Why is Engineering DevOps Important?

This paper is derived from selections from my book.



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from [**https://engineeringdevops.com/documents**](https://engineeringdevops.com/documents)

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

May 8, 2020

© Engineering DevOps Consulting 2019, 2020

All rights reserved

**Why is Engineering DevOps Important?**

Why should you engineer DevOps rather than simply “do a DevOps transformation”?

As a leader or practitioner of any enterprise, product or service that involves technology, ignoring DevOps engineering may be a career blunder. You may think I jest. After all, DevOps in and of itself is not a product. It is not important to nature or the survival of mankind. It does not fix world hunger, save a species, heal the sick, or even babysit the kids. Yet DevOps’s proven track record for delivering a nearly magical array of benefits rightly compels wise technology leaders and practitioners to pursue it because it has power to greatly affect the value of their work and competitiveness for organizations and projects.

In my experience, many organizations have pushed ahead with a DevOps transformation with the expectation of accomplishing the benefits of DevOps and failed because they did not have a clear, end-to-end DevOps engineering blueprint and they did not have or follow an engineering disciplined approach using engineering practices for each of Nine Pillars of DevOps or Three Dimensions (People, Process, and Technology).

Development and delivery methodologies that preceded DevOps have, when properly implemented and carefully followed, yielded improvements to time-to-market and efficiencies; added stability, quality, and security of products and services; and improved satisfaction for some business, development, or operations stakeholders, compared to the alternative of not using them. However, there were usually trade-offs. Speed versus quality. Security versus satisfaction. Stability versus efficiency.

The magic of engineering DevOps is that it has the power to provide drastic improvements to all these benefit categories at once for all stakeholders, including you. To realize the “magical” benefits of DevOps requires a practical, disciplined, step-by-step engineering approach that is described in this book.

**Engineering DevOps Myths and Realities**

There are many popular myths surrounding DevOps that are important from a DevOps engineering point of view. This is understandable given that DevOps definitions and engineering practices have not been “nailed down.” It is important to clarify the truth about myths that are relevant to DevOps so that we can see how an engineering perspective can help. These myths can get in the way of seeing the engineering possibilities.

**Myth:** “*DevOps is a cultural movement.”* The truth is that implementing DevOps requires a careful and balanced engineering approach that considers a wide array of people, process, and technology aspects. While culture is a critical success factor, DevOps is much more than culture. Furthermore, DevOps is not just a movement. In the larger context of software engineering history, the word “DevOps” is new, and perhaps it will be replaced by some other word someday, but the underlying engineering principles and practices that constitute DevOps apply to almost any engineering project throughout the ages.

**Myth:** “*DevOps requires continuous deployments.”* The truth is that DevOps enables continuous delivery of release artifacts to a staging environment to be ready to deploy safely, but frequent deployments to production is not always a goal. There are valid customer use cases that do not desire frequent deployments to production. For example, customers may not want to receive a new version of a software platform product frequently because each new version may be disruptive and risky for reasons beyond the scope of the product itself. The engineering benefits of delivering complete releases to staging without deployment to production are still considerable. Each release is produced and validated in a pre-production environment and made ready for deployment upon demand without disrupting customers that do not want to take a new release.

**Myth:** “*DevOps applies to any software product, service, or application*.” The truth is that DevOps engineering yields high ROI in many circumstances, but some cases do not warrant investment in the people, process, and technology changes needed for DevOps. For example:

• Applications that rarely change do not warrant the cost of DevOps changes.

• Organizations unable to change from organization silos will be too frustrating and counterproductive if they try to implement DevOps.

• The application of DevOps to some Commercial Off-the-Shelf (COTS) products may not yield sufficient ROI because business to business barriers may behave as impenetrable organization silos.

• Applications that have few tools in their toolchains suitable for DevOps may find the transition to DevOps too expensive.

**Myth:** “*DevOps requires Agile.”* The truth is that DevOps and Agile can be complementary or advisories. The notions of working with small incremental changes is common to both Agile and DevOps. However, organizations that embrace Agile are focused on optimization of the development end of the pipeline and may struggle with DevOps, which emphasizes engineering of end-to-end process optimizations.

**Myth:** “*The scope of DevOps is the same as CI/CD.”* The truth is that a well-engineered DevOps implementation encompass CI/CD but extends beyond integration and delivery into planning and operations also.

**Myth:** “*DevOps does not apply to platform products such as software code embedded in manufactured products*.” The truth is that DevOps does apply to embedded code products just as well as web services, business applications, and other types of products. It even applies to development of software and services for DevOps itself! In the case of platform products, typically deployments are planned. They are not continuous, but instead continuous deliveries of candidate releases are delivered to a staging environment to make sure the changes that are implemented for the platform are fully developed and verified as the platform is built instead of waiting for a big-bang release.

**How Will I Know When I Have Engineered DevOps?**

A key tenet of engineering is that processes, projects, and products are specified and measurable. You can tell when you are “done” because when you achieve measurements within specified tolerances those indicate that you are done. This idea of having clear specification and measurable attributes is vital for DevOps processes, projects, and products also. It is completely reasonable for a business manager to ask, “Why should I invest a specific amount of resources into engineering DevOps projects until progress is measurable and it will be clear when DevOps has been accomplished?” This is easy to ask, but without some engineering specifications and measurable definitions, it is impossible to provide an answer. To satisfy this requirement I define minimal, measurable conditions that constitute each of “*The Three Ways of DevOps.”*

I define the following to be the minimal, measurable conditions that constitute “*The First Way of DevOps”—Continuous Flow*:

• Continuous Flow exists from planning to operations without interruptions.

• A defined pipeline implements a defined value stream.

• Outcomes are deterministic and repeatable.

• Each stage in the value stream has measurable exit and entry criteria and gates.

• The work of each stage in the pipeline is defined and bounded.

• Automation is employed in each pipeline stage and between pipeline stages where needed to prevent serious bottlenecks.

• People and manual work may be employed in the pipeline, but delays caused by human interactions with the pipeline are minimal compared to the total time for a change to transit the end-to-end pipeline.

• Processes to identify and handle quality problems and other interruptions to flow are defined.

I define the following to be the minimal, measurable conditions that constitute “*The 2nd Way of DevOps”—Continuous Feedback*:

• *The First Way of DevOps—Continuous Flow,* as described in the last paragraph, is in place and stable.

• Metrics for Service Level Indicators, Objectives, and Agreements (SLI, SLO, and SLA) are in place for the application, the pipeline, and the infrastructure. These SLI, SLO, and SLA metrics are the used to direct application release decisions.

• Metrics analysis tools such as dashboards and algorithms are used to aggregate metrics, log data, and produce trends charts for reactive analysis.

I define the following to be the minimal, measurable conditions that constitute “*The 3rd Way of DevOps”—Continuous Improvement:*

• *The Second Way of DevOps—Continuous Feedback*, as described in the last paragraph, is in place and stable.

• Retrospectives are routinely conducted for every release to identify improvements.

• Metrics are proactively analyzed to identify improvements for Continuous Flow and Feedback. • The organization is routinely searching for and experimenting with new DevOps solutions and improvements through research and industry outreach. By identifying and tracking new solutions, progress towards *Continuous Improvement* becomes measurable.

**Beneﬁts of Well-Engineered DevOps**

How can DevOps, which does not even have a standard definition, be attributed any benefits at all, let alone the lofty benefits people like me attribute to it? Thankfully, you do not have to take my word for it. The benefits of DevOps are very well researched and documented in issues of the *State of DevOps Report* thanks to the excellent work of Puppet Labs® and the DevOps Research and Assessment Organization (DORA), led by Nicole Forsgren, Ph.D., and her book Accelerate,RB8 co-authored by two of my personal DevOps heroes, Jez Humble and Gene Kim. Analysis of data presented in the State of DevOps Report indicates there are at least six types of measurable benefits reported by high- performance organizations that are using practices associated with more mature DevOps compared to organizations that are using practices associated to less mature DevOps. The six benefits of DevOps can be summarized in six categories: Agility, Stability, Efficiency, Quality, Security, and Satisfaction. This granular level of benefit analysis makes it easier to identify goal priorities and specific changes that can be realized with specific engineering practices.

**Agility**

The benefit category of Agility indicates the ability of an organization to move and react quickly to produce changes relevant to products and services. The category of Agility, as represented in the State of DevOps Report, includes the following measurable benefits:

• Lead-time measured as the duration from one point in the value stream until code is ready to be deployed to production

• Frequency of producing deployable releases to live production

• The percent of time employees spend on new work, such as producing new features or code, compared to time spent on other types of activities

• The extent that product teams break work into small-batch increments • The extent that workflow is visible throughout the pipeline

The 2016 State of DevOps Report presents comparison data showing that high-performing IT organizations indicate they are deploying 200 times more frequently than low-performing organizations and have 2,555 times shorter lead times.

**Stability**

The benefit category of Stability indicates the extent to which products and services can maintain an operational state despite disturbances. The State of DevOps Report indicates the average cost of an outage is $500,000 per hour and can be much higher. The category of Stability, as represented in the State of DevOps Report, includes the following measurable benefits:

• Mean-Time-to-Recover (MTTR) from failure/service outages in production

• The percent of code merges from development break the trunk branch

The 2016 State of DevOps Report presents comparison data showing that high-performing IT organizations indicate they are achieving 24 times faster recovery from failures and 3 times lower change failure rates than low-performing organizations.

**Efﬁciency**

The benefit category of Efficiency is a measure of the ratio of useful output to total input for the processes involved in producing a release of a product or service. The category of Efficiency, as represented in the State of DevOps Report, includes the following measurable benefits:

• The percent of time employees spend on all types of unplanned work, including rework

• The extent that comprehensive metrics are available for capital costs of development and operations

• The extent that comprehensive metrics available for keeping track of the non-capital costs of development and operations

• The extent to which lean product management is practiced using highly visible, easy-to-understand presentation formats that show work to be done

The 2016 State of DevOps Report presents comparison data showing that high-performing IT organizations indicate they are spending 22% less time spent on unplanned work and 29% more time on new work than low-performing organizations.

**Quality**

The benefit category of Quality is a measure of excellence of a product or service. In engineering, quality is a measure of deficiencies and significant variations that cause failures against specific customer or user requirements identified during testing and operational experiences. The category of Quality, as represented in the State of DevOps Report, includes the following measurable benefits:

• The frequency of failures that require immediate remediation occurring in live production

• The extent that quality tests and test data are sufficient and readily available when needed

• The extent that the organization regularly seeks customer feedback and incorporates the feedback into design

**Security**

The benefit category of Security indicates the practice of assuring information and managing risks related to the use, processing, storage, and transmission of information or data and the systems and processes used for those purposes. The category of Security, as represented in the State of DevOps Report, includes the following measurable benefits:

• Number of times per year that a serious, business-impacting security event occurs

• Number of times per year that an unauthorized user accesses unauthorized information

• Average percent of time that employees spend remediating security issues

The 2016 State of DevOps Report presents comparison data showing that high-performing IT organizations indicate 50% less time spent on remediating security issues than low-performing organizations.

**Satisfaction**

The benefit category of Satisfaction refers to fulfilling the need of employees to feel good about their working environment. The category of Satisfaction, as represented in the State of DevOps Report, includes the following measurable benefits:

• The extent that employees are likely to recommend their team as a great place to work

• The extent that employees are likely to recommend the organization as a great place to work

• The extent that the organization culture is practicing good communication flow, cooperation, and trust

• The extent that leaders promote personal and/or team recognition by commending better-than-average work, acknowledging improvements in the quality of work, and personally compliment individuals’ outstanding work

The 2016 State of DevOps Report presents comparison data showing that high-performing IT organizations indicate they are 2.2 times more likely to recommend their organization as a great place to work and 1.8 times more likely to recommend their team as a great working environment than low-performing organizations. So how can DevOps do all this?

 If you doubt the above research analysis, then just consider that DevOps is to software-based businesses what lean manufacturing engineering practices was to the automobile industry when Japanese automobile manufacturers leap-frogged American automobile sales in America during the 1980s and 1990s. The Toyota Production System, the benchmark example for lean practices, reduces waste, increasing efficiency and reducing costs. The high-quality and cost-competitive products Toyota produces are causally linked to Toyota’s ability to reduce waste throughout the production process. To this day, Toyota is the auto industry leader in many categories.

**Engineering DevOps Beneﬁts**

**When lean manufacturing engineering practices are applied with engineering precision and discipline to the Nine Pillars of Engineering DevOps and The Three Dimensions, you do indeed get agility, stability, efficiency, quality, security, and satisfaction.**

**Costs of Not Engineering DevOps Properly**

Despite the compelling benefits of DevOps, I suggest that before you dive headlong into a DevOps transformation, you approach DevOps with the utmost respect for engineering. Doing DevOps incorrectly can be just as perilous as not doing DevOps at all.

It is worth reviewing the remarkable DevOps story of *Knight Capital*. If you are developing a continuous delivery pipeline, you will find the story an interesting read. With scary sub-subtitles like “Attack of the Killer Code Zombies” and “45 Minutes of Hell,” it sounds like a script from a fictional Hollywood thriller. Except it was not a fiction. It was a documentary. A real horror that ended a top financial firm and caused unrecoverable financial havoc for investors, other firms, and the entire worldwide stock trading industry. Ripple effects of lessons learned from a single event continue to affect the way the trading software is deployed to this day.

A summary of an article posted April 17, 2014 recaps “*Knightmare: A DevOps Cautionary Tale.”* Some of the main points from the article are recounted below because this is such an important example of the costs of not engineering DevOps properly. Grab some popcorn and read on. In 2012 Knight was the largest trader in US equities and managed an average daily trading volume of more than 3.3 billion trades daily, trading over 21 billion dollars . . . daily. That is no joke! Between July 27, 2012, and July 31, 2012, Knight manually deployed new software to a limited number of servers per day—eight servers in all. During the deployment of the new code, however, one of Knight’s technicians did not copy the new code to one of the eight computer servers. At 9:30 a.m. EST on August 1, 2012, the markets opened, and Knight began processing orders from broker-dealers on behalf of their customers. The seven servers that had the correct deployment began processing orders correctly. Orders sent to the eighth server triggered a supposable repurposed flag and unexpectedly “brought back from the dead” old code, which began routing child orders for execution but wasn’t tracking the number of shares against the parent order—somewhat like an endless loop.

Imagine what would happen if you had a system capable of sending automated, high-speed orders into the market without any tracking to see if enough orders had been executed. Yes, it was that bad! By 9:32 a.m., many people on Wall Street were wondering why it had not stopped. This was an eternity in high-speed trading terms. Why hadn’t someone hit the kill switch on whatever system was doing this?

As it turns out, there was no kill switch. During the first 45-minutes of trading, Knight’s executions constituted more than 50% of the trading volume, driving certain stocks up over 10% of their value. As a result, other stocks decreased in value in response to the erroneous trades. During the 45 minutes of hell that Knight experienced, they attempted several countermeasures to try and stop the erroneous trades. There was no kill switch (and no documented procedures for how to react), so they were left trying to diagnose the issue in a live trading environment where eight million shares were being traded every minute. Since they were unable to determine what was causing the erroneous orders, they reacted by uninstalling the new code from the servers it was deployed to correctly. In other words, they removed the working code and left the broken code. This only amplified the issues, causing additional parent orders to activate the bad code on all servers, not just the one that was not deployed to correctly.

Eventually they were able to stop the system—after 45 minutes of trading. Altogether, four million bad transactions were executed against 154 stocks total: more than 397 million shares. Knight Capital Group realized a $460 million loss in 45 minutes. Knight only had $365 million in cash and equivalents. **In 45 minutes, Knight went from being the largest trader in U.S. equities and a major market maker in the NYSE and NASDAQ to bankrupt.**

NASDAQ and SEC fines and payments to investors ensued. Needless to say, some employees were let go, and the employability of some were affected. Okay, it’s time to put down the popcorn bowl or clean the popcorn you spilled while squirming in your seat reading about the Knightmare. The entire failure event could have been prevented had the DevOps implementation taken a more comprehensive engineering approach following an end-to-end DevOps blueprint and the engineering practices of the Nine Pillars of Engineering DevOps. Here are some lessons learned about not engineering DevOps appropriately:

• Put version management of applications, infrastructure, and pipeline code in a version management system.

• Implement and automate a version roll-back and roll-forward process.

• Put circuit breakers in the code and deployment processes.

• Automate deployment—not just the installation but also the recovery procedures.

• Test deployment processes before they are put in use.

• Instrument application code and deployment processes with monitoring tools.

• Make sure all key stakeholders have visibility to release deployment changes and release deployment process activities.

• Train the staff!

I could go on and tell you other interesting “DevOOOPs” stories and engineering lessons learned, but I hope this one is sufficient to impress upon you that you must engineer DevOps properly or you WILL suffer horrible consequences.

**Summary**

In summary, the answer to the question: “Why is Engineering DevOps Important?” has been explained in this white paper.

**DevOps, when implemented according to engineering practices, yields dramatically improved business outcomes including time-to-market, efficiencies, stability, quality, security, and satisfaction when compared to the alternative of not using them.**

**Learn More**



The book “Engineering DevOps” can be obtained here: [**mybook.to/engineeringdevops**](http://mybook.to/engineeringdevops)

This white paper and other useful papers, glossary checklists, slides, blueprints, templates and engineering practices can be downloaded from <https://engineeringdevops.com/documents>

About the Author:

**Marc Hornbeek** a.k.a. DevOps\_the\_Gray esq.

[linkedin.com/in/marchornbeek](https://www.linkedin.com/in/marchornbeek)

© EngineeringDevops 2019, 2020

All rights reserved

**What Is Engineering DevOps?**