IT REVOLUTION PRESS

presents

Top 11 Things You Need to Know About DevOps

by Gene Kim
Co-author of *The Visible Ops Handbook* and the upcoming books,
The DevOps Cookbook and *When IT Fails: A Business Novel*

About the Author

Gene Kim is a multiple award winning CTO, researcher and author. He was founder and CTO of Tripwire for 13 years. He has written two books, including *The Visible Ops Handbook*, and is now writing *When IT Fails: A Business Novel* and *The DevOps Cookbook.* Gene is a huge fan of IT operations, and how it can enable developers to maximize throughput of features from "code complete" to "in production," without causing chaos and disruption to the IT environment. He has worked with some of the top Internet companies on improving deployment flow and increasing the rigor around IT operational processes. In 2007, ComputerWorld added Gene to the "40 Innovative IT People Under The Age Of 40" list, and was given the Outstanding Alumnus Award by the Department of Computer Sciences at Purdue University for achievement and leadership in the profession.

Table of Contents

1. What is DevOps and where did it come from?

The term “DevOps” typically refers to the emerging professional movement that advocates a collaborative working relationship between Development and IT Operations, resulting in the fast flow of planned work (i.e., high deploy rates), while simultaneously increasing the reliability, stability, resilience and security of the production environment.

Why Development and IT Operations? Because that is typically the value stream that is between the business (where requirements are defined) and the customer (where value is delivered).

The origins of the DevOps movement are commonly placed around 2009, as the convergence of numerous adjacent and mutually reinforcing movements:

- The Velocity Conference movement, especially the seminal “10 Deploys A Day” presentation given by John Allspaw and Paul Hammond
- The “infrastructure as code” movement (Mark Burgess and Luke Kanies), the “Agile infrastructure” movement (Andrew Shafer) and the Agile system administration movement (Patrick DeBois)
- The Lean Startup movement by Eric Ries
- The continuous integration and release movement by Jez Humble
- The widespread availability of cloud and PaaS (platform as a service) technologies (e.g., Amazon Web Services).

One of the DevOps Cookbook co-authors, John Willis, wrote a fantastic piece on the “Convergence of DevOps” here:

2. How does DevOps differ from Agile?

One tenet of the Agile development process is to deliver working software in smaller and more frequent increments, as opposed to the the “big bang” approach of the waterfall method. This is most evident in the Agile goal of having potentially shippable features at the end of each sprint (typically every two weeks).

High deployment rates will often pile up in front of IT Operations for deployment. Clyde Logue, founder of StreamStep, is attributed as saying “Agile was instrumental in Development regaining the trust in the business, but it unintentionally left IT Operations behind. DevOps is a way for the business to regain trust in the entire IT organization as a whole.”

DevOps is especially complementary to the Agile software development process, as it extends and completes the continuous integration and release process by ensuring the code is production ready and providing value to the customer.

DevOps enables a far more continuous flow of work into IT Operations. When code is not promoted into production as it is developed (e.g., Development delivers code every two weeks, but is deployed only every two months), deployments will pile up in front of IT Operations, customers don’t get value, and the deployments often result in chaos and disruption.

DevOps has an inherent cultural change component, as it modifies the the flow of work and local measurements of Development and IT Operations. John Willis and Damon Edwards (both DevOps Cookbook co-authors) wrote extensively about this here.

3. How does DevOps differ from ITIL or ITSM?

Although many people view DevOps as backlash to ITIL (IT Infrastructure Library) or ITSM (IT Service Management), I take a different view. ITIL and ITSM still are best codifications of the business processes that underpin IT Operations, and actually describe many of the capabilities needed into order for IT Operations to support a DevOps-style work stream.

Agile and continuous integration and release are the outputs of Development, which are the inputs into IT Operations. In order to accommodate the faster release cadence associated with DevOps, many areas of the ITIL processes require automation, specifically around the change, configuration and release processes.

The goal of DevOps is not just to increase the rate of change, but to successfully deploy features into production without causing chaos and disrupting other services, while quickly detecting and correcting incidents when they occur. This brings in the ITIL disciplines of service design, incident and problem management.

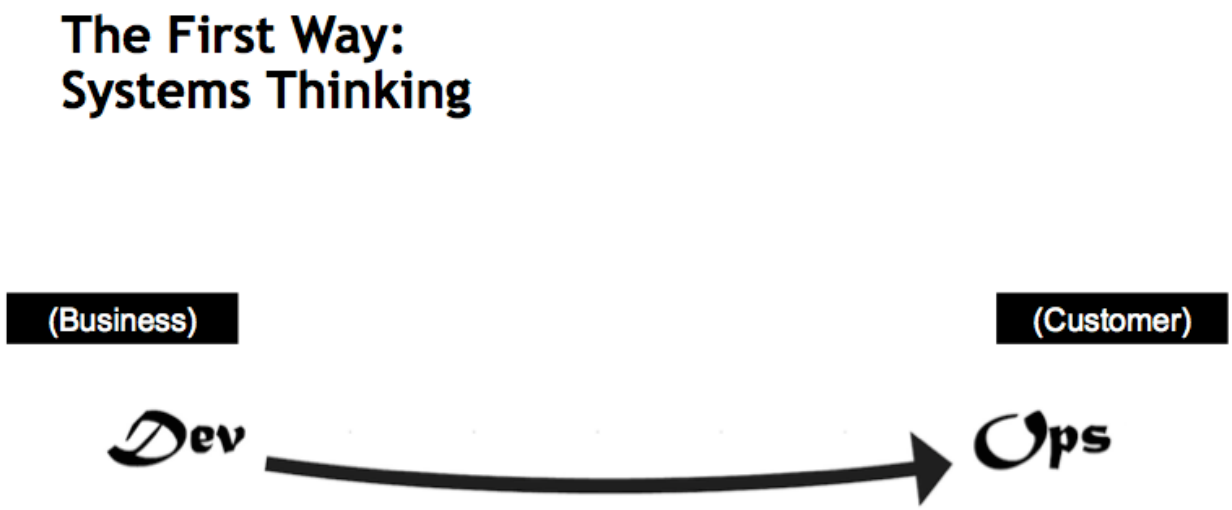
4. How does DevOps fit with Visible Ops?

I co-wrote the “Visible Ops Handbook” in 2004 with Kevin Behr and George Spafford (my fellow co-authors of the upcoming book "When IT Fails: A Business Novel"). Visible Ops is a prescriptive guide to capture the “good to great” transformations of the high performing IT Operations, and one of the key concepts was the notion of how to control and reduce unplanned work.

In many ways, I view DevOps as the inverse, focusing primarily on how to create fast and stable flow of planned work through Development and IT Operations. However, DevOps also has a more holistic approach to systematic eradication of unplanned work, addressing principles of resilient engineering, and the responsible management and reduction of technical debt.

5. What are the unpinning principles of DevOps?

In the *DevOps Cookbook* and *When IT Fails: A Business Novel*, we describe the underpinning principles which all the DevOps patterns can be derived from as "The Three Ways." They describe the values and philosophies that frame the processes, procedures, practices, as well as the prescriptive steps.

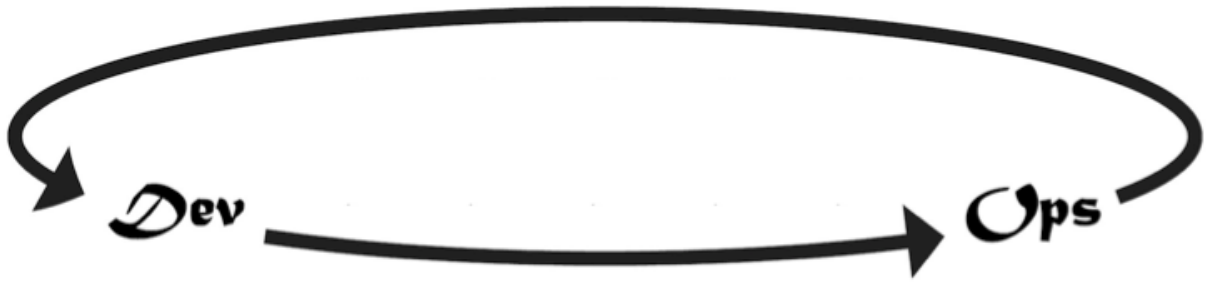


The First Way emphasizes the performance of the entire system, as opposed to the performance of a specific silo of work or department — this as can be as large a division (e.g., Development or IT Operations) or as small as an individual contributor (e.g., a developer, system administrator).

The focus is on all business value streams that are enabled by IT. In other words, it begins when requirements are identified (e.g., by the business or IT), are built in Development, and then transitioned into IT Operations, where the value is then delivered to the customer as a form of a service.

The outcomes of putting the First Way into practice include never passing a known defect to downstream work centers, never allowing local optimization to create global degradation, always seeking to increase flow, and always seeking to achieve profound understanding of the system (as per Deming).

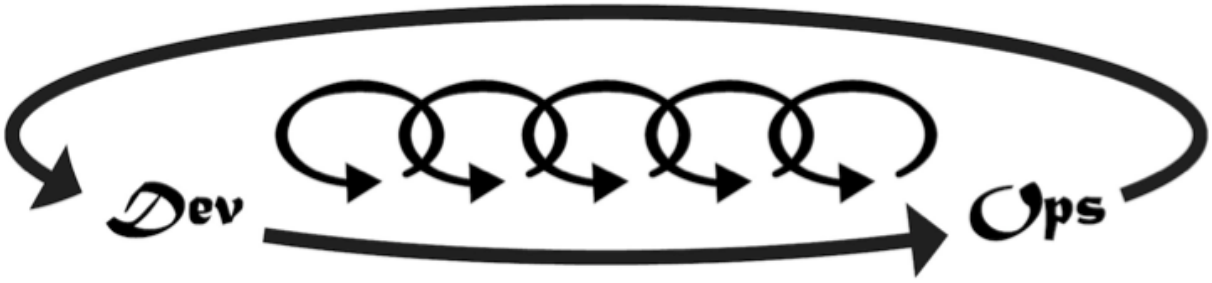
The Second Way:
Amplify Feedback Loops



The Second Way is about creating the right to left feedback loops. The goal of almost any process improvement initiative is to shorten and amplify feedback loops so necessary corrections can be continually made.

The outcomes of the Second Way include understanding and responding to all customers, internal and external, shortening and amplifying all feedback loops, and embedding knowledge where we need it.

The Third Way:
Culture Of Continual Experimentation And Learning



The Third Way is about creating a culture that fosters at two things: continual experimentation, which requires taking risks and learning from success and failure; and understanding that repetition and practice is the prerequisite to mastery.

We need both of these equally. Experimentation and risk taking are what ensure that we keep pushing to improve, even if it means going deeper into the danger zone than we've ever gone. And we need mastery of the skills that can help us retreat out of the danger zone when we've gone too far.

The outcomes of the Third Way include allocating time for the improvement of daily work, creating rituals that reward the team for taking risks, and introducing faults into the system to increase resilience.

6. What are the areas of DevOps patterns?

For the “DevOps Cookbook,” we divide up the DevOps patterns into four areas:

- Area 1: Extend Development into Production: this includes extending the continuous integration and release function into production, integrating QA and infosec into the work stream, ensuring production readiness of the code and environment, and so forth. (Internally, we call this “harnessing your inner Jez Humble”)

- Area 2: Create Production feedback into Development: this includes creating a complete timeline of Development and IT Operations events to aid in incident resolution, integrating Development into blameless production post-mortems, enabling Developer self-service wherever possible, and creating information radiators to show how local decisions affect achievement of global goals.

- Area 3: Embed Development into IT Operations: this includes putting Development into the production escalation chain, assigning Development resources to production problem management and to help retire technical debt, and Development cross-training IT Operations to reduce the number of escalations.

- Area 4: Embed IT Operations into Development: this includes embedding or liaising IT Operations resources into Development, creating reusable user stories for the IT Operations staff (including deployment, management of the code in production, etc.), and defining the non-functional requirements that can be used across all projects.

Patrick Debois, one of my “DevOps Cookbook” co-authors, wrote more about this here.

7. What is the value of DevOps?

I believe there are three business benefits that organizations get from adopting DevOps:

- Faster time to market (i.e., reduced cycle times and higher deploy rates)
- Increased quality (i.e., increased availability, increased change success rate, fewer failures, etc.)
- Increased organizational effectiveness (e.g., increased time spent on value adding activities vs. waste, increased amount of value being delivered to the customer).

Faster time to market:

In 2007, at the IT Process Institute, we benchmarked over 1,500 IT organizations and concluded that high-performing IT organizations were on average 5-7x times more productive than their non-high performing peers. They were making 14x more changes, with one-half the change failure rate with 4x higher first fix rates, and 10x shorter Severity 1 outages times. They had 4x fewer repeat audit findings, they were 5x more likely to detect breaches by an automated internal control, and had 8x better project due date performance! (You can read more about the findings and the research here).

In our research, the highest deploy rate we observed was approximately 1,000 production changes per week, with a change success rate of 99.5%. We thought this was fast...

One of the characteristics of high performers is that they accelerate away from the herd. In other words, the best continue to get even better. This is absolutely happening in area of deploy rates. Organizations who are employing DevOps practices are out-performing our fastest high-performer by orders of magnitude. Accenture recently did a study about what Internet companies are doing, and Amazon has gone on record stating that they're doing over 1,000 deploys a day, sustaining a change success rate of 99.999%! You can see Jeff Jenkins 2011 Velocity talk about the Amazon deployment and IT Operations model here and his slides from the presentation here.

The capability to sustain high deploy rates (i.e., fast cycle times) translates into business value in two primary ways: how quickly the organization can go from an idea to value being delivered to the customer (i.e., "concept to kaching" as Damon Edwards and John Willis say), and how many experiments can the organization be doing simultaneously.

There is no doubt in my mind that if I'm in an organization where I can only do one deployment every nine months, and my competitor can do 10 deploys in a day, I have a significant, structural competitive disadvantage.

High deploy rates also enable rapid and constant experimentation. Scott Cook, the founder of Intuit, has been one of the most outspoken advocates for a "rampant innovation culture" at all levels of the organization. One of my favorite examples is quoted below:

> "Every employee [should be able to] do rapid, high-velocity experiments... Dan Maurer runs our consumer division, including running the TurboTax website. When he took over, we did about seven experiments a year. By installing a rampant innovation culture, they now do 165 experiments in the three months of tax season. Business result? Conversion rate of the website is up 50 percent. Employee result? The folks just love it, because now their ideas can make it to market."

To me, the most shocking part of Scott Cook's story is that they were doing all these experiments during peak tax filing season! Most organizations have change freezes during their peak seasons (e.g., retailer may often have holiday change freezes from October until January). But if you can increase conversion rates, and therefore sales, during peak seasons when your competitor cannot, then that's a genuine competitive advantage.

The prerequisites to do doing this include being able to do many small changes quickly, without disrupting service to customers.

Reduced amount of IT waste:

Mike Orzen and I have long talked about the enormous waste in the IT value stream, caused by long lead times, poor hand-offs, unplanned work and rework. In an article for Michael Krigsman we estimated how much value we could recapture by applying DevOps-like principles.

We calculated that if we could just halve the amount of IT waste, and redeploy those dollars in a way that could return five times what was invested, we would generate $3 trillion dollars of value per year.

That's a staggering amount of value and opportunity that we're letting slip through our fingers. That's 4.7 percent of annual global GDP, or more than the entire economic output of Germany.

I think this is important, especially when I think about the world my three children will inherit. The potential economic impact to productivity, standards of living, and prosperity almost makes this a moral imperative.

However, there's an even greater cost. Working in most IT organizations is often thankless and frustrating. People feel as if they're trapped in an ever-repeating horror movie, helpless to change the outcome. Management abdicates their responsibility to ensure that IT is managed

well, resulting in endless intertribal warfare between development, IT operations and information security. And things only get worse when the auditors show up.

What inevitably results is chronic underachievement. The life of an IT professional is often demoralizing and frustrating. It typically leads to feelings of powerlessness and is rife with stress which seeps into every aspect of life. From stress-related health problems, to social issues, to tension at home, being an IT professional is not only unhealthy, but likely unsustainable.

As people, we're wired to contribute and to feel like we're actively making a difference. Yet, all too often when IT professionals ask their organization for support, they're met with "you don't understand," or worse, a barely masked, "you don't matter."

At the IT Revolution Press, our mission is to improve the livelihoods of one million IT workers by 2017. We hope that "When IT Fails: A Business Novel" can help the business and IT gain a shared understanding of the problem, and that the "DevOps Cookbook" can help people fix the problem.

8. How do Infosec and QA integrate into a DevOps work stream?

High deployment rates typically associated with DevOps work streams will often put enormous pressure on QA and Infosec. Consider the case where Development is doing ten deploys per day, while information security requires a four month lead time to turn around for application security reviews. At first glance, there appears to be a fundamental mismatch between the rate of code development and security code testing.

An example of the risk posed by insufficiently tested deployments is the famous 2011 Dropbox failure, where authentication was turned off for four hours, which enabled unauthorized users to access all stored data.

The good news for QA and Infosec is that Development organizations capable of sustaining high deploy rates are likely using continual integration and release practices, which often goes hand in hand with a culture of requiring continuous testing. In other words, whenever code is checked in, automated tests are automatically run, and issues must be fixed right away, just as if a developer checked in code that didn't compile.

By integrating functional, integration and information security testing into the daily operations of Development, defects are found and fixed more quickly than ever.

There are a growing number of infosec tools such as Gauntlet and Security Monkey that help test security objectives in the development and in production processes.

A genuine concern is that static code analysis tools take too long to run to integrate into a continuous integration and testing process, often requiring hours or even days to complete. In these cases, infosec should designate the specific modules that has security functionality being relied upon (e.g., encryption, authentication modules). Whenever those modules change, a full retest is run, otherwise, deployments can proceed.

One last note: we observe that DevOps work streams often put more reliance on detection and recovery, than standard functional unit testing. In other words, when doing development for packaged software where it is very difficult to deploy changes and patches, QA relies heavily on testing the code for functional correctness before it is shipped. On the other hand, when software is delivered as a service and defects can be fixed very quickly, then QA can reduce its reliance on testing, and instead rely more on production monitoring to detect defects in production, as long as they can be quickly fixed.

Quick recovery from code failures can be aided by using “feature flags,” which enable and disable code functionality via configuration settings, instead of having to roll out an entire new deployments.

Relying on detection and recovery for QA is obviously far more applicable when the worst that could happen is the loss of functionality or required performance. However, when failures risk the loss of confidentiality or integrity of systems or data, then reliance cannot be put on detection and recovery -- instead, it must be tested before code is deployed, because a production failure would generate a genuine security incident.

We’ll be writing more about how we’re codifying the new patterns of QA and infosec testing on the blog in the future.

9. My Favorite DevOps Pattern #1:

All too often in software development projects, Development will use up all the time in the schedule on feature development. This leaves insufficient time to adequately address IT Operations issues. Shortcuts are then taken in defining, creating, testing everything that the code relies upon, which includes the databases, operating systems, network, virtualization, and so forth.

This is certainly one primary causes for the perpetual tension between Development and IT Operations and suboptimal outcomes. The consequences of this are well-known: inadequately defined and specified environments, no repeatable procedures to deploy them, incompatibilities between deployed code and the environment, and so forth.

In this pattern, we will make environments early in the Development process, and enforce a policy that the code and environment be tested together. When Development is using an Agile process, we can do something very simple and elegant.

According to Agile, we're supposed to have working, shippable code at the end of each sprint interval (typically every two weeks). We will modify the Agile sprint policy so that instead of having at the end of each sprint just shippable viable code, you also have to have the environment that it deploys into -- at the earliest sprint, so we're talking sprint 0 and sprint 1.

Instead of having IT Operations responsible for creating the specifications of the production environment, instead, they will build an automated environment creation process. This mechanism will create the production environment, but also the environments for Dev and QA.

By making environments (and the tools that create them) available early, perhaps even before the software project begins, developers and QA can run and test their code in consistent and stable environments, with controlled variance from the production environment.

Furthermore, by keeping variance between the different stages (e.g, Development, QA, Integration Test, Production) as small as possible, we will find and fix interoperability issues between the code and environment long before production deployment.

Ideally, the deployment mechanism we build is completely automated. Tools that can be used include shell scripts, Puppet, Chef, Solaris Jumpstart, Redhat Kickstart, Debian Preseed, etc.

10. My Favorite DevOps Pattern #2:

One of my favorite quotes is from Patrick Lightbody, former CEO of BrowserMob, who said, “We found that when we woke developers up at 2am it was a phenomenal feedback loop, defects got fixed faster than ever.”

This underscores the problem where Development checks in their code at Friday 5pm, high-fives each other in the parking lot and then goes home, leaving IT Operations to clean up the mess the entire weekend. Worse, defects and known errors keep recurring in production, forcing IT Operations to continually firefight, and the root cause is never fixed because Development is focused on building new features.

An important element of the Second Way is to shorten and amplify feedback loops, and to bring Development closer to the customer experience (which includes IT Operations and the end-users of the service being delivered).

Note the symmetry here: Favorite Pattern #1 about making environments available early is all about embedding IT Operations into Development, while Favorite Pattern #2 is about putting Development into IT Operations.

Here, we put Development into the IT Operations escalation chain, possibly putting them in Level 3 support, or even having Development be completely responsible for the success of the code deployments, either rolling back or fixing forward until service is restored to the customer.

The goal is not to have IT Operations replaced by Development. Instead, it’s to ensure that Development sees the downstream effects of their work and changes, and has walked in the shoes of IT Operations enough to be motivated to fix issues quickly to help with the achievement of global goals.

11. My Favorite DevOps Pattern #3

Another recurring problem that occurs in the DevOps value stream between Development and IT Operations isn't sufficiently standardized. Examples of this is when every deployment is done differently, every production environment is different snowflake. When this occurs, no mastery is ever built in the organization in procedures or configurations.

In this pattern, we define reusable deployment procedures that can be used across projects. There is a very elegant solution in the Agile methodology to this, where deployment activities are turned into a user story. For example, we would build a reusable user story for IT Operations called "Deploy into high availability environment," which then defines the exactly the steps to build the environment, as well as how long it takes, what resources are required, etc.

This information can then be used be used by project managers to accurately integrate the deployment activities into the project plan. For instance, we would have high confidence in the deployment schedule if we knew that the "Deploy into high availability environment" story has been executed fifteen times in the past year, taking an average of three days, plus or minus one day.

Furthermore, we also gain confidence that the deployment activities are being properly integrated into every software project.

Recognizing that certain software projects require unique environments that IT Operations doesn't officially support, we can allow for exceptions where these environments are allowed in production, but are supported by someone outside of IT Operations (i.e., unsupported environments).

By doing this, we get the benefits of environment standardization (e.g., reduced production variance, fewer snowflakes in production, increased ability for IT Operations to reliably support and maintain, etc.) while allowing for special cases that allow the nimbleness that business sometimes requires.

The Future of IT

IT Revolution Press is leading the charge to the next revolution in IT.

We know the current system is not working. We know there is a better way. We know that finding a solution will unlock IT's true potential. At IT Revolution Press, we want to help drive the greatest change in how we manage IT. Without a doubt, one hundred years from now, historians will look back at this decade and say, "this was when the Cambrian Explosion for IT occurred, when people finally figured out how organizations manage IT to win."

Our goal is to positively influence the lives of 1 million IT people over the next 5 years. To make this happen, we're uniting thought leaders in all the relevant domains with a common sense of purpose and passion to help us achieve our goal and improve IT for generations to come.

Visit the IT Revolution website to read our blog, download more whitepapers and signup for our newsletter.