

An aerial, high-angle photograph of a complex railway yard or station. The image shows numerous tracks curving and intersecting. A small, white, box-like control building with a flat roof and a set of stairs is visible on the left side. In the background, there are several multi-story urban buildings. A train is visible on the right side of the tracks. The entire image has a blue tint. The text "A Pragmatic Guide to Getting Started with DevOps" is overlaid in white, sans-serif font in the lower-left quadrant.

A Pragmatic Guide to Getting Started with DevOps

Contents

Is DevOps Right for You?	3
Dev and Ops on the Wrong Tracks	5
7 Signs of a Dysfunctional Process	6
Conclusion: Where to Start Your DevOps Transformation	24



Is DevOps Right for You?

In the application economy, every business is a software business. This is why DevOps is quickly becoming one of the most valuable disciplines for your business. It's focused on improving the quality and speed of delivering new apps to market. And it's about tightly integrating development and operations in order to do so.

This is driving businesses everywhere to take a second look at what they may have initially thought was just a buzzword – here one day and gone the next.

Now everyone is starting to wonder, “Can we adopt a DevOps method ourselves? And will it work for us?”

The average organization adopting DevOps sees a 20 percent improvement in time-to-market, a 22 percent improvement in software quality, and a 17 percent improvement in frequency of application deployments—all leading to 22 percent more customers and a 19 percent increase in revenue.¹

So, what can DevOps do for you?

¹ TechInsights Report: What Smart Businesses Know About DevOps, September 2013



What Is DevOps?

Before we decide whether we can use DevOps, we must define what DevOps is, and what it is not.

DevOps is not a product, or even a particular technology. **DevOps is a methodology** that unites the often separate functions of software development (Dev) and production and operations (Ops) into a single, integrated, and continuous process.

DevOps is about **breaking down the barriers between Dev and Ops**. It leverages people, processes, and technology to stimulate collaboration and innovation across the entire software development and release process. Dev and Ops must act and feel like they are a single team.

But DevOps is never finished. Like a symphony orchestra learning a new score, or a sports team that has reached the playoffs, Dev and Ops have to keep pushing, collaborating in the pursuit of perfection.

Dev and Ops on the Wrong Tracks

Dev is focused on faster innovation and doing new things. The mandate to Ops is about stability, control, and predictability. They often don't even report to the same places in the organization. It is like they are on two different train tracks. No matter how fast they go, they never meet.

Left to themselves, Dev and Ops will often struggle to talk to each other, much less collaborate, and will remain mired in manual processes. The result is employees who don't work well together, software that doesn't work reliably, and customers that are thinking about moving to your competitors.

No ifs, ands, or buts about it.



[Next
Section](#)

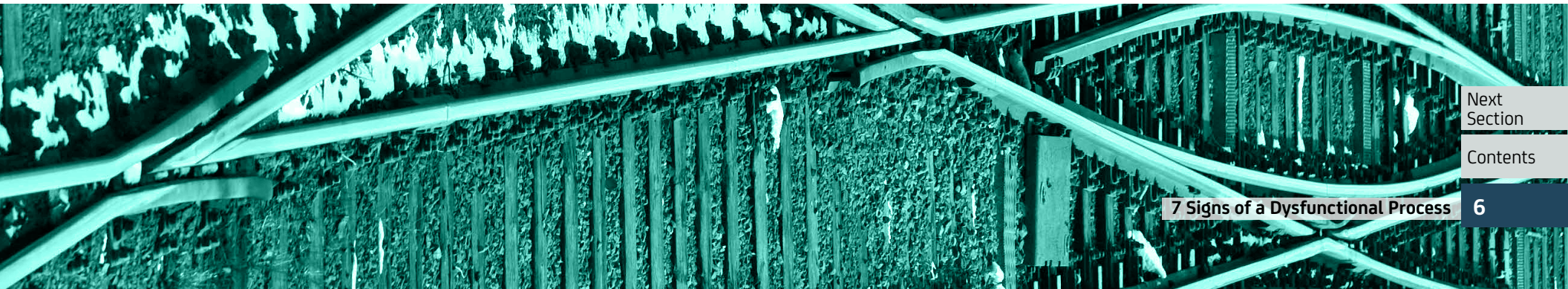
[Contents](#)

7 Signs of a Dysfunctional Process

Dev and Ops on board and heading in the same direction —can be difficult. But first, you have to recognize if your teams are on different tracks to begin with. Some of the warning signs of a dysfunctional process include:

1. You don't discover software defects until late in the lifecycle—or worse, in production.
2. You use Agile to speed development, but any gains evaporate once the app goes into production.
3. Your developers and testers are constantly waiting to access the resources they need, causing delays.
4. You can't pinpoint problems across development, testing, and production operations.
5. You see simple human errors wreaking havoc during development and deployment.
6. Development views their job as finished once the app is in production.
7. Anytime a problem arises, everyone starts pointing fingers to lay blame on someone else.

In the pages that follow, we will review some of the most frequent signs of a dysfunctional process, along with some pragmatic advice on how to move beyond dysfunction into full DevOps maturity.



Warning Sign:

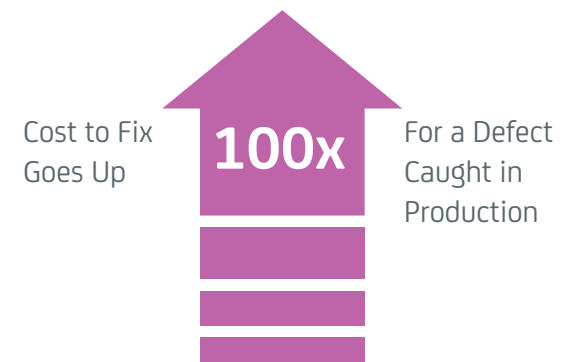
Software Defects Aren't Discovered Until Late in the Cycle

It is impossible to create complex software without defects.
The trick is to find them early in the process.



The cost to fix a defect goes up for each step along the classic waterfall approach to software development lifecycle. It can be one hundred times more expensive if an end-user finds a major defect in production than if you find it in development.²

Unfortunately, with the waterfall method, when the development schedule starts to slip and there is a hard deadline looming, organizations start to run out of time for testing. This situation is ripe for error, and the pressure to meet the deadline means that the application suddenly gets thrown over the wall into production, no matter how complete the testing.



²Software Defect Reduction Top 10 List, Computer, January, 2001. <http://www.cs.umd.edu/projects/SoftEng/ESEG/papers/82.78.pdf>

How to Get Back on the Right Track

Do Development and Testing in Parallel

This is made possible through a new technological approach called service virtualization that can mimic the real-world behavior of the production environment. The real infrastructure resources are not available to developers and testers, so this approach recreates the whole environment in a virtual environment. The back-end infrastructure, databases, servers, and the rest of the environment are available as a virtual service, so a large number of people can test a large number of components all at once without impacting each other or the production environment. Suddenly, development and testing can take place in parallel.

Even better, we can build automation of that testing, making the process even faster.



User Acceptance Testing

Development

QA Testing

Performance Testing

Production

Regression Testing

Integration Testing

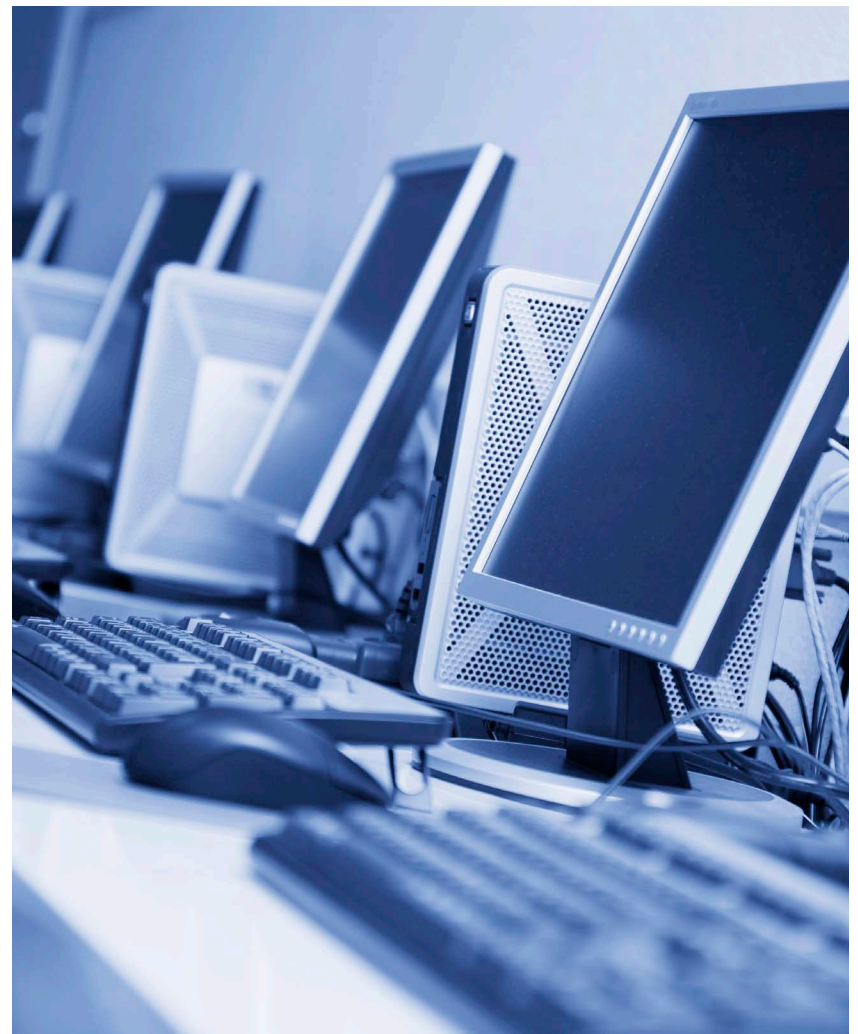
Next
Section

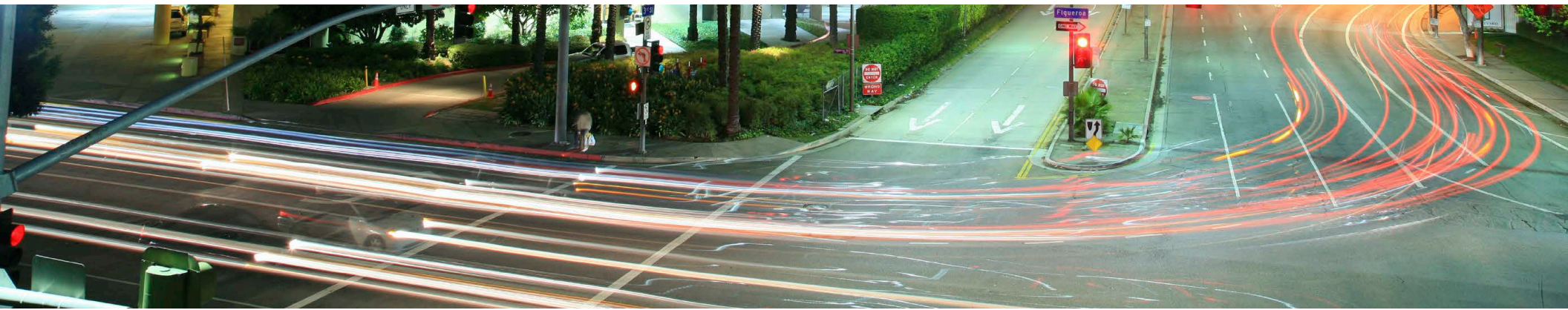
Contents

This ability to mimic the performance of the production environment is made even more realistic through collaborative data mining, where the performance information embedded within the production environment is “mined” to uncover hidden patterns and circumstances.

Data mining gives developers and quality engineers virtual service models that behave even more like the real world, bringing all the benefits of service virtualization with performance profiles mimicking business conditions and scenarios like the ones seen in production. Now, developers and performance engineers reduce their constraints and do so with an increasingly life-like model emulating behavior and the performance characteristics of real components in real-world situations as well.

Find the problems earlier, and you build better software, faster, with less effort.





Warning Sign:

The Speed Gains of Agile Evaporate Once the App Goes into Production

Agile development is a fabulous approach to speeding software development. **But building the software is only one step in getting an app into the hands of the users.**

Because of the lack of collaboration between Dev and Ops, apps often aren't ready for "prime time" when they arrive in production. Because Ops often hasn't been involved in development and therefore doesn't fully understand how to deploy the application to production, there is a lot of trial and error in moving into production, wasting time.

Dev alone doesn't get the app into the hands of the customer. No matter how fast and agile you make the development process, **it is difficult to speed time to market if nothing is done to speed time to production.**

[Next
Section](#)

[Contents](#)

How to Get Back on the Right Track

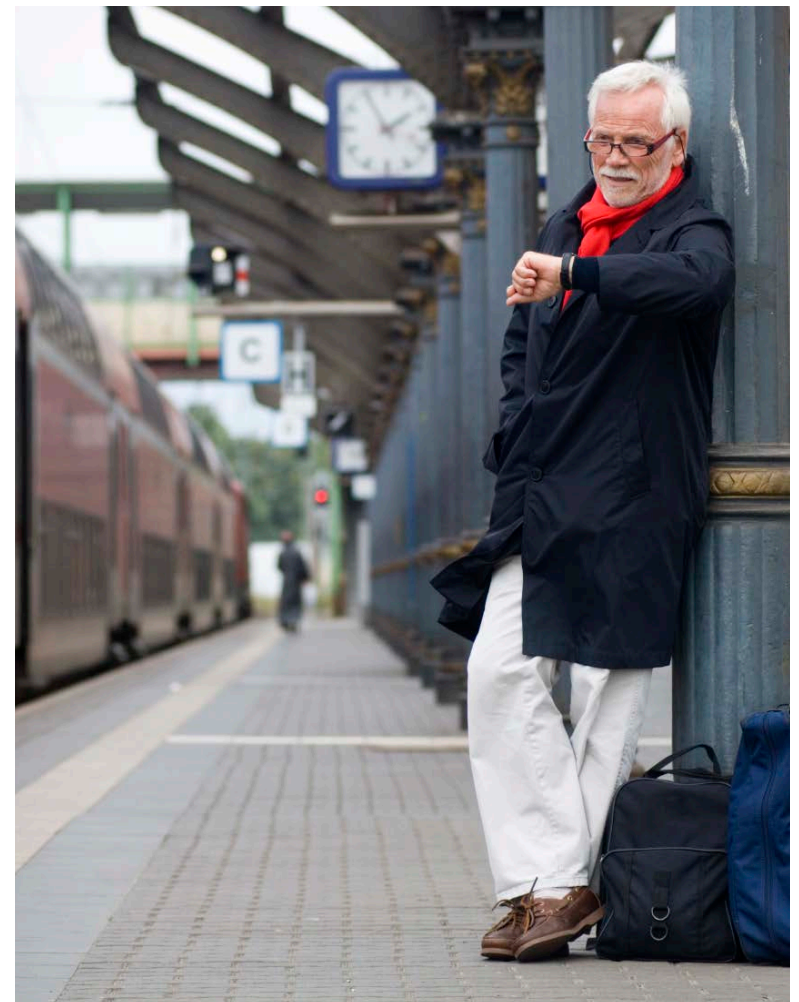
Get Ops Involved Early

Agile is built around an iterative process that responds quickly to market needs. The idea is to roll out frequent incremental changes rather than save all the changes up for a big splash. Ops is simply part of that market need.

Agile shops would never dream of developing an app without paying close attention to market needs. They should pay similar attention to the needs of the production environment. Development and testing should also take place in as life-like (production-ready) an environment as possible.

Another potential source of trouble is completing steps serially.

Even in the most agile shops, there are times when work is largely done in stages. Such serial processes can slow both development and deployment. **Teams should be working in parallel** as much as possible, and in places where you can't break things into parallel efforts, look for ways to move steps earlier into the process. There is no reason why Ops should be waiting for Dev to finish before it starts working on deploying the app; involve Ops early and plan for deployment during the development process.





Warning Sign:

Schedules Are Constrained as Teams Wait for Resources

Whenever a complex application is being built—whether it’s a front-end service for a mobile app or something for the integration layer in the middle of the overall architecture—it is probably making use of information pulled from other systems and services throughout the enterprise.

Developers and testers often can’t get easy access to these services. It could be that back-end systems, like the inventory system, are only available for test access once a week for two hours at a time. Or maybe there’s sensitive data in a production system that either shouldn’t or can’t be touched. All of those are serious constraints on the process of development and testing an application.

The CTO of one company once remarked,

“I can’t do anything until I have everything, and I never have everything.”

Next
Section

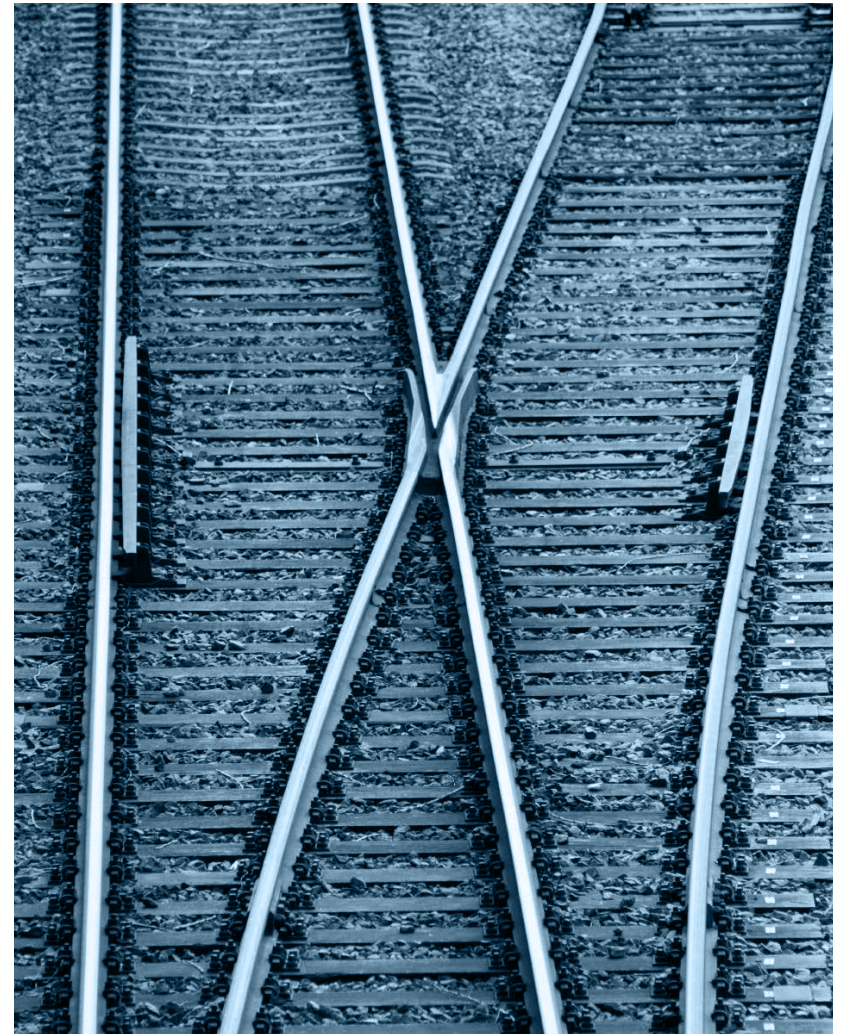
Contents

How to Get Back on the Right Track

Service Virtualization Can Help Address This

Because service virtualization can mimic the real-world behavior of the production environment it enables a large number of people to develop and test a large number of components at once without impacting each other or the production environment.

This removes the constraints that are holding back app development and testing in many organizations. Not only is the testing environment now a much more realistic reflection of real-world conditions, it also enables testing of different components to be done simultaneously. **Steps that once had to be done serially can now be done in parallel.** The entire development cycle **shifts left** as the constraints and bottlenecks are eliminated.



Warning Sign:

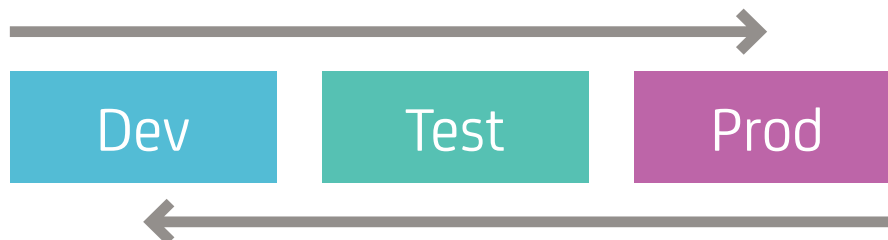
Problems Are Difficult to Pinpoint

Because of Lack of Collaboration Across Development, Testing, and Production Operations

Teams in Dev, Testing, and Production often work in very different environments and manage their work on different systems. This can make software defects difficult to correct.

Test results not predictive

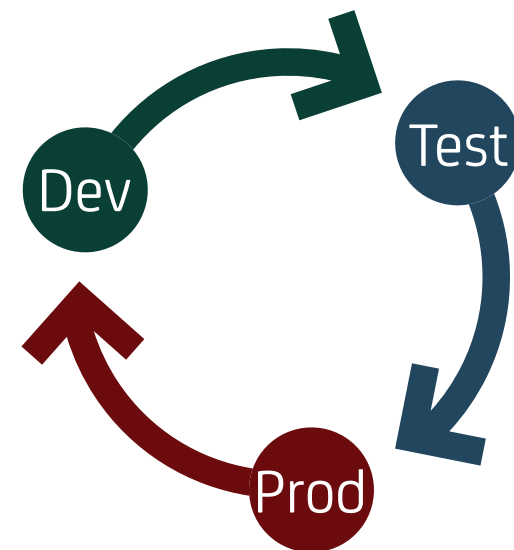
No opportunity to test under realistic conditions



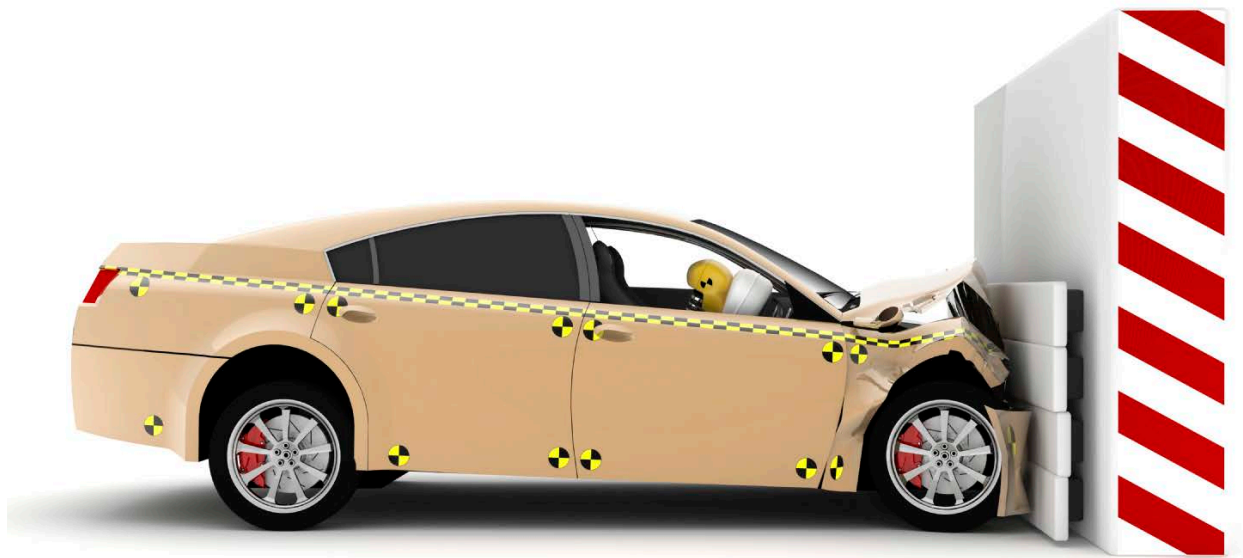
No sharing of feedback

No communication between

- Systems
- Processes
- People



Because development teams and testing teams often don't have reliable access to the production environment, they rely on test environments that, while they might offer the functionality of the production environment, lack the full scale of production. **These are not realistic testing environments** that duplicate the conditions found in production, so apps often fail when taken to production because the earlier **tests are not predictive of the app's real-world behavior.**





Second, because Ops, Dev, and Testing all use different systems to manage their environments, **the feedback loop** that should be in place between Ops and Dev **breaks down**.

Ops knows there are problems with the app, but unless someone in Ops bothers to tell Dev about the issue, Dev will never know about it. Dev cannot monitor the app in production itself, so it only gets a second-hand perspective on any problems.

How to Get Back on the Right Track

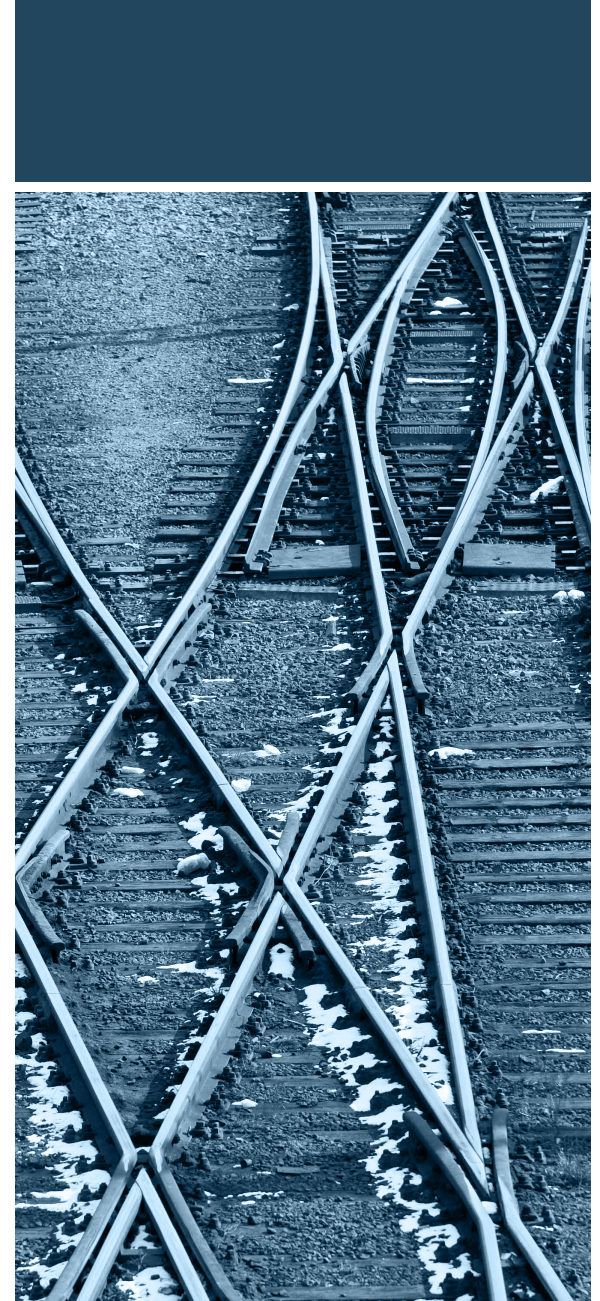
There are Several Ways to Attack This Dysfunction

First, recognize that Dev and Ops are a single team. Mash them together to ensure that the **people** work as a team. Even with this organizational change, establish a **process** to gather feedback and then track and share progress on addressing the issues.

You can take the next step and use **technology**. It's all well and good to address people and process, but if your systems are working at cross-purposes, it will be that much more difficult.

Technology can take several forms. First, **ensure that the systems** that Dev, Test, and Ops use to manage their workflow (and any problems) are **interoperable**, so that Dev has visibility into Test and Ops, etc. This can be done by integrating existing systems, but often it makes more sense to implement something that is designed purposely to create an integrated workflow across the entire process of development, test, and release.

It also makes sense to **put testing environments in place** that duplicate the conditions to be found in the production environment —such as service virtualization, backed by the detailed performance scenarios uncovered by data mining the logs of the production environment.. This ensures that any testing is **more predictive of the app's real-world performance**.



Warning Sign:

Human Errors Cause Havoc and Wasted Time During Deployment



Software release is largely a manual process in many organizations. Would it surprise you to find that **most software issues are due to configuration errors?** That is, there is nothing wrong with the application itself, but the software fails because it wasn't properly configured. When configurations are done manually, it is inevitable that there will be a mistake somewhere along the way.

To err is human, but to really foul things up takes a computer. Errors during deployment are especially damaging because the error has the potential to be replicated across thousands or even millions of servers, desktops, and mobile devices. **Testing won't help, because the configuration needed during testing is different** than the one needed in production.

Next
Section

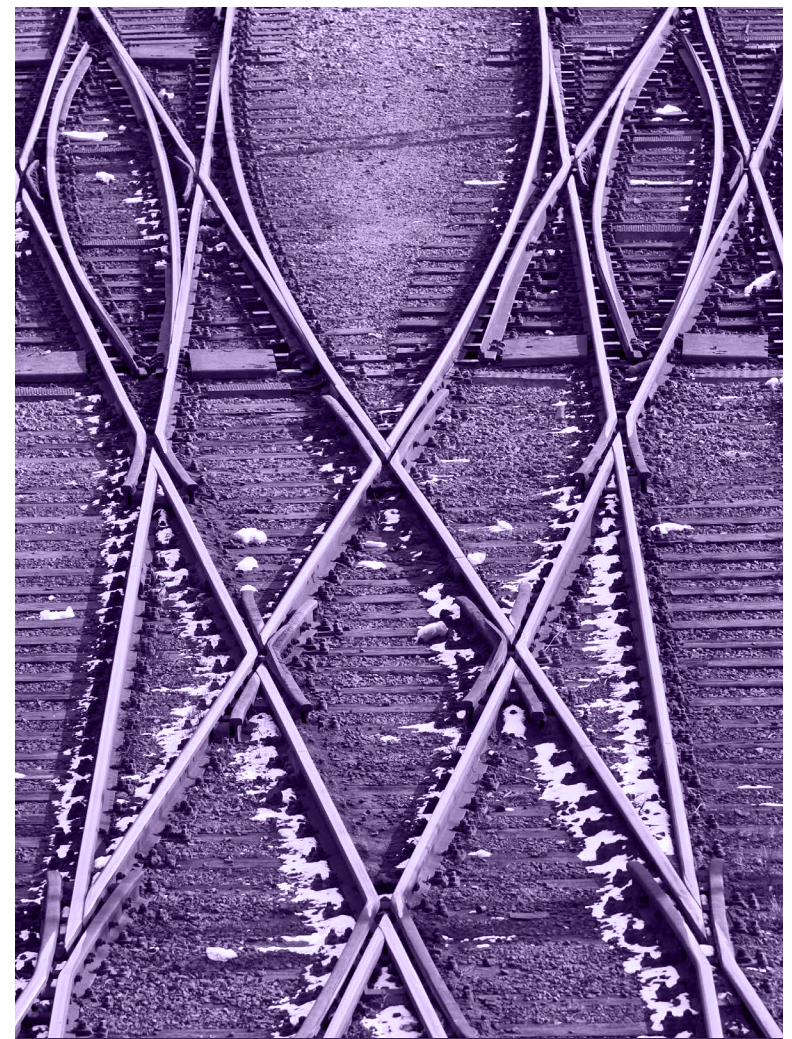
Contents

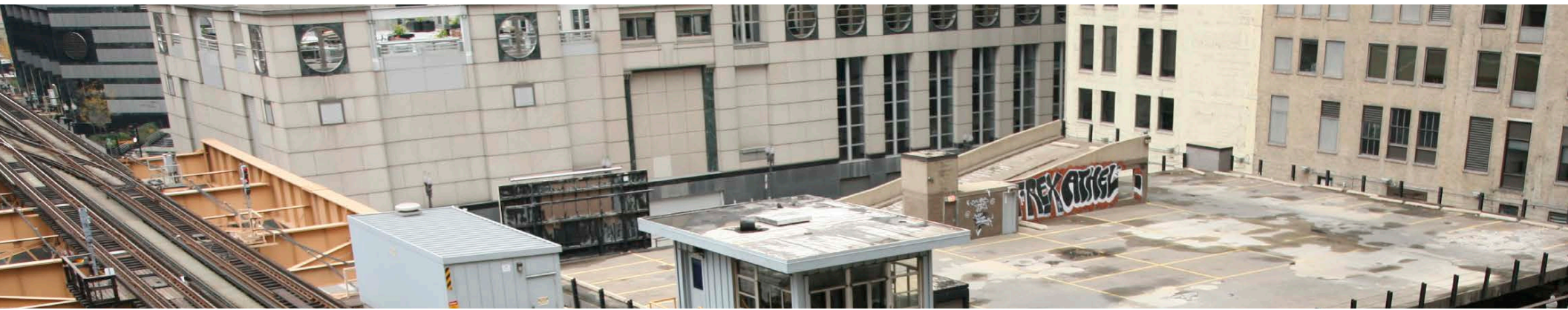
How to Get Back on the Right Track

Many organizations have attempted to solve this problem by relying on scripting. While this is certainly better than an entirely manual process, it suffers from many of the same problems as a manual process. Because of all the “moving pieces” and the configuration differences between environments, the scripts end up being as complicated as the systems they are trying to deploy.

You need more than scripting. You need systems. You need **tools that automate the release of new code into production, which include documentation** internally so that, especially with big teams, it actually enforces the DevOps mindset of collaboration, integration, and communication. This release automation enables you to ensure that a configuration that works properly during testing will be translated faithfully into something that will work in production. It eliminates human errors while simultaneously speeding up the whole software development lifecycle. This increases your readiness for continuous delivery, where apps are updated every few minutes, not in days, weeks, or months.

One of the biggest benefits of release automation is rollback, which allows you to return to a known good state if something goes wrong during deployment. With manual processes, and even scripting, rollback is a nightmare.





Warning Sign:

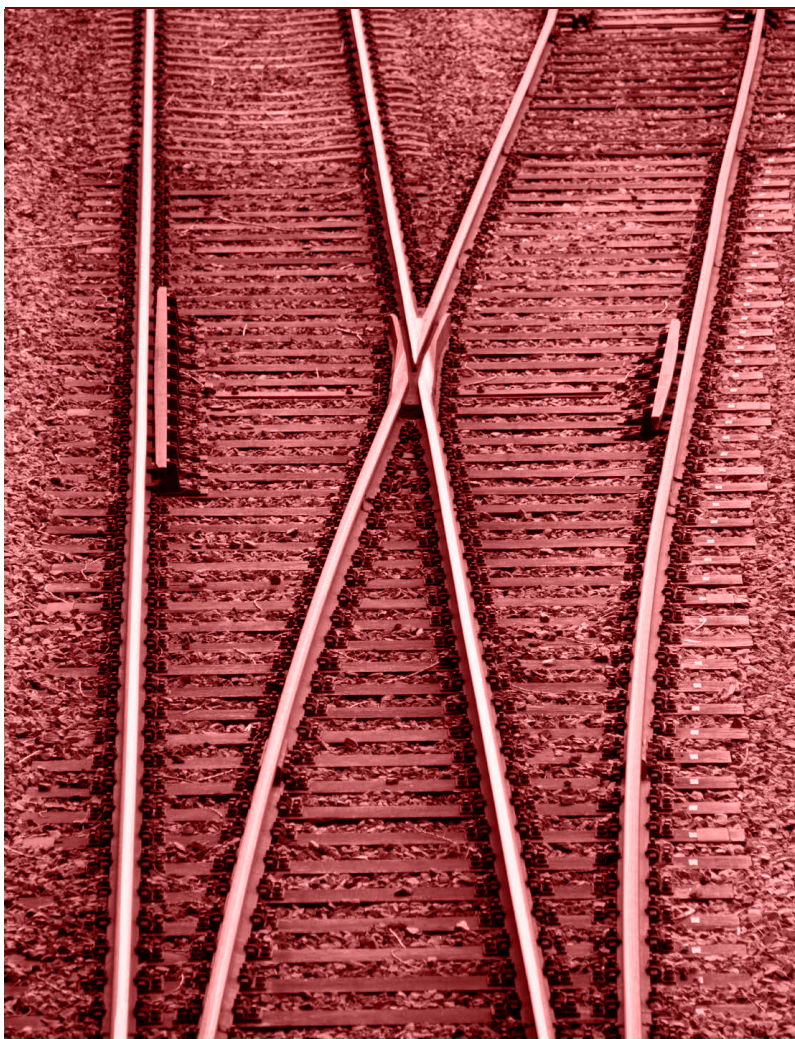
Development's Job Is Finished Once the App Is in Production

Traditionally, application development projects were monster efforts that took 18 to 24 months for the planning, coding, and testing. Then the application was turned over to production, the development team celebrated, and everybody went off to the next 24-month project.

If you are serious about DevOps, there is no better indicator of a dysfunctional process than having the development team spike the ball and leave the field.

[Next
Section](#)

[Contents](#)



How to Get Back on the Right Track

Part of what makes DevOps so powerful is that it enables you to release your applications into production on an almost continuous basis.

Instead of saving a laundry list of features for a big release, the new model is to be releasing new features continuously. The advent of the smartphone app has set expectations that apps will be refreshed frequently. If an app doesn't get updated every few months with one or two new features, people wonder if the app is being properly maintained.

With continuous release, the Dev team finishes one version and immediately starts on the next, adding new features based on user requirements and, ideally, guided also by feedback from Ops on how this new version is performing. Dev and Ops truly are one integrated, collaborative group now. Ops knows what is expected from Dev, and Dev can actually stream its own code into production using load-based tools with features like rollback and version control.

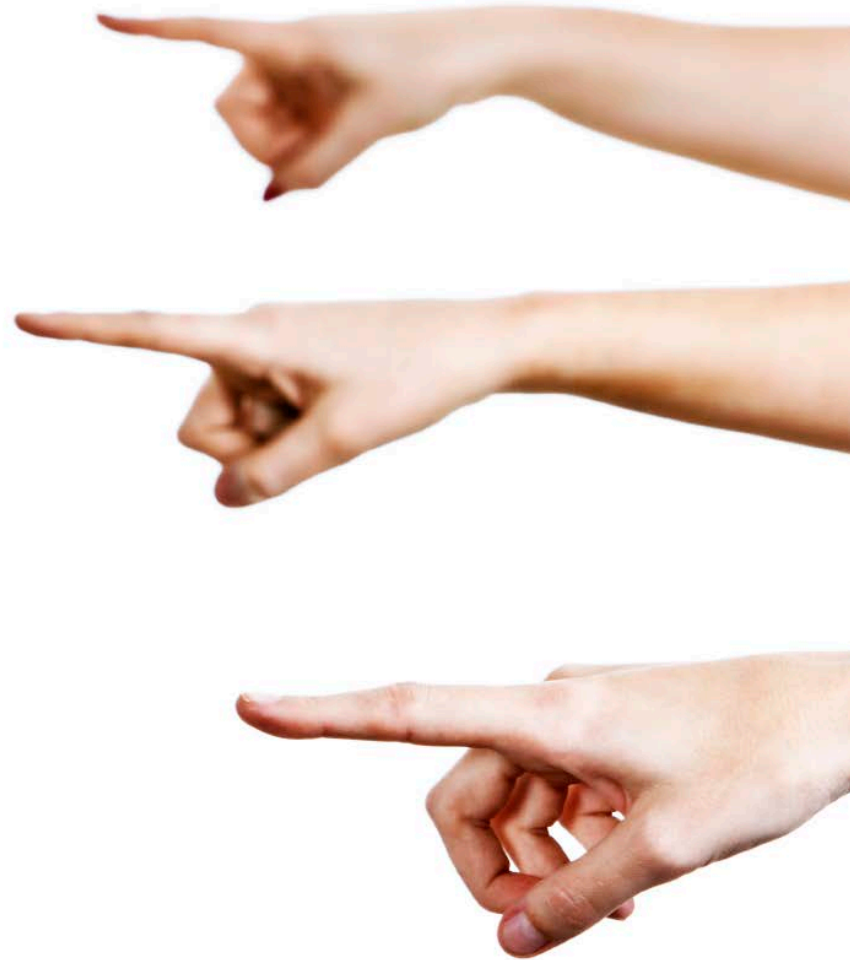
Warning Sign:

Anytime a Problem Arises, People Start Pointing Fingers

If things go wrong, does this start a round of finger pointing, where everyone scrambles to show that it was someone else's fault?

DevOps is all about collaborating and cooperating as a team, and finger-pointing is poisonous to a collaborative working environment.

This is a sign of major dysfunction. You will not be successful in implementing DevOps unless you eliminate the culture of finger pointing.



How to Get Back on the Right Track

Finger pointing is a learned behavior. While this can be difficult to eradicate, there are ways to reduce its effect.

Make it clear that success or failure is a team effort. It takes more than one person to succeed, and it takes more than one to mess up, too.

Set common goals and objectives for all aspects of the development, testing, and production process. The goals should make it clear that Dev cannot be considered a success unless Ops also succeeds, and vice versa.

If everyone—Dev and Ops—has a seat at the table from the outset of the software development lifecycle, then everyone is “brought in” to the process and it is easier to anticipate and remediate problems before they slip into production.

Look at mistakes as an opportunity to learn. You can spend time analyzing what went wrong, what can be learned, what should be done differently, and whether this is a symptom of a larger problem that needs to be addressed. Many so-called mistakes are simply a symptom of a flawed process. Improve the process, and eliminate the mistakes.

Automate things to eliminate as many potential errors as possible.

This both improves process quality and enables people to focus on high-value activities that really make a difference to the organization. Automate everything, and use analytics to track and adjust. Imagine Boeing making planes that didn't have a black box. Without automation and analytics, we are crashing our planes but have no way to reconstruct what went wrong.



Conclusion:



Where to Start Your DevOps Transformation

Recognizing that you have a problem is the first step to making meaningful change. By seeing the value that DevOps can bring to your organization, it might now strike you that you have been mired by dysfunctional processes without realizing it.

So, how do you get out of dysfunction and get on the right track to DevOps?

From our work implementing DevOps at a wide range of different organizations, there are five things that all organizations will eventually end up doing:

- Form application teams that integrate every discipline--from Dev, Testing, and Ops--together
- Improve education, communications, and cross-skilling
- Re-evaluate and rebuild your service delivery cycle
- Evaluate new technology to support DevOps
- Pick the right app or the right line of business to start with DevOps



It doesn't really matter in what order that you do these things; the important thing is to get started.

Pick a critical application that has everyone's attention as your starting point with DevOps. The temptation is to start small—but in DevOps, you will get the biggest payback from addressing a highly critical, highly visible app that is already causing problems. This will help alleviate much of the internal inertia about “doing things differently,” and success with DevOps will have a big impact on the organization. If you can do it on something that is big, ugly, and important, you can replicate that success anywhere in the organization.

An aerial, high-angle photograph of a complex railway yard or station. The image shows numerous tracks curving and intersecting. In the upper left, there is a small, white, box-like control building with a staircase. To its right, a train is visible on one of the tracks. The background features a multi-story urban building. The entire image is overlaid with a semi-transparent blue filter.

To learn more about how you can benefit from
a DevOps methodology visit

ca.com/us/why-ca/devops.html