.

3

Reviewing best practices for expanding DevOps across your organization.

1 DETERMINING WHETHER TO START WITH A GREENFIELD OR BROWNFIELD PROJECT

In urban development, many factors can make greenfield projects simpler than brownfield—there are no existing structures that need to be demolished and there are no toxic materials that need to be removed.

In technology, a greenfield project is a new software project or initiative. Likely in the early stages of planning or implementation, greenfield projects are where applications and infrastructure are built anew and with few constraints. Starting with a greenfield software project can be easier, especially if the project is already funded and a team is either being created or is already in place.

Originally used for urban planning and building projects, greenfield development is when we build on undeveloped land. In software development, these are new projects or initiatives with few constraints

Brownfield projects often come with significant amounts of technical debt, such as having no test automation or running on unsupported platforms.

When transforming brownfield projects, teams may face significant impediments and problems, especially when no automated testing exists or when there is tightly coupled architecture that prevents small teams from developing, testing, and deploying code independently.

Although many believe that DevOps is primarily for greenfield projects, DevOps has been used to successfully transform brownfield projects of all sorts. In fact, over 60% of the transformation stories shared at the DevOps Enterprise Summit in 2014 were for brownfield projects.

On the other end of the spectrum are brownfield DevOps projects—these are existing products or services that are already serving customers and have potentially been in operation for years or even decades.

SYSTEMS OF RECORD VS. SYSTEMS OF ENGAGEMENT

The Gartner research firm has popularized the notion of bimodal IT, referring to the wide spectrum of services that typical enterprises support. Within bimodal IT there are systems of record and systems of engagement.

While it may be convenient to divide up our systems into these categories; we know that the core, chronic conflict between "doing it right" and "doing it fast" can be broken with DevOps.

The data from the *State of DevOps Reports*—following the lessons of Lean manufacturing—shows that high-performing organizations are able to simultaneously deliver higher levels of throughput and reliability. Furthermore, because of how interdependent our systems are, our ability to make changes to any of these systems is limited by the system that is most difficult to safely change, which is almost always a system of record.

Systems of record typically have a slower pace of change and often have regulatory and compliance requirements (e.g., SOX). Gartner calls these types of systems "Type 1," where the organization focuses on "doing it right."

Example: Think ERP-like systems that run our business (e.g., MRP, HR, financial reporting systems), where the correctness of the transactions and data are paramount.

Systems of engagement typically have a much higher pace of change to support rapid feedback loops that enable them to conduct experimentation to discover how to best meet customer needs. Gartner calls these types of systems "Type 2," where the organization focuses on "doing it fast."

Example: Think customer-facing or employee-facing systems, such as e-commerce systems and productivity applications.

Scott Prugh, Chief Technology Officer at CSG, observed:

> *We've adopted a philosophy that rejects bi-modal IT, because every one of our customers deserves speed and quality. This means that we need technical excellence, whether the team is supporting a 30-year-old mainframe application, a Java application, or a mobile application.*

Consequently, when we improve brownfield systems, we should not only strive to reduce their complexity and improve their reliability and stability, but we should also make them faster, safer, and easier to change.

Even when new functionality is added just to greenfield systems of engagement, they often cause reliability problems in the brownfield systems of record they rely on. By making these downstream systems safer to change, we help the entire organization achieve its goals more quickly and safely.

3 EXPANDING DEVOPS ACROSS OUR ORGANIZATION

Within every organization, there will be teams and individuals with a wide range of attitudes toward the adoption of new ideas, while others with more conservative attitudes resist them (the early adopters vs. the late majority and laggards).

Your goal should be to find those teams that already believe in the need for DevOps principles and practices, and who possess a desire and demonstrated ability to innovate and improve their own processes. Ideally, these groups will be enthusiastic supporters of the DevOps journey.

Especially in the early stages, do not spend much time trying to convert the more conservative groups. Instead, focus your energy on creating successes with less risk-averse groups and build out your base from there.

Even if you have the highest levels of executive sponsorship, it's best to avoid the big bang approach (i.e., starting everywhere all at once). Instead, focus your efforts in a few areas of the organization, ensuring that those initiatives are successful, and thenexpand from there.

It is also important to follow a safe sequence that methodically grows your levels of credibility, influence, and support. The following list, adapted from a course taught by Dr. Roberto Fernandez, a William F. Pounds Professor in Management at MIT, describes the ideal phases used by change agents to build and expand their coalition and base of support:

1: FIND INNOVATORS AND EARLY ADOPTERS

In the beginning, focus your efforts on teams who actually want to help—these are your kindred spirits and fellow travelers who are the first to volunteer to start the DevOps journey. In the ideal, these are also people who are respected and have a high degree of influence over the rest of the organization, giving your initiative more credibility.

Especially in the early stages, you will not spend much time trying to convert the more conservative groups. Instead, focus your energy on creating successes with less risk-averse groups and build out your base from there.

2: BUILD CRITICAL MASS AND SILENT MAJORITY

In the next phase, seek to expand DevOps practices to more teams and value streams with the goal of creating a stable base of support. By working with teams who are receptive to your ideas, even if they are not the most visible or influential groups, you create a coalition of teams generating more successes, creating a "bandwagon effect" that further increases your influence. It's important to specifically bypass dangerous political battles that could jeopardize your initiative.

You must demonstrate early wins and broadcast your successes. You can do this by breaking up your larger improvement goals into small, incremental steps. This not only creates improvements faster, it also enables you to discover when you have made the wrong choice of value stream—by detecting errors early, you can quickly back up and try again, making different decisions armed with new learnings.

3: IDENTIFY THE HOLDOUTS

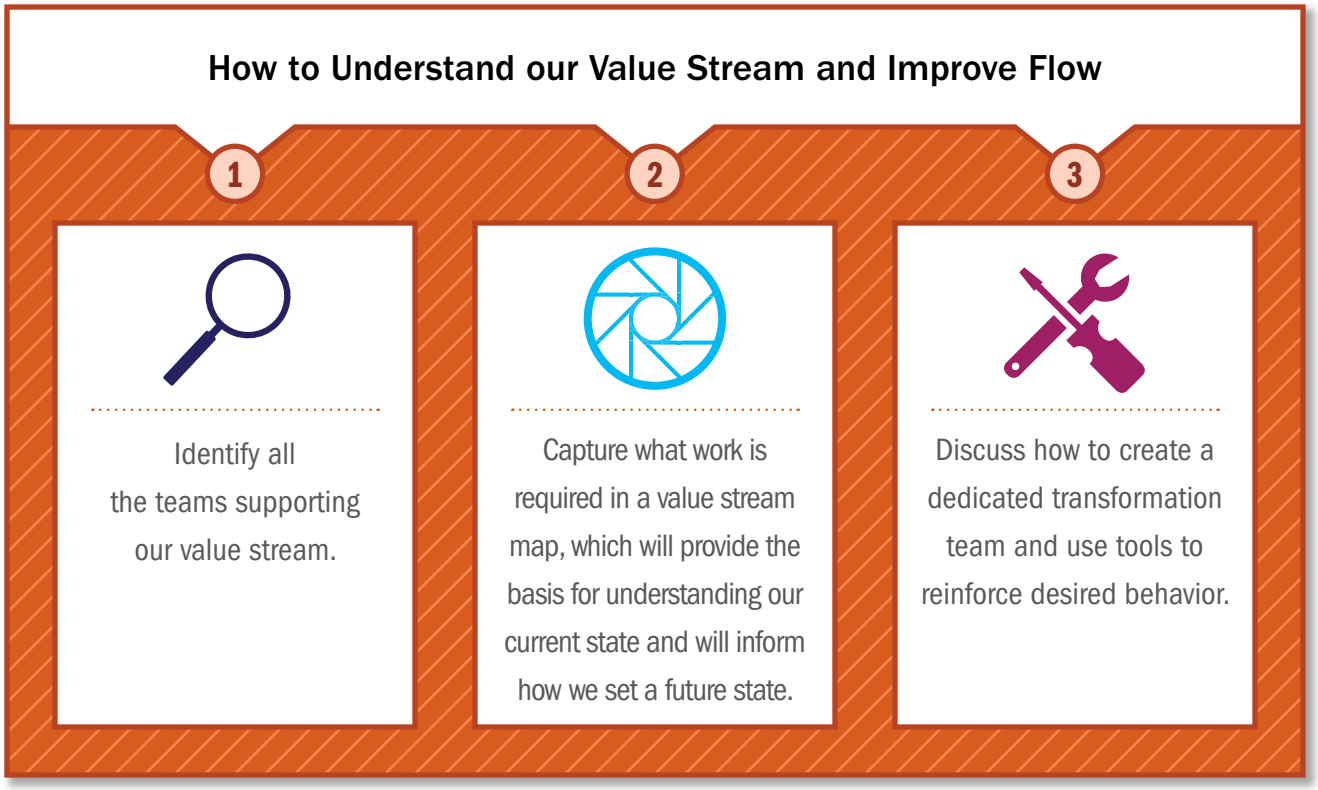
The "holdouts" are the high profile, influential detractors who are most likely to resist (and maybe even sabotage) your efforts. In general, tackle this group only after you have achieved a silent majority, when you have established enough successes to successfully protect your initiative.

Ultimately, expanding DevOps across an organization is no small task. It can create risk to individuals, departments, and the organization as a whole. But as Ron van Kemenade, CIO of ING, who helped transform the organization into one of the most admired technology organizations, said:

> *By choosing carefully where and how to start, we are able to experiment and learn in areas of our organization that create value without jeopardizing the rest of the organization. By doing this, we build our base of support, earn the right to expand the use of DevOps in our organization, and gain the recognition and gratitude of an ever-larger constituency.*

PART 2: UNDERSTANDING THE WORK IN OUR VALUE STREAM AND IMPROVING FLOW

The next step in our DevOps transformation is to gain a sufficient understanding of how value is delivered to the customer, by evaluating what work is performed, by whom, and what steps we can take to improve flow.



1 IDENTIFY THE TEAMS

In value streams of any complexity, no one person knows all the work that must be performed in order to create value for the customer—especially since the required work must be performed by many different teams, often far removed from each other on the organization charts, geographically, or by incentives.

As a result, after we select a service for our DevOps initiative, we must identify all the members of the value stream who are responsible for working together to create value for the customers being served. In general, this includes:

Product Owner	Development	QA
The internal voice of the business that defines the next set of functionality in the service.	The team responsible for developing application functionality in the service.	The team responsible for ensuring feedback loops exist to ensure the service functions as desired.

Operations	Infosec	Release Managers
The team responsible for maintaining the production environment and ensuring required service levels are met.	The team responsible for securing systems and data.	The people responsible for managing and coordinating the production deployment and release processes.

Technology Executives or Value Stream Manager
In Lean literature, someone who is responsible for "ensuring that the value stream meets or exceeds the customer [and organizational] requirements for the overall value stream, from start to finish"

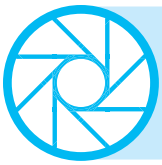
Once we identify our value stream members, our next step is to gain a concrete understanding of how work is performed, documented in the form of a value stream map.

2 CREATING A VALUE STREAM MAP

In our value stream, work likely begins with the product owner, in the form of a customer request or the formulation of a business hypothesis. Some time later, this work is accepted by Development, where features are implemented in code and checked into our version control repository. Builds are then integrated, tested in a production-like environment, and finally deployed into production, where they (ideally) create value for our customer.

In many traditional organizations, this value stream will consist of hundreds, if not thousands, of steps, requiring work from hundreds of people.

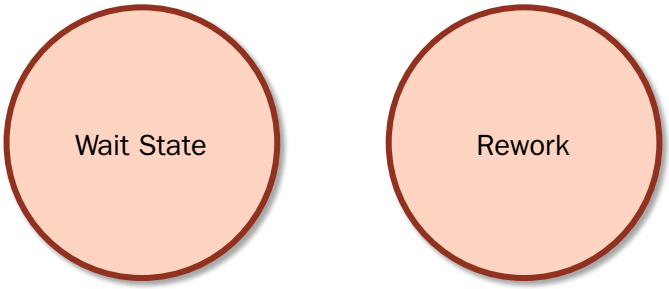
The goal here is not to document every step and associated minutiae, but to sufficiently understand the areas in the value stream that are jeopardizing the goals of fast flow, short lead times, and reliable customer outcomes. Ideally, you will have assembled those people with the authority to change their portion of the value stream.

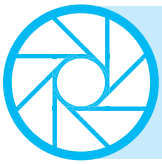


Because documenting any value stream map likely requires multiple days, you may conduct a multi-day workshop, where you assemble all the key constituents and remove them from the distractions of their daily work.

Using the full breadth of knowledge brought by the teams engaged in the value stream, you should focus your investigation and scrutiny on either:

- places where work must wait weeks or even months, such as getting production-like environments, change aproval processes, or security review processes
- places where significant rework is generated





Your first pass at documenting your value stream should only consist of high-level process blocks.

Typically, even for complex value streams, groups can create a diagram with five to fifteen process blocks within a few hours. Each process block should include the lead time and process time for a work item to be processed, as well as the %C/A (percent complete and accurate) as measured by the downstream consumers of the output.

Once you identify the metric you want to improve, you should perform the next level of observations and measurments to better understand the problem.

Next, you will need to construct an idealized, future value stream map, which serves as a target condition to achieve by some date (e.g., usually three to twelve months).

Leadership helps define this future state and then guides and enables the team to brainstorm hypotheses and countermeasures to achieve the desired improvement to that state, perform experiments to test those hypotheses, and interpret the results to determine whether the hypotheses were correct. The teams keep repeating and iterating, using any new learnings to inform the next experiments.

With these metrics in place, the next step in our DevOps transformation will be to create a dedicated transformation team.

3 DEVOPS TRANSFORMATION TEAM

One of the inherent challenges with initiatives such as DevOps transformations is that they are inevitably in conflict with ongoing business operations.

Part of this is a natural outcome of how successful businesses evolve. An organization that has been successful for any extended period of time (years, decades, or even centuries) has created mechanisms to perpetuate the practices that made them successful, such as product development, order administration, and supply chain operations.

While this is good for preserving status quo, teams undergoing DevOps transformations often need to change how they work to adapt to changing conditions in the marketplace. Doing this requires disruption and innovation, which puts them at odds with groups who are currently responsible for daily operations and the internal bureaucracies, and who will almost always win.

So how do you move forward?

Based on the research of Dr. Vijay Govindarajan and Dr. Chris Trimble, both faculty members of Dartmouth College's Tuck School of Business, organizations need to create a

dedicated transformation team that is able to operate outside of the rest of the organization that is responsible for daily operations (which they call the "dedicated team" and "performance engine" respectively).

Create Accountability

First and foremost, hold this dedicated team accountable for achieving a clearly defined, measurable, system-level result (e.g., reduce the deployment lead time from "code committed into version control to successfully running in production" by 50%).

In order to execute such an initiative, try the following:

- Assign members of the dedicated team to be solely allocated to the DevOps transformation efforts (as opposed to "maintain all your current responsibilities but spend 20% of your time on this new DevOps thing."
- Select people who are generalists, i.e., have skills across a wide variety of domains.
- Select team members who have longstanding and mutually respectful relationships with the rest of the organization.
- Create a separate physical space for the dedicated team, if possible, to maximize communication flow within the team and to create some isolation from the rest of the organization.

If possible, you may also want to free the transformation team from many of the rules and policies that restrict the rest of the organization. After all, established processes are a form of institutional memory—the dedicated team needs to create new processes and learnings to generate desired outcomes, creating new institutional memory.

Creating a dedicated team is not only good for the team, but also good for the performance engine. By creating a separate team, you create the space for them to experiment with new practices, protecting the rest of the organization from the potential disruptions and distractions associated with it.

Establish Shared Goals

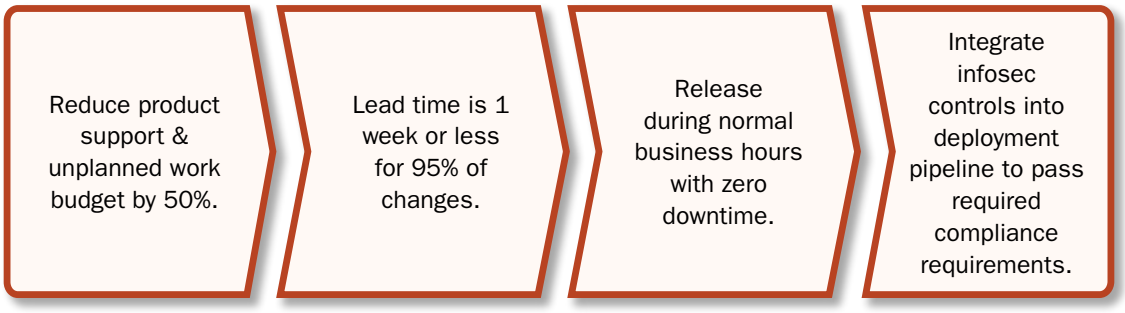
Next, you need to agree on a shared goal for the organization to move toward. One of the most important parts of any improvement initiative is to define a measurable goal with a

clearly defined deadline, usually between six months and two years in the future. It should require considerable effort but still be achievable. And, it should create obvious value for the organization as a whole and to your customers.

These goals and the time frame should be agreed upon by the executives and known to everyone in the organization.

You also need to limit the number of these types of initiatives that are going on simultaneously to prevent you from over taxing the organizational change management capacity of leaders and the organization.

Examples of improvement goals might include:



Once the high-level goal is made clear, teams should decide on a regular cadence to drive the improvement work. Like product development work, transformation work needs to be done in an iterative, incremental manner.

A typical iteration will be in the range of two to four weeks. For each iteration, teams should agree on a small set of goals that generate value and make some progress toward the long-term goal.

At the end of each iteration, teams should review their progress and set new goals for the next iteration.

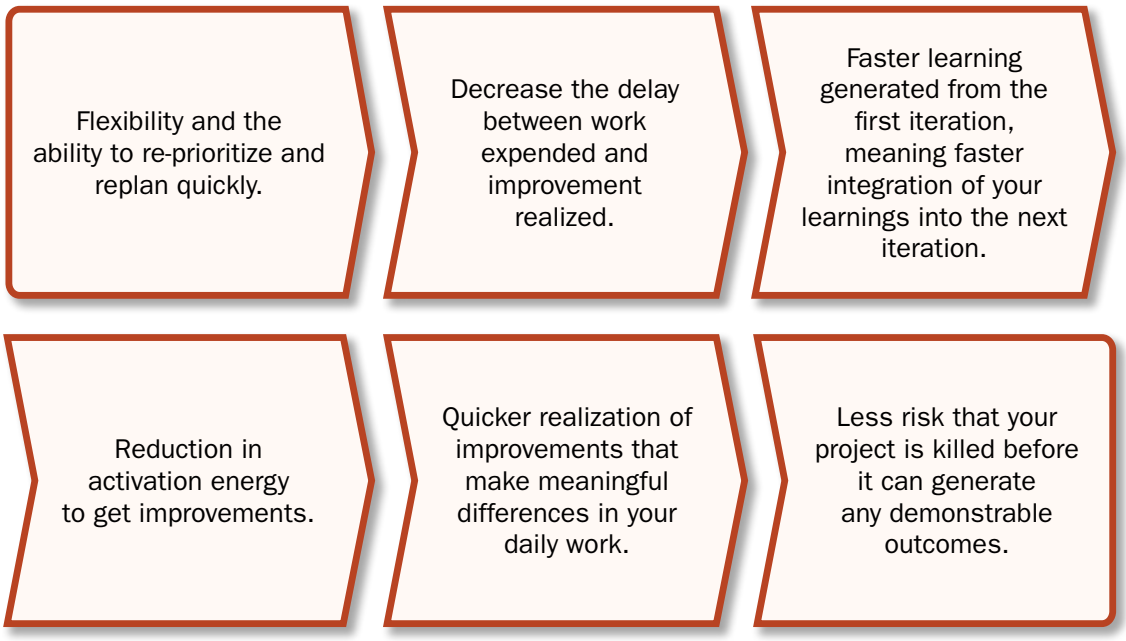
Make Current State of Work Visible

Lastly, in order to be able to know if you are making progress toward your goal, it's essential that everyone in the organization knows the current state of work.

There are many ways to make the current state of work visible, but what's most important is that the information displayed is up to date and that you constantly revise what you measure to make sure it's helping you understand progress toward your current target conditions.

With your goal in place, the next step will be for the organization to keep their improvement planning horizons short, just as if you were in a startup doing product or customer development. Your initiative should strive to generate measurable improvements or actionable data within weeks (or in the worst case, months).

By keeping your planning horizons and iteration intervals short, you achieve the following:



Reserve 20% Time for Reduction of Tech Debt

The final responsibility of the dedicated transformation team is to reserve 20% of cycles for non-functional requirements and reducing technical debt.

A problem common to any process improvement effort is how to properly prioritize it. After all, organizations that need it most are those that have the least amount of time to spend on improvement. This is especially true in technology organizations because of technical debt.

Organizations that struggle with financial debt only make interest payments and never reduce the loan principal, and may eventually find themselves in situations where they can no longer service the interest payments. In other words, they are now only making the interest payment on their technical debt.

You will need to actively manage this technical debt by ensuring that you invest at least 20% of all Development and Operations cycles on refactoring, investing in automation work and architecture, and non-functional requirements (NFRs, sometimes referred to as the "ilities"), such as maintainability, manageability, scalability, reliability, testability, deployability, and security.

Organizations that don't pay down technical debt can find themselves so burdened with daily workarounds for problems left unfixed that they can no longer complete any new work.

By dedicating 20% of your cycles so that Dev and Ops can create lasting countermeasures to the problems you encounter in your daily work, you ensure that technical debt doesn't impede your ability to quickly and safely develop and operate your services in production. Alleviating added pressure of technical debt from workers can also reduce levels of burnout.

Reinforce Behavior with Tools

With your dedicated team in place, you can use tools to reinforce desired behavior. As Christopher Little, a software executive and one of the earliest chroniclers of DevOps, observed,

> *Anthropologists describe tools as a cultural artifact. Any discussion of culture after the invention of fire must also be about tools.*

Similarly, in the DevOps value stream, teams use tools to reinforce their culture and accelerate desired behavior changes. By doing this, Development and Operations may end up creating a shared work queue, instead of each silo using a different one (e.g., Development uses JIRA while Operations uses ServiceNow).

One goal is that tooling reinforces that Development and Operations not only have shared goals, but also have a common backlog of work, ideally stored in a common work system and using a shared vocabulary, so that work can be prioritized globally.

A significant benefit of this is that when production incidents are shown in the same work systems as development work, it will be obvious when ongoing incidents should halt other work, especially when teams have a kanban board.

Another benefit of having Development and Operations using a shared tool is a unified backlog, where everyone prioritizes improvement projects from a global perspective, selecting work that has the highest value to the organization or most reduces technical debt.

As you identify technical debt, add it to your prioritized backlog if you can't address it immediately. For issues that remain unaddressed, use your "20% time for non-functional requirements" to fix the top items from your backlog.

An amazing dynamic is created when we have a mechanism that allows any team member to quickly help other team members, or even people outside their team—the time required to get information or needed work can go from days to minutes.

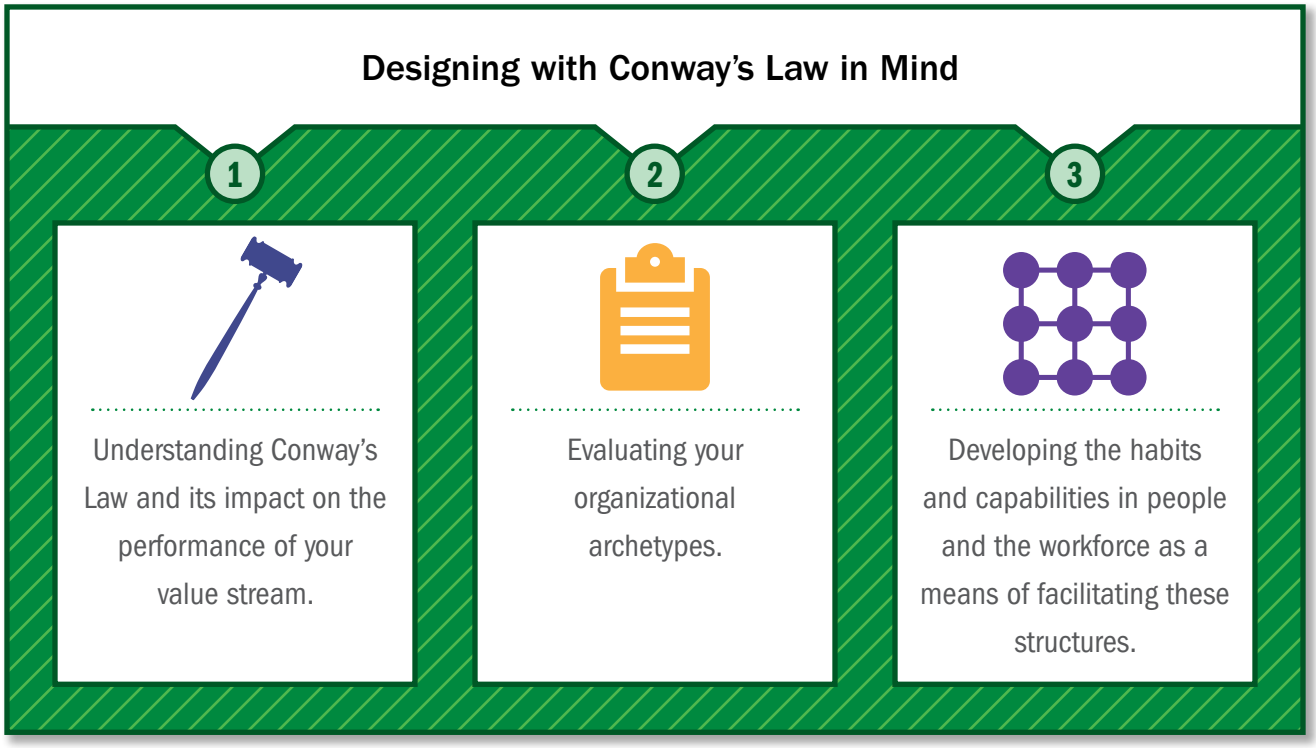
In addition, because everything is being recorded, you may not need to ask someone else for help in the future—you will simply search for it.

With these practices in place, you can enable dedicated transformation teams to rapidly iterate and experiment to improve performance. You can also make sure that you allocate a sufficient amount of time for improvement, fixing known problems and architectural issues, including non-functional requirements.

PART 3: HOW TO DESIGN WITH CONWAY'S LAW IN MIND

Over the past two sections, you have learned the necessary steps to start your DevOps transformation, including the three key components to consider in choosing a starting place and how value is delivered to the customer and how to improve flow.

Next it's time to discuss how and why to design with Conway's law in mind.



1 CONWAY'S LAW

Conway's Law has a tremendous impact on the performance of your value stream. Let's illustrate this with a story—in 1968, Dr. Conway was performing a famous experiment. Together, with a contract research organization of eight people, Conway and his team were commissioned to produce a COBOL and an ALGOL compiler.

During the experiment, he observed:

> *After some initial estimates of difficulty and time, five people were assigned to the COBOL job and three to the ALGOL job. The resulting COBOL compiler ran in five phases, the ALGOL compiler ran in three.*

These observations led to what is now known as Conway's Law, which states:

Organizations that design systems are constrained to produce designs which are copies of the communication structures of these organizations. The larger an organization, the less flexibility it has and the more pronounced the phenomenon.

In other words, how you organize your teams has a powerful effect on the software you produce, as well as your resulting architectural and production outcomes.

In order to get fast flow of work from Development into Operations, with high quality and great customer outcomes, you must organize your teams so that Conway's Law works to your advantage.

This process begins by evaluating the organizational archetypes.

2 ORGANIZATIONAL ARCHETYPES

In the field of decision sciences, there are three primary types of organizational structures that inform how DevOps value streams are designed with Conway's Law in mind: functional, matrix, and market.

They are defined by Dr. Roberto Fernandez as follows:

> *Functional-oriented organizations optimize for expertise, division of labor, or reducing cost. These organizations centralize expertise, which helps enable career growth and skill development, and often have tall hierarchical organizational structures. This has been the prevailing method of organization for Operations, (i.e., serveradmins, network admins, database admins, and so forth are all organized into separate groups).*

Market-oriented organizations optimize for responding quickly to customer needs. These organizations tend to be flat, composed of multiple, cross-functional disciplines (e.g., marketing, engineering, etc.), which often lead to potential redundancies across the organization. This is how many prominent organizations adopting DevOps operate—in extreme examples, such as at Amazon or Netflix, each service team is simultaneously responsible for feature delivery and service support.

Matrix-oriented organizations attempt to combine functional and market orientation. However, as many who work in or manage matrix organizations observe, matrix organizations often result in complicated organizational structures, such as individual contributors reporting to two managers or more, and sometimes achieving neither of the goals of functional or market orientation.

In traditional IT Operations organizations, functional orientation is often used to organize teams by their specialties. However, there are several problems that can occur by overly functional orientation (Optimizing for Cost).

For example: when database administrators are placed in one group, the network administrators in another, the server administrators in a third, and so forth—one of the most visible consequences is long lead times. Especially for complex activities like large deployments where developers must open up tickets with multiple groups and coordinate work handoffs, resulting in work waiting in long queues at every step.

In addition to these long queues and long lead times, this situation results in poor handoffs, large amounts of re-work, quality issues, bottlenecks, and delays. This gridlock impedes the achievement of important organizational goals, which often far outweigh the desire to reduce costs.

Similarly, functional orientation can also be found with centralized QA and Infosec functions, which may have worked fine (or at least, well enough) when performing less frequent software releases. However, as we increase the number of Development teams and their deployment and release frequencies, most functionally oriented organizations will have difficulty keeping up and delivering satisfactory outcomes, especially when their work is being performed manually.

Therefore, reduce the effects of functional orientation ("optimizing for cost") and enable market orientation ("optimizing for speed"), i.e., have many small teams working safely and independently, quickly delivering value to the customer.

Taken to the extreme, market-oriented teams are responsible not only for feature development, but also for testing, securing, deploying, and supporting their service in production, from idea conception to retirement.

These teams are designed to be cross-functional and independent—able to design and run user experiments, build and deliver new features, deploy and run their service in production, and fix any defects without manual dependencies on other teams, thus enabling them to move faster.

This model has been adopted by Amazon and Netflix and is touted by Amazon as one of the primary reasons behind their ability to move fast even as they grow.

To achieve market orientation, don't perform a large, top-down reorganization, which often creates large amounts of disruption, fear, and paralysis. Instead, embed the functional engineers and skills (e.g., Ops, QA, Infosec) into each service team, or provide their capabilities to teams through automated self-service platforms that provide production-like environments, initiate automated tests, or perform deployments.

This enables each service team to independently deliver value to the customer without having to open tickets with other groups, such as IT Operations, QA, or Infosec.

However, having just recommended market-orientated teams, it is worth pointing out that it is possible to create effective, high-velocity organizations with functional orientation.

Cross-functional and market-oriented teams are one way to achieve fast flow and reliability, but they are not the only path. You can also achieve desired DevOps outcomes through functional orientation, as long as everyone in the value stream views customer and organizational outcomes as a shared goal, regardless of where they reside in the organization.

What these organizations have in common is a high-trust culture that enables all departments to work together effectively, where all work is transparently prioritized and there is sufficient slack in the system to allow high-priority work to be completed quickly.

DEVELOPING THE RIGHT HABITS AND CAPABILITIES IN YOUR TEAM

To be able to employ this correctly, testing, operations and security needs to be, first and foremost, everyone's job, every day. In high-performing organizations, everyone within the team shares a common goal—quality, availability, and security aren't the responsibility of individual departments, but are a part of everyone's job, every day.

This means that the most urgent problem of the day may be working on or deploying a customer feature or fixing a Severity 1 production incident.

Alternatively, the day may require reviewing a fellow engineer's change, applying emergency security patches to production servers, or making improvements so that fellow engineers are more productive.

Secondly, we need to enable every team member to be a generalist.

Avoid Siloization of Teams

In extreme cases of a functionally oriented Operations organization, there will be departments of specialists, such as network administrators, storage administrators, and so forth When departments over-specialize, it causes *siloization*, which means they end up operating more like "sovereign states," as Dr. Spear has been created as saying.

Any complex operational activity then requires multiple handoffs and queues between the different areas of the infrastructure, leading to longer lead times (e.g., because every network change must be made by someone in the networking department).

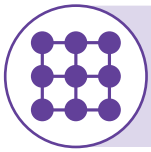
Because we rely upon an ever increasing number of technologies, engineers who have specialized and achieved mastery in the technology areas are needed. However, you don't want to create specialists who are "frozen in time," only understanding and able to contribute to that one area of the value stream.

One countermeasure is to enable and encourage every team member to be a generalist.

The term *full stack engineer* is now commonly used (sometimes as a rich source of parody) to describe generalists who are familiar—at least have a general level of understanding—with the entire application stack (e.g., application code, databases, operating systems, networking, cloud).

When people are valued merely for their existing skills or performance in their current role rather than for their ability to acquire and deploy new skills, organizations (often inadvertently) reinforce what Dr. Carol Dweck describes as the *fixed mindset*, where people view their intelligence and abilities as static "givens" that can't be changed in meaningful ways.

Instead, encourage learning and help people overcome learning anxiety, help ensure that people have relevant skills and a defined career road map, and so forth. By doing this, you help foster a *growth mindset* in your engineers. After all, a learning organization requires people who are willing to learn.



You can do this by providing opportunities for engineers to learn all the skills necessary to build and run the systems they are responsible for, and regularly rotating people through different roles.

How Funding and Team Size Affects Outcomes

One way to enable high-performing outcomes is to create stable service teams with ongoing funding to execute their own strategy and roadmap of initiatives. These teams have the dedicated engineers needed to deliver on concrete commitments made to internal and external customers, such as features, stories, and tasks.

Contrast this to the more traditional model where Development and Test teams are assigned to a "project" and then reassigned to another project as soon as the project is completed and funding runs out. This leads to all sorts of undesired outcomes, including developers being unable to see the long-term consequences of decisions they make (a form of feedback) and a funding model that only values and pays for the earliest stages of the software life cycle—which, tragically, is also the least expensive part for successful products or services.

The goal with a product-based funding model is to value the achievement of organizational and customer outcomes, such as revenue, customer lifetime value, or customer adoption rate, ideally with the minimum of output (e.g., amount of effort or time, lines of code).

Contrast this to how projects are typically measured, such as whether it was completed within the promised budget, time, and scope. You can improve deployment outcomes by creating loosely coupled architectures and designing team boudnaries to enable developer productivity and safety.

When architecture is tightly coupled, small changes can result in large scale failures. As a result, anyone working in one part of the system must constantly coordinate with anyone else working in another part of the system they may affect, including navigating complex and bureaucratic change management processes.

In contrast, having architecture that is loosely coupled means that services can update in production independently, without having to update other services.

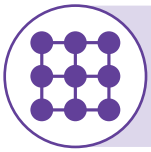
Randy Shoup, former Engineering Director for Google App Engine, observed that

> *organizations with these types of service-oriented architectures, such as Google and Amazon, have incredible flexibility and scalability. These organizations have tens of thousands of developers where small teams can still be incredibly productive.*

As organizations grow, one of the largest challenges is maintaining effective communication and coordination between people and teams.

All too often, when people and teams reside on a different floor, in a different building, or in a different time zone, creating and maintaining a shared understanding and mutual trust becomes more difficult, impeding effective collaboration. Collaboration is also impeded when the primary communication mechanisms are work tickets and change requests, or worse, when teams are separated by contractual boundaries, such as when work is performed by an outsourced team.

Conway's Law helps organizations design team boundaries in the context of desired communication patterns, but it also encourages orgs to keep team sizes small, reducing the amount of inter-team communication and encouraging you to keep the scope of each team's domain small and bounded.



As part of its transformation initiative away from a monolithic code base in 2002, Amazon used the two-pizza rule to keep team sizes small—a team only as large as can be fed with two pizzas—usually about five to ten people.

This limit on size has four important effects:

1. It ensures the team has a clear, shared understanding of the system they are working on. As teams get larger, the amount of communication required for everybody to know what's going on scales in a combinatorial fashion.
2. It limits the growth rate of the product or service being worked on. By limiting the size of the team, we limit the rate at which their system can evolve. This also helps to ensure the team maintains a shared understanding of the system.
3. It decentralizes power and enables autonomy. Each two-pizza team (2PT) is as autonomous as possible. The team's lead, working with the executive team, decides on the key business metric that the team is responsible for, known as the fitness function, which becomes the overall evaluation criteria for the team's experiments. The team is then able to act autonomously to maximize that metric.
4. Leading a 2PT is a way for employees to gain some leadership experience in an environment where failure does not have catastrophic consequences. An essential element of Amazon's strategy was the link between the organizational structure of a 2PT and the architectural approach of a service-oriented architecture.

In 2005, then Amazon CTO Werner Vogels explained the advantages of this structure to Larry Dignan of *Baseline*. Dignan writes:

> *Small teams are fast . . . and don't get bogged down in so-called administrivia . . . . Each group assigned to a particular business is completely responsible for it . . . . The team scopes the fix, designs it, builds it, implements it and monitors its ongoing use. This way, technology programmers and architects get direct feedback from the business people who use their code or applications—in regular meetings and informal conversations.*

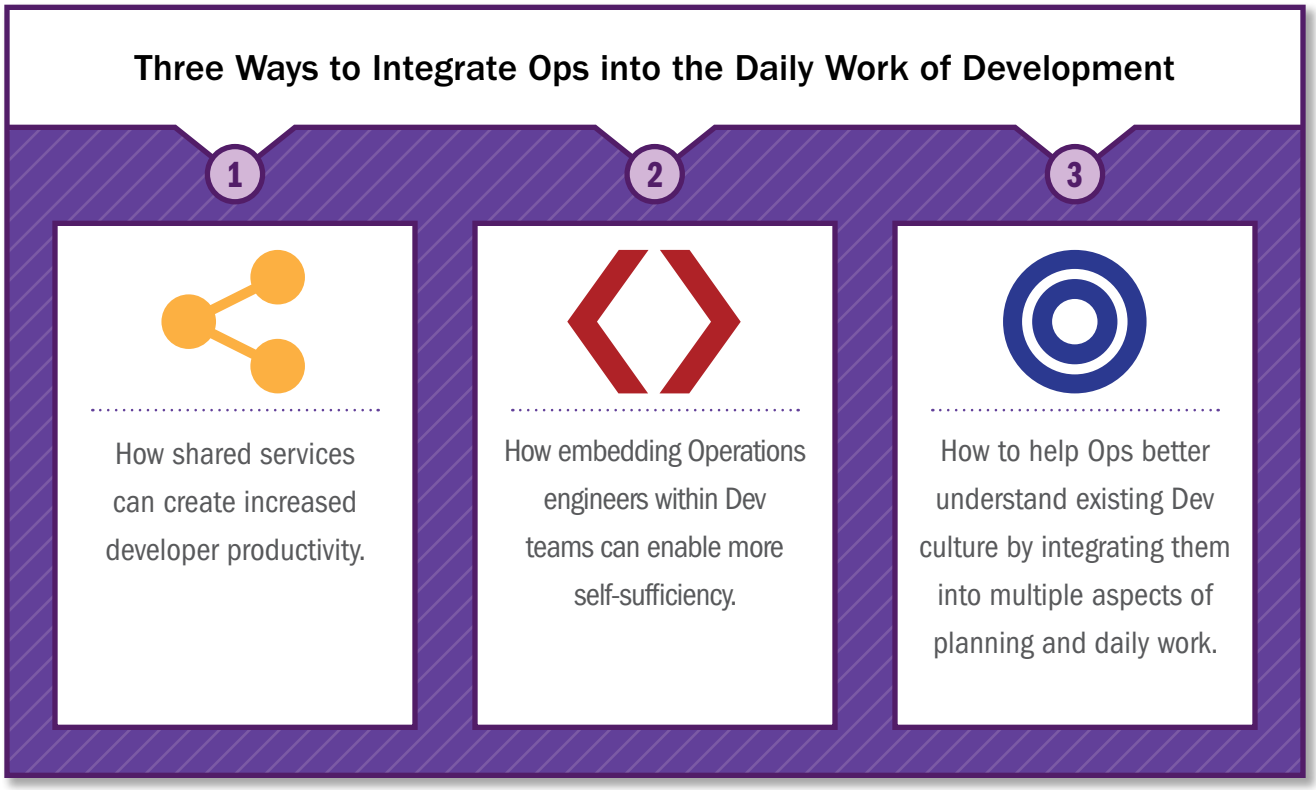
With these pieces in place, architecture and organizational design can dramatically improve outcomes. Done incorrectly, Conway's Law will ensure that the organization creates poor outcomes, preventing safety and agility.

Done well, the organization enables developers to safely and independently develop, test, and deploy value to the customer.

PART 4: INTEGRATING OPS INTO THE DAILY WORK OF DEV

Finally, in any DevOps transformation one goal will be to enable market-oriented outcomes where many small teams can quickly and independently deliver value to the customer.

When done correctly, Ops can significantly improve the productivity of Dev teams throughout the entire organization, as well as enable better collaboration and organizational outcomes.



1 USE SHARED SERVICES TO CREATE INTERNAL MARKETPLACE

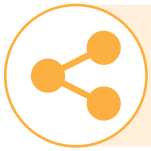
To begin, let's look at how shared services can create increased developer productivity. One way to enable market-oriented outcomes is for Operations to create a set of centralized platforms and tooling services that any Dev team can use to become more productive—such as

getting production-like environments, deployment pipelines, automated testing tools, and so forth.

By doing this, we enable Dev teams to spend more time building functionality for their customer, as opposed to obtaining all the infrastructure required to deliver and support that feature in production.

All the platforms and services we provide should (ideally) be automated and available on demand, without requiring a developer to open up a ticket or manually perform work. This also ensures that Operations doesn't become a bottleneck for their customers.

By creating this effective internal marketplace of capabilities, we help ensure that the platforms and services we create are the easiest and most appealing choice available (the path of least resistance).



Creating and maintaining these platforms and tools is real product development—the customers of our platform aren't our external customer but our internal Dev teams.

Like creating any great product, creating great platforms that everyone loves doesn't happen by accident. An internal platform team with poor customer focus will likely create tools that everyone will hate and quickly abandon for other alternatives, whether for another internal platform team or an external vendor.

Often, these platform teams provide other services to help their customers learn their technology, migrate off of other technologies, and even provide coaching and consulting to help elevate the state of the practice inside the organization.

These shared services also facilitate standardization, which enable engineers to quickly become productive, even if they switch between teams. For instance, if every product team chooses a different toolchain, engineers may have to learn an entirely new set of technologies to do their work, putting the team goals ahead of the global goals.

In organizations where teams can only use approved tools, start by removing this requirement for a few teams, such as the transformation team, so that they can experiment and discover what capabilities make those teams more productive.

Internal shared services teams should continually look for internal toolchains that are widely being adopted in the organization, deciding which ones make sense to be supported centrally and made available to everyone. In general, taking something that's already working somewhere and expanding its usage is far more likely to succeed than building these capabilities from scratch.

2 CREATE SELF-SUFFICIENT TEAMS BY EMBEDDING OPS INTO DEV

Enable product teams to become more self-sufficient by embedding Operations engineers within them. These product teams may also be completely responsible for service delivery and service support.

By embedding Ops engineers into the Dev teams, their priorities are driven almost entirely by the goals of the product teams they are embedded in—as opposed to Ops focusing inwardly on solving their own problems. As a result, Ops engineers become more closely connected to their internal and external customers.

Furthermore, the product teams often have the budget to fund the hiring of these Ops engineers, although interviewing and hiring decisions will likely still be done from the centralized Operations group, to ensure consistency and quality of staff.

For new large Development projects, you may want to initially embed Ops engineers into those teams. Their work may include helping decide what to build and how to build it, influencing the product architecture, helping influence internal and external technology choices, helping create new capabilities in internal platforms, and maybe even generating new operational capabilities. After the product is released to production, embedded Ops engineers may help with the production responsibilities of the Dev team.

They will take part in all of the Dev team rituals, such as planning meetings, daily standups, and demonstrations where the team shows off new features and decides which ones to ship. As the need for Ops knowledge and capabilities decreases, Ops engineers may transition to different projects or engagements, following the general pattern that the composition within product teams changes throughout its life cycle.

This paradigm has another important advantage: pairing Dev and Ops engineers together is an extremely efficient way to cross-train operations knowledge and expertise into a service team. It can also have the powerful benefit of transforming operations knowledge into automated code that can be far more reliable and widely reused.

When Not to Embed Ops into Dev

However, there may be a variety of reasons, such as cost and scarcity, where you may be unable to embed Ops engineers into every product team. In these situations, you can get many of the same benefits by assigning a designated liaison for each product team.

Just like in the embedded Ops model, this liaison attends the team standups, integrating their needs into the Operations road map and performing any needed tasks.

You can rely on these liaisons to escalate any resource contention or prioritization issue. By doing this, you will identify any resource or time conflicts that should be evaluated and prioritized in the context of wider organizational goals.

If you find that Ops liaisons are stretched too thin, preventing the product teams from achieving their goals, then you will likely need to either reduce the number of teams each liaison supports or temporarily embed an Ops engineer into specific teams.

3 PROACTIVE INTEGRATION OF THE DEVOPS TEAM

Once Ops engineers are embedded or assigned as liaisons into product teams, the goal is to help Ops engineers and other non-developers better understand the existing Development culture.

To proactively integrate them into all aspects of planning and daily work, you can:

- Have Ops attend the daily standup.
- Invite Ops to Dev retrospectives.
- Make Ops work visible on shared Kanban boards.

As a result, Operations is better able to plan and radiate any needed knowledge into the product teams, influencing work long before it gets into production.

Have Ops Attend the Daily Standup

One of the Dev rituals popularized by Scrum is the daily standup, a quick meeting where everyone on the team gets together and presents to each other three things: what was done yesterday, what is going to be done today, and what is preventing you from getting your work done.

The purpose of this ceremony is to radiate information throughout the team and to understand the work that is being done and is going to be done.

By having team members present this information to each other, they learn about any tasks that are experiencing roadblocks and discover ways to help each other move work toward completion. Furthermore, by having managers present, they can quickly resolve prioritization and resource conflicts.

A common problem is that this information is compartmentalized within the Development team. By having Ops engineers attend, Operations can gain an awareness of the Development team's activities, enabling better planning and preparation.

For instance, if you discover that the product team is planning a big feature rollout in two weeks, you can ensure that the right people and resources are available to support the rollout By doing this, you create the conditions where Operations can help solve your current team problems or future problems before they turn into a crisis.

Invite Ops to Dev Retrospectives

Another widespread ritual is the retrospective. At the end of each development interval, the team discusses what was successful, what could be improved, and how to incorporate the successes and improvements in future iterations or projects.

The team comes up with ideas to make things better and reviews experiments from the previous iteration. This is one of the primary mechanisms where organizational learning and the development of countermeasures occurs, with resulting work implemented immediately or added to the team's backlog.

Having Ops engineers attend project team retrospectives means they can also benefit from any new learnings. Furthermore, when there is a deployment or release in that interval, Operations should present the outcomes and any resulting learnings, creating feedback into the product team.

By doing this, you can improve how future work is planned and performed, improving your outcomes. Feedback from Operations also helps product teams better see and understand the downstream impact of decisions they make.

When there are negative outcomes, you can make the changes necessary to prevent them in the future. Operations feedback will also likely identify more problems and defects that should be fixed—it may even uncover larger architectural issues that need to be addressed.

Make Ops Work Visible on Shared Kanban Boards

Often, Development teams will make their work visible on a project board or kanban board. It's far less common, however, for work boards to show the relevant Operations work that must be performed in order for the application to run successfully in production, where customer value is actually created.

As a result, organizations and teams may not be aware of necessary Operations work until it becomes an urgent crisis, jeopardizing deadlines or creating a production outage.

Because Operations is part of the product value stream, put the Operations work that is relevant to product delivery on the shared kanban board. This enables us to more clearly see all the work required to move code into production, as wel keep track of all Ops work required to support the product. Furthermore, this practice allows everyone to see where Ops work is blocked and where work needs escalation to highlight areas that may need improvement.

When done well, this will achieve market-oriented outcomes, regardless of how the organizational charts have been drawn.

A CALL TO ACTION

At a time when every technologist and technology leader is challenged with enabling security, reliability, and agility, and at a time when security breaches, time to market, and massive technology transformation is taking place, DevOps offers a solution. Hopefully, this getting started guide has provided an introductory understanding of the problem and a beginning road map to creating relevant solutions.

Next, we invite you to dive deeper into the DevOps practices that can propel your organizatin into the future with *The DevOps Handbook Second Edition*.

The DevOps Handbook can help you create a dynamic learning organization that's able to achieve the amazing outcomes of fast flow and world-class reliability and security, as well as increased competitiveness and employee satisfaction.

DevOps is not just a technology imperative but an organizational imperative. The bottom line is, DevOps is applicable and relevant to any and all organizations that must increase flow of planned work through the technology organization, while maintaining quality, reliability, and security for our customers.

Our call to action is this: no matter what role you play in your organization, start finding people around you who want to change how work is performed. Show them this guide and *The DevOps Handbook* to create a coalition of like-minded thinkers. Ask organizational leaders to support these efforts or, better yet, sponsor and lead these efforts yourself.

DevOps benefits all of us in the technology value stream, whether we are Dev, Ops, QA, Infosec, Product Owners, or customers. It brings joy back to developing great products. It enables humane work conditions with fewer weekends worked and fewer missed holidays with our loved ones. It enables teams to work together to survive, learn, thrive, delight our customers, and help our organization succeed.

We sincerely hope this guide and *The DevOps Handbook* help you achieve these goals.

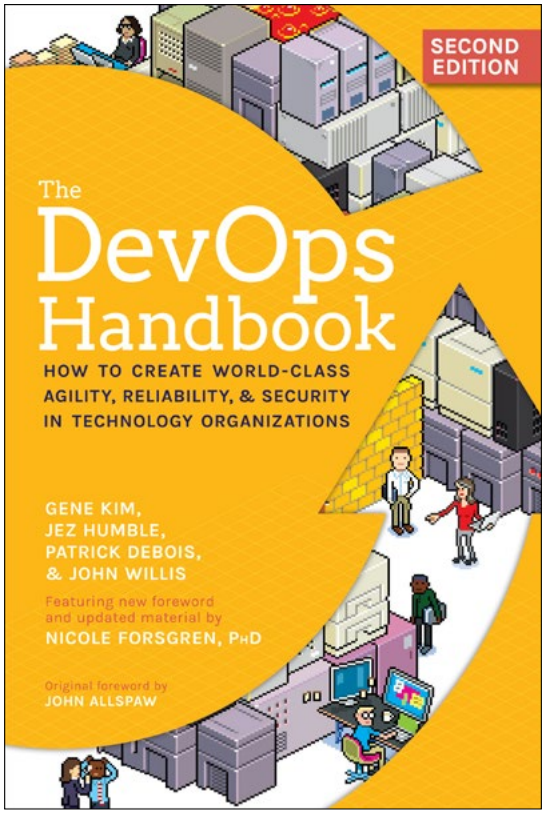
Continue your DevOps tranformation journey with

The DevOps Handbook, Second Edition: How to Create World-Class Agility, Reliability, & Security in Technology Organizations

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

Featuring new foreword and updated material by Nicole Forsgren, PhD

Original Foreword by John Allspaw



Available for purchase at all major book retailers.

Learn more at itrevolution.com/the-devops-handbook