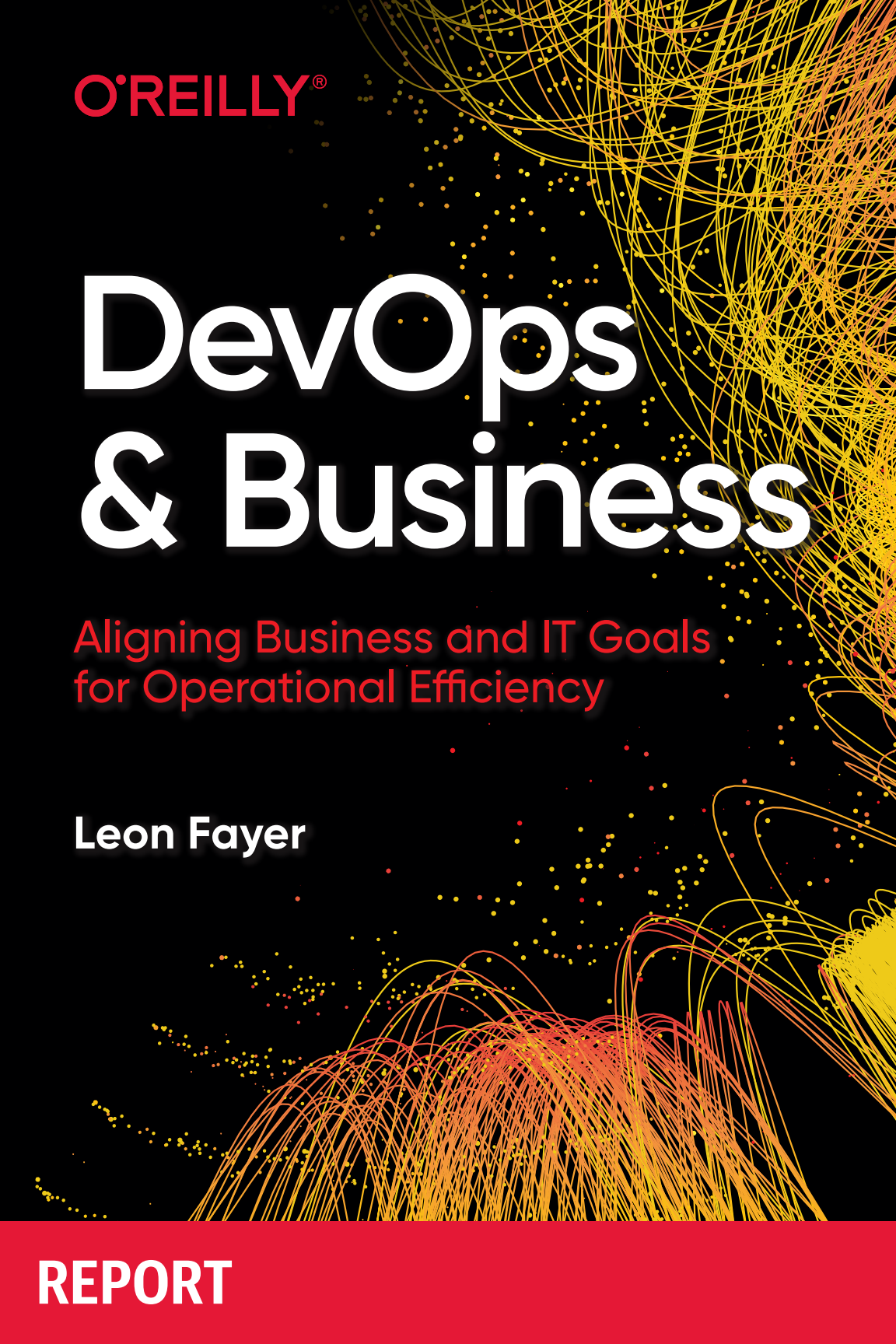
O'REILLY®
DevOps & Business
Aligning Business and IT Goals for Operational Efficiency
Leon Fayer
REPORT

DevOps and Business

Aligning Business and IT Goals for Operational Efficiency

Leon Fayer

Beijing • Boston • Farnham • Sebastopol • Tokyo O'REILLY®

DevOps and Business
by Leon Fayer

Printed in the United States of America.

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

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

Acquisitions Editor: Nikki McDonald
Developmental Editor: Alicia Young
Production Editor: Kristen Brown
Copyeditor: Octal Publishing, LLC

Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

July 2019: First Edition

Revision History for the First Edition
2019-07-24: First Release

978-1-492-03659-3

[LSI]

Table of Contents

CHAPTER 1
Introduction

Every organization has objectives that define its own success, but rarely do technical objectives such as deployment frequency, technology choices, or performance metrics make it to the top of an organization's Key Performance Indicators (KPIs). And although those IT metrics are vital to a company's success, they do not mean much to nontechnical people when presented in isolation.

Here's an example. Imagine that a developer in a large enterprise organization that provides multiple web-based platforms to its users has found a way to improve load time across all properties by 200 milliseconds. She sees an opportunity to deliver value, so she writes up a plan for the development, QA, and deployment effort required, and conservatively estimates that the effort will take two weeks. She presents that plan to her manager, hoping it will make it into the next sprint, only to see her plan rejected. The reason? The manager sees it as a disproportionately large investment for minimal gain. Even though the task is added to backlog, it is given the lowest priority, which means it has little chance of ever actually being implemented.

In this example, the core issue is that load time is a technical metric that means very little to nontechnical people without context. The manager did not understand—and the developer did not communicate—the business impact of performance improvement; without additional context, deprioritizing the effort based on return on investment seems like a logical decision. The developer could have made a much stronger case with business groups if she had

positioned the improvement in terms of business value and told the manager that her plan would lead to a 7% increase to conversion rate and, subsequently, revenue, based on the industry average. This approach would have resonated much better with management and executives, because it would correlate to KPIs familiar to them.

As an IT professional, you need to understand how to align your objectives to business goals to ensure that your decisions help to move the business forward. To do that, you need to understand those business goals and how to align your priorities with them. Getting the IT and business groups to adopt DevOps principles can help you accomplish that goal.

DevOps and Business

DevOps started as a cultural shift aimed at streamlining intergroup communication and improving operational efficiency for—as the name suggests—development and operations groups. Its principles emerged from a need to quickly respond to market and business changes, and over time, organizations began to apply the same principles to other groups under the IT umbrella. Indeed, most organizations starting on a digital transformation journey still tend to confine DevOps transformation to the IT realm, concentrating on automation and monitoring of production environments and streamlining development practices. But adopting these principles in isolation will cause business goals and IT goals to become misaligned, which can cost a large amount of productivity and revenue. The most successful companies blur the traditional line between business and technology domains, collaborating to accomplish joint goals to better the organization as a whole.

In this report, you learn about core DevOps principles and how to bring together the business and technology domains, collaborating to accomplish joint goals and better your organization as a whole. I provide you with best practices for how to understand, align, and support business objectives, how to set up feedback loops to improve intergroup communication, and how to improve insight into technical and business operations to ensure organizational success. I also provide a framework for helping IT professionals to better understand what drives their business, and present processes for aligning IT initiatives with business goals to increase efficiency and improve the bottom line.

CHAPTER 2
Setting the Stage

To create a successful DevOps culture, you need support and commitment from both IT and business executives, which can be a challenge to obtain. Many managers, technologists, and business leaders are still not sure what DevOps is, much less why they need it, and the abundance of materials from vendors trying to capitalize on DevOps' status as something of a buzzword does not help the confusion. As such, education on the benefits of adoption of DevOps core values must be a high priority for anyone looking to lead a digital transformation at their company.

Although you'll be faced with many challenges in this process, the major ones you're likely to encounter can be boiled down to two generic points: uncertainty about what DevOps is and uncertainty about how it will benefit the organization. In this chapter, we discuss these challenges, and I provide you with practical strategies for educating stakeholders on how DevOps can optimize business.

Defining DevOps

The term *DevOps* has many different perceived meanings, which often leads to misunderstanding about what it actually is. When the term was first introduced, it was intended to define a relationship between development and operations groups. Over time, the definition has been expanded; DevOps is not just about interdepartment communication; rather, it's about building the culture and providing value to an organization. DevOps represents an organizational culture aimed at delivering *business value* quickly.

For the purpose of this discussion, we use a specific set of DevOps values as the framework for defining the term: CAMS, an acronym defined by Damon Edwards and John Willis. This acronym identifies the major areas of potential improvement for organizations: *culture*, *automation*, *measurement*, and *sharing*. Over the next few chapters we dive into each of these four concepts in order to understand their importance in the context of DevOps and discuss organizational value gains that you can achieve by adopting these ideas across the entire company.

Educating Stakeholders

The principles of CAMS and the impact of each component can sometimes be tricky for stakeholders to understand. For example, although *automation* is a commonly understood term with a measurable value, the implications of culture changes are much more difficult not only to quantify, but also to explain to the uninitiated.

I've worked with many organizations, large and small, that asked for assistance with their digital transformation. After initial review of their value chain, I would outline initiatives that would provide the highest value from the most economical investment standpoint. Sometimes, those initiatives centered around operational and development improvement, but most of the time, they were focused on improving the efficiency of existing processes. About 70% of the time, the response I got from business leaders was: "Oh, we can't change any processes. We were looking to buy and integrate some automation tools. And maybe look into putting our developers on call. But that's it."

Of course, I am paraphrasing, but the point stands. Many companies and decision makers don't fully understand what DevOps brings to the organization, and because of that, they don't understand where the changes need to be made to truly make a positive impact. They are looking for a silver-bullet solution or product that doesn't exist.

As such, when bringing the concept of DevOps to organizations where decision makers are not sure of what DevOps really is or the value it provides, I tend to start from the opposite end of the information spectrum and educate on what DevOps is *not*. This approach helps to clear up some of the reductive ideas that many people have about DevOps.

DevOps Is Not a Product (or Suite of Products)

We're doing DevOps by using a configuration management tool.

DevOps is a philosophy. An approach. A method. It is not a product. You can't "buy" DevOps (or its accessories) just to say that you have it. No matter how many companies are trying to sell you a "complete DevOps solution," DevOps is an entire culture that must be adopted and embraced by the entire organization.

DevOps Is Not Interchangeable with Other Industry Buzzwords

By using Agile techniques for automation in the cloud, we've successfully achieved DevOps.

DevOps is not a synonym for other industry terms like "Agile," "cloud," or "automation." Even though certain elements of these practices and tools do intersect, you cannot equate them to one another.

DevOps Is Not a Separate Team

We've created a new DevOps team to make communication between our Dev and Ops more effective.

One of the goals of DevOps culture is to help eliminate siloed structure, improve communication between development and operational groups, and cross-educate those groups make them more efficient and effective. Creating a new siloed group does not contribute to any of those goals.

DevOps Is Not a Staff Consolidation Technique

We've given our developers pagers and made them responsible for operations.

DevOps is not a method to reduce staff and stack cross-department responsibilities on remaining people. It logically expands the area of responsibility for each group and defines the processes for all technology groups to work efficiently together. As example, it extends developers' responsibility for success of the application after deploying it to production, both reducing the load on operations team and providing a subject matter expertise in production troubleshooting.

DevOps Is Not New

The DevOps model is not mature yet. We'll give it a few more years.

Both engineers and organizations have been applying DevOps principles for years prior to when the term was coined. Companies that understand the need of cross-department collaboration successfully streamlined execution and increased the success rates in time-to-market and quality of products. The toolset that is most commonly associated with DevOps-related concepts are (arguably improved upon) derivatives of what were used well more than a decade ago.

DevOps Is Not Just About Development and Operations

We finished our reorg; now, IT consists of DevOps, QA, Security, Database, and Networking groups.

DevOps is, in a certain sense, a misnomer. Initially intended as a method to improve efficiency between development and operations group, the concept has logically expanded to cover other technology units, as well. And with the end goal being optimization of service delivery for organizations, the more successful companies have begun to span principles beyond technology groups, including marketing, finance, HR, and other nontechnical business units (a topic that we examine in more detail in the following chapters).

DevOps is about a culture; a set of processes and practices that optimize service delivery for organizations. The other often-discussed concepts, such as automation, tooling, and staff responsibility, should be tactical implementations for the overall digital transformation strategy.

Defining Your DevOps Goals

After you've established a baseline definition of DevOps for teams at your organization, you need to make sure that both technical and business units agree upon and understand the overall strategy and goals of a DevOps transformation and are comfortable with the changes required to put DevOps principles into practice. To accomplish this, you can use the CAMS principles as a blueprint to assess and outline areas of impact and potential improvement for your organization (we take a look at all of these principles in greater depth in later chapters):

Culture
: Remember that you are not picking tools or implementing an isolated processes. DevOps is about a culture shift, to promote collaboration and empathy both within and across diverse business units.

Automation
: Focus on removing manual steps from your value chain (where appropriate). Keep an eye towards the end goal of improving the time from inception to delivery to customer.

Measurement
: Much like performance and security, threat measurement should not be merely a feature or an afterthought, but a core requirement for each component in your delivery model. Keep in mind that you can't improve what you can't measure.

Sharing
: Creating robust feedback loops is key to a successful continuous improvement model. And being transparent with information made available (success and failure alike) will contribute to culture of collaboration and empathy.

After the key areas are outlined, it's time to focus on the effects the changes would have for individual groups and people.

Understanding the Impact of Change

Change is crucial to long-term organizational success, yet it is often met with resistance by individuals. Despite being vital to success, change can easily be perceived as having negative connotations and consequences, such as increasing individuals' workloads, creating potential failures, compromising product quality, and reducing the size of the staff. Overcoming resistance to change is especially difficult when working with organizations and individuals who don't believe that they have an immediate, acute problem and those who perceive change as a threat to their current role.

There are a number of common reasons for resistance to change that tend to fall into three main categories. You should address each of these reasons as part of a buy-in acquisition from all groups.

Complacency

Complacency is often a problem in organizations and individuals with a history of success without recent reliance on innovation—"If it ain't broke, don't fix it," as the saying goes. As one of the more egregious examples, many traditional retail businesses either passed on or procrastinated on a strong push from many organizational leaders to shift the priorities to online venues. On an individual level, not keeping current with your industry can make your skills quickly outdated and, as a byproduct, have a negative impact on any solutions you propose based on then-current knowledge. Remember, continuity when you aren't going anywhere is stagnation.

Personal Impact

Fear for job stability is perhaps the most common reason that contributes to resistance to change on an individual level. Some people are worried about stepping out of their comfort zone, either technically or socially, or hesitant to change their routines. Others feel that they won't be able to make the transition very well, either because the change may force them out of the job if they don't pick up new skills, or due to automating themselves out of a job—a frequent concern during a DevOps transformation.

Lack of Trust

Trust (which I expand on in Chapter 7) is a key factor in getting a cross-departmental buy-in for change. If management doesn't believe IT can successfully manage change, they will be skeptical of any innovations that require their buy-in. Similarly, if organizations continuously reject grassroots innovation, it disincentivizes engineers from providing value through change. Not trusting that change will better your situation creates strong resistance.

Helping Teams to Embrace Change

The previous three categories, at their core, essentially boil down to an aversion to change due to risk and money factors. Though those concerns can be valid, they are often driven by a fear of the unknown, which is amplified by the lack of understanding of reasons for change. And a lack of understanding stems from a lack of clear communication.

As such, when preparing a compelling case to get a buy-in, there are a number of strategies that you can use to communicate clearly and directly with your stakeholders across teams to help them embrace change. Let's take a look at some of them.

Address Personal Concerns

Consider each audience. When you're presenting to executives, you should focus on the impact of change to improve organizational goals, but when you're trying to get a buy-in from the people in your group, you need to consider their individual concerns. When presented with a change to their routine, people's first reaction is usually, "What does it mean for me?" Address that question first.

- Discuss how individual job descriptions will change.
- Outline a plan of action to train people if needed.
- Talk through group structure changes.
- Perhaps most important, address any potential personnel changes head on.

Be Transparent

Explicitly tell people what's going on. Given incomplete information, people will speculate and make assumptions, forming a resistance to change before the change is even communicated or understood. Even if you don't possess all the details, update the people involved and acknowledge that information will be made available as soon as it's available.

Involve Affected Groups

Involve the people in conversations who will be affected by the proposed change about the purpose and effects of change, and involve them early and often. Get them excited about the change by focusing on how the changes will improve the business, which will benefit everyone at the organization. It's less likely for individuals to oppose to change when they are part of the decision, let alone if they feel there is a personal connection or benefit associated with it.

NOTE

Of course, as with anything dealing with human-factor impact, this is not always easily scriptable. One of the major resistance factors is a potential shift in balance of organizational power. When pushing a change that has rippling effects across the organization, loss of power and control by management and individual groups can be a real deterrent to initiative. A threat to status, position, or autonomy (even if it's only a perceived threat) can be a major source of pushback. Although applying the above tactics should diffuse a lot of power balance issues, major organizational changes often require restructuring, which can shift power and control for both individuals and groups, making fear of change justifiable. Understanding and addressing roles and areas of potential contention early can help to diffuse or eliminate a source of resistance.

After you have secured buy-in for the overall strategy, you can begin focusing on defining individual implementation tactics with an eye out for the strategic outcomes. As part of this exercise, you should define goals and outcomes that would signal incremental success of changes in your group. (We get into what these should look like in later chapters.)

Throughout the remaining chapters, I dive into the specifics of championing and helping with adoption of each of CAMS' principles in order to build the culture of cross-departmental collaboration and sharing in support of desired organizational outcomes. But first, it's important to understand and define measurable criteria for organizational success, which is the focus of Chapter 3.

CHAPTER 3
Quantifying Success

Now that you have educated the teams and have established a shared vision for what DevOps would mean for your organization, it's time to define measurable goals, both organizational and departmental. In this chapter, I provide you with tactics for quantifying the return on investment (ROI) value of DevOps culture changes and implementation.

Organizational versus Departmental Goals

A key thing to remember is that although organizational and departmental goals tend to look vastly different on the surface, departmental goals are intended to support overall organizational goals, and the two must therefore be closely aligned.

Consider a common example: a board mandate to lower Operational Expenditure (OpEx) across an organization. Such a mandate might, for instance, result in a directive for IT to look for cloud alternatives in order to lower monthly hosting and operational costs. However, departmental directives need to be presented in a context of organizational goals in order to offer perspective and avoid incorrect execution. Otherwise, the operations team might decide to switch their architecture to another cloud provider—but without explicit instructions to lower cost, they could reengineer the same architecture with a similar monthly spend. To make matters worse, there is not only an additional labor cost investment associated with the move, but also an increase in error frequency associated with a new architecture, making this initiative a costly exercise. In the end,

IT has met its departmental goals—it has successfully switched to a different cloud platform—but it completely missed organizational targets of lowering operational expense (OpEx).

To line up everyone's objectives, you need to clearly define the success of all the departmental initiatives by tying the goals back to organizational Key Performance Indicators (KPIs). Remember, when defining goals, it is important to position them with the primary focus on *value delivered* instead of *deliverables*. The DevOps community has adopted the terms *output* and *outcome* to define the difference. Output is what is traditionally expected to be delivered, be it code, boxed product, or plans. Outcome, on the other hand, is the value that outputs provide: better user experience (UX), revenue increase, improved quality. Even though individual departmental initiatives will still produce *outputs*, the quantitative goals should be *outcomes* that benefit the organization and its customers. In this chapter, we dive deeper into the definition and measurement of outputs from various groups and their correlation to organizational outcomes.

Quantifying Success

In any DevOps implementation, teams need to understand the *value of change*. This premise assumes that every change, large or small, carries a certain value. The value can be either positive or negative. In context of a company, every change made by any of its employees has a quantifiable impact of the organizational bottom line. Again, that impact can be positive or negative. To claim success as a result of a change, more often than not you need to illustrate a net positive value of that change.

Every change, large or small, carries a certain value.
The value can be either positive or negative.

Some changes carry straightforward value. A developer changing the code to fix a bug in a checkout process that errors out for people outside of the United States because of different postal code syntax has provided a very positive value that can be measured in international currency. Conversely, the developer who deployed the initial change, adding a validation to a zip-code form field to allow only

five-digit zip codes, added a negative value to the bottom line, measured in the same currency. But not every change is that binary.

Take, for instance, downsizing. Often, companies undergo restructuring or rounds of layoffs to reduce operational cost. Naturally, reducing headcount carries a positive value of change to an organization from a fiscal perspective. However, there are long-lasting impacts of any downsizing, both on productivity and culture. Layoffs have a documented history of having a degrading effect on productivity, especially in the environments of "high involvement workplaces," which organizations embracing DevOps culture tend to be. These are workplaces where greater emphasis is placed on the importance of human beings compared to traditional workplaces, and employees have more decision-making authority and responsibility.[1] Additionally, company restructures are often intended to replace higher-paid employees with cheaper, less-experienced counterparts. As such, in order to derive a true net value of personnel change, you would sum up recurring negative value of dropped productivity and decreased quality of labor with the positive value of cost savings.

> -(Δ productivity) – (Δ quality of labor) + cost savings =
> net value of personnel change

To add to the complexity of that calculation, a decline in productivity and quality can build incrementally over an extended period of time, especially if it's not addressed, further reducing the value of change over time.

That having been said, many organizations make complex decisions consistently with arguably demonstrable success. These companies approach their decisions from organizational need perspective, developing strategies that are projected to yield the highest return on investment (ROI) value. Multiple sources, such as historical data, multichannel metrics, and fiscal projections, contribute to every decision that go into the calculation of projected value of proposed change for any organization (more on this in Chapter 6).

1 Robert Sutton, "The Last Word on Layoffs: Evidence on Costs and Implementation Practices," *Harvard Review*, August 1, 2007, *https://hbr.org/2007/07/the-last-word-on-layoffs-evide*.

Undergoing DevOps transformation is no different.

Applying the Appropriate Metrics

To demonstrate the positive ROI of the initiative for your organization, you should define, collect, and eventually present tangible metrics. Those should go into the decision on the viability of any new initiative. Although actual metrics vary from organization to organization, there are a number of universal metrics that you should collect, examine, and measure as part of DevOps adoption. We can divide these metrics into two broad categories: those that measure delivery speed and those that measure quality.

Delivery Speed

Perhaps the biggest advertised benefit of DevOps is the promise of speed. From the removal of the proverbial wall of confusion between departments, to the frequency of deployments to the idea of continuous code delivery, speed is a central theme when discussing DevOps goals. Measuring speed improvement would be a logical stat to benchmark as part of your initiative.

There are two major areas you should consider when defining measurable goals for speed: time from inception to market, and time from commit to deploy.

Time from inception to market

I've heard DevOps defined (narrowly) as "delivery of application changes at the speed of the business." In nontechnical terms, this refers to the *time from inception to market*, a metric of great value that represents the velocity of your innovation cycle and correlates directly to ROI.

Let's take the video game industry as an example. A popular trend in online games is to release periodic expansions or Downloadable Content (DLC), providing players with new content and game experiences at a modest price point. So let's examine a company that releases new content every six months:

With every DLC release, the game developer not only earns revenue from active players, but the new content also attracts gamers who have moved on to other games. Speaking financially, the company would experience revenue spikes every six months, tailing off rap-

idly over the next few weeks after the release, just to be repeated in six months when a new release is pushed out.

Following a base premise that adopting key DevOps principles can accelerate delivery, shortening DLC development cycles to five months would project a 17% increase in revenue—a number that could buy a lot of goodwill within the organization.

But remember, context is important. In contrast, the magazine industry works in a fashion that might appear similar to the video game industry but is actually very different. Although magazines also deliver their product (issues) on a cycle, their delivery cycle adheres to an immutable interval—after all, you cannot release a monthly issue every 25 days. However, even though earlier release is not the outcome to aim for, shortened production could still provide positive value to the organization, which we will look at in greater depth in the next section.

Time from commit to deploy

A common metric in the world of software development is *time from commit to deploy*, or the time interval between completing development to making the change available to customers. Like most DevOps-related improvements, this one can (and should) be applied to *any* inefficient processes, technical or not. To frame the concept in nonsoftware terms, it's the *efficiency of the deployment pipeline*. Or, to put it even more simply, what does it take to *put a finished product in front of your customer*?

> DevOps principles don't just apply to technical processes; they should be applied to all organizational processes.

Let examine the same example we just looked at, the monthly magazine company. Every month, after all the content has been collected and written, there is a significant effort that goes into producing the magazine. First, the content needs to be finalized and organized for review. Then, the content goes through a rigorous proofreading and editing process. From there, it goes to designers to be arranged to fit the magazine's layout and style. When it's approved, digital files are sent to the printer. Finally, the issue is delivered to distribution channels, all before the magazine makes its way to customers.

If the process sounds familiar, it should. It closely resembles the software deployment process: commit → QA → integrate → deploy to production.

To improve on the process, you can apply concepts that often are associated with DevOps principles, like shorter feedback loops and automation of repeatable tasks, shortening production time, and allowing for more time for quality content creation.

In this case, speed translates to process efficiency, which is a viable metric against which to benchmark value.

Time is a critical metric of DevOps adoption. Yet, people often make the mistake of constraining speed improvements to software development cycles without considering the end goal. The speed of generating technical *output* becomes much less important if it doesn't result in business *outcome* of value. In the world of business, speed equates to process agility, which is applicable to any organizational process, not just processes within the software development life cycle. Identifying the processes that require the most improvement is the primary job of a DevOps leader, focusing on the largest ROI areas.

Value Stream Mapping

When reviewing the organizational delivery pipeline, you should pay attention to processes that can benefit from injection of DevOps principles. Value stream mapping can help you to collect and analyze data about the efficiency of individual steps and their value contribution to overall business outcomes. A value stream map illustrates your process from idea inception to delivery to customers. It provides you with visibility into the current state of your processes and can help to identify delays, blockers,and inefficiencies in your workflows. Keep in mind that value streams should involve multiple groups within your organization, not just IT, so make sure to include members from appropriate teams in the evaluation.

Evaluating the total time between handoffs and the processing time of each step should help to assess individual processes' efficiency. Pay attention to areas of potential improvement, such as quicker handoff time, opportunities for automation of repeatable tasks, and even reduction in steps or individuals involved in the particular step of the process.

Learn from the current-state value stream map and outline your ideal future-state value stream map by focusing on value delivery. From there, prioritize optimizations based upon their value-add to defined outcomes, focusing on highest-value items first. Be sure that you create feedback loops to continuously monitor improvements to key defined metrics.

As mentioned earlier, delivery velocity represents one important set of metrics for measuring the success of DevOps adoption. Quality is another important set.

Quality

Another primary criterion of measurable success for an organization is the quality of its products or services.

Often, when speed is positioned as a primary focus of DevOps initiative, quality suffers due to conflicting early goals. Yet quality is one of the key components that should be measured and calculated into a net value gain to validate the success of a DevOps initiative.

The significance of quality is widely acknowledged, yet the specific metrics remain a question. Much like speed, quality should be a quantifiable measure against which an organization can measure its financial success. However, unlike the measure of velocity, the value calculation should be done from a standpoint of cost reduction rather than revenue gain. Depending on the business, this metric can apply to the end-product quality, the effort to get to a desired product quality, or both.

Defect rate

Defects in a delivered product or service can be correlated directly to the gross cost of that product. In a traditional, scheduled software release development model, a large focus is often placed on testing in order to reduce a heavy remedy cost incurred from potential defects.

Agile (and later DevOps) organizations have approached the problem from a different angle: reducing the size and scope of each release, promoting iterative delivery of smaller features instead of large releases. Limiting the scope of a change also limits the scope for a potential error, reducing risk and cost associated with each release. Smaller scope shifts the focus from maximizing defect pre-

vention to minimizing remedy time. Smaller changes also allow organizations to not only iterate through and get ideas to market faster and cheaper, but also to receive, learn from, and act on users' feedback more quickly, increasing velocity of value delivery (as we'll discuss in Chapter 5).

But not all errors are valued equally. Errors found in the earlier stages of the delivery process require less effort to remedy, whereas malfunctions manifesting themselves in the final stages carry the highest resolution cost. The impact and remedy cost of a bug caught by an automated unit test would be significantly less than that of a bug that made it all the way through the delivery pipeline to consumers.

Errors found in the earlier stages of the process require less effort and cost to remedy.

The cost of failure is a very tangible unit of measure. There are certain metrics that can (and should) be collected to calculate an actual financial impact of flaws or failures of the product, including *lost revenue* and *cost of repair*. The combination of those metrics can provide a financial value of defects, resulting in actionable changes to improve the profit margin.

Factoring in hard cost for defects should be part of the bottom-line calculation for any organization delivering products or services. And then there are the intangibles.

Potential revenue is often regarded as an intangible metric, one that cannot be well measured. Although there is some truth to that, when it comes to immediate understanding of negative value, there are metrics that let you calculate long-term negative effects on consumers:

Returning customers
: Collecting and measuring the rate and value of returning customers provides a solid projection trend of what the expectation should be over time. Correlating that to the defect rate would give you a great insight into the financial impact of improving (or worsening) your quality control.

Customer acquisition attrition
: Similar to the returning customers metric, the customer acquisition cost and rate can be a powerful set of metrics when combined with the defect rate. Understanding both of these values can help measure the intangible: customer perception of your offering.

Change failure rate

As we were talking about business agility, we discussed that one of the main goals of DevOps is to increase the speed of delivery. Needless to say, to truly maximize the value from speed gains, the failure rate should be low; otherwise, you're just increasing the velocity of failure, bringing a negative overhead to the projected value. After all, to err is human; to err 1,000 times per second is automation. And no one wants that. We anticipate that with the addition of automated testing and delivery procedures, one should expect a decrease in the defect rate over time, especially as the team refines the process. An increasing failure rate, or one that is high and does not go down over time, is a good indication of problems in your process.

As an example, consider an online entertainment company that has a 105-step build pipeline to deliver an update to its customers every two days that takes 23 hours to complete. For this company, there is little margin for error. Unfortunately, the process failed about 30% of the time, blocking the work for multiple teams and requiring additional hours of work from the engineering team just to compensate for failures in the process, instead of concentrating on the next release.

In this case, the quality of the process was not only below the acceptable standard, but it also incurred additional cost in required (and lost) labor to support the broken process. On top of that, the overall time to completion was borderline unacceptable, incurring both opportunity costs and actual costs with periodically missed deadlines.

The good news is that this failure scenario provides three tangible metrics that improvements can be measured against: *failure rates*, *cost of labor*, and *process time*.

Automating more repetitive and labor-intensive steps of the process and adding early automated failure detection tests reduced the failure rate for this company by 22%, a change carrying more than 70%

reduction labor cost associated with failures. Parallelizing independent tasks where appropriate allowed teams to reduce a typical build time by more than 50%.

The tangible value of freed-up resources and guaranteed timely delivery resulted in improved customer satisfaction and provided a solid case for continuation of organizational DevOps adoption, based solely on measurable improvements to organizational KPIs.

Mean Time To Recovery

Often ignored by business groups, Mean Time To Recovery (MTTR) is a metric that technology groups often concentrate on during DevOps adoption. This is the time from problem occurrence (or, in some cases, detection) to resolution of the problem. Much like change failure rate, the overall time should trend downward over time because it represents a good measure of the team's maturity and effectiveness in handling changes and responding to new challenges.

NOTE

Do not confuse MTTR with MTTD (Mean Time To Detection). MTTD indicates the time from the problem occurrence to problem detection. This metric can be useful both independently and as a submetric of MTTR and is improved by better observability practices, such as top-bottom monitoring, intelligent logging, and actionable alerting.

There are a number of tactical implementations to improve MTTR, from better documentation to cross-departmental information flow improvement to improved delivery pipeline with defined rollback procedure for each deploy. Each of those (and many other) steps are initiatives that, although they can improve MTTR, require time and money to implement. And, let's face it: it can be difficult to get executive buy-in for that investment in potential future error response procedure. Effectively, you're offering to buy an insurance policy.

The way to explain the need for proactive investment into MTTR reduction is to frame the question in the terms that would be appealing to the CEO. You need to determine what monetary value downtime presents to your organization.

> Question to ask: What monetary value does one minute of downtime carry for your business?

I used to have a standard practice for my development team on one of the online entertainment projects I ran. I would run a query for every new developer on the project, showing average hourly revenue from online channels. Then, I would divide the number by 60 and say, "Every minute the checkout functionality is down, even if the rest of the site is up, this is how much actual money the company is losing. Not theoretically or potentially: in hard dollars." This approach might sound harsh, but that would educate developers on the actual business impact of downtime and encourage them to improve development and deployment procedures to minimize both the risk and MTTR.

A similar approach works in the other direction. Using MTTR metrics for correlation with revenue or cost impact can help you make your case for "insurance" investment. Here's an example. I recently witnessed a large ecommerce site, making just south of five million dollars per month in online revenue and originating from a traditional catalog company, suffer a big revenue blow during Black Friday. The owners, coming from years of successful brick-and-mortar operations, made a conscious decision to separate the operations and development groups both logically and physically, and maintain the traditional software development life cycle, limiting each group's responsibilities specific to its respective domain. Those choices contributed to the company's inability to accept online orders for more than eight hours—Time to Recovery—on Black Friday, which cost it hundreds of thousands of dollars in lost revenue. You can use these kinds of tangible metrics in helping you to frame your plans for improvements. In this case, you could present a plan to the CEO to reduce revenue loss in future failure situations as a function of MTTR, thereby presenting your case through the lens of the business outcomes they're most interested in.

The metrics described in this chapter provide a base set of performance benchmarks that bridge the gap between technological and business goals. All of them, although technical or procedural in nature, support business-centric KPIs such as revenue and cost control. However, as an organization, there are many more core metrics that you should collect across different functional groups (which we discuss more in Chapter 6) that are essential to supplement high-level KPIs.

The bottom line is this: if you're going through DevOps transformation, you need metrics not only to make sure you're changing in the

correct direction, but also to help with acquiring buy-in from different groups and aligning KPIs to work toward the shared goals. Without metrics, you won't have any way of knowing whether your initiatives are beneficial to the adopter group or organization as a whole.

CHAPTER 4

Defining Responsibilities

Once you have established strategic Key Performance Indicators (KPIs) and defined measurable organizational success criteria, it's time to coordinate individual responsibilities to support the effort. For that, you need to make a decision on how to position technology groups to best enable business outcomes. Every organization has its own optimal path to delivery of its goods and services; your challenge is to understand it and inject or adopt technology to support it.

To determine the best approach for positioning technology groups, you need to truly understand your organization's mission statement and strategic objectives. In most cases, however, technology focus is not included in a mission statement because most companies are not technology companies (even though they rely on technology to support and grow the business). In this chapter, we discuss assumed responsibilities for technical and nontechnical groups within an organization and the separation between local and strategic decision making.

Technology's Place Within the Organization

As an example of how technology fits into an organization's strategy, let's consider a modern media company whose goal is to obtain the industry leadership in production and distribution of quality digital content. Although the business would not exist without technology, the company's business objective is not software development; it is the production and delivery of its media content. In this scenario,

technology is an enabler of business, and the organizational structure should reflect that.

There are a number of approaches in which technology can be integrated into the overall organizational structure, each with its own set of strengths and weaknesses. Depending on business mission and operational practices, an organization's team structure and interaction methods will differ from company to company. Different organizational structures reflect different strategies and business models. There is not a "silver bullet" approach, but there are certain operational models that you can consider.

Technology is an enabler; organizational structure should reflect that.

A classic role for technology groups is to be treated as standalone units that report to the CTO and are responsible for their own P&Ls within the organization. This approach is generally taken in organizations whose value delivery processes don't directly rely on technology. This is the most traditional model, though it often incentivizes the wrong goals. With the technology unit being responsible for generating revenue from internal groups and adhering to its own budget, its KPIs are not to deliver the value to business units, but to optimize its own internal margins. Focusing on group *outputs* instead of business *outcomes* will misalign objectives and misrepresent the value of your group.

The first sign of a broken model would be a perception of technology group as a pure cost center. Usually, in compensation, a department's focus is on its own cost reduction as a measure of value, instead of maximizing the business value that technology enablement brings. Refocusing the group purpose from being a self-sustained department to a support model for different nontechnical groups would make technology a stakeholder in the success of business initiatives, breaking down the isolation and realizing objectives.

Another approach comes from applying services thinking to your delivery process. It has been a common practice, especially in enterprise organizations with large and distributed technical groups, to have a shared services or platform group that's responsible for building and maintaining infrastructure available for use and consumption to other groups. In this model, the delivery method shifts *from*

pipes to platform,[1] adapting a model that's been prevalent with the growing availability of web services. Instead of operating in a model in which the value is produced by a provider and consumed by users, it allows both value creation and consumption by users themselves.

We can apply an approach similar to service delivery models to that for nontechnical groups. Creating a business platform, a version of internal Software-as-a-Service (SaaS) collection offering, would shift the focus of the technology group from building and delivering code for consumer-facing initiatives to building and supporting a platform to enable nontechnical groups to create consumer-facing offerings. As I've mentioned in previous chapters, one of the biggest draws of DevOps is the promise of speed, and a platform concept certainly plays to that. Furthermore, the up-and-coming trend of "low-code" takes the concept of enabling nontechnical staff to produce consumer-facing applications and tools through graphical user interfaces (GUIs) on top of a provided platform. It reduces the time from application inception to getting the product into customers' hands. It supports perhaps the most drastic organizational shift from a traditional perspective: the "marketing technology" concept, in which technology is a function of the business unit that it supports, like marketing. In this structure, the group reports to a head of the marketing unit (CMO) because its main purpose is to support the marketing group's revenue-generating initiatives. It has been explored more and more in the industry and has allowed business units, like marketing, to dictate the platform needs to support their initiative development effort even further.

Choosing an operating model is critical to business success and requires an understanding of business needs to which technology services or a platform should be tailored. But even with an established model, there needs to be a clearly defined set of responsibilities for each technical and business group. These responsibilities will dictate both internal and cross-departmental processes and communication methods.

1 Sangeet Paul Choudary, "Why Business Models Fail: Pipes vs Platforms," *Wired*, October 2013, *https://www.wired.com/insights/2013/10/why-business-models-fail-pipes-vs-platforms*.

Decision-Making Structure

Even with an existing culture of shared responsibility and streamlined communication, any decision, proactive or reactive, that requires escalation to another group introduces a delay. Even though core DevOps principles promote a culture of shared responsibility and cross-functional collaboration, decentralizing decision-making by area of domain knowledge and maximum value contribution is essential for shortening time-to-value. Trusting individual groups and departments to make local decisions in support of an umbrella strategy shortens feedback cycles, reduces the human resources, and keeps the decision in the hands of experts.

However, not all decisions need to be decentralized. Critical decisions, such as incident responses, should not carry a decision-by-committee overhead. Conversely, including all of the involved parties in more global, strategic decisions—changes that affect organizational processes, for example—is a good idea. Asking yourself the following five questions will help to identify the type of responsibility for particular decision areas:

- Who is affected by the decision (department, organization, customer, etc.)?
- Does this decision require domain knowledge outside of the primary (local) group?
- How frequently does this type of decision need to be made?
- How quickly should the decision be made?
- What is the risk and cost of making a mistake?

Decisions that have only local impact, decisions that require a single area of domain knowledge, decisions that need to be made frequently, and decisions that require a quick feedback should be entrusted to local decision makers. Note that these are not disqualifying questions. These are merely a baseline for optimizing the process for solving certain types of problems. These questions also refer to immediate impact point, not chain value. I've emphasized that every decision should be made in support of shared goals. Based on that, theoretically, "everyone" should always be the response to the first question. Practically, however, there is a distinction to be made between "who is affected" and "who benefits."

By way of example, a quality-of-life improvement (such as switching operating system from Windows to macOS for a developer) does not directly affect anyone but that developer. However, potential productivity improvement due to the developer's familiarity with the tools benefits the organization as a whole. In this case, a small local decision contributes to the strategic goal of improved productivity. But even with a potentially high-profile impact, this tactical decision should be made locally by the software team lead without invoking a chain of command or soliciting feedback from other departments.

Similarly, as I discuss in Chapter 5, decision makers from all groups can benefit from cross-departmental domain knowledge. Having the director of technology involved in a new initiative brainstorming session with marketing might uncover technical limitations with an idea, inflating the originally projected time and budget scope, and forcing the group to rethink (or modify) the idea. Use these questions as a preliminary guide and adjust your strategy to best reflect your organizational model and culture.

To reiterate, not all decisions need to be decentralized, but understanding the scope and impact of decisions can help you to determine which groups need to be involved. As part of organizational transformation, you need to establish the guidelines for decision making and educate and empower the involved groups. Those guidelines should define individual responsibilities and communication channels for each group.

IT Responsibilities

Tactical responsibilities, often discussed as part of DevOps transformation, are local to technology teams. These responsibilities might include process changes to improve Mean Time To Detection (MTTD), new tool stacks to build out the delivery pipeline, or restructuring and augmenting teams with skills to support those initiatives. For that reason, the natural allocation of these responsibilities should fall solely on technology groups—not a major shift from well-run traditional organizations.

But ensuring autonomy for those decisions is key to DevOps adoption. The ability to make quick local changes in support of changing requirements allows quicker response and adoption, thereby helping the technology landscape to change at the same pace as business dic-

tates. But the only way to reap the benefits of this approach is to gain an understanding of organizational goals and ensure that all of the local decisions support those goals.

Process implementation

Agile-like methodologies are a cornerstone of successful DevOps adoptions. However, the framework and methodology that is most optimal for the delivery of services differ based on each group's mode of operation, personnel, and a number of other factors. Organizationally, companies can make a strategic decision to adopt Agile methodology company-wide, but the specifics of implementation should be delegated to individual groups.

If your group is responsible for scheduled, iterative deliveries of incremental features, a Scrum framework might be a good fit. However, if the groups are much more interrupt-driven, dealing with unforeseen issues and requests, Kanban might be a more suitable framework.

Similarly, a decision as to whether developers should participate in on-call rotation and the capacity in which they augment that team should be left to the discretion of the group that understands the immediate pains and best remediation steps.

Personnel decisions

Personnel decisions to support DevOps initiatives should also be delegated to the implementation teams. "DevOps Engineer," a common title in recent years, has even less commonly agreed upon meaning than the term "DevOps" itself.

DevOps is neither a team nor a role. It's an operational approach to solving problems as a single, cross-functional unit. As such, much like with IT groups in the pre-DevOps era, you need to define and understand the process and determine what skills would be needed to support it. There are newly formed roles within the teams adopting new methodologies; roles that are becoming essential in the DevOps organization, but are intended to supplement the team with the needed skill sets.

That having been said, when reorganizing the department to embrace the new processes, oftentimes both the structure of the organization itself and distribution of roles may need to change (both from a skill and headcount perspective). When increasing

automatic test coverage, for example, a reduction in the QA group might be appropriate in lieu of test automation engineers. Adopting a Scrum framework might require investing in Scrum Master training for leads and managers. The changes can even affect the resource distribution, such as hiring more developers to work on new features because the new architecture and delivery pipeline has allowed for better parallelization of tasks. Those are just a few examples of personnel changes that might be required to successfully realize DevOps adoption plans. However, much like with process implementation, local understanding of the problem can help provide a better solution for the desired outcome than a generic strategic mandate would.

Technology selection

Although DevOps, despite common misconception, is not purely about tools, selecting the appropriate tools is an important step in building a successful DevOps organization.

There are a number of valid considerations for selecting a product over its competitors, or as an addition to (or replacement of) part of an existing architecture:

- Documented benefits of technology to improve aspects of your business
- Ability to scale better with company growth
- Technology giving a competitive advantage
- Taking advantage of new and cutting edge technology

The challenge with these criteria is that each can also be presented as a list of *invalid considerations* without proper context and domain knowledge. As an example, an initiative like a cloud migration needs to be backed by performance, growth planning, and cost-benefit analysis as it relates to its own architecture. Perceived benefits need to be validated by subject matter experts and tied back to organizational KPIs in order to determine the validity of the assumption and the value to the organization.

In a DevOps organization, the responsibilities of technology groups do not differ much from those of traditionally structured organizations. The group should be responsible for local process implementation, hiring needs, and technology choices. A major DevOps

culture shift lies in trust and enablement that the organization provides to its technology backbone. Business decision makers need to both trust that all the local decisions will be done in support of the umbrella strategy, and they need to provide appropriate context to allow technology teams to make educated decisions. It's the responsibility of the organization to cultivate both.

Business Groups Responsibilities

Business groups must define the context for an organization's strategic vision. Context is important. It establishes a common baseline for the communication. Without it, you're creating disjointed channels for data sharing, each with its own context. When your message is delivered in one context but received in another, it creates a set of false expectations for both parties. Without full understanding of strategic goals and without being kept in a loop when those goals shift or change, IT cannot effectively plan its support tactics and will fail to deliver value.

Strategic direction

With implementation being delegated to individual groups responsible for their segment of the delivery process, one of the primary responsibilities for business units is to define desired outcomes to which individual group goals should align. For IT, choosing the appropriate processes and tools to accomplish the goals is important. However, to achieve that, the IT team must answer a question: "What is the goal?" And the business group needs to be the one providing the answer. After the business group defines and (perhaps more importantly) communicates its success measures, each group can define local KPIs in support of those shared goals.

One of the major breakdowns occurs when high-level decision makers are unable to separate strategic decisions from tactical decisions. Let's examine real directives that were given by nontechnical executives and leaders to technical teams for execution:

> Bad: "We need to buy Ansible Tower and automate everything."
>
> Bad: "We need to move to the AWS cloud."

These examples illustrate poor strategy communication. Neither of these statements frame a problem that needs to be solved or an outcome that needs to be reached. Instead, they attempt to define a solution that needs to be implemented. Purchasing a particular tool or

switching a platform is not a goal; it's a means to achieve a goal that has not been communicated. Requirements like these raise more questions than they answer: why a particular product and not another? What is "everything"? And perhaps the most important question, "What is the goal we are trying to accomplish by these changes?" is left unanswered.

> Better: "We need to automate our testing."
>
> Better: "We need to scale our application better."

These requirements are formulated a little better in the sense that they move one step up from tool selection, but they still leave a lot to be desired. They still provide solutions instead of defining problems. Do we need to scale horizontally or vertically? Why is concentrating on test automation is a priority? Do we really need to scale or is performance a problem we're looking to solve? And if we do need to scale, what are the metrics for projected growth? Ultimately, the same question ("What is the problem we are trying to solve?") is not answered. Instead, a solution is requested without defining an actual problem or a goal.

> Best: "We need to deliver product 30% faster to consumers."
>
> Best: "We need to reduce our operational expense by 20% and prepare for ten times growth over the next year."

The preceding statements are an example of what properly formulated strategic goals looks like. The desired organizational metric that needs to be improved is clearly communicated, and the means to accomplish the goal is left for local decision makers, allowing them to define local KPIs in support of the strategic goal.

One of responsibilities of a DevOps leader who tries to bridge the gap between business and IT is to make sure the information flow is accurate and, most important, meaningful. Making sure that strategic directions are measurable and actionable is the first step in establishing that feedback loop. The next step is ensuring that data flow is context complete and continuous.

Transparency

Because IT KPIs need to support specific objectives or initiatives originating in one of the nontechnical groups, those objectives, along with changes to them, need to be clearly communicated down to supporting groups. And that information needs to be complete and timely.

Information completeness. One of the common landmines in communication is the assumption that a piece of information, often outside of the receiver's domain, is not important to context. Contractual requirements and business operation compliance are good examples of domains that are often kept away from IT.

Suppose that, as part of improving the delivery pipeline for managed customers, you plan a move from Subversion to Git to simplify your Continuous Integration (CI). In choosing a tool, GitHub seemed like an obvious choice. Not only is your team already familiar with the tool, but the SaaS nature of GitHub offers the lowest cost option. A no brainer.

But, what if you knew that each customer's contract contained a standard legacy clause stating that no part of their code or data can leave your network? This piece of information, seemingly unrelated to the initiative at hand, would immediately put a restriction on tool choices and eliminate SaaS options at selection process.

When sharing information between groups, err on the side of more information being better. Naturally, there is information that shouldn't be shared with everyone due to its sensitivity, but shifting communication models to allow for more transparency, within legal and security guidelines, should reduce decisions made based on incomplete data.

Information timeliness. Another often-encountered issue in communication is time sensitivity. DevOps is all about improving velocity of time-to-market initiatives, but with increased velocity comes increased demands for timely information exchange. The model where technology is the primary backbone of businesses is a relatively new concept, and at times, nontechnical groups either don't have "notify IT" on their checklist or else do so too late in the process. The reason is not malice. It is the lack of understanding of impact a business change has on technology plans.

Technology working in support of business initiatives is a two-way road. Just as IT must be aware of the effects of changes to the technology stack and processes, business units need to be mindful that technology changes might be required to support changes in business operations.

Perhaps the most common example is the marketing group notifying the operations group about a projected surge of traffic to the website. From a PR perspective, being featured on the front page of a major news site is a big success. From a technology perspective, it's a disaster that pushes the system over the projected capacity, causing a poor user experience and forcing the entire team into firefighting mode in order to accommodate the traffic spike. Such issues can be avoided with adequate advance notice, allowing some time to prepare to scale with the surge.

The biggest challenge with enforcing business-specific responsibilities is getting buy-in from respective business units to change their processes, which is not always easy, even considering the value the change provides. However, the change has to happen in order for IT to provide the most value to the organization.

With increased velocity of change comes increased demand for timely information exchange.

Separating decision making and responsibilities and aligning local outcomes to shared objectives is crucial in the positioning of your technology team as a supporting pillar for the success of your business. To successfully gain (and maintain) an understanding of strategies and priorities, you need to implement a combination of pull and push techniques at different points of your process to obtain complete and timely information on a consistent basis. In Chapter 5, we talk about establishing clear and reliable communication channels and feedback loops at every step of the delivery life cycle.

CHAPTER 5
Improving Communication

In a DevOps-centric organization, communication and collaboration are the fundamentals for success. When an organization focuses on agility, the transfer of accurate and complete information must match the pace of delivery—which, in turn, increases value delivered. However, many organizations don't invest enough into improving communications. For example, the Project Management Institute (PMI) reported that, on average, 40% of all projects that do not meet their original goals and business intent do not succeed due to ineffective communications.[1] That's a staggering number of failures tied to day-to-day communications.

Better, more streamlined communication channels and feedback loops allow teams to make higher-quality decisions more quickly and to identify and respond to mistakes, improving efficiency in service delivery. Improvement in those areas carry tangible boosts for business Key Performance Indicators (KPIs), such as operational costs and profit margins. Framing the issue in terms of organizational value places even greater urgency on improvement in communication efficiency. The first step in the right direction, as we discussed in Chapter 4, is communicating shared goals across teams.

1 PMI, "Pulse of the Profession, The High Cost of Low Performance: The Essential Role of Communications", *Pulse of the Profession In-Depth Report*, May 2013. *https://www.pmi.org/-/media/pmi/documents/public/pdf/learning/thought-leadership/pulse/the-essential-role-of-communications.pdf*.

The next step is to ensure that different teams still manage to speak the same language. In this chapter, we examine the importance of shortening feedback loops and cover process points most appropriate for continuous communication channels.

Education

Everyone is a master in their own silo. Specialists have always been highly sought-after resources. However, the same narrow specialization silos of operational knowledge often become a hindrance to the overall delivery process. Breaking down the proverbial wall between the groups, keeping specialization but expanding the breadth of knowledge in order to blur the hard transition lines, is what DevOps culture is about. However, the gap between domain knowledge of technical and nontechnical groups is much wider than the one between, for example, development and operations groups. Some concepts that are accepted as common sense to one group can be completely foreign and counterintuitive to another, which can cause friction between groups. Without proper context, even the best ideas can be viewed as "not applicable" to people in other groups. Furthermore, improvement decisions made based on incomplete or missing context can be harmful to desired outcomes.

It's important that you make an effort to share domain knowledge that will help to align goals for different groups and provide the context needed for supporting decisions.

Educating IT Groups

Engineers are great at building solutions to problems. But to do so, they need to understand the problem that they need to solve; otherwise, the result will be a great solution for someone else's problem. The key is to make sure engineers understand not only the problem that needs to be solved, but also the thought process that went into the definition of that problem. Results are always better when you understand the reasons for work.

The world's largest digital lottery company, daily sending upward of 100,000,000 emails to its vast userbase, was always trying different approaches to improve click-through conversions and, ultimately, revenue with those emails. They experimented with multitarget A/B splits to gauge user responses, allowing collected information to inform their future decisions.

A request came from the marketing group, asking to design and build a new email template with defined specifications to test on a subset of users. The digital team, following the company's brand guidelines and industry best practices in user experience (UX), came up with a slick template with some innovative ideas intended to appeal to the target audience. Emails went out, causing a conversion rate on par with other variations, marking another success. The team was preparing to roll out the template to a larger audience when it received an email from the new director of marketing.

The email contained a number of stylistic changes to the template, including some that violated the company's style guide and many that went against the industry best practices. Furthermore, from an aesthetic perspective, the requested design would be significantly subpar to the one that was created by the design team. In the team's expert opinion, the "bad" design would turn people away from the email, lowering click-through rate.

The director of marketing appreciated the feedback and was happy to share his logic for the decision. He believed that the changes he proposed, as "ugly" as they appeared to be, would increase the *revenue* by 3%—not click-throughs or conversions, but resulting revenue. It was an educated guess based on his experience.

The team didn't fully buy the director's explanation, which it saw as more of a hunch than an actual educated guess; its collective experience and industry knowledge led the team to believe that the changes would have a cascading negative effect, causing a drop in revenue instead of an increase. But the director of marketing was adamant about the need for those changes, and because marketing was an internal customer, the team reluctantly decided to implement them. The team also decided to follow the success of the new email promotion closely in preparation for an "I told you so" moment when the numbers came in. However, much to their surprise, within the first 24 hours, the numbers showed a 2.6% increase in revenue compared to the "clean" version and other historical emails of the same type.

Despite the outcome, the UX team was not wrong. They provided the best solution they could based on the *information that was available*. The team was not privy to the information that the marketing group had available to it—deterministic analysis of data collected *in addition* to perceived best practices in the world of marketing. As

such, operating with partial data, the team came up with a great generic solution that wasn't optimal for the specific problem. Continuously sharing marketing analysis excerpts and domain-specific reasons for decisions would have provided the design team with context to inform its decisions and shortened the feedback loop, allowing it to ask on-demand supporting questions to clarify areas of uncertainty.

"Best practices" or "industry standards" don't take local context in consideration. Use them as guiding principles instead of rigid guidelines, adjusting to your specific use cases.

Educating Business Groups

In the same way that IT doesn't always understand business, business groups naturally often lack an understanding of technology. To effectively communicate, you need to speak the common language. And given the nature of business operations, this means that everyone—technical and business—needs to speak in terms of money.

Framing the discussion around the value of change resonates much better with business units. "We've installed [insert name of the product] in our production environment to mitigate [insert name of the technical problem]" does not give much context to a nontechnical person, nor does it relay the value of change. "We've spent [insert small dollar amount] to mitigate a loss of [large dollar amount] from periodic system disruptions" frames the value in terms that is understood by all.

Years ago, a completely nontechnical CEO with whom I had the pleasure of working, operating a successful multimillion-dollar digital business, summarized it in the most concise way. It was during a particularly bad outage when the team was working to bring the system back up; I was trying to explain how a failed disk on a proxy appliance had caused a trickle-down effect bringing the system down, but users were largely unaffected, except for a short period. I went on to explain what would be done to fix the technical problem going forward, discussing redundancy, tested failovers, and so on. His response to my explanation was, "I don't give a **** if my whole datacenter is on fire, as long as I am still making money." And there is a lot of truth to that statement. As was discussed in previous chap-

ters, concentration on the business value of any change should be a focus for the technical group. But IT is not the only group that makes a mistake of framing a problem in technical terms.

Chapter 4 discussed how nontechnical people tend to frame the solution they need in the context of technology goals. A great example of this would be the ever-elusive 99.999%, or "five-nines." This refers to the high-availability requirement for systems and services. Often, you hear an executive mandate to technology groups to harden the system to get to that mark. And the onus is on the technology group to educate that executive on the impact, the cost, and the value of that request.

The very first question that needs to be asked is, "why?" However, asking "why do we need this?" often doesn't yield the answer you're looking for. Instead, try framing the question around the specifics of Return on Investment (ROI). In this case, asking, "can our business sustain outages totaling more than *5.26 minutes per year*?" would move the conversation in the direction of discussing business-dictated requirements, which in turn should result in a decision whether five-nines is a hard requirement.

Second, remember that every change carries not only the potential desired outcome value, but also a cost associated with it. Everyone wants to be as resilient as Netflix, but not everyone has the budget to achieve it, nor do they usually have a need for it. Introducing a technical cost requirement to business will offer additional context, modifying the perceived ROI of the requested change.

The value of technical domain knowledge often comes in the form of an optimal solution to support business-defined objectives. Educating non-technical stakeholders on current technical capabilities and offering cost-factored options provides the missing context for value evaluation of ideas in terms of actual ROI.

Follow these guidelines when coming up with a solution:

- Define your business need instead of defining a technical problem.
- Set realistic goals.
- Always consider cost.
- Build what you need, not what you want to need.

Everyone is a master in their own silo and a novice in others. By breaking down knowledge silos and sharing expertise and knowledge, you can create an organization of experts with familiarity in other operational areas, helping tomake the optimal decisions based on the most complete information available.

Continuous Communication

In a DevOps environment, as I've covered in previous chapters, change is continuous. As such, an organization needs a communication framework to match the speed of change. Agile methodologies have made a step in the right direction by emphasizing simplicity of information exchange and promoting iterative feedback loops. However, to maximize the value from DevOps principles, you must integrate and optimize business feedback loops at every stage of your delivery and management process.

Before the Start of the Delivery Cycle

Perhaps the most common point of intersection between technical and nontechnical groups is the requirements-gathering stage of the process. The important thing to remember, however, is that the first step in requirements definition is to define *business requirements*. And this is what's often neglected during the process.

Engineers have a tendency to start thinking about the solution before fully understanding the problem. They often think their job is to develop software, but, really, that is just the means to an end. The end is to empower business to reach its goals. The solution can be elegant from a technical perspective, but if it doesn't meet the business objectives or time constraints, it doesn't work. And it's that on which a consumer (internal or external) will judge the result.

> IT responsibility is not to build and operate better software. It is to empower the business.

To align the focus, engineers must understand the target objectives and, no less important, the reasons behind the requirements.

Asking the right questions (and getting complete answers) is key. Keep in mind, as mentioned earlier, technical people often do not possess the expertise and knowledge specific to the nontechnical group that requests a solution. That lack of context can cause them to make a local decision to design or "improve" the solution, basing it on the *incomplete* information available to them—which, in turn, leads to an unsatisfactory solution.

Let's examine another example, the one in which the engineering team tries to take initiative and make improvements without fully understanding the scope of change.

The finance group noticed that one of the dozens of reports it runs on a regular basis was becoming slow and sometimes unresponsive. It created a ticket, asking IT to look into the performance problem. A database engineer investigated and made a number of changes to improve both current and future performance, planning for growth on the database and the query sides. Also, as part of that investigation, she noticed that the query generating the report was wrong.

Working in an organization that encouraged initiative to improve the system and process where possible, the engineer fixed the report-generation query and noted it in a ticket to make sure that consumers of that report were aware of the long-standing problem that was fixed.

The next post in the ticket was a request to revert the change and put the bug back immediately.

The reason, after some discussion, was dependencies. The data from the report in question was used as a base for a number of other financial reports and analyses. "Fixing" the report would change certain calculations, causing confusion across organizations among groups that relied on the data from this report. The organization had to run both reports in parallel for months in order to incrementally reconcile the numbers while maintaining stability.

Asking the finance group about the purpose of the report and history behind it would've given engineers the context they needed to identify the complexity behind the found "bug." Having a culture of continuous communication would also provide a shortened feedback loop during the development to bring up a potential issue to domain experts upon discovery and collectively discuss the best course of action for a remedy.

Always ask for the purpose of each request. Not only will that information help to better understand the problem, but understanding the end goal will help with coming up with the best solution.

During the Delivery Cycle

One of the primary purposes of effective communication is to continuously align everyone's current views so that everyone is looking at the same big picture at all times. Agile has embraced the model of continuous communication by promoting regular synchronization points, reviewing the current state and project-affecting changes and offering all relevant groups an opportunity to adjusts plan of action according to the new reality. Even so, with a DevOps approach pushing for higher velocity of change across the organization, limiting chances for feedback to the end of each delivery cycle, no matter how short, is not enough for effective communication. Communication strategy must evolve at the pace of business.

One of the key premises of DevOps is Continuous Delivery (CD). The approach helps deliver value more quickly by allowing for smaller, incremental changes. More features reach consumers faster, more behavior data is collected, and, as a result, the rate of change for business needs increases. If your team releases features whose real-time analytics inform future business decisions daily and your only chance to discuss the potential changes with business groups is at the end of a two week sprint, your feedback loop is too long. When requirements change to reflect new business needs, there is a high likelihood that a change in supporting technical plans and operations will be required. Waiting to share this information until the next scheduled checkpoint can result in a missed opportunity to capitalize on "hot" data or, worse, cause the supporting team to be unprepared for change.

Let's take a team responsible for the stability of a production system as an example. For the team on-call, its job consists of responding to production issues and finding a remedy in the fastest possible manner (given that, as Chapter 3 informs us, every minute of downtime carries a monetary deficit). To minimize the loss, the team needs to be equipped with all of the tools and—perhaps more important—access to information required for a quick identification and solution of the problem. I talk more about information collection and correlation strategies in Chapter 6, but for now, let's examine a common production issue. Figure 5-1 shows a common problem pattern in which system latency increases by multiples of the norm, causing the application to become unresponsive. This is a critical business problem because it disrupts customer experience.

Even if the operations team has been collecting all of the relevant system telemetry, it still needs to understand the cause of the spike to mitigate it. Correlating various system metrics can help to isolate and mitigate the problem, but it doesn't really answer the "why" question, leaving the system vulnerable for repeat problems.

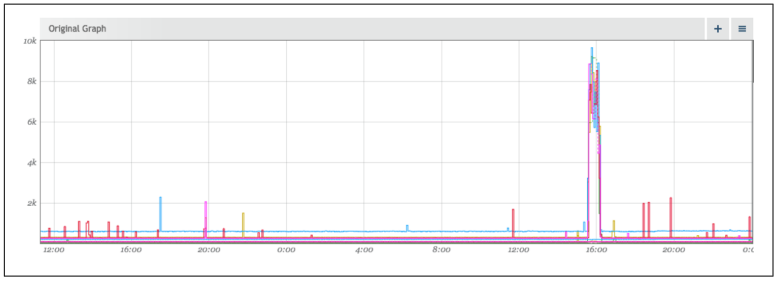

Figure 5-1. A user-perceived latency spike

As the technical team is frantically trying to reduce the load on the system to resume business operations, three floors above, the marketing team is cracking open champagne to celebrate the success of a social media campaign. That team has been continuously monitoring responses and adjusting the message based on real-time behavior data, anticipating incremental increases in traffic, which culminated in marketing success and operational failure, as shown in Figure 5-2.

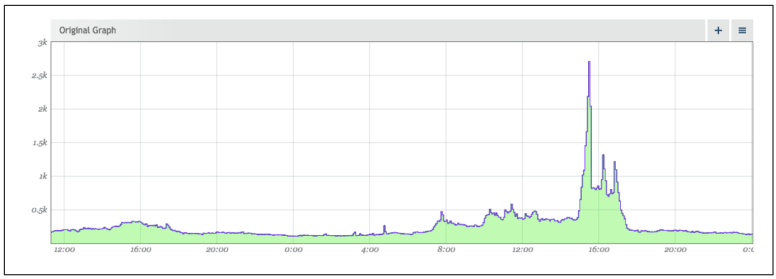

Figure 5-2. Marketing campaign traction in relation to a user-perceived latency spike

Had the information about the expected incremental traffic increase been relayed to IT, not only could it have made compensating changes to the system, but it also could have been prepared to scale the architecture as soon as the traffic had begun to climb—before the team was alerted about it after the traffic reached a critical threshold.

It's the operations team's responsibility to respond to unexpected events, but it is significantly less disruptive to business when the team is notified about expected events, allowing time to prepare support and a mitigation plan. As I mentioned earlier in this chapter, business units need to be educated on the value of communication changes to technology groups supporting their initiatives, even if they don't require immediate action.

Contributions should be judged based on the value they bring, not on a source of origin.

However, be aware that the onus for open and continuous communications doesn't fall strictly on business teams. Engineers have a tendency to shy away from all nontechnical communication during development or troubleshooting. Many technology teams have "developers only" and "no managers" communication policies so not to disrupt the flow of delivery. And although certain channels should focus on technical topics only, removing the team members from exposure to other areas of business is the wrong cultural approach—one that promotes silo isolation and cross-departmental distrust. The primary mistake lies in the dismissal of information based on the role or title of a person delivering it. Contributions

should be judged based on the value they bring, not on a source of origin. Different perspectives are extremely valuable, even in retrospect.

After the Delivery Cycle

One of the biggest communication challenges across teams is often building and sustaining an understanding of the value of technical effort among business stakeholders. Conversely, IT often lacks the understanding of the impact their successes and failures have on the organizational bottom line. Providing understandable and visible value correlation between technical and business outcomes cross-indoctrinates both groups. And one of the key feedback loops in the process for understanding the value of success and failure is *retrospectives*.

A retrospective is a periodic team review, or a review of an event such as an outage or a "near miss" or a missed delivery goal, with an actionable outcome for future improvement. Retrospectives offer a great venue for understanding the impact of change (whether expected or unexpected), educating the team, and maximizing the forward value. However, a list of people invited to them is often limited to technical groups. Much like development cycle communication, retrospectives are perceived as a technology-only feedback loop. But if we agree that technology's purpose is to support business, the goal of *understanding the impact of change*, as technical as it might be, becomes impossible without the actual understanding of the impact an event can have on business. Inviting the people who possess domain knowledge to educate the rest of the team is valuable not only from an educational perspective, but also from an angle of framing prioritized action items that would provide maximum business value gain. Additionally, collaborative retrospective is a great tool for encouraging collaboration and building a strong culture that increases transparency and builds trust within and between teams.

Conduct retrospectives even after events with a positive outcome. There is valuable information to be shared and learned for all of the groups, regardless of the type of outcome.

Let's examine a common event: a latency spike. An alert came in because something happened, unbeknown to the operations group, that caused degraded performance for users. The team responded quickly and kept the outage to a minimum. All in all, Mean Time To Recovery (MTTR) was commendable. Nonetheless, an outage is an event that deserves a retrospective.

Based on the telemetry that was collected, it looks like there was a period of about 20 minutes during which users' experience was degraded or they were not able to interact with any website features at all, due to connection timeouts caused by latency in content servicing (Figure 5-3). That's a sizable outage. However, after reviewing the overall incident rate and performance trends, it appeared that overall performance over the longer interval is well within the 99th percentile (a measurable KPI for the group), which meant that its Service-Level Agreements (SLAs) were not violated, which, in turn, meant that although there should be improvements made to flatten the spikes going forward, it is not considered a major outage requiring prioritization of effort.

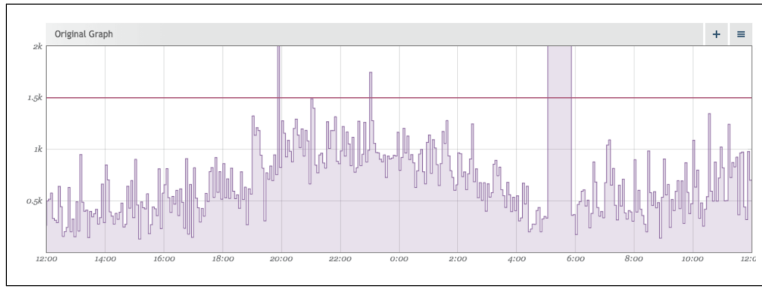

Figure 5-3. A chart showing the spike in degraded user experience due to latency in content servicing

From a technical perspective, the outcome of the retrospective is viable. The team validated the current state of the system against SLAs and adequately prioritized resulting tasks based on the information they had. However, a retrospective is another decision point for which context is important.

The scope of inspection of the outage was limited to local KPIs. Without access to additional telemetry and without understanding of event correlation, the team made a decision based on information available to it. Having a person who is responsible for monitoring conversion and revenue trends would paint a different picture.

Figure 5-4 shows actual users who were affected by the aforementioned latency variance. For anyone dealing with finances, each of the users represents a monetary value. During the partial outage, which was regarded as minor by the technical group according to its understanding of the impact, the company suffered thousands of dollars in lost revenue. What's even more telling, a past occurrence of revenue loss due to latency was almost as bad from a financial perspective but was disregarded as a false positive by the operations team due to a quick recovery.

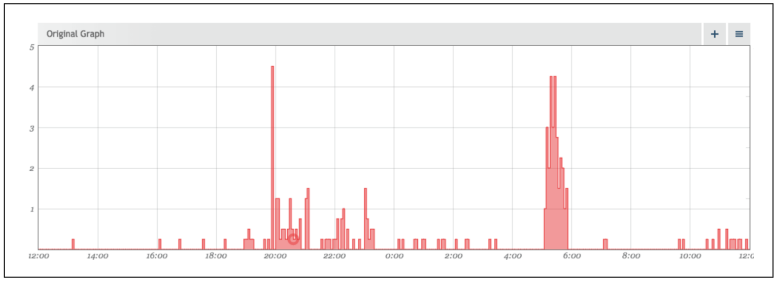

Figure 5-4. Actual users affected by the latency in the system

Understanding the impact of change is crucial in the world of continuous changes. A singular event can have variable perceived value (positive or negative) for different groups within the same organization. Having representatives of all groups collaborate in order to understand a holistic impact of an event helps to understand a true value of change. Don't be hesitant to extend a standing invite to retrospectives for every group in the organization. It is an opportunity to both teach different groups about technical challenges and successes and learn from experts outside of the technical domain. Having a mixed representation achieves two goals often attributed to DevOps culture: exposure and continuous education.

Don't be hesitant to extend a standing invite to retrospectives for every group in the organization. It is an opportunity to both teach different groups about technical challenges and successes and learn from experts outside of the technical domain.

Communication and collaboration between business and IT is one of the cornerstones of effective and efficient DevOps culture. Agility and frequency of information exchange must match the agility and

frequency of value delivery that an organization is trying to achieve. You should integrate feedback loops into every stage of the delivery pipeline. These should not serve as current information exchange but rather as opportunities for continuous cross-education and forward progress. If the only venue for different groups to come together to discuss requirements, problems, or even to brainstorm solutions is a series of mandatory meetings, your communication strategy will hinder the velocity of your business.

Equally important, communication fosters teamwork, and effective teamwork leads to better solutions, better efficiency and better culture. Setting up communication channels between groups that allow continuous communication will promote the culture of teamwork throughout the organization, improving delivery value.

When assessing the effectiveness of your communication strategy, there are three questions that you should ask:

- Is my communication strategy flexible enough to change with my Agile processes?
- Is my communication strategy effective for my organization?
- Is my communication strategy contributing to delivery efficiency?

Answers to these questions would determine the positive or negative value that your feedback loops provide to the overall process. It is also worth noting that the benefits of both individual feedback loops and communication strategy as a whole, much like other process performance metrics, can be measured and optimized for efficiency. Tracking metrics like wasted cycles because of misunderstandings, time spent redoing tasks, and points of lost momentum, especially when correlated with other performance metrics (which we discuss in Chapter 6) will help you optimize communication strategy for your organization.

CHAPTER 6
Improving Visibility

Even with transparent communication, you're going to encounter roadblocks and issues both during the development and the operational phases of every initiative. Issues can stem from inadequate processes, bugs, edge cases, or unforeseen user behavior. As the group responsible for managing applications and systems in production, the question that your group should ask is, "How can we minimize the impact of a mistake or an unforeseen event?"

In a previous chapter, I've discussed the value of reducing Mean Time To Recovery (MTTR). The first step in that process would be identifying that there is an issue that needs to be resolved. The second is to quickly understand relevant information about the event. That's where observability comes in. It provides an organization with an ability to inspect the state of the systems at every step and detect changes, allowing you to assess and provide feedback on suboptimal processes and changes in behavior. Monitoring should be treated as a continuous feedback loop for determining the performance of each metric at any given time.

However, because a system's purpose is to support business objectives, focusing solely on the system's health is not sufficient to guarantee the business' performance. While major issues such as outages that substantially disrupt user experience have a clear impact on business performance, more subtle issues can have an even larger impact on business operations despite the appearance of technical success. For example, an error in business logic that does not properly act on a failure condition can prevent a small percentage of web

application users from completing a purchase for an extended period of time. Although a change in revenue trends might eventually be noticed by the finance group, system metrics such as uptime and network latency would appear within normal levels, allowing the issue to persist, potentially leading to a more substantial loss of revenue than would occur from a widespread but short outage.

Data without context can lead to an inaccurate understanding of what's actually affecting your systems and your business. This chapter covers both the importance of continuous insight into business performance—including what kinds of monitoring and alerting metrics you should consider—and the importance of data correlation.

The Importance of Holistic Monitoring

Monitoring has traditionally been a function of IT, focusing on *system health*. However, the health of the system is not valuable as a standalone measure. For organizations that rely on technology for their business success, monitoring efforts that are isolated to technical symptoms are at high risk for continuous failures. Focusing on metrics related to business success is vital for teams responsible for the health of the supporting system.

> I don't give a **** if my whole datacenter is on fire, as long as I am still making money.
>
> —CEO

The operations group needs to have access to business metrics of success in order to have a holistic view of the *business's health* and to be able to properly detect, diagnose, remedy, and (potentially) predict business-disrupting issues:

Technical monitoring
: Ensures that the *application and systems* are functioning as expected. Used for identifying failures in *technical* operations and capacity planning. Metrics can include network latency, CPU utilization, database, application performance, deployment frequency, and more.

Business monitoring
: Ensures that *business processes* are functioning as expected. Used for identifying changes in *business* trends. Metrics can include

business KPIs such as revenue, profitability, conversions, subscriptions, and communication efficiency.

When working on improving visibility into different aspects of an organization's performance, three "C"s should inform the process: collect, collaborate, and correlate:

- *Collect* data from all the sources, technical and nontechnical
- *Collaborate* on data quality and relevance with different groups
- *Correlate* different data points for a holistic view

Traditionally, teams collect the data that is critical to understanding progress for *local* KPIs, as shown on Table 6-1. Periodically, the pieces of information are shared in support of a particular conversation, but continuous visibility of data is limited to a group with the perceived benefit of having it. The operations group sees the metrics showing the health of the system, developers are interested in application performance statistics, and business groups focus on their own KPIs. This means that every group is making their decisions based on a different dataset. The separation of data availability can provide a false sense of normalcy at different parts of the responsibility chain.

Table 6-1. Sample metrics of interest by group

Key metrics	Group
Resource utilization, availability, MTTR	Operations
Delivery velocity, defect rates, application performance	Development
Profit, organizational KPIs, growth	CEO
Revenue, expenses, efficiency	Finance
Click-throughs, conversions, visits	Marketing

Let's look at a case from one of the top social media sites, averaging hundreds of million events per day. A system with 24/7 user demands needs to have high availability Service-Level Agreements (SLAs), and the operations team has invested a significant effort into both system resilience and visibility and the application in order to assure continuous service delivery. For every interruption of service, caused by either an API failure or unavailability of a portion of architecture, the appropriate team would be notified immediately. An extensive rule set for "everything is working" criteria has been established, including HTTP checks returning a 200 code from tar-

get API checks, system health being within norm, and the user browsing trends being within defined thresholds. The assumption was that while those metrics were within the norm, from a system operations perspective, the site was operational. That criteria was serviceable until one of the issues the team encountered was silent loss of event data.

The site was up and available for browsing and the API accepted post requests and returned success codes to users, but the posts themselves never registered in the database due to a faulty code deployment. From the standpoint of the group responsible for operational stability, the available data showed that the site was up and running; frustrated users and business analysts had a different experience.

The issue, in this particular case, was that the team responsible for service availability had concentrated on monitoring system and application metrics. Even though these metrics are important for isolating and troubleshooting the problem, the first symptom of the problem should be a change in a business metric trend. Setting the system to collect data on average events per second and setting an appropriate threshold alert on it would immediately raise flags after deployment, allowing for much faster Mean Time To Detection (MTTD).

A Top-Down Monitoring Strategy

There are a number of reasons an effective monitoring strategy is beneficial. For one, because software can never be bug-free, constantly observing system behavior is the only way to detect production problems early. The complexity of modern systems is another reason, where a glitch in one part of the architecture can have a trickle-down effect throughout the entire system, manifesting as a completely different cause. And, of course, effective monitoring allows you to proactively detect potential problems such as user-detected failures, security breaches, and performance spikes.

But perhaps the most important reason to monitor and trend the behavior of your system is because things change. And changes affect business.

A change anywhere in the organization can have a trickle-down effect involving processes from completely different groups. As a

person whose responsibility it is to maintain the health of the underlying systems that support business, you need to be able to detect the changes as well as understand the complete scope of their effects. Follow the steps laid out in the following subsections when defining your monitoring strategy.

Understand the Business Objectives

As discussed in Chapter 3, the first step toward establishing an effective monitoring program is to understand your organization's business objectives. Clearly identifying these objectives helps to come up with a list of business-critical metrics that need to be continuously observed for change. Understanding the success baseline values of those metrics enables the changes in data patterns to be evaluated based on the value of change.

For example, an organization that relies on site registration (beta sign-ups, membership services) might want to begin with a business check to make sure the hourly number of registrations does not drop below a set threshold. Similarly, an ecommerce application could benefit from a business metric that monitors credit card transaction success-versus-failure ratio to ensure that the sales process works as expected. Metrics such as these not only provide considerable business intelligence for forward decision making, they also provide a framework of focus for technical metrics in support of these benchmarks.

Map Business and Technical Metrics

After you've mapped the business objectives, the next step is to identify the technical metrics that affect the associated business metrics. Mapping the relationships between technical metrics and business objectives provides context and highlights mutual impacts. For instance, the credit card transaction success-versus- failure ratio measured by an ecommerce business depends on the availability of the application server, connectivity to a third-party authorization service, and availability and performance of that service itself. If an outage of the authorization service occurs, it will translate into a higher ratio of card transaction failures. Conversely, a higher ratio of card transaction failures might be caused by an issue highlighted by one of the associated technical metrics.

Correlate data

Continuous monitoring can (and should) collect large volumes of data for analysis. After the metric sets for each level have been established, make sure that the related metrics can be easily correlated. Collecting multiple business and technical metrics helps to ensure that the necessary information is *available*, but having the ability to review the related metrics together helps to ensure that the necessary information is *useful*. Understanding the relationships accelerates troubleshooting by enabling issues identified through a business metric to be mapped to the related technical metrics (and vice versa). Don't underestimate the importance of cross-silo correlation. More often than not, interpreting the reason behind a change in one metric requires following a chain of impact through multiple metrics, often collected via different channels. Having that ability to easily traverse the dependency tree on demand would improve MTTR.

Let's examine these premises by the way of a practical example. Most incident responses start with a call (or a page), and this one was no different. In this case, a call came from the CEO of a large online marketing company, operating a very complex system with multiple revenue-generation points supporting more than 100 million users. As he was reviewing his business health dashboard, the CEO noticed that the revenue from online channels had dropped by more than 20% from the normal daily threshold. He immediately escalated the issue to his web operations team.

As any engineer knows, step one of troubleshooting the problem is to confirm the problem, so looking at the revenue trends seemed like a good starting point. Fortunately, the WebOps team had access to business KPI metrics, so it was able to pull up the information in its monitoring system, as shown in Figure 6-1.

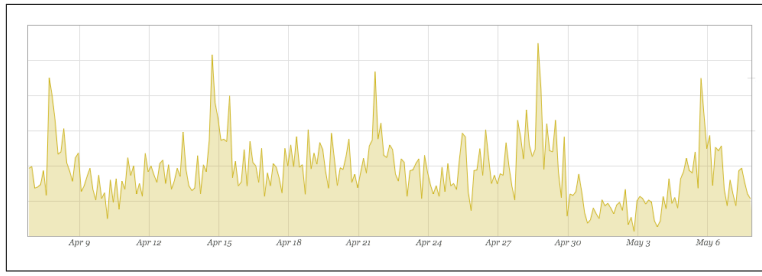

Figure 6-1. Revenue data over the course of a month

The graph clearly shows that, starting around April 30th, the trend looked abnormal in comparison to the previous few weeks. It seemed like there was an actual problem with revenue and not just a fluke in the report. Potentially, the issue could lie in the payment processor or another place in the system. However, it could also be a result of a nontechnical issue such as an unsuccessful marketing campaign or a drought in the sales process. The next logical step would be to correlate revenue to user traffic, collected from Google Analytics, to see whether there were any common trends, as shown in Figure 6-2.

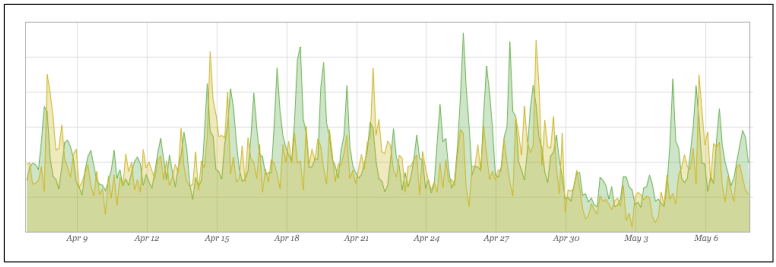

Figure 6-2. Revenue and user traffic data from the same period

Even though the traffic showed a clear drop at the same time as revenue, the ratio of revenue/visitor remained the same, allowing the team to exclude the payment processor and other application-specific logic from the list of immediate suspects.

NOTE

Often, this is the first potential breaking point in the process. It is very tempting to look at the ratios, attribute revenue decrease to natural traffic decrease, and stop the investigation.

The next question that the team asked was "what would be a logical cause for a drop in overall traffic to the site?" Numerous studies have shown that even a minimal increase in latency can cause a significant drop in conversions, interactions, and, subsequently, revenue. There are a number of potential performance bottlenecks in any web system, and this DevOps team had access to current and historic metrics for them, but the metric of the immediate concern in this case was user-perceived performance (as shown in Figure 6-3).

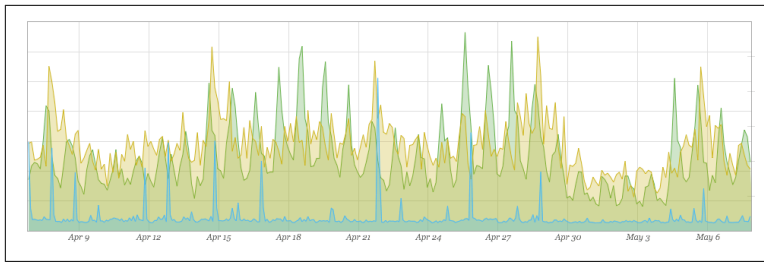

Figure 6-3. Revenue, user traffic and user-perceived performance data

Load times didn't seem to be deviating from the norm, but, as mentioned in the earlier example of the top social media site going down, the HTTP response metric doesn't provide full visibility into the load times for a dynamic application, so checking the health of the database and CPU usage on the server(s) to validate that the underlying platform was not the cause was a reasonable step.

NOTE

There are numerous metrics to monitor database and system health that should be collected, as they are in this case, but when researching the root cause of the elusive problem, diving deep into a specific component can waste time early in the process.

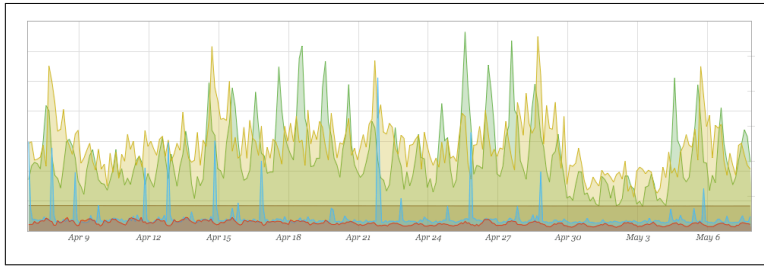

Figure 6-4. Revenue, user traffic, user-perceived performance, and database load data

The high-level metrics showing overall system performance appear well within the norm (see Figure 6-4), so at first glance, it seems like the original revenue problem was not a systems issue.

NOTE

This is the second point of the investigation where the process can break down due to the assumption that a problem doesn't have a known system origin.

A lot of engineers would conclude that there was no confirmation to the problem reported; the reported problem was just an anomaly because the technical system monitors didn't exhibit any issues. This is exactly why it's vital that technology teams understand the business.

After confirming that there were no clear indications of system issues, the team turned its collective eye to the business metrics collected, particularly the marketing metrics. It is not uncommon for a site to see a drop in purchases if a company stops promoting or if its marketing campaign is ineffective: traffic to the site slows down, subsequently decreasing the number of transactions. This company, in particular, sends out tens of millions of emails each day, bringing new and returning users to the site and subsequently guiding them through revenue-generating channels. So, looking at the email deliverability and bounce rates collected from the company's email engine was a good next step, as shown in Figure 6-5.

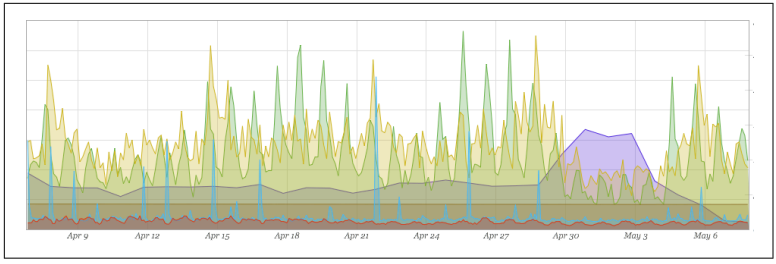

Figure 6-5. Data for revenue, user traffic, user-perceived performance, database load, and email bounces

Bingo! The bounce rates skyrocketed at the same time as the drop in traffic and revenue stream occurred. Upon closer investigation, it appeared that one of the major Email Service Providers (ESPs) accidentally blocked the delivery domain, thus preventing marketing emails from being delivered to the recipients. The issue was resolved (after some heated discussions with the ESP) and the trends returned back to the expected level.

Fortunately, the company's policy was one of transparency, and the DevOps team had sufficient data to target and correlate during troubleshooting. The availability of cross-domain metrics and the ability for the team responsible for incident response to merge and compare trends across business and technical domains enabled rapid resolution of the issue with little wasted effort across all teams.

If email deliverability was not the issue, there were over 1,500 other development, operational and system metrics that were on a list to be verified. Having access to those metrics significantly speeds up troubleshooting and, subsequently, time-to-resolution.

The primary goal for an effective monitoring program is to improve business performance. It also minimizes negative impacts by enabling faster MTTR, and maximizes opportunities by providing information to guide planning and to support strategy development. As such, when defining your monitoring strategy, begin with a focus on a change in trends to top-level strategic KPIs as actionable trigger events. From there, work down the logical stack to collect supporting metrics to have data available for contextual analysis.

To limit the quantity of data to a reasonable level and avoid stale or stray data points, for each metric added for analysis, you must ask two questions: "what are the key decisions that this metric will enable?" and "under what condition this metric is no longer needed?"

Continuous Instrumentation

I should note here that monitoring, much like security and performance, is not a feature. It cannot be treated as a line item on a delivery checklist or project plan. Adding metric collection as a post factum to a project delivery will result in significant holes in coverage of both business and technical metrics.

Many DevOps practices incentivize smaller, more frequent changes. Changes could include feature enhancements, marketing initiatives, or process improvements. But, as was noted in previous chapters, every change carries a certain positive or negative value. To determine that value, you need to have the composite measurable data

collected for both pre- and post-change state. The delta of those two provides an incremental value of a specific change.

That having been said, determining and instrumenting a check for every change criterion is impossible. There are three types of change events that can occur:

Known knowns
: These are the metrics and success criteria that are known from the start, such as returned 200 codes from the API endpoint.

Unknown knowns
: These are the predicted problem-indicating edge cases that can be caught by observing top-level overtime trends, such as revenue thresholds.

Unknown unknowns
: These are the points of change that no one anticipates. These changes can initially be noticed only by their effect on other behavioral metrics.

Let's examine a situation very similar to the example with email deliverability affecting the bottom line. Like the previous example, this incident also started with the CEO noticing a visible revenue drop. However, unlike the success story in which the DevOps team managed to find a cause after traversing through metric dependencies, this time the team did not find anything out of the ordinary. It too had an abundance of metrics at its disposal, but none of those metrics could be attributed as a root cause of the problem.

The problem persisted for two days. IT and marketing teams continued to analyze logs and to collect data in search of anomalies, to no avail—until a call came in from accounting to the developers, asking them to remove the American Express logo from the checkout page because they were in a middle of renegotiating rates and therefore not currently accepting American Express cards.

Again, this was a process change, manifesting as a technical problem, resulting in a revenue loss.

Putting aside a clear lack of communication that needed to be addressed organizationally, IT should not have been faulted for not collecting credit card type usage metrics. No one could have foreseen this particular manifestation of the problem. However, after the incident, the team added a check that would catch similar criteria in

the future, such as hourly usage threshold or credit card distribution ratio.

In each of these examples, the team had access to more than 1,500 metrics to review when the incident occurred. A lot of those metrics were added over time, after either the initial problem occurrence or in response to business-driven changes.

New data points should be added to the collection system with a discovery of a new variable condition or with a change to base assumptions. Make it a part of your standard operating procedure to include visibility coverage assessment and improvement as part of any incident retrospective or new requirement implementation.

As mentioned earlier, monitoring strategy should not solely focus on measuring the availability and success of the service offered. Internal process inefficiencies can often be more costly than system outages. Changes don't just happen in software. Organizationally, KPI, process, even personnel changes can cause a value-altering chain reaction.

Monitoring Processes

The main reason why monitoring is important for any organization is because you cannot improve what you can't measure. This mantra applies not only to systems, but also to processes. The ability to observe processes helps not only to respond to change-induced issues, but also to detect inefficiencies. Implementing key metrics across the organization helps you to recognize areas that are most in need of improvement. If the metrics show that certain workflows are inefficient due to bloated processes or interactions between multiple groups, those workflows need to be reviewed and changed. Having insight into cross-departmental process efficiency provides a holistic picture of the areas in need of improvement.

You cannot improve what you can't measure.

Much like with technical metrics, a cascading approach should be taken when monitoring processes. A process, for all intents and purposes, is a workflow. Workflow consists of multiple steps that result in a desired outcome. From a metric collection perspective, that outcome would be the top-level business value that needs to be

observed and measured. Each individual step in that workflow should also be monitored and treated as a supporting metric.

Applying value stream mapping to each measurable point in a process can help to isolate areas in need of the most attention. Measuring the success of individual steps in a process against the value delivered and creating an actionable feedback loop to the team will allow you to continuously optimize those processes. Make sure to constantly assess the value of any action. If it contributes to net negative value of the overall process, change it.

An obvious example of a process with multiple individual steps that can be optimized would be a development process. It's a great way to gauge how effectively your team is operating and covers a broad spectrum of areas with potential need for improvements. Collecting metrics on top-level criteria such as time to market and defect rate (see Chapter 3), gives a baseline performance indication of your development process. However, to improve the outcome, efficiency should be injected into individual steps of that process. The metrics collected are an indication of how efficient each step is.

Break down each process into smallest possible measurable events and collect data on each.

If time to market is one of your KPIs that is in need of optimization, supporting metrics (such as feedback loops and time from commit to deploy) need to be examined. Assuming that commit-to-deploy time is the metric that negatively stands out, it can be broken down even further into smaller measurable events. Digging further, the reason for inefficiency might be that one of many steps of a feature approval process is contingent on an individual who travels a lot. If you can eliminate that step or delegate it to a committee, it would significantly shorten commit-to-deploy time and, subsequently, improve the overall metric of time to market.

This chapter talks specifically about monitoring, but organizational observability strategy should not be limited to systems and application monitoring. Every group should be responsible for collecting (or making available for collection) data specific to the group's domain. The resulting superset should provide visibility into any

process, technical or not, that contributes to the overall value delivery with a focus on business-first metrics via monitors, logs, or any other means. Moreover, your alerting strategy should follow the same path.

Alerting

A holistic monitoring approach encourages expanding your monitoring strategy to areas outside of IT and logically forces teams to increase the number of metrics collected. With the incrementally increasing complexity of modern systems and the broadening scope of observable behavior, the "monitor everything" mantra can exponentially increase a number of metrics collected. On the surface, this is a good thing. Having more data at your disposal provides more granular information for analysis, increasing the velocity of improvements. In theory.

In practice, however, applying data collection principles to notifications can generate alert volumes that are a curse for many teams. Can you imagine having an alert on every one of the 1,500-plus metrics in the previous example—from IOPSs to component metrics to individual user page views? In addition to the sheer number, false positives and misplaced alerts can place a significant burden on the teams responsible for supporting the business instead of helping them to detect and fix issues quicker. To make matters worse, indiscriminate alerting increases the chance of misprioritization, or altogether omittance, of a critical alert. For this reason, the concept of "actionable" alerts has been a familiar term in the world of operations.

Monitor everything. Alert on what's important for business.

An actional alert principle dictates a baseline requirement for every new production alert. Anyone adding an alert must answer two top-tier questions:

What is the business impact of this alert?
: This is the single most important question to answer before adding a production alert for a potential failure condition. When this failure occurs, does it materially affect the business?

If the answer is no, it should not be an alert that wakes someone at 3 a.m. By way of example, a failed node in a cluster of web servers, while requiring investigation and correction, does not critically impact business operation.

Can something be done to remedy the problem?
: If immediately fixing or implementing a workaround is not obvious or is outside of the control of a person woken up at 3 a.m., it should not be an alert. Let's revisit the American Express decline rate example. Although having data available would significantly decrease discovery time in the case of a repeat problem, immediate alert on that failure would not be actionable because the solution is outside the control of an on-call team. For issues that can be addressed by only non-operational staff, a separate alert contact group or an escalation chain should be defined to make sure the appropriate people are notified about the potential problem.

There are additional questions that are beneficial for the team, such as, "Can this problem persist until tomorrow with minimal/no negative impact?" and "Is someone better equipped to resolve this problem?", that would send notifications to designated groups via different alerting channels—but the primary questions would dictate the necessity and validity of each alert that goes into the system.

Making alerts actionable can significantly improve the quality of life and service for the team. Following top-down design as a guiding principle for alert structure will also focus your efforts on things that are important to the business. Additionally, like most things in DevOps culture, it is important to make continuous improvement to your alerts. Incident post-mortems and "near miss" retrospectives are good times to identify opportunities to improve monitoring, and especially alerting. Improving the signal-to-noise ratio by filtering out certain alerts, making alerts clearer, and directing alerts to the person who is more likely to address it is required to keep the team focused on the right objectives.

Remember: monitor everything. Alert only on what's important *for the business*.

CHAPTER 7

Improving Culture and Reaping Rewards

DevOps fosters a culture of continuous learning and improvement Throughout this report, I've talked about techniques and methods for organizations to improve cross-departmental processes and cooperation. However, at its core, DevOps is about culture (as noted by the CAMS abbreviation, in which "C" stands for culture). And culture, first and foremost, is about people.

For that reason, one of the primary factors of success or failure of DevOps adoption is the human factor. All of the suggestions in this report are aimed at creating an environment that encourages collaboration and empathy because these qualities help to create business value through better quality. However, many organizations suffer from distrust among the groups, which creates a challenge when moving toward a more open culture. To achieve a culture that supports the technical and process changes of DevOps, you need to not only show local value to everyone involved, but also create an environment that encourages collaboration, trust, and empathy among people across the entire organization.

Fostering Empathy

The most nonconcrete concept that's often associated with DevOps is empathy.

All people are different. They have different backgrounds, beliefs, and experiences. To make matters even more complicated, those individual experiences and points of view can change over time. Yet to maximize the organizational value, individuals with different backgrounds, titles, and goals must effectively collaborate and work toward shared goals.

Traditionally, there is an inherent lack of trust between technical and nontechnical groups due to their diverse domains of expertise and misaligned objectives. Legal doesn't understand why engineers push back on a standard, because in their eyes, "all deliverables will be free of all defects." Sales people can become frustrated that the development team is hesitant to release a big feature at 5 p.m. on Friday before a holiday weekend. And engineers are skeptical that user experience (UX) changes that the marketing team requests will increase the revenue. This lack of understanding of decision-drivers for different groups leads to a sense of superiority, which is not conducive to an empathetic environment.

> In a culture in which individual groups feel superior to others, collaboration efforts are doomed to fail.

Everyone is a master in their own domain and a novice in others. It doesn't mean people in one group are more or less qualified in their domain than in another. It just means their areas of expertise are different. Statements like "marketing people don't understand how technology works" are detrimental to breaking out of knowledge silos. Of course marketing folks don't understand the nuances of technology; it is not their job to be an expert in that area. It is yours. It is also your job to mentor and educate other groups, including marketing, on the benefits that technology can provide in support of their objectives.

The best way to start the transformation toward better collaboration is to assume good intent. Assume that everyone is trying to do good.

When different groups come up against misaligned objectives, don't assume malice—assume misinformation, and work on correcting that. That's where empathy comes in.

Assume that everyone is trying to do good.

Empathy, in essence, is the ability to relate to other people and a willingness to see things from the point of view of another person. There are a number of reasons why empathizing with others is important in the context of DevOps:

- Better appreciation for the challenges that cross-departmental groups and teammates face
- Deeper understanding of motivation and context for decisions
- Less cross-departmental conflict due to lack of understanding
- Continuous insight into other parts of organization

A transparent and collaborative cross-departmental environment creates empathy between different groups. Empathy improves understanding and better enables people to relate to other groups' problems. By empathizing with others, you will be a better teammate.

Empathizing with customers is just as important as with people in your own organization. Customer satisfaction is an ultimate success criterion, and making a point to truly understand their needs should improve the value delivered.

Building Trust

Jeff Sussna, a prominent DevOps expert, in his interview for DevOps.com, made an interesting observation that Agility + Empathy = Quality, arguing that quality has a direct correlation to both the effectiveness and the culture of all of the teams involved in execution. Cross-departmental teamwork, in addition to goal (mis)alignment, poor communication, and lack of transparency, can be affected by different motivation incentives and trust among various teams.

Departmental distrust, both rational and irrational, is caused by a variety of factors and can have cascading negative effects. In addition to quality, other factors that can be affected are communication, sharing, and competency perception. To gain the trust of other groups, you need to present the data that is missing to give validity to your input.

Be transparent. *Transparency is key to building trust*. When decisions are made behind closed doors, without substantiating data, they are frequently perceived as invalid, especially when coming from groups with limited domain knowledge.

Show value in the language the recipient will understand. As Chapter 5 discusses, seeing data in a familiar context provides a sense of relatability and creates contextual competence, improving trust.

You must also understand the effects of a culture change on different groups. For example, project managers are a group that is often disliked and distrusted by other groups, technical and business alike. Even good project managers find it difficult to gain trust and not be perceived as a blocker in an Agile organization. The reason for that is competing goals.

Following Agile principles, the objectives of technical and executive teams in a DevOps organization are to constantly innovate and improve. Continuous improvements mean frequent changes. In contrast, a primary function of the project management office (PMO) is to maintain stability and to control the rate of change. It is not that managers don't want innovation; it is that their function is to manage risk and to enforce consistency and predictability in the process of innovating. Yet the decisions they make in service of that goal are often perceived as resistance to change by other groups, causing the trust in their support for organization vision to be diminished.

In addition to having innately conflicting objectives, when defining a DevOps adoption strategy, organizations often focus on drivers, incentives, and collaboration methods with input from technical and business teams, leaving middle management's needs outside of the scope of the conversation. Isolating a group creates resentment and loss of trust, resulting in degradation not only of that team's contribution value, but of the organization as a whole.

Refocusing objectives in terms of group-by-group incentives and value added will help to receive buy-ins and cooperation from each team. In this example, educating managers on the benefits of lean processes and the managers' roles in making the process a success will help to align objectives and show the importance of their group in the context of the overall plan. Involving managers in the discussion on details of day-to-day implementation of changes gives them a sense of inclusion and gives you the benefit of insight into their perspectives.

Every group involved in the delivery of organizational value needs to be included as part of a culture shift. Follow these steps when trying to create a culture of trust (or to regain or strengthen trust):

- Identify trust gaps
- Understand causes for distrust
- Show value by identifying with groups objectives
- Be open about goals
- Create information transparency

Also keep in mind that the preceding list is a blueprint and not a catch-all solution to trust issues. Things can and will go wrong. When they do, the focus must be on the problem and not the people. Trust breaks down when issues arise, and that is when you need trust the most. Building a blameless culture creates a safe environment to enable and encourage people to work together to find a path forward without shifting blame onto others.

The best way to build empathy and trust is to get people working together toward common goals and solving problems together—look for opportunities to bring people together across groups in brainstorming, in retrospectives, and in celebrations. Again, without a culture of mutual respect and collaboration, most improvement initiatives would fall flat. DevOps is about people, not tools.

CHAPTER 8
Conclusion

The principles described in this report have been successfully used by IT groups to streamline intergroup communication and improve operational efficiency for technical groups. But technology is useless unless it supports business needs. The same principles can (and should) be applied across all the teams to improve *organizational* value.

As with any major organizational shift, this one is not without challenges. As a DevOps leader championing change from a grassroots angle, especially if you do so without full executive support, you would likely see resistance to change from various groups. You might encounter reactions ranging from direct dismissal of proposed ideas to unwillingness to share information, to refusal to include IT in nontechnical matters. Resistance to change is a normal human reaction; the onus will be on you to educate each group about the benefits this change will bring *them*.

Perhaps the best method to nontechnical groups excited about DevOps is to show success, even a small one. Find a point of frustration for that group or a person and improve on it. Whether it's automating a report for finance, creating an online form for HR to replace email submissions, or adding a direct line of communication for sales. Small wins help to build organizational trust in you and in your process, and encourage people from different groups who have benefited from these wins to champion for change.

In this report, I've talked about the need for a business-first approach to technology decisions, cross-departmental observability,

and continuous feedback loops as methods to bring business decision makers and technology teams together and align their goals and priorities. These tips should not only improve operational efficiency, but enable your team to help the business succeed by building and delivering the best technology service in the most efficient way. However, each organization's culture and approach is different, and even though the recommendations in this report will hold true for most, implementation details will vary. Use the following recommendations as a guideline to map out an adoption strategy that best fits your situation:

- Embrace organizational goals.
- Understand motivation and incentives for different groups.
- Eliminate blame.
- Concentrate on techniques that enable the other group.
- Encourage open and frequent communication.
- Measure everything.
- Make data-driven decisions.
- Use contextual correlation.
- Approach change as a series of small improvements.
- Celebrate your success.

Remember, as a DevOps leader, your job is not to provide the best technical solution and optimize the output of your team, but to enable your organization to thrive. I hope the information in this report will prove useful not only in your efforts to bring DevOps culture and processes to your organization, but also help you to succeed. Best of luck to you!

About the Author

A senior technology executive with two decades of experience in technical and business operations, **Leon Fayer** has expertise around architecting and operating complex systems and defining approaches to maximize technical decisions' value. Through the years, Leon has had a unique opportunity to design and build systems that run some of the highest trafficked applications in the world—in both the public and private sectors. In his past life, he brought a fully searchable digital library online for NRL, was an architect behind what is now IBM Enterprise CMS and OpenText BPM platforms, and was responsible for systems powering brands such as National Geographic and Wikipedia. Presently, Leon advises companies about critical aspects of technology adoption and delivery strategies.