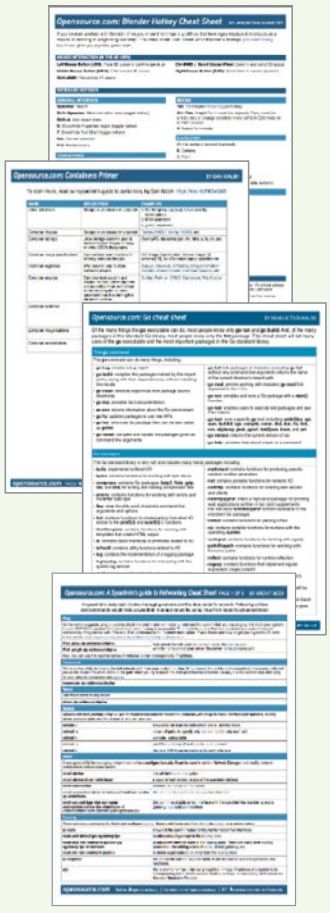
Starting a DevOps Transformation

Breaking down walls between people, process, and products

Open Source Cheat Sheets

Visit our cheat sheets collection for free downloads, including:



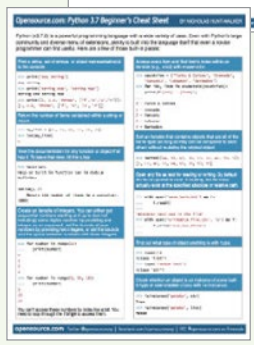
Blender: Discover the most commonly and frequently used hotkeys and mouse button presses.

Containers: Learn the lingo and get the basics in this quick and easy containers primer.

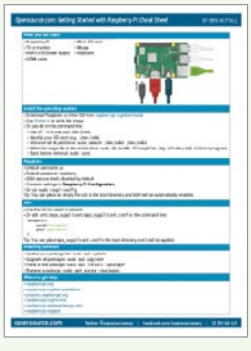
Go: Find out about many uses of the go executable and the most important packages in the Go standard library.

Inkscape: Inkscape is an incredibly powerful vector graphics program that you can use to draw scaleable illustrations or edit vector artwork that other people have created.

Linux Networking: In this downloadable PDF cheat sheet, get a list of Linux utilities and commands for managing servers and networks.



Python 3.7: This cheat sheet rounds up a few built-in pieces to get new Python programmers started.



Raspberry Pi: See what you need to boot your Pi, how to install the operating system, how to enable SSH and connect to WiFi, how to install software and update your system, and links for where to get further help.

SSH: Most people know SSH as a tool for remote login, which it is, but it can be used in many other ways.

ABOUT OPENSOURCE.COM

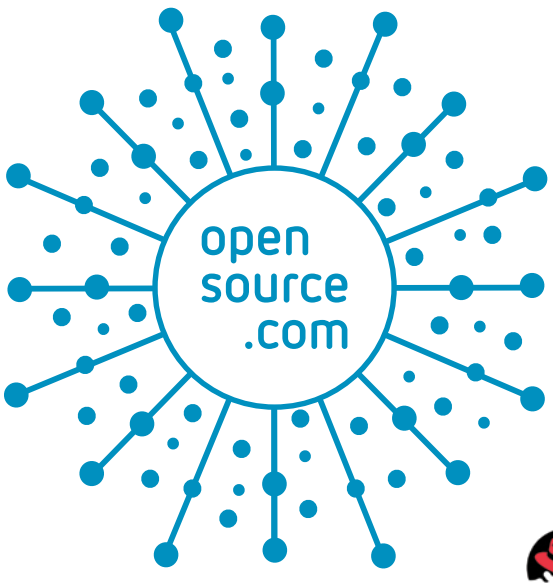
What is Opensource.com?

OPENSOURCE.COM publishes stories about creating, adopting, and sharing open source solutions. Visit Opensource.com to learn more about how the open source way is improving technologies, education, business, government, health, law, entertainment, humanitarian efforts, and more.

Submit a story idea: https://opensource.com/story

Email us: open@opensource.com

Chat with us in Freenode IRC: #opensource.com



SUPPORTED BY RED HAT

WILLY-PETER SCHAUB

SINCE MID-'80S, I have been striving for simplicity and maintainability in software engineering. As a software engineer, I analyse, design, develop, test, and support software solutions. I am passionate about continuous innovation and to share learnings from the digital transformation, by Microsoft and the ALM | DevOps Rangers, to a DevOps culture to adapt people, process, and products to continuously deliver value to our end users.

CONTACT WILLY-PETER SCHAUB

Website: https://willys-cave.ghost.io
https://www.agents-of-chaos.org

Linked In: http://www.linkedin.com/in/wpschaub

Twitter: https://twitter.com/wpschaub

INTRODUCTION

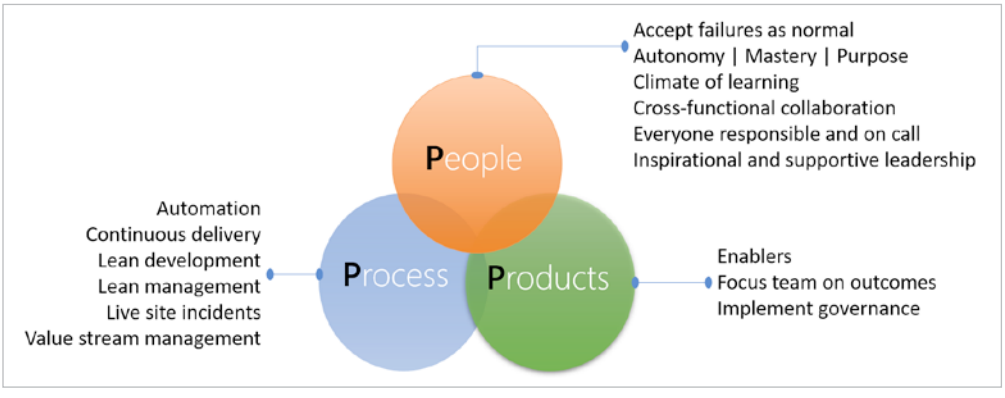
CHAPTERS

GET INVOLVED | ADDITIONAL RESOURCES

Breaking down walls between people, process, and products

DevOps transformation success hinges on removing the barriers inherent in an organization.

OVER THE PAST FEW MONTHS, I published a few articles that explore the three **P**s that are core to your digital transformation, enabling you to move from a resource-optimized business model based on capital expenses (CAPEX) to a market-optimized model based on operational expenses (OPEX). Among other benefits, the union of **P**eople, **P**rocess, and **P**roducts helps you drive core business values, such as increasing the flow of business value, shortening delivery cycle times, and enabling everyone to continuously learn, adapt, and improve.



Engineers (like me) typically focus on **products**, tools, and technologies. While these are pivotal to the success of DevOps, it is important to realize that you cannot buy or install DevOps! Products are enablers, not silver bullets. They allow you to focus on outcomes, such as automation, consistency, reliability, maintainability, progressively enabling or disabling features, and visualizing the flow of value.

When we talk about **process**, we often gravitate to automation to enable efficient, stable, and consistent value streams. It is important to also include goals to celebrate success as a team and organization, focus on quality from ideation to deprecation, create a lightweight and responsive change-management process, embrace loosely coupled architectures to enable scaling, and strive for multiple new-feature releases per day.

At the core of the transformation are **people** and their culture, not products, process, or even the organization's size. People need to buy into DevOps, understand how their roles will be affected, and take responsibility for their part of the transformation. People need to realize that DevOps is not limited to development and operations, even if its very name excludes other stakeholders. You need to break down all barriers and walls within your organization, bringing together all stakeholders, including development, data services, operations, security, and business.

To help you explore the three Ps, we've bundled a few articles for you to read.

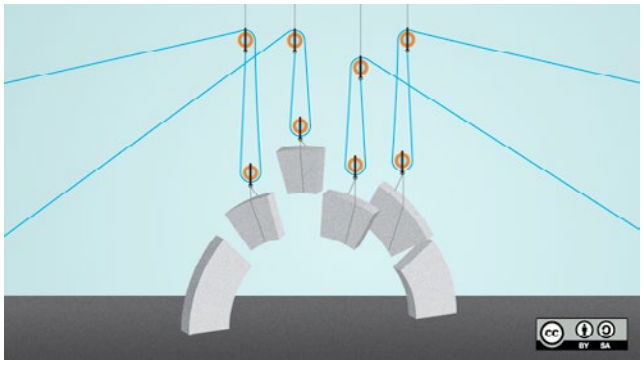
Blueprint for a team with a **DevOps mindset**

Culture is the greatest challenge when embarking on a DevOps transformation.

I'VE HAD THE PRIVILEGE to work with some of the brightest minds and leaders in my 33 years of software engineering. I've also been fortunate to work for a manager who made me question my career daily and systematically broke down my passion—like a destructive fire sucking the oxygen out of a sealed space. It was an unnerving period, but once I broke free, I realized I had the opportunity to reflect on one of the greatest anti-patterns for effective teams.

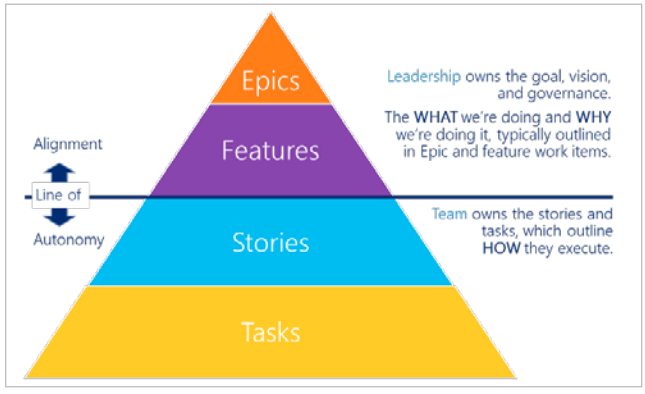
It should come as no surprise that the culture of an organization and its engineering teams is the greatest challenge when embarking on a DevOps mindset [1] transformation. The organization needs to influence through leadership and autonomy, promoting a culture of learning and experimentation, where failure is an opportunity to innovate, not persecute. Fear of retribution should be frowned upon like the archaic Indian practice of Sati [2]. Teams need to feel they are operating in a safe environment, understand what the transformation entails, and know how they will be affected.

So, what is the blueprint of an effective team? The concept of autonomy, self-organization, and self-management is core to agile practice. In addition, lean practices promote reducing waste, creating short feedback loops, using lightweight change approval, limiting work in progress (WIP), reflecting and acting on feedback, and transparently visualizing work management. All these strengths need to be reflected in your blueprint.



Let's review an effective team blueprint's genetic information.

As shown in the graphic below, **self-organization** is a natural process that creates order within the team. It outlines HOW the autonomous team collaborates and coordinates. **Self-management** defines how the diverse team members work together in their own way, aligned with a shared vision and governance, owned by the leadership.



Line of autonomy and governance.

Team size is a topic that creates vibrant discussions. Some say the ideal size is 7±2, others say 6±3, while others maintain that there is no upper limit. Another argument comes from anthropologist R.I.M. Dunbar [3], who says in his paper [4] on the relationship between humans' neocortex size and group size, "there is a cognitive limit to the number of individuals" in an effective team. He argues that

if you want a highly cohesive team, keep your team size smaller than 12.

Amazon CTO Werner Vogels' famous quote "You build it, you run it" [5] is reminiscent of the great thematic quote from Spider-Man: "With great power comes great responsibility." We need to foster ownership, accountability, and responsibility. Everyone in the team must be empowered, trained to run the business, responsible, and on call. When there is a live site incident, all designated response individuals, which includes members from the associated feature team, must join the 2am call to gather evidence, investigate, and remediate at the root-cause level.

EPIPHANY: The 2am wakeup call is the best motivation for a production-ready mindset. Pull any engineer into an early morning live site incident a few times, and the quality bar magically shall improve.

The 2am wakeup call is the best motivation for a production-ready mindset. Pull any engineer into an early morning live site incident a few times, and the quality bar magically shall improve.

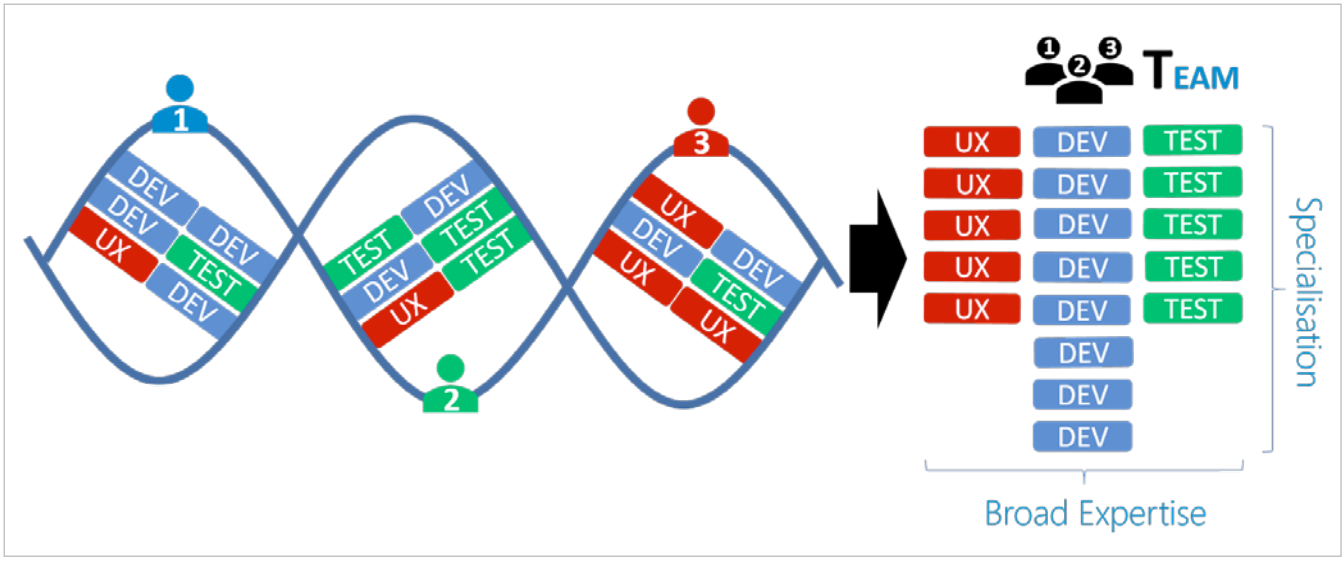
Cross-functional is another key part of genetic information for an effective team blueprint. Contrary to common belief, it does not imply that everyone in the team can do everything. Instead, as shown in the graph below, the cross-functional team is based on the concept of T-shaped skills or T-shaped persons. The horizontal bar of the T stands for broad expertise and the ability to collaborate with others, while the vertical bar represents the depth of a single expertise. Assume we have a three-person team comprised of experts in development, testing, and user experience. Each has his or her own T-shaped skills. When we combine the three experts' genetic material, we get a joint T-shaped team with specialization and broad expertise that can design, develop, test, and support features as a cross-functional team. The culture of learning not only ensures that the team's combined broad and specialized expertise is aligned with business needs, but it also empowers and motivates the team and its members. Information sharing and brown bag events are two excellent team-building tools.

Shortage of testers? No problem; with the support of the test expert, the development and user experience experts can temporarily blend into a test role. When testers automate testing, learn and share good coding practices, help improve feedback loops, and help the team broaden their skills beyond unit and regression testing, the line between developers and testers begins to blur. The developer and tester roles evolve and merge into the engineer role.

This raises the question: "What if team members refuse to be cross-functional?" There is no place for heroes or knights in shining armor in an effective cross-functional team—find them another home.

Inspire and **celebrate** success as a team and an organization whenever achieving a goal. It motivates the team, fuels team spirit, and acts as a major source of inspiration for others to excel. Your team can enjoy a round of nachos to confirm a job well done, play and learn with a game of GetKanban, encourage team members to celebrate with their family after a hectic sprint, or review and celebrate fast feedback as part of your regular standups. The options are endless—just ensure you spend time together as a team.

For example, our scrum master always reflects on our achievements whether we pass or fail a sprint. It eventually



dawned on me that it not only bonds us as a team, but others aspire to be like our team.

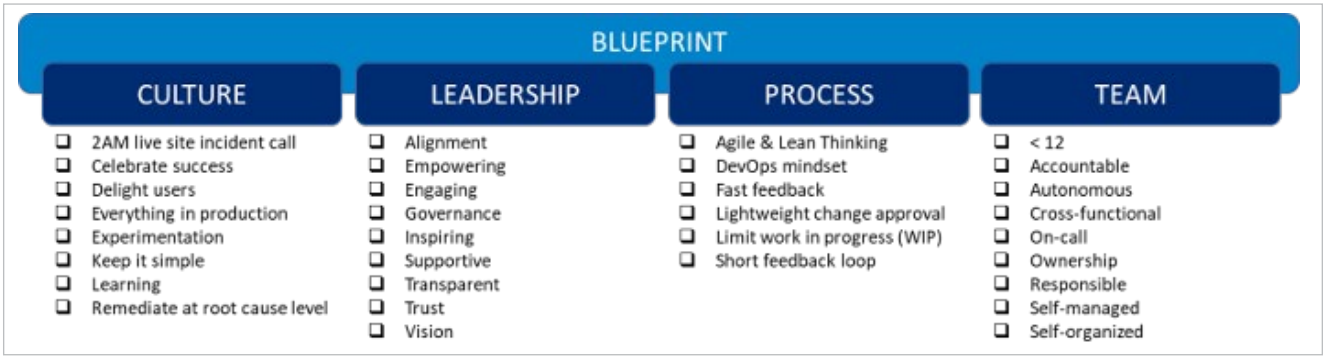
Finally, remember to **keep it simple** to reduce risk, cost of training, process, and products. Enable the just barely good enough (JBGE) approach. Simplicity positively affects maintainability and enables cross-functional teams to collaborate and fill in for each other more effectively.

You might be wondering why none of the above feels like a new revelation. As discussed in Analyzing the DNA of DevOps [6], we believe DevOps has inherited from decades of practices and learning, including waterfall, lean thinking, agile, and "real-world" 2am live site incident calls. We are not inventing anything new; instead, we continue to reflect, learn, and streamline our blueprint. The DevOps mindset is based on a few matured foundations, and DevOps lights up when the team is razor-focused on delivery, telemetry, support in production, and (most importantly) bonding together as a team. A team that doesn't enjoy being together is not a team; they are a group of individuals told to work together.

So, let's get back to the greatest anti-pattern I mentioned. Effective teams, who are autonomous, empowered, self-organizing, and self-managed are based on **trust**, **inspiration**, and **support** of their leadership. If any of these important pillars is missing, toxicity rises, passion declines, and you are an eyewitness to the most destructive anti-pattern that will doom any diverse, collaborative, and effective team.

Links

[1] https://github.com/wpschaub/DevOps-mindset-essentials
[2] https://en.wikipedia.org/wiki/Sati_(practice)
[3] https://en.wikipedia.org/wiki/Robin_Dunbar
[4] http://www.uvm.edu/~pdodds/files/papers/others/1993/dunbar1993a.pdf
[5] https://queue.acm.org/detail.cfm?id=1142065
[6] https://opensource.com/article/18/11/analyzing-devops-dna-traces-waterfall-and-other-frameworks



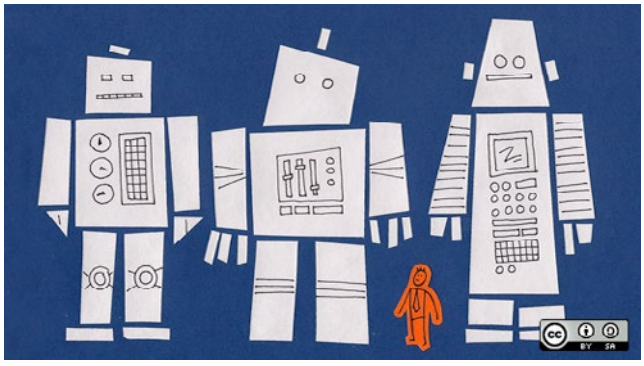
Blueprint for an effective team with a DevOps mindset.

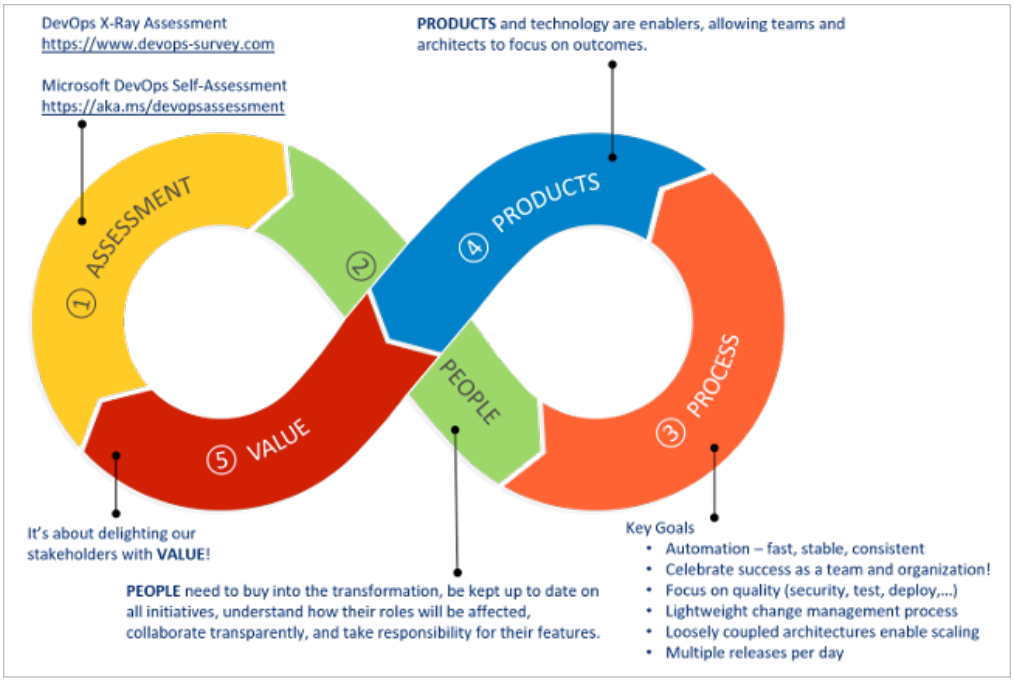
DevOps transformation: Key differences in small, midsize, and large organizations

When embracing a DevOps mindset, does an organization's size matter?

THE CONTINUOUS INNOVATION JOURNEY to DevOps is just that—continuous—meaning it's unlikely you'll ever reach the destination.

As depicted in the graphic below, the journey's steps are: **assess** and compare the organization with the rest of the industry, ensure that the **people** buy into the transformation, introduce an engineering **process** and **products** that enable teams to delight their stakeholders, continuously deliver **value**, and scale solutions from hundreds to millions of users. Conceptually, the transformation should be the same for any organization.





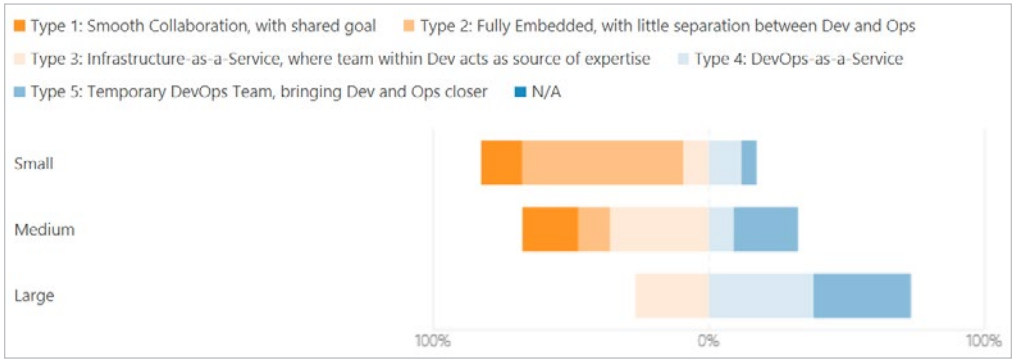
DevOps transformation journey

However, a DevOps transformation in a company with a handful of engineers is quite different from one with hundreds or thousands of engineers.

Instinctively, the DevOps journey should be easiest with small organizations, as they are typically abundant with passion and an appetite for change. However, small organizations tend to be more constrained on resources, infrastructure, and budget, while larger organizations tend to have more policies, governance, and politics that affect the transformation.

So, how does size matter? Here's what members

Ability to embrace DevOps based on organization size



DevOps patterns in organizations by size

of the community said in a poll about how size affects the ease of transforming.

Small and emerging organizations

In small organizations, management and engineering lines of responsibility tend to blur, which naturally creates a lean environment. Similarly, because resources are scarce, engineering typically wears multiple technical and operational hats, which organically creates cross-functional teams that are accountable for their solutions. Also, there is usually an abundance of passion and appetite for new products and processes in small and emerging organizations, which makes them ideal candidates for a transformation to a DevOps mindset. In addition, the DevOps transformation is becoming a necessity to compete with larger competitors.

As management and engineering in small organizations are lean, it is important to ensure there is a clear vision, engineering is empowered and accountable, the line of autonomy is respected, and the feedback and fail-fast processes are active. There also needs to be a clear "2 AM call" process for when a live site incident impacts the user experience—in many cases, even the CEO of a small organization must be on the standby roster. Engineers who are unwilling to fulfill a cross-functional role are a risk for these organizations. Engineers must have the right attitude, and some may need to find another home (within the organization or without).

Consider the **DevOps-as-a-Service** and **Fully Embedded** models Matthew Skelton discusses in "What Team Structure is Right for DevOps to Flourish?" [1] The former model is ideal for small organization and the latter is feasible, as Microsoft's part-time ALM | DevOps Ranger [2] community demonstrates.

Midsize organizations

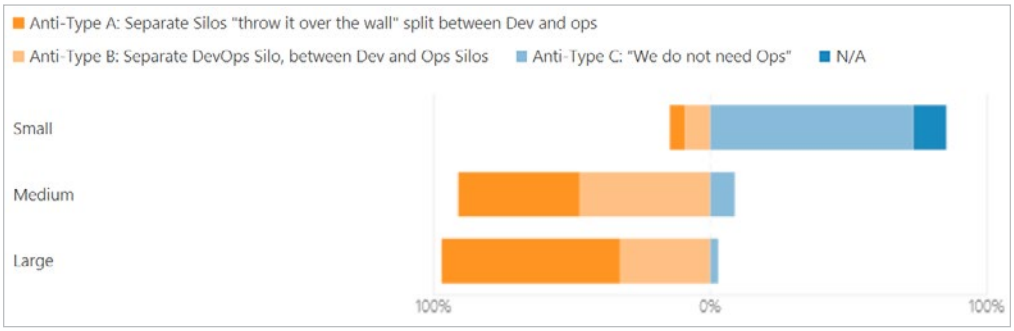
Midsize organizations' rising resources and budget often create an environment where politics, policies, and technical governance raise their ugly heads. You are likely to find a focused IT operations team, one or a few engineering teams, and the dreaded divide between development and operations. These are siloed environments where development builds the solutions and IT ops supports the infrastructure and solutions.

Midsize organizations must focus on breaking down the silos, creating a common language, and establishing technical governance that will unite the business, development, and operations leadership and engineering. Talking about manifestos (instead of governance) will reduce the angst and resistance from engineering.

Consider the **Temporary DevOps Team** Skelton describes in the "team structure" article linked above. It is a good, albeit temporary strategy to bring dev and ops closer together.

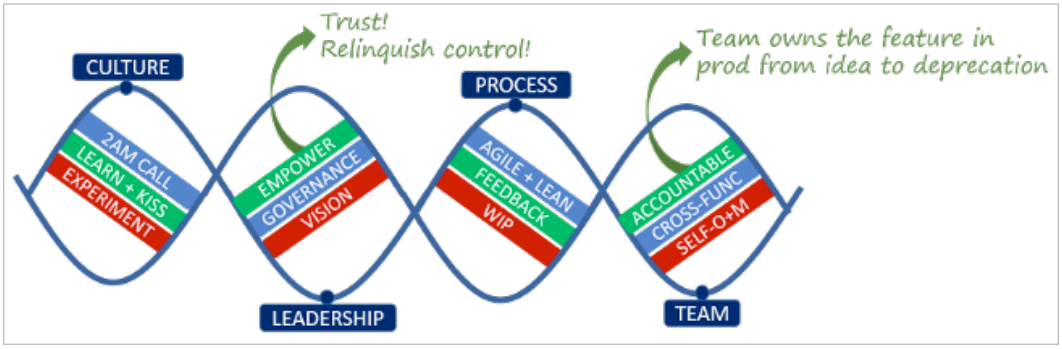
Large and established organizations

Few of us have the luxurious resources and budgets of large organizations such as Amazon, Facebook, Google, or Microsoft. Large organizations have the diversity, experience, and ability to divide their operations and devel-



DevOps anti-patterns in organizations by size

opment teams into many small, focused teams. For example, Microsoft has several product-focused units, such as Windows, Office, and Azure DevOps, each broken down into many small teams using a common engineering system. The teams naturally embrace agile and DevOps practices, supported by a unified enterprise-level vision and transformation strategy.



DNA: Culture, leadership, process, and team

Consider establishing a community of excellence or center of excellence to create special interest groups. Tapping thought leadership by skilled knowledge workers in this way can enable and provide the organization with best practices. These groups also support the concept of a DevOps-as-a-Service pattern to help move the organization to the next level of awareness and maturity, share knowledge, cut waste, and innovate. Ensure you have representation from business, development, and operations! While all types of organizations would benefit from DevOps-as-a-Service, resource requirements—budget and people—make it viable for medium and large organizations only.

Some large organizations feel like midsized organizations, with siloed leadership, business, operations, and development teams, just bigger and more segregated. Their challenges include encouraging the entire organization to consider and embrace DevOps practices, translating organizational business-speak into a common language, and introducing unfamiliar concepts such as scrums, kanban, sprint cadence, short delivery cycles, quick and lightweight feedback loops, and continuous experimentation in production. While many organizations see the value of bringing development and operations closer together, you will experience resistance from waterfall teams that are used to stringent sequences, detailed and predictable scope, and milestone-focused projects. You may also experience leadership unintentionally interfering with engineering teams, blurring the line of autonomy that separates the WHAT and WHY (owned by leadership) from the HOW and WHEN (owned by engineering).

The **Fully Embedded** or **Smooth Collaboration** models Skelton discusses in his blog have proven successful with large organizations that naturally embrace DevOps. For others, the **DevOps-as-a-Service** and **Temporary DevOps Team** models are ideal to bring dissimilar teams closer together.

The right strategy

Every organization is different, and assessing the right strategy is a combination of multiple characteristics, irrespective of size.

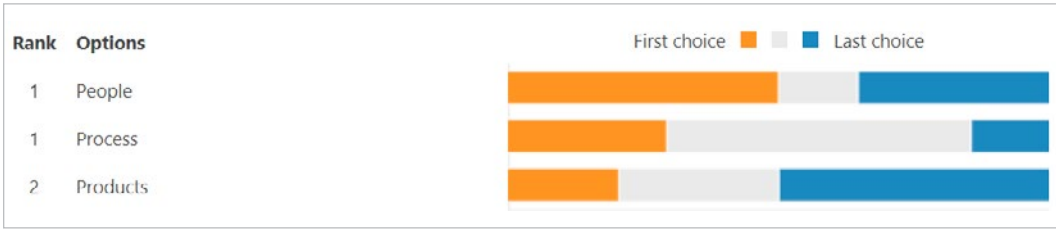
It is important to perform an assessment of the organization's **people**, **process**, and **products** to reveal its culture, leadership, teams, and appetite for change. More importantly, the assessment will highlight the areas that will transform naturally and the areas you need to nurture thoughtfully.

At the core, it is people and their culture, not an organization's size, that molds and differentiates organizations from one other.

Links

[1] https://blog.matthewskelton.net/2013/10/22/what-team-structure-is-right-for-devops-to-flourish/

[2] https://opensource.com/article/17/11/devops-rangers-transformation



People, process, products challenges in DevOps

Analyzing the DNA of DevOps

How have waterfall, agile, and other development frameworks shaped the evolution of DevOps? Here's what we discovered.

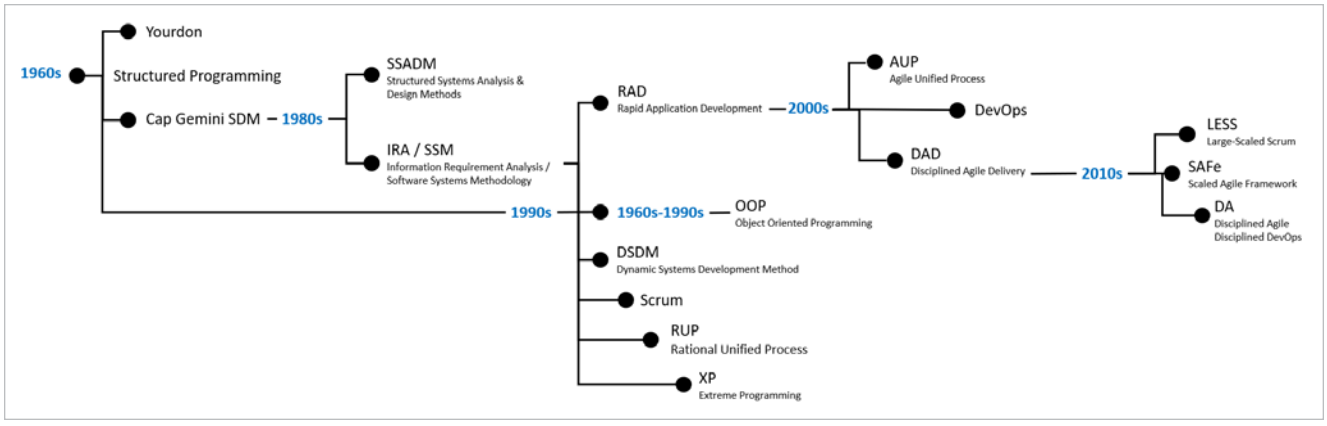
IF YOU WERE TO ANALYZE the DNA of DevOps, what would you find in its ancestry report?

This chapter is not a methodology bake-off, so if you are looking for advice or a debate on the best approach to software engineering, you can stop reading here. Rather, we are going to explore the genetic sequences that have brought DevOps to the forefront of today's digital transformations.

Much of DevOps has evolved through trial and error, as companies have struggled to be responsive to customers' demands while improving quality and standing out in an increasingly competitive marketplace. Adding to the challenge is the transition from a product-driven to a service-driven global economy that connects people in new ways. The software development lifecycle is becoming an increasingly complex system of services and microservices, both interconnected and instrumented. As DevOps is pushed further and faster than ever, the speed of change is wiping out slower traditional methodologies like waterfall.

We are not slamming the waterfall approach—many organizations have valid reasons to continue using it. However, mature organizations should aim to move away from wasteful processes, and indeed, many startups have a competitive edge over companies that use more traditional approaches in their day-to-day operations.

Ironically, lean, Kanban [1], continuous, and agile principles and processes trace back to the early 1940's, so DevOps cannot claim to be a completely new idea.



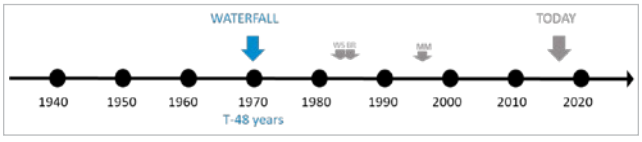
Haplogroup—Paternal line for SDLC

Let's start by stepping back a few years and looking at the waterfall, lean, and agile software development approaches. The figure below shows a "haplogroup" of the software development lifecycle. (Remember, we are not looking for the best approach but trying to understand which approach has positively influenced our combined 67 years of software engineering and the evolution to a DevOps mindset.)

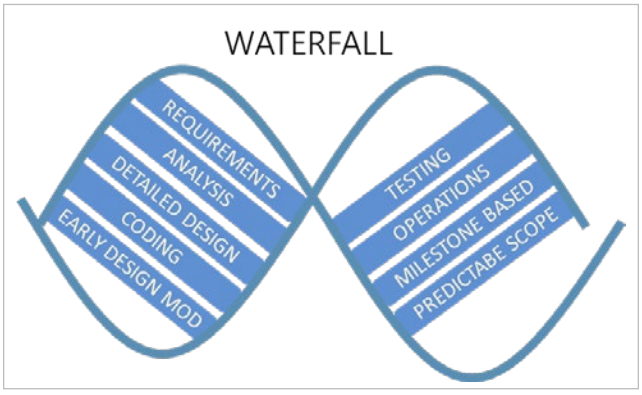
> "A fool with a tool is still a fool." –Mathew Mathai

The traditional waterfall method

From our perspective, the oldest genetic material comes from the waterfall [2] model, first introduced by Dr. Winston W. Royce in a paper published in the 1970's.

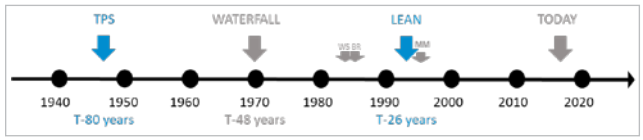


Like a waterfall, this approach emphasizes a logical and sequential progression through requirements, analysis, coding, testing, and operations in a single pass. You must complete each sequence, meet criteria, and obtain a signoff before you can begin the next one. The waterfall approach benefits projects that need stringent sequences and that have a detailed and predictable scope and milestone-based development. Contrary to popular belief, it also allows teams to experiment and make early design changes during the requirements, analysis, and design stages.

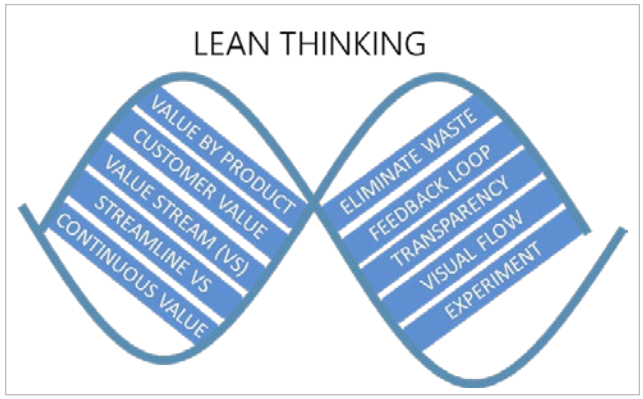


Lean thinking

Although lean thinking dates to the Venetian Arsenal in the 1450s, we start the clock when Toyota created the Toyota Production System [3], developed by Japanese engineers between 1948 and 1972. Toyota published an official description of the system in 1992.



Lean thinking is based on five principles [4]: *value*, *value stream*, *flow*, *pull*, and *perfection*. The core of this approach is to understand and support an effective value stream, eliminate waste, and deliver continuous value to the user. It is about delighting your users without interruption.



Kaizen

Kaizen is based on incremental improvements; the **Plan->Do->Check->Act** lifecycle moved companies toward a continuous improvement mindset. Originally developed to improve the flow and processes of the assembly line, the Kaizen concept also adds value across the supply chain. The Toyota Production system was one of the early implementors of Kaizen and continuous improvement. Kaizen and DevOps work well together in environments where workflow goes from design to production. Kaizen focuses on two areas:

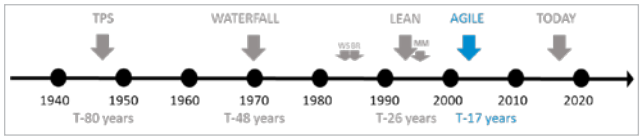
- Flow
- Process

Continuous delivery

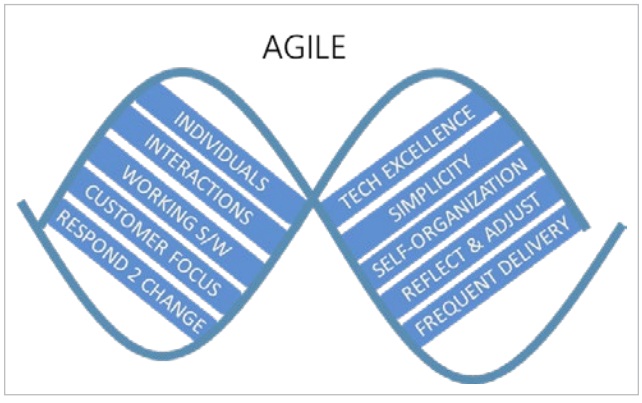
Kaizen inspired the development of processes and tools to automate production. Companies were able to speed up production and improve the quality, design, build, test, and delivery phases by removing waste (including culture and mindset) and automating as much as possible using machines, software, and robotics. Much of the Kaizen philosophy also applies to lean business and software practices and continuous delivery deployment for DevOps principles and goals.

Agile

The Manifesto for Agile Software Development [5] appeared in 2001, authored by Alistair Cockburn, Bob Martin, Jeff Sutherland, Jim Highsmith, Ken Schwaber, Kent Beck, Ward Cunningham, and others.



Agile [6] is not about throwing caution to the wind, ditching design, or building software in the Wild West. It is about being able to create and respond to change. Agile development is based on twelve principles [7] and a manifesto that values individuals and collaboration, working software, customer collaboration, and responding to change.



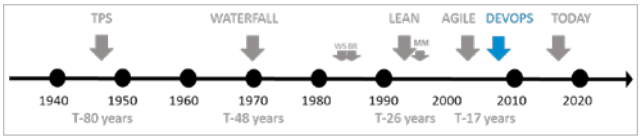
Disciplined agile

Since the Agile Manifesto has remained static for 20 years, many agile practitioners have looked for ways to add choice and subjectivity to the approach. Additionally, the Agile Manifesto focuses heavily on development, so a tweak toward solutions rather than code or software is especially needed in today's fast-paced development environment. Scott Ambler and Mark Lines co-authored Disciplined Agile Delivery [8] and The Disciplined Agile Framework [9], based on their experiences at Rational, IBM, and organizations in which teams needed more choice or were not mature enough to implement lean practices, or where context didn't fit the lifecycle.

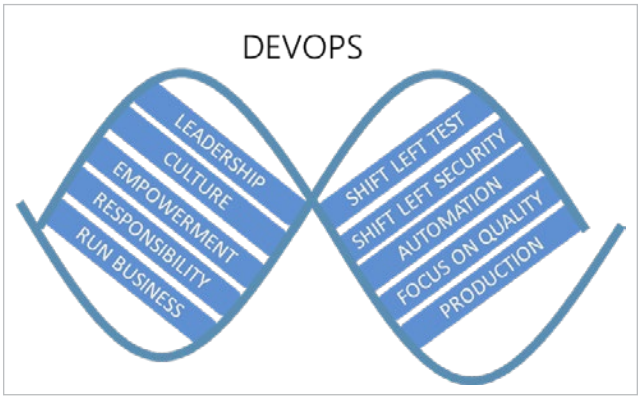
The significance of DAD and DA is that it is a process-decision framework [10] that enables simplified process decisions around incremental and iterative solution delivery. DAD builds on the many practices of agile software development, including scrum, agile modeling, lean software development, and others. The extensive use of agile modeling and refactoring, including encouraging automation through test-driven development (TDD), lean thinking such as Kanban, XP [11], scrum [12], and RUP [13] through a choice of five agile lifecycles, and the introduction of the architect owner, gives agile practitioners added mindsets, processes, and tools to successfully implement DevOps.

DevOps

As far as we can gather, DevOps emerged during a series of DevOpsDays in Belgium in 2009, going on to become the foundation for numerous digital transformations. Microsoft principal DevOps manager Donovan Brown [14] defines DevOps as "the union of people, process, and products to enable continuous delivery of value to our end users."

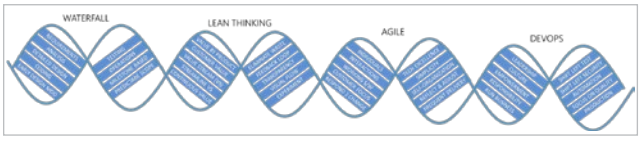


Let's go back to our original question: What would you find in the ancestry report of DevOps if you analyzed its DNA?



We are looking at history dating back 80, 48, 26, and 17 years—an eternity in today's fast-paced and often turbulent environment. By nature, we humans continuously experiment, learn, and adapt, inheriting strengths and resolving weaknesses from our genetic strands.

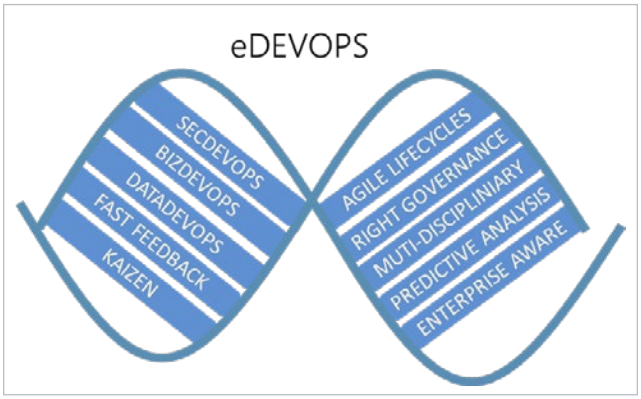
Under the microscope, we will find traces of waterfall, lean thinking, agile, scrum, Kanban, and other genetic material. For example, there are traces of waterfall for detailed and predictable scope, traces of lean for cutting waste, and traces of agile for promoting increments of shippable code. The genetic strands that define when and how to ship the code are where DevOps lights up in our DNA exploration.

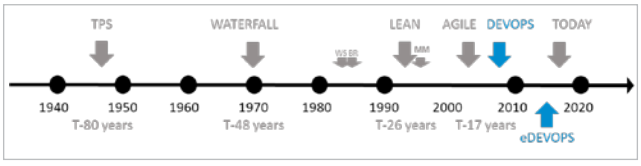


You use the telemetry you collect from watching your solution in production to drive experiments, confirm hypotheses, and prioritize your product backlog. In other words, DevOps inherits from a variety of proven and evolving frameworks and enables you to transform your culture, use products as enablers, and most importantly, delight your customers.

If you are comfortable with lean thinking and agile, you will enjoy the full benefits of DevOps. If you come from a waterfall environment, you will receive help from a DevOps mindset, but your lean and agile counterparts will outperform you.

eDevOps





In 2016, Brent Reed coined the term *eDevOps* (no Google or Wikipedia references exist to date), defining it as "a way of working (WoW) that brings continuous improvement across the enterprise seamlessly, through people, processes and tools."

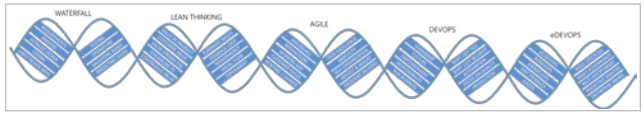
Brent found that agile was failing in IT: Businesses that had adopted lean thinking were not achieving the value, focus, and velocity they expected from their trusted IT experts. Frustrated at seeing an "ivory tower" in which siloed IT services were disconnected from architecture, development, operations, and help desk support teams, he applied his practical knowledge of disciplined agile delivery and added some goals and practical applications to the DAD toolset, including:

- Focus and drive of culture through a continuous improvement (Kaizen) mindset, bringing people together even when they are across the cubicle
- Velocity through automation (TDD + refactoring everything possible), removing waste and adopting a TOGAF [15], JBGE (just barely good enough) approach to documentation
- Value through modeling (architecture modeling) and shifting left to enable right through exposing anti-patterns while sharing through collaboration patterns in a more versatile and strategic modern digital repository

Using his experience with AI at IBM, Brent designed a maturity model for eDevOps that incrementally automates dashboards for measuring and decision-making purposes so that continuous improvement through a continuous deployment (automating from development to production) is a real possibility for any organization. eDevOps in an effective transformation program based on disciplined DevOps that enables:

- Business to DevOps (BizDevOps),
- Security to DevOps (SecDevOps)
- Information to DevOps (DataDevOps)
- Loosely coupled technical services while bringing together and delighting all stakeholders
- Building potentially consumable solutions every two weeks or faster
- Collecting, measuring, analyzing, displaying, and automating actionable insight through the DevOps processes from concept through live production use
- Continuous improvement following a Kaizen and disciplined agile approach

The next stage in the development of DevOps



Will DevOps ultimately be considered hype—a collection of more tech thrown at corporations and added to the already extensive list of buzzwords? Time, of course, will tell how DevOps will progress. However, DevOps' DNA must continue to mature and be refined, and developers must understand that it is neither a silver bullet nor a remedy to cure all ailments and solve all problems.

```
DevOps != Agile != Lean Thinking != Waterfall
DevOps != Tools !=Technology
DevOps ⊂ Agile ⊂ Lean Thinking ⊂ Waterfall
```

Links

[1] https://en.wikipedia.org/wiki/Kanban
[2] https://airbrake.io/blog/sdlc/waterfall-model
[3] https://en.wikipedia.org/wiki/Toyota_Production_System
[4] https://www.lean.org/WhatsLean/Principles.cfm
[5] http://agilemanifesto.org/
[6] https://www.agilealliance.org/agile101
[7] http://agilemanifesto.org/principles.html
[8] https://books.google.com/books?id=CwvBEKsCY2gC
[9] http://www.disciplinedagiledelivery.com/books/
[10] https://en.wikipedia.org/wiki/Disciplined_agile_delivery
[11] https://en.wikipedia.org/wiki/Extreme_programming
[12] https://www.scrum.org/resources/what-is-scrum
[13] https://en.wikipedia.org/wiki/Rational_Unified_Process
[14] http://donovanbrown.com/
[15] http://www.opengroup.org/togaf

Co-authored by Brent Reed

Brent Aaron Reed is a pragmatic leader who has been involved at the forefront of technology and its application in bringing value to people. Brent strives for continuous improvement through education, awareness, collaboration and passion. Brent is certified in Microsoft, Security+, Agile & Disciplined Agile and many other frameworks and technologies.

Visualizing a DevOps mindset

Use this graphical analysis to help develop a DevOps strategy for your organization.

THESE DAYS, organizations are moving from a resource-optimized business model based on capital expenses (CAPEX [1]) to a market-optimized model based on operational expenses (OPEX [2]). What's driving this shift? Reducing time to market and continuously delighting customers with value.

Welcome to digital transformation. Are you ready to embrace a DevOps mindset in your organization?

As defined by DevOps manager Donovan Brown [3], "DevOps is the union of people, process, and products to enable continuous delivery of value to our end users."

DevOps is not about magical unicorns and colorful rainbows. It's a journey of continuous learning and improvement, with a destination you never quite get to. It's the reason that all of the images herein are based on the infinity symbol:

Being a visual-minded person, I created a presentation with posters [4] for the recent Global DevOps Bootcamp (GDBC) [5]. This annual community-driven event is hosted around the globe to create an environment in which participants collaboratively explore digital transformation and DevOps insights.

Let's explore four of the quick-reference posters [6] (also referred to as *visuals* and *infographics*). For a more in-depth discussion of DevOps, refer to The DevOps Handbook [7], by Gene Kim, Jez Humble, Patrick Debois, and John Willis.

Practices

Based on the DevOps Assessment [8], the first two posters are intended to be used when reviewing the assessment results with all stakeholders. The first one introduces five key practices:

Top performers encourage a culture that fosters a growth mindset, reward innovation, collaboration, experimentation, learning, and user empathy. Strive for a process with responsive application delivery, flexible scheduling, and iterative experiments. Monitor, identify and mitigate issues, and continuously eliminate wasteful bottlenecks. Only valuable key performance indicators are measured and used to strive for better outcomes, such as a low change failure rate (CFR),

minimal time to recover (MTTR), and remediation of issues at the root level. Lastly, technology, which is an enabler, is the focus of the next poster.

Technology

Here's a companion of the practices poster, focused on technology:

Version control manages versions of your application, configuration, infrastructure, and other code. It enables team collaboration and monitoring activities such as deployments. Top performers use topic branches for short-term isolation, continuously merge changes into master, review, and audit using Git pull requests, and version everything.

Testing must be viewed as a continuous activity, embedded into both the developer workflow and the continuous integration (CI) and continuous delivery (CD) pipeline.

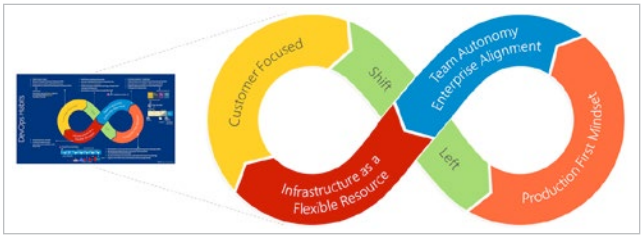
The cloud enables you to effectively provision your infrastructure and move as fast as necessary.

Lastly, monitoring enables you to form a hypothesis, validate or disprove experiments, proactively detect issues as they occur, and understand application health.

The black bar on the right of the poster lists products to consider when you're investigating technology for your development, production, common engineering, and other environments. Provide feedback on the listed products and regularly update this volatile and opinionated part of the visual.

Habits

Based on the Moving 65,000 engineers to DevOps with VSTS [9] story, this poster focuses on the five key habits we learned about during our transformation. The customer-focused, team autonomy and enterprise alignment, and shift-left habits are evolutions of Agile, and the production-first mindset and infrastructure as a flexible resource are particular to a DevOps mindset.



Customer-focused is part of our quest to delight customers and our obsession with delivering customer value. You must actively listen to your users, progressively enable and disable features, perform continuous experiments, and measure key performance indicators. Use all available feedback to maximize learnings and influence value.

Shift left encourages reviews, validations, and approvals for both testing and security as early as possible in the feature delivery cycle to drive quality and a fail-fast mindset. When technical debt exceeds a predefined limit (for example, five bugs per engineer), encourage feature teams to suspend feature work until the technical debt is paid down.

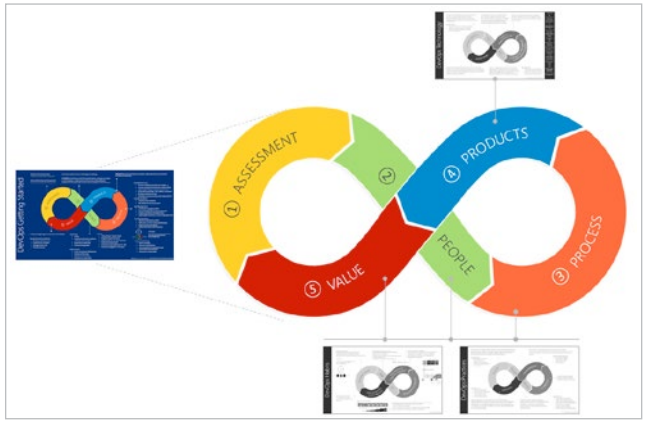
Team autonomy and enterprise alignment are concerned with what, how, and why we build. You need a common cadence, or heartbeat, across your organization to enable all leadership and feature teams to collaborate transparently and effectively. The most effective feature teams own a feature from idea to production, with autonomy on how they develop and support their features.

Production-first is a mindset that does not differentiate how features and bugs are handled during development, testing, and operational support. Everything should be automated, versioned, and fine-tuned in production. Lean on ring-based deployment and rings [10] to limit the blast radius of feature changes in production, remediate all issues at the root cause level, and remember to be transparent about issues, root cause, and resolution (as a user, I'm much more understanding if I have an insight into issues).

Infrastructure as a flexible resource describes how solution architectures are adapted to the cloud, containerization, and microservices. Adopt a pragmatic transformation that makes sense for your organization, goals, products, and culture. As with the previous habits, it's important to favor autonomy over a descriptive architecture and not to transform everything all at once.

Getting started

The last visualization combines all of the above and suggests five steps to getting started with DevOps:



I prefer to start with the **assessment** to help identify key areas that can be improved.

- Assessments provide a benchmark of your DevOps mindset and performance against the rest of the industry. It's important to understand where you're doing well and where investment will help take you to the next level. Both the DORA [11] and Microsoft [12] DevOps Assessments are great starting points. In addition, gather metrics to use as a base to measure progress—for example, deployment frequency, lead time for changes, mean time to repair, and change failure rate.
- People and culture are your biggest challenges. Everyone needs to buy into the transformation, understand how they will be affected, encourage transparency, be engaged, and take full responsibility for their value streams. This includes leadership, which needs to be supportive, inspirational, empowering, and drive a clear vision. You'll make or break the transformation as a team.

Without committed people and an experimental culture, the rest of the DevOps transformation journey is futile.

- Process is your engineering system, which enables the teams to manage live site incidents, use lean management and development, and continuously deliver value. A common engineering system introduces consistency, empowers feature teams, and enables and encourages everyone to support and contribute to each other. Your top process goals should include a focus on quality, a loosely coupled architecture to enable scaling, lightweight management, automation, multiple releases per day, and celebration of success as a team and as an organization.
- Products are the easiest link in the chain. They enable everyone to focus on what's important: Delivering value to end users.
- Value is all about delighting users. Key performance indicators include deployment frequency, lead time for changes, change failure rate, and time to recover.

Whether you tackle all these steps all at once ("big bang"), step-by-step ("peeling an onion"), or gradually innovate your value chain across all steps ("broad-spectrum innovation") is your choice. Just be pragmatic.

Improvement is possible for everyone if leadership provides consistent support and team members commit themselves to the work. –*Accelerate: The Science of Lean Software and DevOps* [13], by Nicole Forsgren, Jez Humble, and Gene Kim

Which visuals do you like? Which ones add no value? What is missing? Let's collaborate [14] to demystify DevOps and help you and your users shine. Users need to understand that they are not alone and know they can rely on proven practices and real-world learning.

Looking forward to your feedback and pull requests!

Links

[1] https://www.investopedia.com/terms/c/capitalexpenditure.asp
[2] https://en.wikipedia.org/wiki/Operating_expense
[3] http://donovanbrown.com/post/what-is-devops
[4] https://github.com/wpschaub/DevOps-mindset-essentials/blob/master/src/presentations/devops-mindset-essentials-gdbc.pdf
[5] https://globaldevopsbootcamp.com/
[6] https://github.com/wpschaub/DevOps-mindset-essentials/tree/master/src/posters
[7] http://a.co/92KSNxJ
[8] https://aka.ms/devopsassessment
[9] https://www.slideshare.net/WillyPeterSchaub/devconf-moving-65000-microsofties-to-devops-with-visual-studio-team-services
[10] https://opensource.com/article/18/2/feature-flags-ring-deployment-model
[11] https://www.devops-survey.com/
[12] https://aka.ms/devopsassessment
[13] https://t.co/smb82Y4i0M
[14] https://github.com/wpschaub/devOps-mindset-essentials

Deploying new releases: Feature flags or rings?

Use deployment rings to progressively expose a new release, and feature flags to fine-tune each release in production.

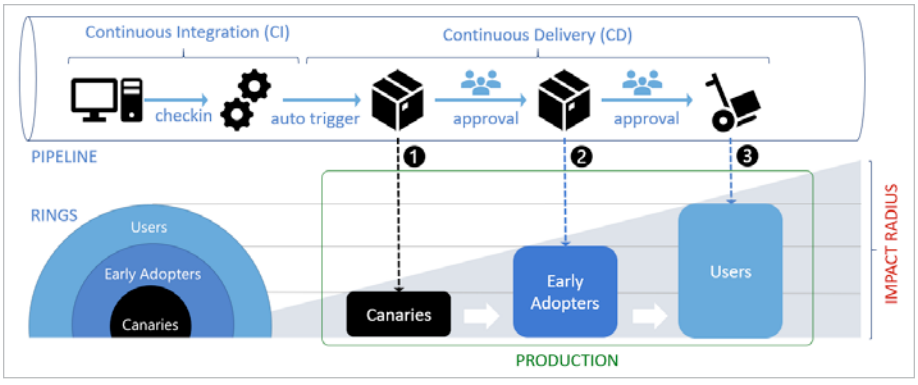
DEVOPS ENABLES us to deliver at speed, learn from production feedback, make better educated decisions, and increase customer satisfaction, acquisition, and retention. We need to fail fast on features that result in an indifferent or negative user experience and focus on features that make a positive difference. Progressive exposure is a DevOps practice, based on feature flags and ring-based deployment, that allows us to expose features to selected users in production, to observe and validate before exposing them to all users.

You're probably asking yourself whether to use ring-based deployments, feature flags, or both to support progressive exposure in your environment. Let's start by exploring these strategies.

Rings and feature flags

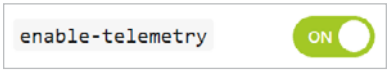
Ring-based deployment was first discussed in Jez Humble's book, Continuous Delivery [1], as canary deployments. Rings limit impact on end users while gradually deploying and confirming change in production. With rings, we evaluate the impact, or "blast radius," through observation, testing, diagnosis of telemetry, and most importantly, user feedback. Rings make it possible to progressively deploy binary bits and have multiple production releases running in parallel. You can gather feedback without the risk of affecting all users, decommission old releases, and distribute new releases when you are confident that everything is working properly.

The following diagram show an implementation of the ring-based deployment process:



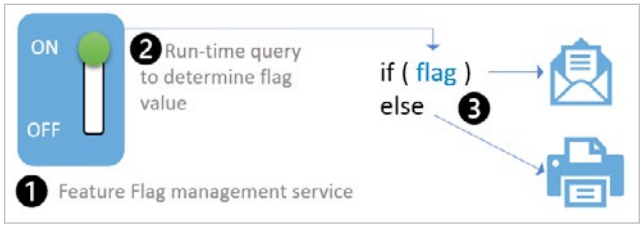
When your developers complete a pull request with proposed changes to the master branch, (1) a continuous integration build performs the build, unit testing, and triggers an automatic release to the Canaries environment in production. When you're confident that the release is ready for user acceptance and exploratory testing in production, (2) you approve the release to the Early Adopters ring. Similarly, when you're confident that the release is ready for prime time, (3) you approve the release to the Users ring. The names and number of rings depends on your preferences, but it's important that all rings are using the same production environment.

Feature flags were first popularized by Martin Fowler [2]. Flags decouple release deployment and feature exposure, give run-time control down to the individual user, and enable hypothesis-driven development. Using and tying feature flags back to telemetry allows you to decide if a feature helps to increase user satisfaction, acquisition, and retention. You can also use feature flags to do an emergency roll-back, hide a feature in a region where it shouldn't be available, or enable telemetry as needed.



A typical feature flag implementation is based on (1) a management service

that defines the flag, (2) a run-time query to figure out the value of the flag, and (3) an if-else programming construct, as shown:



Both the feature flags and ring-based deployment model strategies are invaluable, whether you're working with an open source extension [3] community or moving 65,000 engineers to DevOps [4].

Back to the question: Should you use feature flags, rings, or both?

The quote "You do not respond to a mosquito bite with a hammer," by Patrick L.O. Lumumba, comes to mind.

We use both rings and feature flags to progressively expose a new release in production, whether it's a hot fix or feature release for our commercial product, affecting 65,000 engineers and eventually hundreds of thousands of users as the blast or impact radius of the release increases. Feature flags allow us to progressively reveal [5] new features of each release, perform A/B testing, and experiment in production. Because we're working with cloud services and extensions, we have a continuous feedback loop with our users and the ability to fine-tune each release by toggling feature flags.

For our open source community extensions, we primarily use ring-based deployment to progressively expose a new release to canary, early adopters, and users, as outlined in the table below. We're gradually implementing feature flags in selected extensions to experiment [6] and gather experience in fine-tuning features and managing the associated technical debt.

The quasi-continuous delivery mode [7] is another example of using both strategies to deploy new features to 1% of the users in the first ring, then 20%, 50%, and 100%, continuing with the same pattern to the second ring, and so on.

You can use either ring-based deployment or feature flags to implement the progressive exposure DevOps practice—they are symbiotic. It all boils down to how cautious you want to be when rolling out releases and exposing features. I recommend that you experiment with both. Start by using deployment rings to progressively expose a new release, then use feature flags to fine-tune each release in production.

I think of a  when using the ring-based deployment model to deploy a release and a  when using feature flags to fine-tune the release.

Happy ringing and flagging!

Comparing rings with flags within the context of our open source extensions [8]:

	DEPLOYMENT RING	FEATURE FLAG
Progressive exposure	Yes	Yes
A/B Testing	All users within ring	All or selected users
Cost	Production environment maintenance	Feature Flag database and code maintenance
Primary use	Manage impact "blast radius"	Show or hide features in a release
Blast radius - Canaries	0-9 canary users	0, all, or specific canary users
Blast radius - Early Adopters	10-100 early adopter users	0, all, or specific early adopter users
Blast radius - Users	10000+ of users	0, all, or specific users

For more details, refer to Software Development with Feature Toggles [9], Phase the roll-out of your application through rings [10], and Our Feature Flag Investigations [11].

Links

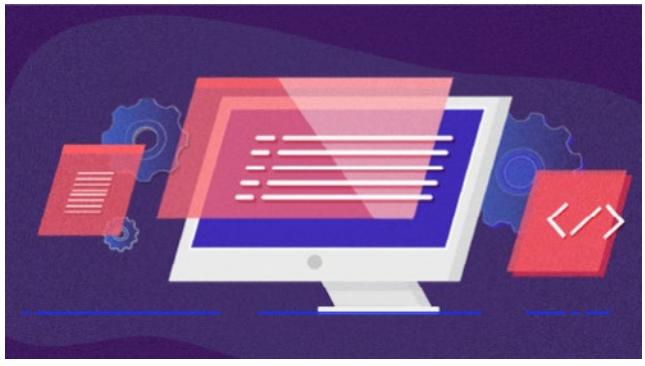
[1] https://www.continuousdelivery.com/
[2] https://martinfowler.com/bliki/FeatureToggle.html
[3] https://aka.ms/vsarsolutions#Extensions
[4] https://aka.ms/devops
[5] https://youtu.be/ed3ziUDq_n0
[6] https://blogs.msdn.microsoft.com/visualstudioalmrangers/tag/launchdarkly/
[7] https://code.facebook.com/posts/270314900139291/rapid-release-at-massive-scale/
[8] https://aka.ms/vsarsolutions#Extensions
[9] https://msdn.microsoft.com/en-ca/magazine/dn683796.aspx
[10] https://docs.microsoft.com/en-us/vsts/articles/phase-rollout-with-rings
[11] https://blogs.msdn.microsoft.com/visualstudioalmrangers/tag/launchdarkly/

What's the cost of feature flags?

Here's what you need to know about managing feature flags in a progressive exposure environment.

IN A PREVIOUS ARTICLE, I introduced feature flags and ring-based deployments, both enablers for the DevOps practice of progressive exposure.

Progressive exposure enables us to mitigate the impact of changes as they occur, perform iterative experiments, assess features, and get rapid feedback on every change—all in production. Feature flags, for example, enable you to perform short-lived experiments, isolate unfinished work, fine-tune continuous releases, and dynamically manage long-lived operational configurations and permissions.



It's a practice that not only leads to happy customers but enables us to create effective and motivated feature teams.

You're probably asking, "What's the catch?"

For **ring-based deployments**, your primary cost is to manage the production environments covered by the rings with a "production-first" mindset. You need to minimize the "blast radius" for each release, monitor each release, and mitigate root issues quickly. For **feature flags**, you need to manage your feature flag product, manage technical debt, and develop an insight into the implications of simply "flipping a flag."

Let's explore some of the costs of feature flags.

Product investment and operational cost

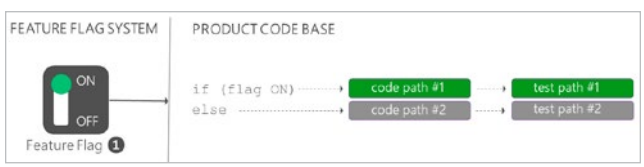
You need to investigate and find the right feature flag solution for your environment. Some important considerations include seamless integration with your DevOps process and products, simple and cost-effective management of flags, ability to perform an emergency roll-back, and support for auditing, fine-tuned permissions, and security. For example, if you're managing feature flags down to a specific user, you're likely capturing personal information and entering the realm of the new Global Data Protection Regulation (GDPR) [1].

Don't build your own custom feature flag solution. There are enough options available, such as the FeatureToggle [2], NFeature [3], FeatureSwitcher [4], togglz [5], and ff4j [6] frameworks, and software as a service (SaaS) such as LaunchDarkly [7]. With the latter, you delegate maintenance, updates, and infrastructure to your SaaS provider so you can focus product features and deliver value to your customers.

Technical debt

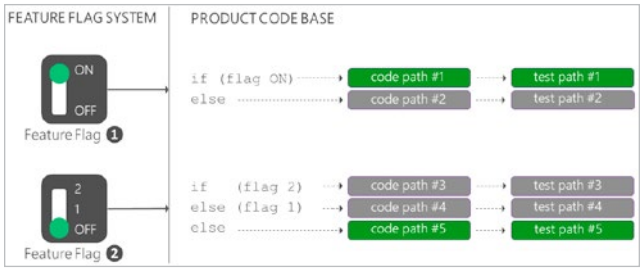
With feature flags, we break up a product into independent parts that can be released separately, giving our feature teams and business control over who gets which feature and when. By breaking up your product, you're adding a level of complexity, which needs maintenance to avoid stale flags and associated code.

For example, when we add a simple ON|OFF feature flag to isolate a feature, we're adding an **if-else** code construct and **doubling** our code and test paths, as shown below. If it's an experimental feature flag, its lifespan is typically weeks, after which it needs to be removed to avoid technical debt. For other feature flags, the lifespan may be longer; however, the same principle applies: Remove the feature flags and associated code as soon as you do not need them.

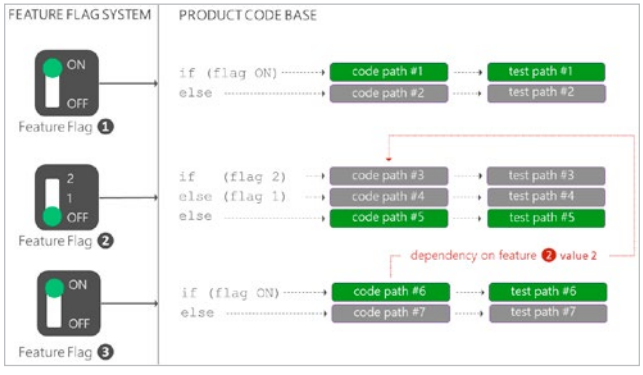


When we add a multi-value OFF|1|2 feature flag, we **multiply** the feature code and test paths. After adding two feature

flags, we are maintaining two features, with five code and test paths. Every path needs to be validated and tested with every change as we have no guarantee when a feature flag will be toggled.

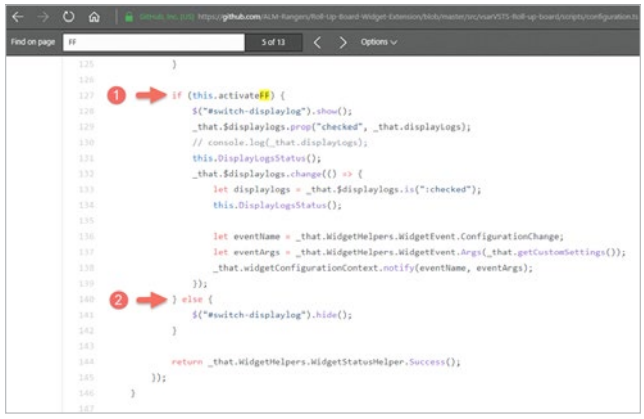


But there's more: Let's add another simple ONIOFF feature flag for a new feature for which we would like to collect telemetry to examine a hypothesis. It again **doubles** the feature code and test paths, incrementing the paths that need to be validated to seven. More importantly, it introduces a dependency on a specific version of the second feature. Do we wait for the dependency to be enabled? Do we rely on an isolated and potentially incomplete feature? Who ensures that a feature flag is not inadvertently toggled to satisfy a dependency? Great questions, which need to be answered as part of your process transformation to encourage flexible scheduling, iterative experiments, and close team collaboration and to facilitate real-time ownership and management of these challenges.

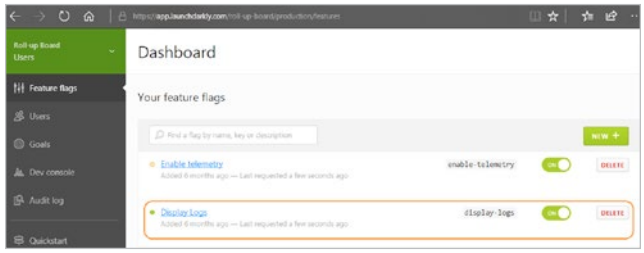


Imagine a product with hundreds of feature flags. How do you identify stale feature flags and associated code and test paths adding to our technical debt (cost)? How do you convince your feature teams to *change* and *remove* code from a fully functional product? The feature teams need to own the feature from sunrise (idea) to sunset (deprecate), use a common engineering process, and apply consistent code and naming conventions. Identifying and removing stale feature flags and code must be simple.

Let's have a quick look at an extract from the Roll-up Board [8] extension, which shows the ON and OFF code paths for a feature flag that checks the value of **activateFF**.



In the feature flag admin system, the feature has a friendly display name, **Display Logs**, and **display-logs** flag.



At a glance, the relationship between **activateFF** and **display-logs** is not obvious. As an engineer, I'm reluctant to make changes to the code without further investigation (cost).

Understanding of the implications of flipping a flag

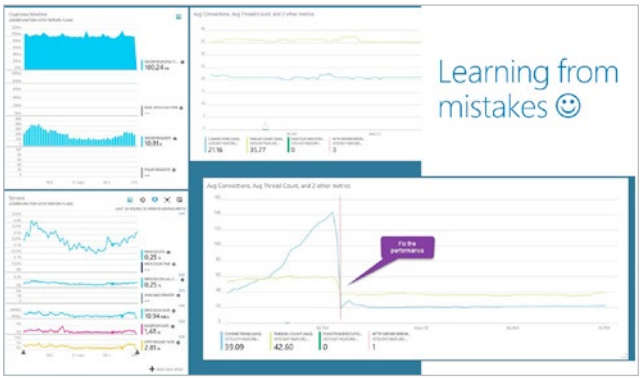
There's one more important cost you need to consider. Flipping a feature flag is simple—the change ripples through production quickly, and your users start using your new feature with excitement. You're happy with the feature you just enabled, but are you confident that you understand all the side effects of flipping the flag?

We often share the following two experiences, which demonstrate that even with the best process, we can have bad days that result in bad customer experiences.

- A rough patch [9] —The team flipped a feature flag at a big marketing event when corporate vice president Brian Harry was on stage. As described in the blog post, it didn't go well. The product experienced unexpected authentication failures and eventually buckled under load.
- How we learned about the 503 meltdown [10] —The team enabled feature flags in one of their most popular extensions. Users experienced 503 errors, followed by severe performance issues, and eventually the Azure Functions handling feature flags failed under load.

In both cases, there was a "failure under load." It's important to flip a feature flag in a canary or early adopter environment and simulate anticipated loads a few days before flipping the feature flag for all users. This is particularly true for big marketing events, where first impressions count.

A solid engineering process and live telemetry enabled us to detect the issues as they occurred, identify the root cause, and mitigate the impact.



It's important to learn from these mistakes, explore potential implications, have user empathy, and be transparent about issues, root cause, and resolution of bad days like ours. Users with an insight are typically more tolerant and supportive of your continuous journey of learning and innovation.

DevOps is a journey of continuous learning and improvement, with a destination you never quite get to!

Once you are cognizant of and manage the risks and costs, your feature teams will be able to progressively expose releases using ring-based deployments and fine-tune them using feature flags.

Enjoy observing your motivated feature teams—and more importantly, your happy customers!

References

- A Rough Patch [11]
- Continuous Delivery by Jez Humble [12]
- Feature Toggles by Martin Fowler [13]
- How we checked and fixed the 503 error and performance issue in our Azure Function [14]
- Moving 65,000 engineers to DevOps [15]
- Phase the features of your application with feature flags [16]
- Phase the roll-out of your application through rings [17]

Links

[1] https://ec.europa.eu/commission/priorities/justice-and-fundamental-rights/data-protection/2018-reform-eu-data-protection-rules_en
[2] https://github.com/jason-roberts/FeatureToggle
[3] https://github.com/benaston/NFeature
[4] https://github.com/mexx/FeatureSwitcher
[5] https://github.com/togglz/togglz
[6] https://github.com/clun/ff4j
[7] https://www.launchdarkly.com/
[8] https://github.com/ALM-Rangers/Roll-Up-Board-Widget-Extension
[9] https://aka.ms/bh-ff-sos
[10] https://aka.ms/vsar-ff-sos
[11] https://aka.ms/bh-ff-sos
[12] https://www.continuousdelivery.com/
[13] https://martinfowler.com/bliki/FeatureToggle.html
[14] https://aka.ms/vsar-ff-sos
[15] https://aka.ms/devops
[16] https://docs.microsoft.com/en-us/vsts/articles/phase-features-with-feature-flags
[17] https://www.visualstudio.com/en-us/articles/phase-rollout-with-rings

GET INVOLVED

If you find these articles useful, get involved! Your feedback helps improve the status quo for all things DevOps.

Contribute to the Opensource.com DevOps resource collection, and join the team of DevOps practitioners and enthusiasts who want to share the open source stories happening in the world of IT.

The Open Source DevOps team is looking for writers, curators, and others who can help us explore the intersection of open source and DevOps. We're especially interested in stories on the following topics:

- DevOps practical how to's
- DevOps and open source
- DevOps and talent
- DevOps and culture
- DevSecOps/rugged software

Learn more about the Opensource.com DevOps team: https://opensource.com/devops-team

ADDITIONAL RESOURCES

The open source guide to DevOps monitoring tools

This free download for sysadmin observability tools includes analysis of open source monitoring, log aggregation, alerting/visualizations, and distributed tracing tools.

Download it now: The open source guide to DevOps monitoring tools

The ultimate DevOps hiring guide

This free download provides advice, tactics, and information about the state of DevOps hiring for both job seekers and hiring managers.

Download it now: The ultimate DevOps hiring guide

The Open Organization Guide to IT Culture Change

In The Open Organization Guide to IT Culture Change, more than 25 contributors from open communities, companies, and projects offer hard-won lessons and practical advice on how to create an open IT department that can deliver better, faster results and unparalleled business value.

Download it now: The Open Organization Guide to IT Culture Change

WRITE FOR US

Would you like to write for Opensource.com? Our editorial calendar includes upcoming themes, community columns, and topic suggestions: https://opensource.com/calendar

Learn more about writing for Opensource.com at: https://opensource.com/writers

We're always looking for open source-related articles on the following topics:

Big data: Open source big data tools, stories, communities, and news.

Command-line tips: Tricks and tips for the Linux command-line.

Containers and Kubernetes: Getting started with containers, best practices, security, news, projects, and case studies.

Education: Open source projects, tools, solutions, and resources for educators, students, and the classroom.

Geek culture: Open source-related geek culture stories.

Hardware: Open source hardware projects, maker culture, new products, howtos, and tutorials.

Machine learning and AI: Open source tools, programs, projects and howtos for machine learning and artificial intelligence.

Programming: Share your favorite scripts, tips for getting started, tricks for developers, tutorials, and tell us about your favorite programming languages and communities.

Security: Tips and tricks for securing your systems, best practices, checklists, tutorials and tools, case studies, and security-related project updates.