Introducing DevOps into Your Project

eGuide

TECHWELL™

In order to adopt DevOps, organizations must fully embrace openness, experimentation, innovation, and collaboration. While you might feel pressure to introduce DevOps into your project, what if your company culture isn't ready? This eGuide rounds up a collection of resources to help you get started on your DevOps journey, which entails creating a generative culture and optimizing processes in a way that will ultimately lead to business success.

In this DevOps eGuide

Making DevOps Evolution Happen

It takes effort to evolve an organization's culture, processes, and technology to optimize performance for a DevOps environment, and it all comes down to the people. Large-scale mobilization requires a focus on people at all levels, empowering them to discover and make the changes that will help them most. Here's how.

You're Ready for DevOps—but Is Your Workplace?

In order to adopt DevOps, organizations need to be able to embrace the openness it requires, encourage experimentation and innovation, and work across departmental silos. You may be ready to encourage collaboration and communication to reap the benefits, but what if your company culture isn't? Here's how you can influence your organizational dynamics to lay the groundwork for DevOps.

7 Ways to Change the Culture for DevOps Success

The hard part of successful DevOps isn't implementing the technology; it's ensuring you have the right culture in your organization. You need to break down silos and align competing priorities and individual incentives to gain real benefits from DevOps. Move beyond thinking about technology alone and look at the people side of the equation. Here are seven ways to create a successful team that delivers the benefits of DevOps.

DevOps in the Trenches: Get Started with Metrics

DevOps initiatives often start with one silo seeking to be more collaborative with others. This "DevOps in the trenches" isn't ideal, but it is a way to get DevOps bootstrapped and begin seeing benefits. Here are some tips for how to get started doing DevOps based on what role you're in, with key metrics to help.

How Continuous Testing Is Done in DevOps

DevOps does speed up your processes and make them more efficient, but companies must focus on quality as well as speed. QA should not live outside the DevOps environment; it should be a fundamental part. If your DevOps ambitions have started with only the development and operations teams, it's not too late to loop in testing. You must integrate QA into the lifecycle in order to truly achieve DevOps benefits.

Shifting Right Offers New Possibilities for Agile and DevOps Teams

The shift-right concept originates from testing. But agile and DevOps teams also can use it to improve their systems and service to the client. However, there is a complicating factor: Different people have different explanations for what shifting right is. Let's look at the different forms of shifting right, what the potential benefits are, and who should ideally be involved in your shift-right process.

Embedding Performance Engineering into Continuous Integration and Delivery

In the world of continuous integration and continuous delivery, the importance of ensuring good performance has increased immensely. While functional and unit testing are relatively easier to integrate into these processes, performance engineering has typically raised more challenges. Here's how you can mitigate them.

Making DevOps Evolution Happen: A Conversation with Helen Beal

Helen Beal, DevOpsologist at Ranger4, chats with TechWell community manager Owen Gotimer about making your DevOps evolution happen, micro-bonus programs, and the neuroplasticity of squirrels.

TechWell Hub Takeovers

Each month a thought leader joins the TechWell Hub for a Slack Takeover. These chats give our community members an opportunity to ask questions, share challenges, and grow as software professionals by having conversations with an expert. Here's a sample of some of the questions and answers from a few of our DevOps-oriented Slack Takeovers.

Additional Resources

Making DevOps Evolution Happen

By Helen Beal

The world is in a state of digital disruption. The World Economic Forum says we are in the Fourth Industrial Revolution, typified by AI and the blurring of lines between human and technology. Carlota Perez thinks we are at the turning point of the Fifth Industrial Revolution: the precipice of a Golden Age. Organizations all over the planet are transitioning their ways of working from project to product to ensure their position in the new order.

And they are all finding it's a constant race to keep up; in some cases, they feel they are sprinting to stand still. It takes effort to evolve an organization's culture, processes, and technology to optimize performance for a DevOps environment, and it all comes down to the people.

Our people need to unlearn behaviors and practices some have spent several decades mastering. We need to unpick onerous processes designed to protect us and break dependencies in order to operate at the speed demanded of us.

We have to reframe failure as an improvement opportunity, build dynamic learning and safety cultures, distribute authority, and expect our leaders to enlighten us to be empowered and autonomous. We must train ourselves to think of end-to-end value streams and to constantly inspect, adapt, and shorten them, elevating value-adding activities above all else. We also need to automate, from idea to the moment value is realized, and ensure we use customer feedback to inform our next iteration in the best way possible.

It's all a very big ask. We know where we are now, but it's hard to see how to disentangle ourselves from the strangulating processes and bureaucracy we've spent years developing for the right reasons.

Our human and technology systems are highly complex and frequently fragile, and we can't expect to reach our long-term DevOps goals overnight. If it were easy, everyone would have already done it.

Every organization looks similar, but different—like a fingerprint. The same patterns appear over and over: governance, regulations, compliance, and security hamstringing us, the impossibility of prioritizing technical debt over much-needed functional changes, and financial models that drive undesired behaviors. The same patterns to solve these challenges also appear over and over.

The key is to not be dismayed or disoriented by the scale of the tasks ahead of us. How do you eat an elephant? One piece at a time.

Large-scale mobilization requires a focus on people at all levels, empowering them to discover and make the changes that will help them most; showing them the long-term vision but enabling them to aim for their next target condition; and experimenting with improvements, not just swimming against a tide of work.

Think evolution, not transformation. Think power of the people, not of the board. Think daily, constant improvement and adaptation, not one big bang. You are not thinking the impossible. This is the art of the possible, supported by science.

You're Ready for DevOps—but Is Your Workplace?

By Matt Hilbert

DevOps is gathering pace. More and more organizations are recognizing that by encouraging collaboration, cooperation, and communication, they can release features faster and offer advantages to customers sooner.

That's the promise, but in order to adopt DevOps, organizations need to be able to embrace the openness it requires, change the way they function, encourage experimentation and innovation, and work across departmental silos.

It's not as easy as it sounds, and the reason lies in an academic paper written by Ron Westrum, an American sociologist, in 2004. Originally published in Quality and Safety in Health Care, an international peer-reviewed journal for health professionals, "A Typology of Organisational Cultures" is as relevant today as it was then. Importantly, the model he created is applicable across every industry sector.

Westrum wrote the paper following research into the belief that organizational cultures shape many facets of performance. He was looking in particular at the safety aspects of health care, but he used his knowledge of organizational dynamics from many industries, going back decades.

What Is an Organizational Culture?

Westrum defines the culture within organizations as "the patterned way that an organization responds to its challenges, whether these are explicit (for example, a crisis) or implicit (a latent problem or opportunity)."

He called out the way organizations process information as a type marker for culture, and he identified three patterns, each created and shaped by the focus of management and the response of the workforce to that focus:

Westrum's Three Cultures Model

Pathological	Bureaucratic	Generative
Power-oriented	Rule-oriented	Performance-oriented
Low cooperation	Modest cooperation	High cooperation
Messengers shot	Messengers neglected	Messengers trained
Responsibilities shirked	Narrow responsibilities	Risks are shared
Bridging discouraged	Bridging tolerated	Bridging encouraged
Failure leads to scapegoats	Failure leads to justice	Failure leads to inquiry
Novelty crushed	Novelty seen as a problem	Novelty implemented

Westrum discovered that each of these organizational types influences activities such as communication and cooperation, which causes organizations to respond characteristically to problems and to opportunities for innovation. Indeed, the way they process information and behave is predictable in almost every case.

Pathological cultures are created within organizations by individuals who focus on personal power and the hoarding of information to gain advantage. The management style is typically domineering, with innovation and creativity discouraged because they threaten the status quo.

> *By its very nature, DevOps requires cooperation, collaboration, and a culture that welcomes change.*

Bureaucratic cultures emerge where the emphasis is on following rules and defending departmental turf. Whether it's good for the organization or not, existing procedures and practices are not questioned, and change is seen as a predicament that must be subjected to intense scrutiny.

Generative cultures arise when the motivation is the broader mission of the organization. This creates a climate that encourages the solving of problems rather than seeking the cause of them and supports innovation and cooperation within and across departments.

It's interesting to note here that the cultures Westrum talks about can exist throughout an organization or within units or departments of an organization. One part of an organization may be pathological while another is generative, the driver being the leadership in place, because it is the preoccupations of individual leaders that shape cultures.

The bureaucratic type of culture is also often the one that organizations or departments default to when there is no politics in play at one end of the spectrum or mission to aim for at the other.

You can probably guess where this is going. You may even recognize which column of Westrum's Three Cultures Model the organization you work for fits into … and also where DevOps is more likely to take hold.

By its very nature, DevOps requires cooperation, collaboration, and a culture that welcomes change. So the farther right your organization sits in the model, the better the fit with DevOps.

Can Organizational Cultures Change?

If you work in a department or organization with a pathological or bureaucratic culture, you might be thinking DevOps will never be an option. Encouragingly, however, in the conclusion to his paper, Westrum writes, "By changing the culture, virtually everything can change—trust, openness, confidence, and even competence."

The key is to influence the leadership style by, for example, referencing sources like The State of DevOps Report from Puppet and DevOps Research and Assessment (DORA) to demonstrate the business benefits DevOps can bring to an organization. You also could try using a DevOps approach to resolve a small but difficult issue. Doing so may open eyes to a better way of doing things.

And cultures in the same organization can change over time, depending on the management style in place. Westrum himself, for example, uses two different examples from NASA to illustrate how a generative culture solved a problem under one flight director and how a bureaucratic culture prevented problems being solved under the leadership of another.

When an oxygen tank exploded on board the Apollo 13 spacecraft in April 1970, it left the three-man crew stranded in the Command Module with rising levels of carbon dioxide and an apparently impossible journey home. Mission Control didn't give up. While the famous line from the Apollo 13 movie, "Failure is not an option," was never actually said, that was the attitude.

Mission planners worked out a way to use the moon's gravity to slingshot the spacecraft back to Earth. NASA engineers improvised a carbon dioxide filter using a hose taken from a space suit and a myriad of other odd bits and pieces available to the crew. A team of six engineers from the University of Toronto were called on to calculate the exact pressure required to separate the Lunar Module from the Command Module on re-entry.

At 12:07 p.m. on April 17, 1970, the Command Module of Apollo 13 splashed down in the South Pacific Ocean. All three crew members survived—a generative culture at work.

Fast-forward to January 16, 2003, and the launch of the space shuttle Columbia from Kennedy Space Center. During launch, a large piece of thermal insulation from the external fuel tank broke off and damaged the left wing of the orbiter spacecraft.

While the two-week mission went ahead, some NASA engineers were concerned about how the damage would affect re-entry when the orbiter returned. They requested imaging of the orbiter to determine the level of damage. Repeated requests were ignored and, in some cases, quashed. Worried, a former NASA flight director worked outside official NASA channels to get the imaging. He, too, was ignored.

Instead, staff relied on a damage prediction spreadsheet created to calculate the impact severity of ice pellets the size of cigarette butts. The piece of thermal insulation that struck the wing was the size of a suitcase.

The attitude of senior NASA managers was influenced by a belief that nothing could be done if severe damage was detected, so they opted to keep the orbiter crew in the dark about the situation.

At 9:00 a.m. on February 1, 2003, the orbiter disintegrated in the sky over Dallas, Texas, on re-entry. Hot atmospheric gases had penetrated the damage in the left wing and destroyed its internal structure, causing the spacecraft to break apart. All seven crewmembers died. This is a tragic example of a pathological culture at work.

Can You Introduce an Organizational Culture?

We don't all work at NASA, or even do jobs that involve saving lives. Most of us, however, want to work for an organization where the culture encourages innovation and collaboration and lets us do the right thing. So, what can you do to introduce a culture that is more responsive to DevOps?

If you work for an organization with a pathological culture, DevOps probably isn't an option. Cooperation and collaboration are actively discouraged, so the glue needed to connect dev and ops is missing. If you really want to do DevOps, you're out of options here, and it might be worth moving on.

In bureaucratic organizations, introducing DevOps is possible, but there will be problems. Think soft DevOps here, with long lead times and planning meetings before a tentative first step is taken. Patience is key, but conversely, once DevOps is successfully introduced, it will become the new norm and take its place in the rulebook.

Generative organizations are where DevOps will be seen as a natural next step, if it's not in place already. Every facet of their makeup matches the advantages of DevOps. Perhaps unsurprisingly, Westrum mentions a case study and anecdotal evidence that demonstrate how the cooperation and empowerment within generative organizations makes them more effective.

So if you're ready for DevOps, take another look at Westrum's model and decide where your organization sits within it. The next step will be down to you.

7 Ways to Change the Culture for DevOps Success

By Steve Jones

The hard part of successful DevOps isn't implementing the technology; in fact, that's relatively straightforward to do. Essentially, the technology side is about automation, focusing on infrastructure as code, and creating (and enforcing) a pipeline process.

The difficult part is ensuring you have the right culture in your organization. You need to break down silos and align competing priorities and individual incentives to gain real benefits from DevOps.

First off, why should you move to DevOps? There's clearly an increasing push behind it, coming from all levels of the organization, as well as major business thinkers. Economist James Bessen says that when software is core to business operations, it tilts the playing field in favor of those who use it most effectively. DevOps is one way to do this.

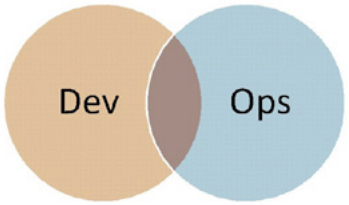
DevOps creates a more agile business that can respond more quickly to changing customer needs, helping ensure the company adds value and grows. At a developer level, we all want to work for cool companies like Spotify and Netflix that are built on DevOps and use technology as a business differentiator. And make no mistake, these companies know that encouraging a DevOps culture attracts talent that, in turn, helps them deliver more value to their business.

However, a major point about many of these cool companies is that, as digital-first businesses, they've grown up with DevOps. For those who work in established, more traditional organizations, there are three key challenges to face when implementing DevOps:

- **A need to overcome departmental silos:** Silos inevitably create issues, like development not communicating with operations, projects competing for resources, and teams jealously guarding information. This leads to problems when updates are deployed, and a blame culture emerges
- **Clashing incentives:** If your individual or team objectives (and bonus) are based on specific targets, it's human nature that this is what you'll focus on. That doesn't normally fit with the more inclusive, open vision that DevOps requires. One team, for example, may want to focus on one feature, while another thinks a different feature is important and, in the end, both will be delayed
- **Competing priorities:** Different roles have different priorities that can appear to be in opposition. For example, a database administrator or other member of an ops team is charged with keeping systems up and running and minimizing downtime. That means they're going to be suspicious of faster development approaches that might lead to performance or dependency issues conflicting with their own goals and targets

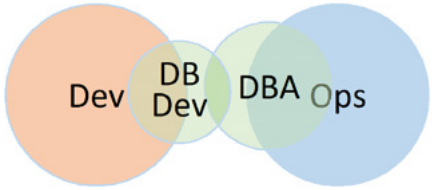
Overcoming these challenges is about building the right type of DevOps team to suit your business and people. This will vary depending on your culture and on factors such as your product set (fewer products means fewer silos), the effectiveness of tech leadership, the appetite and ability to change, and your capacity and skills.

The excellent Team Topologies book and website gives some examples of the types of teams that can help you successfully introduce DevOps—and the antipatterns that undermine them. Of the nine examples they list, some of the most common models I've seen include:



Dev and Ops Collaboration: Think of this as a perfect Venn diagram, with dev and ops overlapping just so. While this is highly effective, embedding this model fully does require strong technical leadership and an open culture.

Fully Shared Ops Responsibilities: This is more of an eclipse, with little or no separation between dev and ops teams. This model tends to appear in digital-first businesses that have grown up around a single product and is difficult to sustain in more traditional organizations.



Dev and DBA Collaboration: This aims to bridge the gap between developers and database administrators (DBAs) by incorporating a database capability from the DBA team, complemented with a database capability (or specialism) from the dev team. This gives a more holistic view of the database and thus aids communication and DevOps flows.

> *The biggest enemy of productivity is noise—not noise as in a loud working environment, but noise as in distractions.*

In my experience, whichever model you choose, there are seven ways you can maximize your chances of DevOps success.

1. Allow autonomy

Empowering teams is one of the key tenets of DevOps. That means giving them the freedom to decide how they build software, rather than micromanaging them. Clearly this isn't a recipe for anarchy; they still need to be held responsible and accountable, and to ensure their systems thinking and workflow matches that of other teams. It's more about letting them agree among themselves how they're going to work.

Will they be pair programming, for example, or using formal code reviews? Who are the best people to work on different elements of the project? If they're using branching, how often will new code be committed to the shared repository? The answers to questions like these will give the team real ownership of the project, while at the same time improving the way they work.

2. Improve productivity

The biggest enemy of productivity is noise—not noise as in a loud working environment, but noise as in distractions like telephone calls, emails, meeting requests, meetings, questions from other teams, "urgent" messages on channels like Slack … you get the picture. All the stuff a normal working day throws at you.

The problem is that while this noise might appear to improve communication, it can also drive down the quality of code. Every interruption and unexpected diversion introduces stumbling blocks to the flow of coding, which can result in errors.

Solutions are to minimize meetings (and the length of meetings), introduce "no-interruptions time" on calendars so team members can focus on only the work, and have one person on the team each day handling any questions or problems that come up.

> *Changing your culture and breaking down silos is at the heart of successfully adopting DevOps.*

3. Require transparency

Information sharing and transparency is vital to breaking down silos and encouraging collaboration. Every team member should have access to source code, for example, so they can see what's been done before, and the backlog, so they know what to prioritize next. Similarly, metrics should be shared, as well as information about incidents and lessons learned. This openness around code, knowledge, data, and best practices will bring walls down and contribute to real teamwork.

4. Encourage diversity

DevOps brings together people with different skill sets, and it only delivers results when you embrace this diversity. The alternative, often called groupthink, is a natural result of people congregating around others with the same mindset, but it can lead to teams either heading in the wrong direction or going nowhere because no one wants to upset the applecart.

Teams made up of people with diverse skills, styles, and ways of thinking lead to higher productivity because strengths in one area are balanced with strengths in others. This is particularly important with the rise of full-stack developers and the need for everyone to have a variety of skills.

5. Learn from failures

In the slow-moving, silo-based model of working, people are often afraid to shout out when they make a mistake or to share any lessons they've learned because it might hurt their careers. This is the antithesis of DevOps and is the perfect way to discourage innovation and prevent the kind of creativity that results in great code.

Instead, DevOps calls for recognizing mistakes and seeing them as an opportunity for everyone to learn, rather than an issue to be blamed for. A good approach is to embrace blameless root-cause analysis techniques and to enable constant improvements.

6. Recognize your peers

Everyone likes and responds well to justified praise, so make sure that the whole team recognizes and celebrates achievements. And this isn't just about the consistent top performers; everyone in the team has a different level of experience and skill, so praise their major accomplishments equally.

7. Have fun!

You can't expect to build a team by just bringing together a group of people from diverse backgrounds. You need to encourage a team ethos through bonding sessions that enable your people to get to know their colleagues as individuals in a less formal, more social environment. That will help team dynamics and underpin openness and understanding.

Changing your culture and breaking down silos is at the heart of successfully adopting DevOps. Move beyond thinking about technology alone and look at the people side of the equation. Then you'll be able to create a successful team that delivers the benefits of DevOps, whatever type of organization you are.

DevOps in the Trenches: Get Started with Metrics

By Jeffery Payne

While it is nice when an enterprise recognizes the value of getting everyone working together in an end-to-end value stream, the reality is that's not where most DevOps initiatives start. Often, one particular silo decides there is value in working more closely with others and seeks ways to do so.

I call this "DevOps in the trenches," and while it's not ideal, it is a way to get DevOps bootstrapped and gain some benefits from a more collaborative delivery process.

Here are some tips for how to get started doing DevOps in the trenches, with key DevOps metrics to help.

If you are in software development, work with testing to decrease the cycle time it takes to build and test changes. Benchmark how long it takes for a change to get through your build and test process, and then brainstorm with your testing organization about what you ultimately want your cycle time to be.

Use this metric as a forcing function to discuss progress, either during one of your daily standups or at a quick meeting each week. Help improve this number by putting in place a robust continuous integration capability that incorporates appropriate testing (static and dynamic) into the many builds you should be doing each day.

If your test organization does not have automation skills, dedicate time for someone on your development team to serve as a software development engineer in test (SDET), and help them create a maintainable regression suite that can be run frequently. Also build a simple smoke test for your application that allows testers to validate that a new build works well enough for them to spend time testing it.

If you are in software testing, work with operations to decrease the failure rate of your application in production. Often a root cause of these failures is the manual, untestable, poorly documented procedures for setting up production environments and installing and configuring an application for use.

Benchmark how frequently such deployments fail in the various environments your application is set up in—QA, staging, and production. Ask operations if you can "test" these deployment procedures and more clearly specify the steps involved.

Since testing production deployments can impact production quality, do these tests in any production-like environment, like staging. Then have a tester sit with operations the next time they do a deployment and observe how testable the production deployment process is.

Meet regularly with operations to discuss the deployment failure rate you are tracking and other ways to improve the testability of your deployments.

If you are in operations, work with development to decrease the time it takes to fix defects found in production. Doing so will begin to reduce the mean time to repair (MTTR) an application when it fails.

Benchmark how long it takes tickets to be closed, and use this number to frequently discuss with development ways to better collaborate to restore service quicker. Evaluate your ticketing process and look for inefficiencies, points of confusion, and queues where progress is blocked behind a process bottleneck, and seek to reduce or remove them. Invite someone from development to a weekly meeting to discuss outstanding tickets and progress. Review the time it is taking to close tickets, whether it is trending up or down, and other ways to improve the process.

How Continuous Testing Is Done in DevOps

By Junaid Ahmed

The adoption of DevOps practices is dramatically increasing throughout many different industries, mostly due to companies recognizing the numerous benefits DevOps is able to deliver.

DevOps does speed up your processes and make them more efficient, but companies that solely focus on speed and ignore quality aspects are likely to suffer a huge blow. Teams should focus first on quality by reducing defects and bugs before working on speeding up their operations.

DevOps is all about better enabling the software testing process to deliver quality results within a shorter time frame. Consequently, quality assurance is an important aspect of the DevOps methodology. Integrating QA within DevOps helps companies focus on giving their clients quality software before it ships.

Integrating QA within DevOps also plays a vital role in managing risks by ensuring the application is robust and stable throughout the development process. Continuous testing as part of a DevOps method helps detect bugs quickly, when they are easier and less expensive to fix, ensuring your application is fit for usage and enhancing a good user experience.

QA should not live outside the DevOps environment; it should be a fundamental part. But if your DevOps ambitions have started with only the development and operations teams, it's not too late to loop in testing. You must integrate QA into the lifecycle in order to truly achieve DevOps benefits.

Why Should You Perform Continuous Testing In DevOps?

Software projects, websites, and applications are not static. They require regular updates and real-time changes to fulfill all the set requirements of the clients.

These changes used to be time-consuming and perilous, but now they can be attained more easily through continuous integration, continuous deployment, and continuous testing.

DevOps allows software testing teams to easily upgrade and deliver various products without interfering with their quality. That's why most DevOps enterprises begin with the adoption of continuous integration practices: to ensure that everything works together.

Continuous testing involves testing a software application beginning in its early stages and automating testing throughout the development lifecycle. This helps the team examine the quality of the product at every stage of the continuous delivery process. And this process is not limited to only testers and developers; it also involves the contribution of stakeholders, operations, and even the customer. Continuous testing is an integral factor in the DevOps equation.

If continuous testing is performed properly, it delivers quick and uninterrupted insight into the quality of every new build of your software. This information can help you analyze whether the application is prepared to go through the delivery pipeline.

For example, when the code in a source code server like Jenkins is verified by developers, a set of automated unit tests is executed in the continuous process. If the tests don't pass, the build will be rejected, and the developers will be notified about it. If the tests pass, the code will be sent to the QA servers for functional and load testing. Then those tests are executed in parallel, and if they pass the build, the application will be deployed in production.

Key Points for Continuous Testing Adoption

Before you begin implementing continuous testing in your team, there are a few points to keep in mind.

The idea of continuous testing is to implement testing early in the software development lifecycle and at each branch involved in your CI/CD (continuous integration and delivery) pipeline. So before you transition a code change to the production environment, you should already have it validated at the staging ones.

This could be challenging, as you need to make sure that all your staging environments are exact replicas of your production. This requires more resources, bandwidth, and infrastructural cost. And even if you do have all the changes pushed to the staging environments, you can't be sure about pushing them to production just because they worked in the staging environments, as your production web application will usually be facing a considerable amount of user interaction. The fact that there is more web traffic in your production environment compared to the stage environment is one of the common things testers often forget. If you are short on the resources and investment required to maintain different stage testing environments, then continuous testing is probably not a good idea for you.

You also must be ready with your tools arsenal. You need to have the right automation tools on board for effective implementation of continuous testing in DevOps.

This means you need to have the tools required for each layer of the test automation pyramid—UI testing, API testing, and unit testing—because you can't expect one tool to cover everything for you. For example, if you are performing automated browser testing, you would require unit testing frameworks to validate your code changes at an earlier stage in the continuous deployment pipeline. However, at a later stage, you would need an end-to-end test automation framework.

> *You need to have the right automation tools on board for effective implementation of continuous testing in DevOps.*

In continuous testing, you are testing a single change on multiple test environments before deploying into production. A release may contain a bucket of feature changes, so you will be testing numerous code changes at multiple test environments, and for each code change you also need to perform regression testing on each stage environment. All of this could be time-consuming unless you know how to put parallel testing to best use. Many automation frameworks can execute multiple test scripts simultaneously, which will speed up the continuous testing through your CI/CD pipeline.

Also be sure to recognize false negatives and false positives. They are more common than you may know! When you execute a test automation script, it may show an execution error even if the system is working fine, or the automation testing script may show the test execution as successful even when the system met with errors. While either is dangerous, false positives can be more devastating because you believe everything is good to go and then you get an outage.

Plan your rollbacks thoroughly. Even if you use all the best practices for continuous testing and CI/CD, there is no guarantee that the build won't break in production. Always be ready for the worst scenarios. Make sure your data is backed up before you push changes to production. In case things go south, you can roll back to the previous production version quickly and perform a round of smoke testing to ensure the web application functions well again.

> *Though automated testing techniques are advanced and fast, they also require continuous human intervention.*

Keep in mind that when you roll back from the production environment, you can't just validate your pre-production situation and commit the changes again. You need to evaluate the entire pipeline to make sure there are no loose ends, as well as all your test environments, because you don't want to postpone the same release cycle twice.

Integrating QA and DevOps

The testing and technical teams should work hand in hand to ensure quality throughout every step in the software development lifecycle.

When integrating QA into DevOps practices—known by the term "QAOps"—the testing team should adapt and adopt certain quality processes:

- They should strive to detect bugs at the earliest point in the development lifecycle and prevent potential bugs from reappearing in the production cycle
- They are responsible for highlighting issues in the process and recommending necessary changes
- They must ensure that all the environments required for testing are standardized with automated deployment
- Apart from finding and preventing bugs, they should also focus on improving the overall quality of the product

Some people in the software industry think the requirement for QA is decreasing with the rise of DevOps due to automated processes. But that's not completely true. Though automated testing techniques are advanced and fast, they also require continuous human intervention. Companies that lack enough QA professionals and resources are not likely not to achieve their customers' requirements, mostly when it comes to updates and continuous changes.

DevOps was created to make developers think in synchronization with software testers, not to replace them. Software development processes are becoming faster through continuous deployment to meet ever-growing customer demands, and integrating QA with DevOps can help you fulfill all your objectives.

The integration of the QA process into DevOps also helps you manage various risks, making sure the end results are robust and more stable. QAOps is like a fitness regimen for software, as it makes it easy to detect bugs frequently and on time so you can tell when your applications are fit to run.

There are a couple of patterns for integration of QA into DevOps that teams can adopt:

- Try conducting an exploratory test before any new feature is merged into the master codebase. The tryout tests are designed to ensure the system is adequately covered to deliver accurate results
- Before you transmit a code change, make sure that your QA, developers, and other stakeholders involved in the continuous testing process are ready with their testing checklist. This checklist should include all the valid and invalid test scenarios they need to consider along with the expected behavior over specific test environments

Every business venture, company, and enterprise operates differently, so the ways of adopting QAOps may also differ. But the two suggestions above can be implemented across any software team.

You cannot deliver a comprehensive and quality service without a QA testing strategy, so QA is essential to the DevOps process. Integrating QA processes into DevOps operations helps both the testing and technical teams handle dynamic software environments and situations, deliver at speed throughout the CI/CD pipelines, and ensure the quality of the product—as well as customer satisfaction.

Shifting Right Offers New Possibilities for Agile and DevOps Teams

By Jan Jaap Cannegieter and Tijs Latiers

The shift-right concept originates from testing. But agile and DevOps teams also can use it to improve their systems and service to the client. However, there is a complicating factor: Different people have different explanations for what shifting right is.

Let's look at the different forms of shifting right, what the potential benefits are, and who should ideally be involved in your shift-right process.

The Different Forms of Shifting Right

The idea of shifting left, or testing earlier in the development lifecycle, has been around for a while. Analogous to this is shifting right, or performing test-like activities later in the development lifecycle and beyond. This includes getting feedback from the users.

We have identified three different forms of shifting right.

The first is testing in production. Some people, especially testers, start to panic when they hear that phrase—the client could be confronted with bugs! But there are definitely situations in which testing in production can be a good option.

First of all, testing in production is mostly on top of unit tests, functional tests, and nonfunctional tests, so we already know the product is working. But do the users use it in the right way? Can they find the new feature and understand it? And is it working on all devices used by our customers? These are the questions that can best be answered by testing in production. At the same time, we can monitor the services we deliver in order to make sure we deliver them correctly.

A lot of organizations already do a relatively limited form of testing in production. After a deployment, or when the organization goes into production several times a day, the testers make sure the system is available, the interfaces are up, and the functionality is working. Some organizations "follow" the first transactions though the system, sometimes even continuously.

We sometimes get questions about whether this should be called testing or monitoring, but we think this discussion is irrelevant. One way or the other, the team is responsible for these activities, so they should be done if they add value. The main goal is to be sure the system works properly, and it's undeniable that testing skills are useful for activities like this.

Case 1

A big online web shop scheduled a course to be delivered. The course was planned for the day before a new release was to be deployed, and the team members were supposed to have time. But the course day was not a success.

The new functionality was a shop-in-shop on the main web shop, the first time the organization deployed something like this. Because of the complexity and the inexperience with this kind of functionality, the DevOps team was more focused on the deployment. On big screens the team members could see the first visitors going to this new part of the website and ordering products. The deployment was not fault-free, and the team had to correct some bugs and deploy new versions immediately.

At the end of the day some minor things were disabled in production, but the main process worked well. We had a drink and rescheduled the course.

> *Finding defects is not the main goal of shift right; measuring the extent to which the intended value is realized is.*

Another form of shifting right can be pretty close to testing in production but is not necessarily: A/B testing and canary testing. With A/B testing, the team develops two solutions for a feature and presents both to the users, who then give their opinions on the best option. This can be a group of user representatives, such as user acceptance testers, or it can be real users in production. Canary testing is making a new feature available to a small number of users, with the team monitoring how they use it. Both A/B testing and canary testing could be done in production, but they don't necessarily have to be.

Drawbacks of A/B testing and canary testing in production is that it is sometimes difficult to ask the user what they liked more, simply because you sometimes don't know who the user is. And in the case of A/B testing in production, every user sees only one solution, so they can't compare it to anything. Sometimes you can measure things like conversion rate or the number of times a process is completed, but this is not always possible.

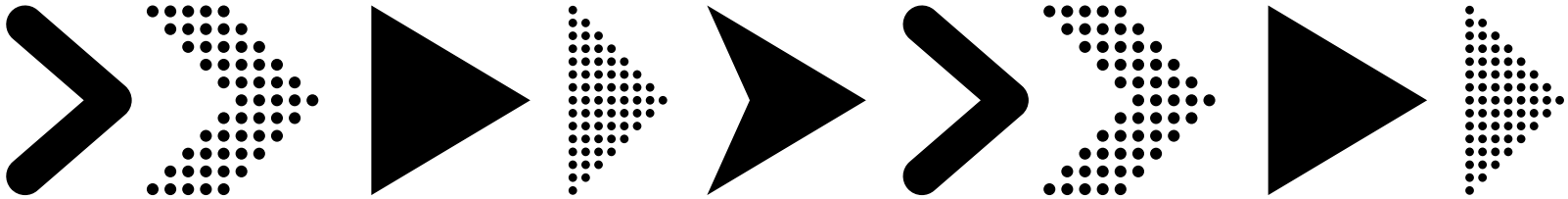
For this reason A/B testing and canary testing are sometimes done in a user acceptance test, where the testers see both solutions (A/B testing) or see the old and the new feature (Canary testing).

Case 2

An A/B test was performed with a controlled group of users. This was not done in production, but in a user acceptance test environment. We gave half the users solution A and half the users solution B. After they tested the new features (which included much more than this A/B test), we asked the users how they liked the solution by means of some questions they had to rate on a scale of one to five.

The two solutions were almost equally rated, and the general idea was, who cares, they are both good. So it didn't really matter which solution we chose. But we spent a lot of time designing, developing, and testing the two solutions that could have been spent on building another feature. What we learned is that A/B testing only adds value when you have two fundamentally different solutions. Otherwise your time could be better spent flipping a coin.

The third form of shifting right has to do with measuring the effect of a change—not only directly after deployment, but over a longer period of time. Is the conversion rate higher? Do more users stay with the process until the end? Do more users use the feature after the initial launch? Is the intended value delivered with the system or as an adjustment to an existing system?



This form of shifting right is done in the operations phase and is about measuring relevant data and translating it into valuable information for the team and other stakeholders. The development of a feature costs money, but some organizations don't bother to measure whether benefits are realized from the feature. This is what this third form of testing in production is all about.

Some people say that looking back on whether benefits are reached is not necessary because the feature is already in production, but we don't agree with this. DevOps and agile are about learning in order to do things better in the future, and the information we get out of measuring the effect of a change can help us to estimate and prioritize changes better. System development is not a one-time project anymore, but a continuous activity.

Case 3

A development team was responsible for realizing a web application for a marketplace for specific products. There already were some comparable marketplaces available online, but this new one had some unique selling points. Before development was started, the team defined the minimum viable product (MVP), along with measurable benefits in terms of market share, minimum number of visitors, minimum number of transaction,s and minimum number of providers.

This set of measurable benefits and the definition of the MVP was important guidance for both the product owner and the team. Benefits were measured after implementation, and about three months after the deployment of the MVP, the goals were met. This generated enough funds to develop the marketplace further.

It's All about Value

Regardless of the form your shifting right taks, the aim should be measuring the value of the system or the change for the customer. Finding defects is not the main goal of shift right; measuring the extent to which the intended value is realized is.

Customer value is not always easy to predict up front, and in order to evaluate it, we need feedback—feedback from real customers, maybe in production, maybe before we go to production, or maybe after a longer period of time. In this way shifting right is essential in the inspect-and-adapt cycle.

The cycle of inspecting and adapting is a useful practice in DevOps and agile. In the Scaled Agile Framework (SAFe) in particular, inspecting and adapting is a significant event, held at the end of each program increment, where the current state of the solution is demonstrated and evaluated by the train. According to this definition, inspecting and adapting is limited to a product increment, but in our opinion, shifting right broadens the use of inspecting and adapting to user acceptance testing and production.

The inspect-and-adapt cycle focuses on the product, process, and performance of the team as well as on adjusting the way teams work to get more value. Seen this way, inspecting and adapting is a team activity, but shifting right adds value to the inspect-and-adapt practice from a customer perspective. And customer value is what it's all about.

Case 4

A development team had the task of migrating a big, monolithic system to a new, multilayer architecture. The maintenance costs of the old system were high and the system was hard to test, which led to a long time to market. Due to the amount of time needed for this migration, it was financially not possible to do the migration in one release.

The solution was to migrate in small, incremental steps. In order to decide which parts to migrate first, the team installed a logging system that gave information about which functional parts were used most by the users—that was the inspect part. Based on the analysis of the user data, which was gathered for a few months, it was easy to determine what the MVP was and to make a user-driven roadmap for the migration—this was the adapt part.

The main lesson learned here was that gathering this kind of information is easy to realize and essential for making well-founded, user-driven decisions.

The Tester in Shifting Right

Before looking at the question of whether the tester could have a role in shifting right, let's look at another question first: Can all testing be done by means of shifting right?

The answer to that is no. Testing is also finding bugs, investigating the system, and determining how well the system is working. Shifting right will always be on top of unit testing, functional testing, and nonfunctional testing.

The second question is whether the tester should execute or organize the shift-right approach. The theoretical correct answer would be no, because there is no tester in agile or DevOps. But we all know that in practice most teams have a team member with testing skills, at least when a complex product is being built or the quality of the product is important to the customer.

> *Shifting right will always be on top of unit testing, functional testing, and nonfunctional testing.*

In our opinion, every team member could have a hand in executing or organizing shift-right activities. But at the same time, we often see that testers have the right skills and focus to do these activities. Good testers have analytical skills, and they know how to investigate things. The reporting about the benefits of a feature has similarities to modern test reports. The transition for the tester when it comes to shifting right involves measuring and evaluating the value after the usual tests, or even after the system is in production to their work. If possible, pairing a tester with an operations or DevOps team member will boost the speed in which the team can shift to the right.

Even if testers are very involved in the shift-right activities, they can't do it alone. The tester will have to involve different people from inside and outside the team, including the product owner or business owner, other team members, and clients. Ideally, shifting right starts with a stakeholder analysis in order to decide who should be involved, whether shifting right should be used in the inspect-and-adapt cycle, and how the outcome will be reported. This way the right people will be involved in the right way.

Shift Right to Serve Your Customers

Shifting right as part of the inspect-and-adapt cycle offers agile and DevOps teams techniques to measure the value of the system. Teams should consider each of the possible ways to shift right and their applicability to the team's specific context, then proactively take the initiative and make the shift-right approach a part of their job. When the right stakeholders are involved, shifting right can help teams serve their customers better.

Embedding Performance Engineering into Continuous Integration and Delivery

By Anjeneya Dubey

In the world of continuous integration and continuous delivery (CI/CD), where technology companies are rapidly deploying code and infrastructure changes so their application can gain a competitive advantage, the importance of ensuring good performance has increased tremendously. While functional and unit testing are relatively easier to integrate into these processes, performance engineering has typically raised more challenges, especially in analyzing the results and pass/fail decision-making.

My company is currently going through this transformation, and embedding performance evaluation of the services as part of CI/CD is a must. Here are some of the challenges we faced and changes we made across the board to make this happen.

Cultural challenges: If you want to evaluate the performance of every line of code that is checked in, culture is the most important yet difficult change required. There were several areas where we struggled. Performance was an afterthought; teams were focused on functionality more than performance. There was also a lack of understanding of performance and scalability needs of the product, and performance engineers were not part of the agile teams and weren't included in the agile ceremonies.

To help alleviate these challenges, we needed to shift the performance lifecycle to the left. We empowered our developers to test, allowing good performance to be baked into the code.

Process changes: We also had to make significant changes in our Scrum process to create awareness of performance. We made nonfunctional and performance requirements part of the functional requirements, which required us to have scalability needs, API contracts, and service-level agreements clearly defined at the story level. This enabled us to learn what "performance-ready" services mean to our customers.

We mandated performance as part of the definition of "done" for sprints and included performance results during sprint demos with all stakeholders.

Tools and accelerators: At this point we had to make sure we had the right tools and accelerators to create, execute, and analyze performance tests at the sprint level, ready to be integrated into the CI/CD pipeline.

We built a performance engineering platform that manages the testing cycle of our services, leveraged functional test scripts to collect browser-side response time, created self-contained tests that create test data before each test and destroy it after each test, developed a central repository for all performance metrics, created an algorithm for dynamic thresholds for pass/fail of each build and individual REST APIs for test status and decision-making, and automated defect creations and notifications through our performance engineering platform.

All our services are in the cloud, making it easier for the performance environment to expand and contract as needed and adding flexibility to build and kill new environments. Such benefits can go unnoticed, but they are a huge driver to implement our test framework that can scale to address business needs.

Making DevOps Evolution Happen

A Conversation with Helen Beal

By Owen Gotimer

Helen Beal, DevOpsologist at Ranger4, chats with TechWell community manager Owen Gotimer about making your DevOps evolution happen, micro-bonus programs, and the neuroplasticity of squirrels.

Owen Gotimer

The first question I want to open the discussion with is, what is DevOps?

Helen Beal

What a brilliant question. Yeah, it's a tough one, isn't it? And the reason it is so tough actually harks back to the origins of DevOps really, and its relationship with agile and ITSM, in particular. So agile, as everyone knows, has got a manifesto, and you can look at it on the internet, and ITSM has something called the ITIL, the IT Infrastructure Library, which is pretty well known. But both of those were codified in some ways and the guys that originated the DevOps movement tried very purposefully not to create similar things and have a similar definition. Because they wanted it to kind of be allowed to evolve in its own way. And that's been really powerful I think and absolute the right decision to allow the DevOps movement to grow and evolve and find new ways of talking about things. So it's allowed it to really move from something that was very focused initially around agile system administration, to a place where it is now which is where we're really looking at the end-to-end value stream and not just the end-to-end technology value stream, the end-to-end business value stream for a product or a service. So it is very hard to define. But ultimately, it's about delivering value faster and more safely. So it's that throughput and stability and balance around value outcomes.

What challenges do you face in trying to both deliver things safely and quickly?

Owen Gotimer

Helen Beal

With technologists, watching the automation piece is relatively easy, so the big barriers we often have are around culture. With them, they've got some leadership challenges, which again, are fairly common, where it looks like you've got leadership support, but they're kind of talking the talk but not walking the walk. So we have conversations about how to really tackle their concerns. Like many people they can't kind of come to terms with the fact that we can balance these two things. They're just seeing speed, and speed is a risk to their stability. So stop talking about deployment frequency, and actually start talking about approaches to protect production. So start talking about things like limited blast radius approaches, and architectural approaches, and focus a bit more on tools like application performance management and deployment automation and locating and things that really bolster that end of it.

As we get more and more distributed around the world, it's nice to be able to connect with people who aren't in the same geographical region as you are necessarily working with you on a day to day basis, and reward those people, your colleagues that are working on it with you on a day to day basis. So I love the idea of micro bonuses, and I can certainly see the value in not working towards that end of year performance review. But for a lot of people, that's probably scary, because traditionally, that's how it's gone. They've sat down at the end of the year for a performance review. What are some steps teams can take to get away from the idea that performance reviews are the best way to go about leading those kinds of bonus or incentives?

Owen Gotimer

Helen Beal

So fundamentally, what we're trying to do in DevOps and agile is move away from big batch to small batch processes, right? And whether we're talking about a requirements document or system or performance review or a funding model, they all could be big batch or small batch. So there's steps you can take. And if your big batch in terms of performance reviews is every 12 months, then maybe start making it every three months. So move to the rolling quarterly wave, as we often call it, in the finance model. That's what we often try and move to. And then you experiment with breaking it all down further. And if you're a very large organization, you don't actually have to do it with everybody, immediately. You can experiment with smaller parts of the organization. In technical terms, if we were talking about this, we'd call it a canary deployment, right? So you can do a canary deployment of micro bonuses or quarterly performance reviews in a part of your organization as well and test it out, experiment with that, inspect the results, get the feedback, and make a decision on what experiment to try next.

Do you see that a lot where organizations bring in DevOps teams, and how do you think that either helps or inhibits their ability to move through their journey?

Owen Gotimer

Helen Beal

Yeah, great question. Something I am quite passionate about, and I'm not alone. I can be a little bit of a purist, and I kind of don't apologize to that in some respects, because part of my job is to try and help organizations advance to the best pattern for them. So the DevOps team, it happens. There's a couple of problems with creating a DevOps team, but you can give some organizations the impression that DevOps is done. We've got a DevOps team, we've done DevOps. You can create another silo, and that creates all sorts of bad behaviors like handoffs. When the DevOps team pattern works, it's when they're seen as like a tiger team or an evangelizing team that are starting the DevOps evolution across the organization. So if they become this kind of miscellaneous bucket for work that other people can't really be bothered to do, it stimulates the whole growth of that pattern across the organization.

With DevOps, you often have those naysayers who aren't aren't ready to get on board, who don't think it's going to work. What do you do with the naysayers?

Owen Gotimer

Helen Beal

There comes a point in time where you're like, right,"this is what we're doing now, this is who we are. If you don't want to be on this bus with us doing this thing, then maybe you want to find another bus." So it sounds pretty harsh, but in their heads, they must be getting to the same place. We don't all fit everywhere all the time. The critic role is actually very useful to us because they're like the anti-mirror that tells us all the things we need to know about why people are resisting the change. So we should spend more time with them and really help them understand how it's going to better their lives.

READ THE FULL INTERVIEW

Each month a thought leader joins the TechWell Hub for a Slack Takeover. These chats give our community members an opportunity to ask questions, share challenges, and grow as software professionals by having conversations with an expert. Here's a sample of some of the questions and answers from a few of our DevOps-oriented Slack Takeovers.

TECHWELL HUB
A SLACK COMMUNITY

VISIT THE HUB

Q **@John Schultz**
DevOps seems to be a buzzword-ish sort of term that a lot of companies say they are doing, but I'm not really sure what it means.

A **@Lisa Crispin**
It is a culture of the whole team, including operations, working together to not only deliver tested code, but to be engaged with the code that is already in production, learning how customers really use it, and responding super fast to their issues.

READ THE FULL CONVERSATION

Q **@Tom Stiehm**
What do you see as the biggest challenges to adopting security practices into DevOps?

A **@DJ Schleen**
Great question! There are a ton of challenges to gain successful adoption. It depends on which team is trying to implement the practice. Is it the development team or the security team - and that differs between companies large and small. The biggest challenge is to ensure everyone is comfortable with change and can communicate with candor. Once the culture is there (or growing), the challenge is to find the best place in an automated pipeline to put security controls without sacrificing the speed of delivery and deployment.

READ THE FULL CONVERSATION

VISIT THE HUB

Q **@msowers**
Understanding security is extremely challenging. There are so many dimensions and so much to learn. How should someone get started learning about security?

A **@Larry Maccherone**
One of the guiding principles in my updated DevSecOps Manifesto is, 'Adopt a few key practices deeply and universally more than a comprehensive set poorly and sporadically.' The idea is to pick just a few things and focus on those before you spread. The absolute best bang-for-the-buck thing to start with is analysis for code imported, aka software composition analysis, or misnamed open source security.

READ THE FULL CONVERSATION

Q **@Jessica Romero**
How do you start injecting DevOps in a team who is not familiar?

A **@Nathen Harvey**
I think it depends on your perspective of the idea of "DevOps". Part of the problem, as I stated early this morning, is that there is not really an adopted, universally agreed on definition of that word. But selling ideas, DevOps or otherwise, starts with understanding what's important to the people you're selling the ideas to and challenging them to imagine a new world.

READ THE FULL CONVERSATION

Q **@Sam N**
Have you been in a situation where the environment didn't encourage DevOps adoption?

A **@Melissa Benua**
So my first instinct is to always try and get down to the root of the problem. Do they have an idea of why they need to implement DevOps, from a monetary standpoint? There are significant competitive, market advantages to companies who have strong continuous delivery / DevOps practices, such as increased market agility, increased market cap, increased dev satisfaction and retention, and increased time-to-mitigate for security issues.

READ THE FULL CONVERSATION

Additional Resources

MORE INFORMATION FOR SOFTWARE PROFESSIONALS

TECHWELLHUB
A SLACK COMMUNITY

The TechWell Hub is a great resource to get your questions answered, help others with problems they're stuck on, and engage with experts in software. Follow channels like #DevOps, #DevSecOps, and more.

JOIN HERE

NARROW YOUR SEARCH TO A SPECIFIC TYPE OF RESOURCE:

AgileConnection Community

AgileConnection brings you the latest in agile and DevOps principles, practices, and technologies. Check out articles and interviews from experienced software professionals and thought leaders, and join the community to gain access to member-exclusive content such as conference presentations, weekly newsletter updates, Q&A discussions, and more.

DevSecOps Articles

AgileConnection is home to thousands of DevOps software resources, including articles, archived *Better Software* magazine articles, conference presentations, and interviews with industry notables. Check out these DevSecOps articles to read about how to reduce risk, protect data, and build security into your DevOps pipeline from the start.

TechWell Conference Presentations

Couldn't make it to a TechWell conference to sharpen your security and DevSecOps skills and knowledge? TechWell conference presentations are available to AgileConnection members soon after conferences end. **Click here** to join AgileConnection and access conference presentations related to security and DevSecOps.

Interviews

Each year, TechWell interviews dozens of software professionals, including well-known thought leaders, seasoned practitioners, and respected conference speakers. **Click here** to read, listen to, and watch interviews with DevOps and security experts.

Agile + DevOps Virtual

In light of recent events, TechWell has morphed the popular Agile + DevOps West conference into a fully virtual experience this year! From the comfort of your own digital device, you will have access to all of the same great content and experts you have come to expect from an Agile + DevOps West conference.

DevSecOps Summit at Agile + DevOps Virtual

The DevSecOps Summit is a fully-virtual, multi-day series of first-person talks, giving an ideal perspective on how you and your team can enable faster application development with more rapid deployment to production while integrating security into your DevOps initiatives. Explore the program **here**.