FACILITATING THE SPREAD OF KNOWLEDGE AND INNOVATION IN PROFESSIONAL SOFTWARE DEVELOPMENT

SCALING DEVOPS

eMag Issue 51 - May 2017

ARTICLE
Respect Your Organisational Monoliths

Q&A
Starting and Scaling DevOps in the Enterprise

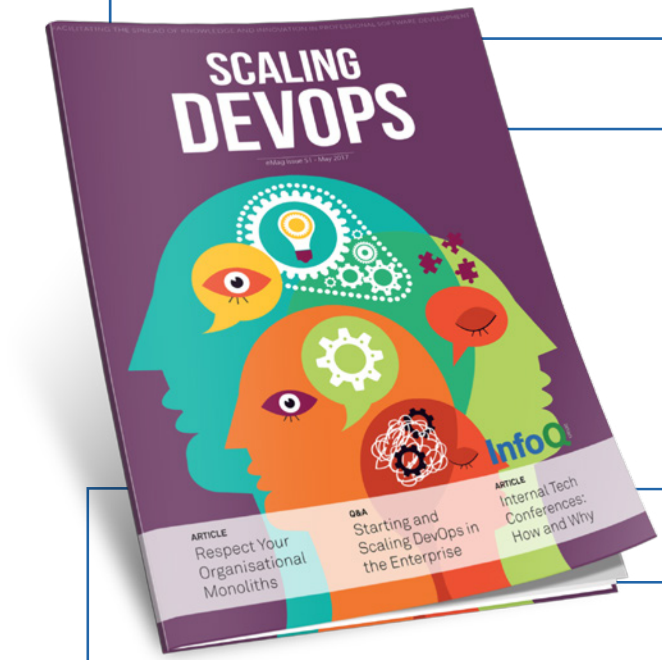
ARTICLE
DevOps Lessons Learned at Microsoft Engineering

Respect Your Organisational Monoliths

There is a lot of information about DevOps, the technology, the culture, the behaviour. There is not a lot of information about tackling DevOps in large enterprises and there is certainly very little about tackling DevOps in large financial organisations. This article presents lessons learnt rolling out DevOps in a large insurance organisation.

DevOps Lessons Learned at Microsoft Engineering

Thiago Almeida from Microsoft shares how adopting DevOps practices resulted in better engineering and happier teams, and the lessons learned in that journey.


SCALING
DEVOPS
InfoQ
ARTICLE
Respect Your Organisational Monoliths
Q&A
Starting and Scaling DevOps in the Enterprise
ARTICLE
Internal Tech Conferences: How and Why

Q&A on Starting and Scaling DevOps in the Enterprise

The book Starting and Scaling DevOps in the Enterprise by Gary Gruver provides a DevOps based approach for continuously improving development and delivery processes in large organizations. It contains suggestions that can be used to optimize the deployment pipeline, release code frequently, and deliver to customers.

DevOps and Product Teams: Win or Fail?

Peter Neumark found a new world when he moved from a DevOps infrastructure team to a Lean product team. How to experiment frequently while keeping operational performance? Platform teams to the rescue!

Communities of Practice: The Missing Piece of Your Agile Organisation

Communities of practice bring together people who share areas of interest or concerns. They have specific applications in agile organisations: scaling agile development and allowing individuals to connect with others who share similar concerns. Communities of practice bring people together to regain the benefits of regular contact while keeping the value of multidisciplinary agile teams.

Internal Tech Conferences: How and Why

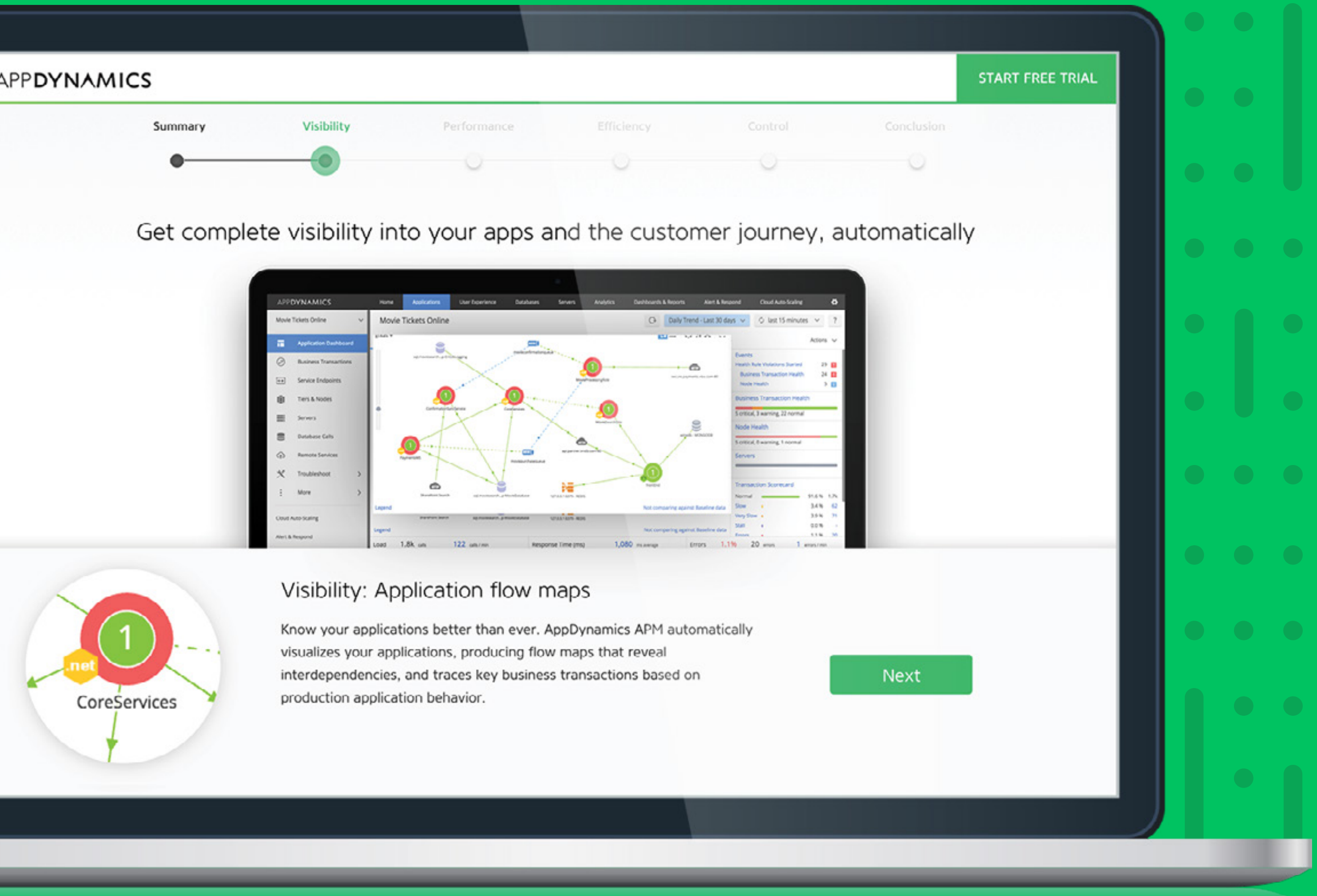
Software engineering today is every bit as much about the people as it is about technology - empowered teams don't appear overnight. We need to oil the wheels of collaboration so they roll smoothly. Here, Matthew Skelton and Victoria Morgan-Smith discuss how to use internal conferences to boost your organisation's social capital, the currency by which relationships flourish and businesses thrive.

APPDYNAMICS

Get complete visibility into your apps and the customer journey, automatically.

Optimizing customer experience with end-to-end visibility and management of even the most complex and distributed applications with auto-generated flow maps, dynamic baselining, and code-level diagnostics.

Take a Tour at **AppDynamics.com**



Get complete visibility into your apps and the customer journey, automatically

Visibility: Application flow maps

Know your applications better than ever. AppDynamics APM automatically visualizes your applications, producing flow maps that reveal interdependencies, and traces key business transactions based on production application behavior.

MANUEL PAIS

Team-first technologist at Skelton Thatcher Consulting. DevOps advocate with a diverse background as developer, build manager and QA lead. Manuel enjoys helping organizations adopt test automation, continuous delivery and cloud, from both technical and human perspectives. He has worked on a range of technologies (Java, .Net, web, mobile) and industries (banking, telecom, legal, defense and aviation). Co-author of the books "Team Topologies" and "Team Guide to Software Releasability". Also InfoQ DevOps lead editor, co-organizer of DevOps Lisbon meetup. Tweets @manupaisable, blogs at skeltonthatcher.com

A LETTER FROM THE EDITOR

As the DevOps movement grew and permeated enterprises around the world, looking for faster and more reliable delivery, scaling became the "holy grail" of DevOps (as with Agile).

"We've done a pilot, we've seen a couple of teams grow autonomous and high-performant, we've seen the benefits of responding quickly to business and customer needs. Now we just need to scale this to all our teams."

Alas, DevOps is not a framework or a method that we can copy and paste. It is a movement focused on changing culture because it makes business sense (as widely reported in the yearly State of DevOps Report). But culture is context. And the larger the organization, the more co-existing cultures there are, each with its own challenges.

This eMag collects articles that explore how to go about scaling DevOps in large organizations – effectively identifying cultural challenges that were blocking faster and safer delivery – and the lessons learned along the way. We include a couple of practices that can help disseminate those lessons.

Margo Cronin, head of technology architecture at Zurich Insurance, reminds us that an organization can have not only technical, but also cultural monoliths that need to be understood to effectively introduce lasting changes, and not just sporadic improvements. Cronin also touches on the importance of bringing suppliers aboard DevOps initiatives, at the risk of reverting progress when incentives for suppliers contradict the cultural changes we want to see in the organization.

Thiago Almeida from Microsoft takes us on the multinational's journey of DevOps adoption. Understanding the nature of the product/service being provided and aligning teams and roles to support delivery and constant feedback from customers were some of the keys to success. Along the way, they learned that promoting accountability ("you build it, you run it") and practices such as canary releases and feature flags helped increased employee and customer satisfaction.

Read the interview with Gary Gruver, of Hewlett Packard and Macy's fame, to get practical gems from an executive who has been through DevOps journeys at multiple enterprises. Gruver reminds us that releasing software in large organizations is hard because of coupled system architectures (and team organization reflects/re-enforces them, as explained by Conway's Law). And that the end goal is not DevOps itself, but identifying and fixing sources of waste and inefficiencies.

Peter Neumark tells us an engaging story about how Prezi evolved their thinking on DevOps and team organization as the business grew from a startup to a mid-sized company. Growth caused a shift in priorities, from being feature-driven to worrying much more about availability. Neumark tell us first-hand how he experienced DevOps in an infrastructure team vs a product team, and how platform teams came to the rescue.

Finally, we include two articles that take a close look at healthy practices for disseminating knowledge and culture and fostering collaboration outside people's comfort zone. Matthew Skelton, co-founder at Skelton Thatcher Consulting, and Victoria Morgan-Smith, agile delivery coach at the Financial Times, go through the goals and logistics of internal tech conferences. Emily Webber, an independent agile coach, explains how communities of practice help connect people in different parts of the organization trying to scale Agile and DevOps.

We hope you enjoy this eMag and hopefully can take some lessons back to your organization!

Respect Your Organisational Monoliths

Margo Cronin is an open group certified master architect and certified program manager. She worked for 10 years for IONA Technologies as a middleware consultant before moving to the FinServ sector in Switzerland. There, over the last 10 years, she has worked for Credit Suisse and Zurich Insurance, in both delivery and enterprise architecture roles. Cronin started working with DevOps to enable scrum and agile to be successful in large distributed organisations. She is co-founder of an enterprise architecture platform in Switzerland called EntArchs.

When starting the DevOps journey in our organisation, we found that there was a lot of information on technologies and even a lot about the culture, but there was little information on adopting DevOps in large distributed organisations and the associated enterprise specific challenges.

In this article, I talk about my experience approaching DevOps for the first time in 2014 onsite at a large insurance company.

Every story has a hero. In this story, it's the monolith. Let's start with a couple of definitions of monolith:

Definition one: A great big rock.

Definition two: A large impersonal political, corporate or social structure regarded as indivisible or slow to change.

Controversially, this definition could be applied directly to some of our large distributed organisations. Our organisations are very political. I love the phrase "slow to change". What strikes me in this sentence is, it does not say "unwilling to change" or "won't change". It made me think, what are the features of our organisations that make them monolithic? What creates that viscosity and resistance that slows down our change so much that it nearly seems we cannot change?

We now design our organisations' model, roles and applications not to be monolithic. We favour modularity - but our organisations have existing mono-

KEY TAKEAWAYS

In a large distributed organisation, you are at risk of creating DevOps in a vacuum unless you respect the characteristics of your organisation (the real monoliths)

The technology problems are the easiest to solve

Bring the supplier on the journey

Engage the correct stakeholders (not just the IT and business managers, but also GIS, PMO, procurement, and legal)

Culture change is about behaviour change. The first behaviour change you can enable is your own

lithic features that need to be considered and understood in order to enable change.

It's All about Behaviour

A video on the transformational changes that happened in Yellowstone park due to the reintroduction of a pack of wolves has been making the rounds of social media.

Wolves went extinct in Yellowstone in the 1920s. In 1995 biologists reintroduced a pack of 14 wild wolves into the park with the goal of regenerating the wolf population. The outcome was miraculous. The park flourished with the return of animal species, flora and even the rivers changed direction.

Previously, when the wolves were extinct, deers and elks over grazed in the park, eroding many areas. Upon the return of the wolves, the deer and elk stopped grazing in certain parts of the park where they could be easily hunted by the wolves. Indeed, a short while into the clip the scientist excitedly explains that the deer and the elk changed their behaviour. Programs arranged by humans - culling the deer and elk - did not have an impact. Indeed, even the number of deers and elks killed by the wolves themselves did not have much of an impact. It was the behaviour change that did it!

Like Yellowstone, our organisations need transformational change. Like Yellowstone, this requires behaviour change.

I believe, with behaviour change you need to lead by example. Before you look at the behaviour of your teams, you need to begin with your own behaviour. Before you can change that, you first need to understand why you behave the way you do.

Characteristics Influencing Behaviour Change

In my large FinServ organisation these are the characteristics that influenced my behaviour:

Outsourcing

We outsourced a lot of our IT, both application and infrastructure. This very much influenced how our teams were constructed. It influenced how I designed and delivered applications.

Global IT Standards

We had global IT standards that dictated what technology we used for a specific technical capability. This influenced how I designed applications and with what technology. It influenced how my solutions were governed and approved.

Business Units

We had 50+ business units, with some IT autonomy to respond to local demand or regulation; this influenced how we behaved at the group and business unit level. It also cultivated a shadow IT culture. These shadow IT teams can perform very well in isolation. As these teams frequently use non-standard technologies, methodologies or suppliers, the success and culture of the team exists in isolation within the business unit and is typically not replicated across the organization. Sharing and collaboration does not happen beyond the team, and frequently the team feels under threat of being "shut down". They achieve success only in a vacuum.

Near-Shore and Off-Shore Teams

Whether using near-shore or off-shore teams, my behaviour was influenced as it affected how I communicated.

Different Dedicated Departments

In large enterprises we often have different departments with different mandates. There are many examples: portfolio management, sourcing, development and operations, as a few examples. These are departments with a dedicated remit which "own" certain activities or functions. They affect how we behave, who we engage, how we engage and when.

These are a few features of a large distributed organisation that I have worked with on a daily basis. There are many more. Not only do they influence how we behave, but they do not lend themselves well to a global DevOps or Continuous integration & delivery strategy.

To enable behaviour change - which will bring about culture change - we need to understand these characteristics of our large distributed organisations. These features and characteristics are our true monoliths.

Monoliths Influencing Culture Change

These are the monoliths I encountered while owning the delivery of a strategic initiative for my large distributed financial organisation:

Us vs. Us

A financial product is not a commodity product like a car or a handbag, nor is it like Twitter or Facebook - a platform you engage on, reach out and connect to people on. Insurance is something you purchase and then you put it in a digital drawer or physical drawer and forget about it, until it needs to be renewed or until you need to use it - because something has happened to your car, your business, your health or your family. Therefore insurance companies are about mitigating risk, about protecting customers. Banks have the same challenge. When you are about protection and protecting customers, many departments crop up in your organization around protection. This might be budget protection, data protection, security protection, system protection, etc. These teams are typically "always right". They are right because they are protecting the customer which is what the organization is all about. Have you ever noticed how difficult it is to negotiate with someone who is always right? (You are thinking about your partner now, right?)

DevOps and agile initiatives (like Scrum) are being more widely adopted in our organisations. However, to successfully enable these methodologies and way of working, it is important to engage the teams and departments in the company that are responsible for "protection". Like your partner, these are the teams where you need to show the love. Without engaging these teams you are working against your own group and run the risk of creating an "us vs. us" environment.

To enable DevOps to be successful in my organisation, I had to engage with many such groups. I will specifically mention the security group. Most (financial) organizations have a dedicated security group. I went to my security group with a DevSecOps story. However, their big concern was around testing. How do you continuously evolve an asset in production while maintaining traditional organizational milestones around testing? To get engagement from group security we had to examine and explain our product testing and test driven development strategy more closely.

Engage your security, legal, asset management and operations release management groups. Explain what DevOps is, but most importantly listen to their concerns. Their remit is important to the success of the organization. Addressing their concerns leads to buy-in. It will enable the wider organisation to adopt DevOps instead of creating DevOps in a vacuum.

Project vs. Product

Over time the platform my team was building started to mature and achieve excellent velocity. I then realized that my organisation was fully structured around projects. We had application teams building a project, go live, and then hand the asset over to a support team. The asset was in a Business As Usual (BAU) state. We ▶

Engage your security, legal, asset management and operations release management groups. Explain what DevOps is but most importantly listen to their concerns.

had project managers, program managers, portfolio managers. All governance pivoted around projects.

As my team adopted DevOps to become more and more autonomous, we moved from building a project to building a product. We were not going to hand it over to go into BAU state, but rather continuously evolve it. A project-focused organisation is not structured to support this. Even simple tasks like securing funding or getting approval at stakeholder meetings became impediments due to this monolith.

Addressing how your organisation does governance and portfolio management is key for DevOps adoption outside the Scrum room in large organisations.

The Supplier

Many organisations use suppliers. Some outsource everything, others only certain functions. I experienced first-hand what happened when an organisation heavily depended on a supplier, yet was not engaged with teams working in a DevOps way. Our cycle time went from two days to three hours thanks to adopting DevOps,mainly by evolving our toolset away from restrictive global standards to a complementary mix of tools and technologies that supported our test driven development strategy and enabled continuous integration and delivery. Our asset went live, and the business was very satisfied. However, once the asset was in production for a couple of sprints, some changes were introduced around our automated release processes. All the automation remained but the supplier added their manual processes around the automation, postponing releases to production. This brought the cycle time back up to two days.

In large organisations, suppliers often have restrictive contracts that dictate how they and the organisation engage. In this case, the supplier was being penalised when there were incidents in production. Of course, the supplier was going to be all over any team that had access to production servers.

We hear a lot about moving from Mean Time Between Failure (MTBF) to Mean Time to Recover (MTTR), but such a transition is not possible if a team, person or supplier is being penalised due to issues in production.

DevOps does not mean that you should ignore supplier management. Should your organisation have suppliers, understand their contracts and how they influence the supplier's behaviour. How can you adapt your organisation's behaviour to enable the supplier to support or allow your DevOps product to evolve in production? For example, contract notes can be raised on vendor contracts removing stringent penalty clauses. This is a conversation to have with your wider IT management team and strategic sourcing. It has to be an organizational decision to make such a change to enable the supplier to come on the DevOps journey with you.

The Traditional Application Monolith

I believe that in the future there will not be much of an enterprise's portfolio that needs to remain monolithic. Bi-modal and two speed IT should not have to apply to much of an enterprise's application portfolio. That said, there could be monolithic applications in the organisation that will not wish to adopt DevOps or agile initiatives.

Some applications will remain monolithic in nature as they are meeting their objectives and are successful (or perceived to be!). If your product is successful it is likely that your backlog could expand and introduce stories that will need to engage with such an application monolith. This can cause problems, especially in the area of testing where test data from the monolith may not be updated as rapidly as the DevOps product requires. Solutions vary based on the application in question (automation, moving payload to a QA cloud, strangler architectural pattern, etc). However, as the owner of a DevOps-driven product, you need to understand your enterprise architectural landscape in which your product lives, and the potential two-speed-IT style monoliths that you may have to interface with.

Conclusion

This is the famous scene from Space Odyssey, the monkeys scrambling around the monolith:

Avoid creating DevOps in a Vacuum in your organisation by understanding what influences your (and your peers') behaviour. These are your true monoliths. Once understood, you can start looking into changing them. By changing them, you enable pro-active behaviour change, both your own and your peers'. This behaviour change will bring about transformational change in your organisation. ■

Avoid creating DevOps in a Vacuum
Identify the monoliths of your organization to enable behaviour change
Transformational change requires behaviour change

DevOps Lessons Learned at Microsoft Engineering

Thiago Almeida grew up in Brazil and lived in New Zealand for many years before joining the Microsoft team in Redmond, Wash. He's part of the team that drives adoption of new technologies, focusing on cloud computing, open source, and DevOps practices. He tweets at @nzthiago and blogs at http://talmeida.net.

Microsoft engineering groups have adopted DevOps practices in the past few years, learning and benefiting from this change.

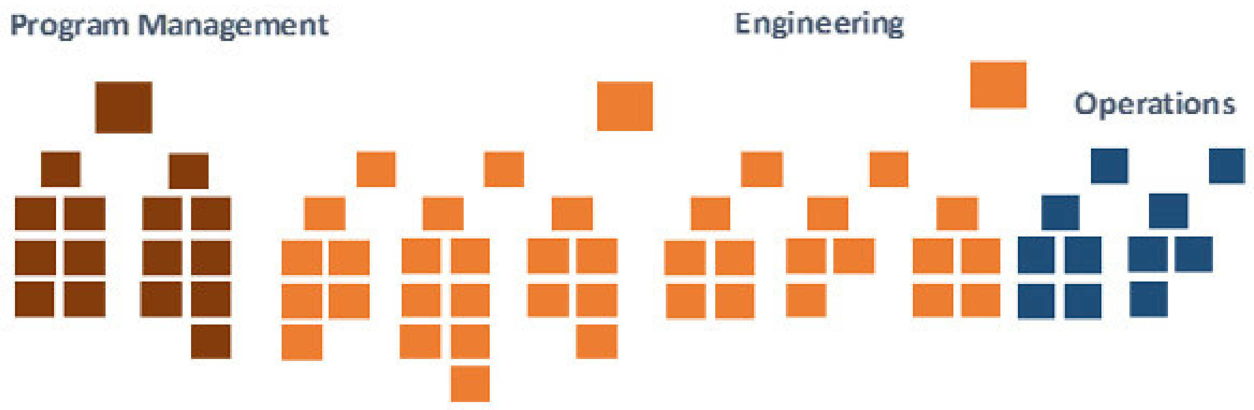
As we have observed in the software industry and, frankly, drawn from the pain we have experienced, DevOps practices and habits have been essential for improving at delivering services and other products across the board. We have found that the organizational changes and cultural shifts required to embrace these practices have been just as significant. This resulted in a change in the structure of our teams, their responsibilities, and the culture across development, operations, and the business.

For example, Microsoft used to have a separation of engineering practices and tools between Windows, Office, and the rest. We now have over 43 thousand internal daily users of Visual Studio Team Services across multiple engineering teams, and that number is rapidly climbing. Microsoft intends it to be the default tooling for teams to use to support their practices.

This article summarizes stories and presentations from Microsoft engineering teams as well as internal conversations, especially from the Cloud and Enterprise and the Bing teams.

Team organization

In the past, we had three distinct roles in engineering in what we call "feature teams": program managers, developers, and testers, with a complete separation of development and testing from an organizational and team perspective. Also, the operations team was in an organization separate from the rest of the engineering team.



We wanted to reduce delays in handoffs between developers and testers and focus on quality for all software created, so we combined the traditional developer and tester roles into one discipline called "software engineering". Software engineers have become responsible for every aspect of bringing their features to life and performing well in production. This does not mean that we abandoned testing — quite the contrary. This means testing and quality is everyone's responsibility.

To deliver the best set of services to our customers, we need engineering and operations to work closely together throughout the entire lifecycle of development, from design to deployment in production. One of our first steps was to bring the operations teams into the same organization. Operations staff underwent a significant change from a traditional mentality and accountability so we now call our operations team members "service engineers".

We need to be one team if we are going to deliver the best services. The close coupling between the individuals who are writing the code and the individuals who are operating the service allows us to get capabilities into production much more rapidly.

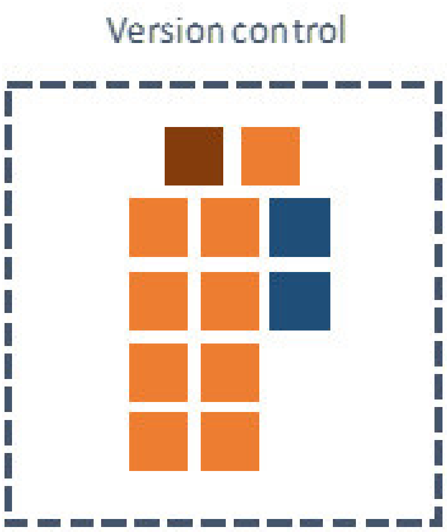
The new organizational chart looks like the image shown above, with combined development and test teams and with operations in the same organization as the rest of engineering.

From this, we form feature teams, which are cross-discipline teams that focus on a single solution, feature, or product. In the Developer Division, the group that builds tools for developers and development teams, feature teams are made up of 10 to 12 people and are self-managed, have an autonomous backlog, and remain mostly intact for 12 to 18 months. There are currently 4,307 people in the Developer Division; 436 of those are in the Team Services team and Team Services has 35 feature teams. To the right is the organizational view of the "Version Control" feature team, with a program manager, an engineering lead, software engineers, and service engineers. Service engineers support more than one feature team, but never more than one product.

Another interesting change is in the physical location of the teams. Feature teams across engineering are now located in their own team rooms dedicated to working together, sometimes called "neighborhoods". These are open-plan areas that teams can customize however they like. All work conversations in the team room should be relevant to everyone in the room. Team rooms also have meeting and focus areas for extended conversations and phone calls. It's a great combination of open plan and offices.

Team accountabilities

The main goal of all this is the significant change in accountabilities. These accountability changes were introduced across software and service engineering to achieve the best results ▶

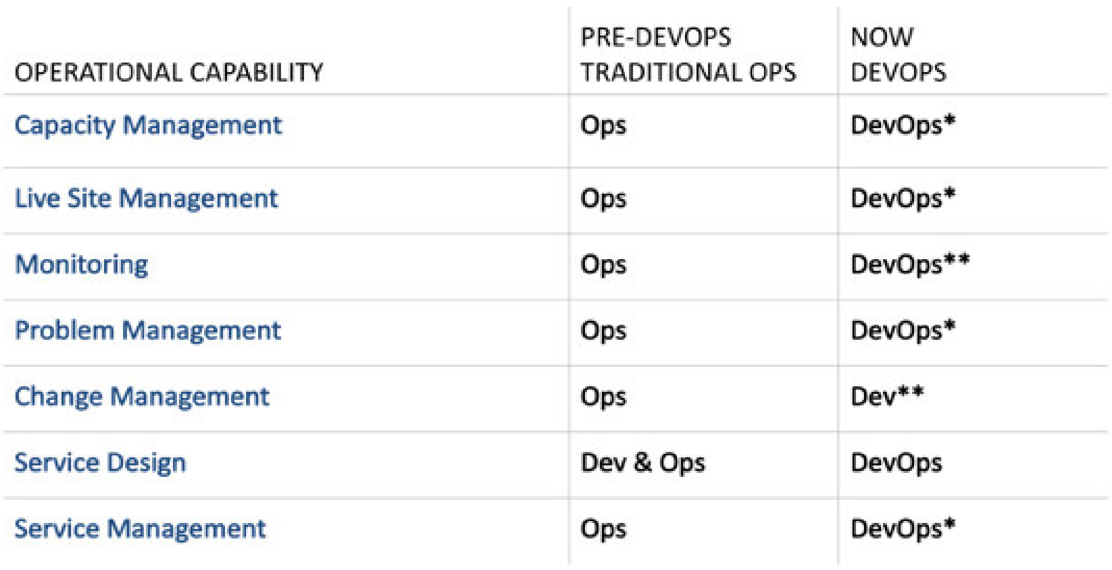


for our customers. We use metrics to measure progress and encourage a positive cultural shift. For example, tests coverage and customer SLAs are shared responsibilities.

Software engineers have become accountable for not only building and testing but ultimately for the health of production. This accountability shift has two aspects. First, we want the feature teams obsessed with understanding our customers to get a unique insight into the problems they face and how they can be raving fans of the experiences those teams are building. Second, we need the feature teams and individual engineers to own what they are delivering into production. The feature teams have the power, control, and authority over all of the parts of the software process.

Service engineers have to know the application architecture to be more efficient troubleshooters, to suggest architectural changes to the infrastructure, to be able to develop and test things like infrastructure as code and automation scripts, and to make valuable contributions that impact the service design or management. Automation is a key theme continually being improved upon for all aspects of the software lifecycle and has enabled Microsoft to scale and deliver value faster to customers. For example, we automated previously manual efforts like testing, environment creation, and release management. The service engineers bring invaluable skills to the team, especially since there are more moving parts and more opportunities for failure.

This table shows the operational capabilities and the shift of responsibilities.



OPERATIONAL CAPABILITY	PRE-DEVOPS TRADITIONAL OPS	NOW DEVOPS
Capacity Management	Ops	DevOps*
Live Site Management	Ops	DevOps*
Monitoring	Ops	DevOps**
Problem Management	Ops	DevOps*
Change Management	Ops	Dev**
Service Design	Dev & Ops	DevOps
Service Management	Ops	DevOps*

* This capability has been partially automated.
** This capability has been majority or fully automated.

Note the shift in change management from Ops to Dev. That is because new services and hot fixes are automatically deployed into production with a peer-based review system. We introduced automated tests and deployments and feature flags, reducing the risks.

These changes have been well received. I have worked with a lot of high-potential startups in the past as part of the Microsoft BizSpark program and in talking to the feature teams inside Microsoft engineering, I get the same sense of drive and excitement as I did when I was involved with those startups.

These changes have brought the following benefits:

- increased sense of accomplishment,
- feature teams obsessed with understanding our customers,
- decoupled services with clear contracts, and
- focus on automation and telemetry.

For further information about these changes, see Our DevOps Journey and the "DevOps at Scale" session from Build 2016.

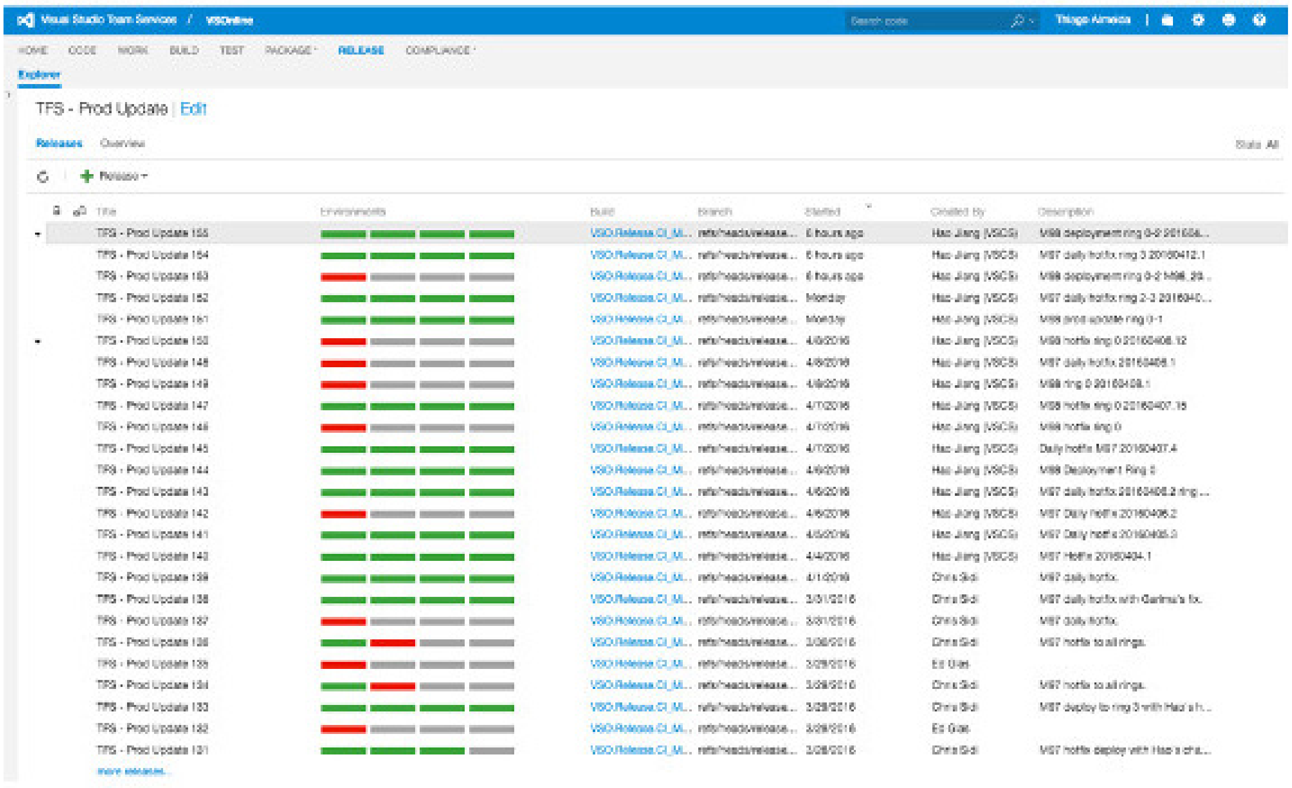
Flighting deployments

Being responsible for a hosted service like Visual Studio Team Services (VSTS) or a mobile app like OneDrive for iOS, teams at Microsoft have realized the benefits of canary releases, in which deployments occur in batches. The VSTS team calls canary releases "deployment rings". Teams automate their builds and tests and push these onto real but internal or early feedback accounts, or to developers' physical devices (a.k.a. "dogfooding"). This allows for controlled exposure, early feedback, and experimentation.

For example, VSTS currently releases updates to the services with four deployment rings into 12 scale units at different Microsoft Azure regions. The deployments occur in batches, with the team's own VSTS account being in the first scale unit deployed by the first deployment ring, before the other three deployment rings push the changes to other 11 customer scale units across the world. Lead engineers in the feature teams approve the release into the first ring, and the rest is automated. Because the teams themselves get the updates first, they are testing within their own team during work hours, with the right engineers there to make fixes. If anything is going to break, they want it to break on themselves first.

Most of the code being deployed is still behind feature flags for another level of controlled release.

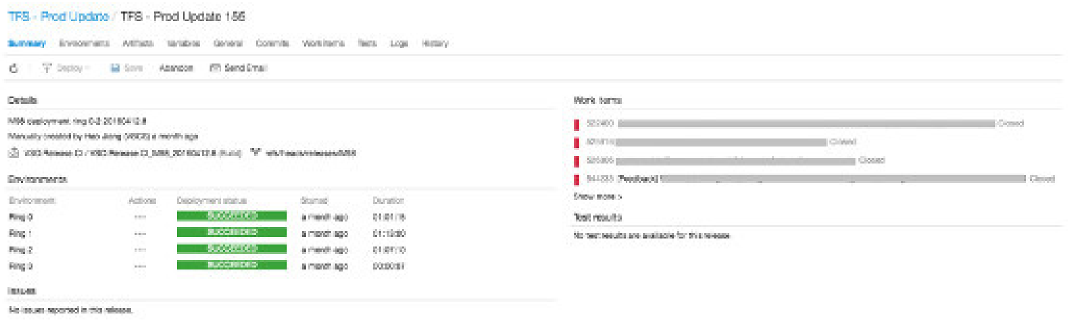
As the saying goes, the mark of a good compiler is one that compiles itself. So the VSTS team uses VSTS to deploy updates to the service. Each row in the next image is a release of daily hot fixes. They use the environments concept of VSTS to deploy to the different deployment rings. Environments in VSTS are logical groupings of a series of tasks that might need approvals before and after, which can be executed in parallel or in sequence, so it works well with the deployment rings concept of the VSTS team. ▶


Visual Studio Team Services / VSOnline
Search code
Thiago Almeida
HOME CODE WORK BUILD TEST PACKAGE RELEASE COMPLIANCE
Explorer
Releases Overview
Release
State All
more releases...

TFS - Prod Update | Edit

Title	Environments	Build	Branch	Started	Created by	Description
TFS - Prod Update 155	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	6 hours ago	Hao Jiang (VSCS)	M98 deployment ring 0-2 201604...
TFS - Prod Update 154	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	6 hours ago	Hao Jiang (VSCS)	M97 daily hotfix ring 3 20160412.1
TFS - Prod Update 153	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	6 hours ago	Hao Jiang (VSCS)	M98 deployment ring 0-2 M98_20...
TFS - Prod Update 152	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	Monday	Hao Jiang (VSCS)	M97 daily hotfix ring 2-3 201604...
TFS - Prod Update 151	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	Monday	Hao Jiang (VSCS)	M98 prod update ring 0-1
TFS - Prod Update 150	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	4/8/2016	Hao Jiang (VSCS)	M98 hotfix ring 0 20160408.12
TFS - Prod Update 148	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	4/8/2016	Hao Jiang (VSCS)	M97 daily hotfix 20160408.1
TFS - Prod Update 149	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	4/8/2016	Hao Jiang (VSCS)	M98 ring 0 20160408.1
TFS - Prod Update 147	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	4/7/2016	Hao Jiang (VSCS)	M98 hotfix ring 0 20160407.15
TFS - Prod Update 146	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	4/7/2016	Hao Jiang (VSCS)	M98 hotfix ring 0
TFS - Prod Update 145	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	4/7/2016	Hao Jiang (VSCS)	Daily hotfix M97 20160407.4
TFS - Prod Update 144	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	4/6/2016	Hao Jiang (VSCS)	M98 Deployment Ring 0
TFS - Prod Update 143	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	4/6/2016	Hao Jiang (VSCS)	M97 daily hotfix 20160406.2 ring ...
TFS - Prod Update 142	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	4/6/2016	Hao Jiang (VSCS)	M97 Daily hotfix 20160406.2
TFS - Prod Update 141	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	4/5/2016	Hao Jiang (VSCS)	M97 Daily hotfix 20160405.3
TFS - Prod Update 140	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	4/4/2016	Hao Jiang (VSCS)	M97 Hotfix 20160404.1
TFS - Prod Update 139	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	4/1/2016	Chris Sidi	M97 daily hotfix.
TFS - Prod Update 138	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	3/31/2016	Chris Sidi	M97 daily hotfix with Garima's fix.
TFS - Prod Update 137	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	3/31/2016	Chris Sidi	M97 daily hotfix.
TFS - Prod Update 136	🟩 🟥 ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	3/30/2016	Chris Sidi	M97 hotfix to all rings.
TFS - Prod Update 135	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	3/29/2016	Ed Glas	
TFS - Prod Update 134	🟩 🟥 ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	3/29/2016	Chris Sidi	M97 hotfix to all rings.
TFS - Prod Update 133	🟩 🟩 🟩 🟩	VSO.Release.CI_M...	refs/heads/release...	3/29/2016	Chris Sidi	M97 deploy to ring 3 with Hao's h...
TFS - Prod Update 132	🟥 ⬜ ⬜ ⬜	VSO.Release.CI_M...	refs/heads/release...	3/29/2016	Ed Glas	
TFS - Prod Update 131	🟩 🟩 🟩 ⬜	VSO.Release.CI_M...	refs/heads/release...	3/28/2016	Chris Sidi	M97 hotfix deploy with Hao's cha...

Release 155, for example, was successful across the four deployment rings, deploying several associated work items to all 12 scale units.



TFS - Prod Update / TFS - Prod Update 155

Summary Environments Artifacts Variables General Commits Work items Tests Logs History

Deploy Save Abandon Send Email

Details

M98 deployment ring 0-2 20160412.8

Manually created by Hao Jiang (VSCG) a month ago

VSO Release CI / VSO Release CI_M98_20160412.8 (Build) refs/heads/releases/M98

Environments

Environment	Actions	Deployment status	Started	Duration
Ring 0	···	SUCCEEDED	a month ago	01:01:16
Ring 1	···	SUCCEEDED	a month ago	01:10:30
Ring 2	···	SUCCEEDED	a month ago	01:07:10
Ring 3	···	SUCCEEDED	a month ago	00:00:51

Issues

No issues reported in this release.

Work items

553400	[redacted]	Closed
509716	[redacted]	Closed
525306	[redacted]	Closed
544233	[Feedback] [redacted]	Closed

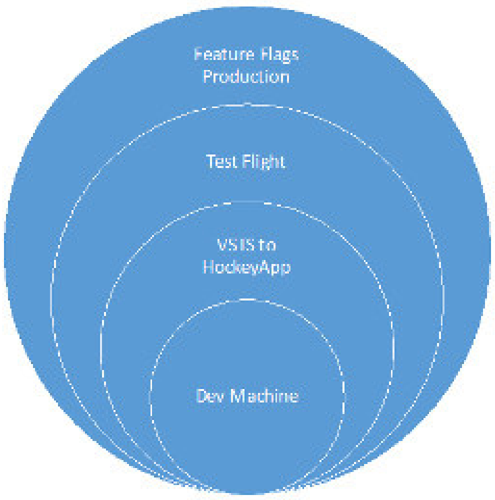
Show more >

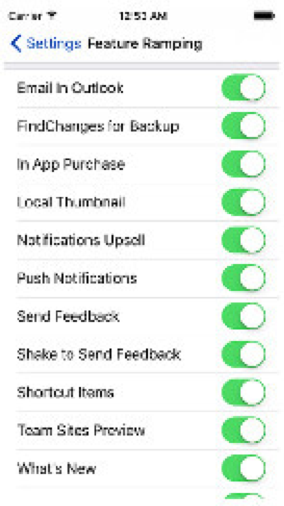
Test results

No test results are available for this release.

Another example is the OneDrive mobile team. They use VSTS to automatically build and test their iOS app and then VSTS automatically pushes those builds to their physical devices via a product called HockeyApp. HockeyApp not only helps with the deployment to devices, it also instruments all of the crash data and analytics so that anyone on the development team can resolve problems. The team uses HockeyApp to release the updates to the team members themselves and to internal adopters.

After that, the OneDrive team scales that feedback to users beyond developers and internal/corporate via HockeyApp, then to Apple's TestFlight for more beta testers, and finally in production with feature flags. Once a feature has enough positive feedback and testing, they roll it out to all users.





Carrier 12:53 AM

< Settings Feature Ramping

Email In Outlook	On
FindChanges for Backup	On
In App Purchase	On
Local Thumbnail	On
Notifications Upsell	On
Push Notifications	On
Send Feedback	On
Shake to Send Feedback	On
Shortcut Items	On
Team Sites Preview	On
What's New	On

This brings the following benefits:

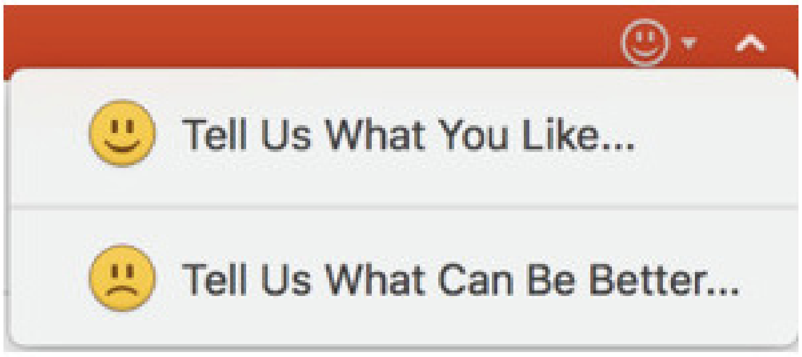
- It brings in early feedback.
- It promotes experimentation.
- It allows for controlled exposure.
- Testing first within your own team Improves code quality.
- It reduces the time of problem resolution.

Learning from customers: Direct feedback

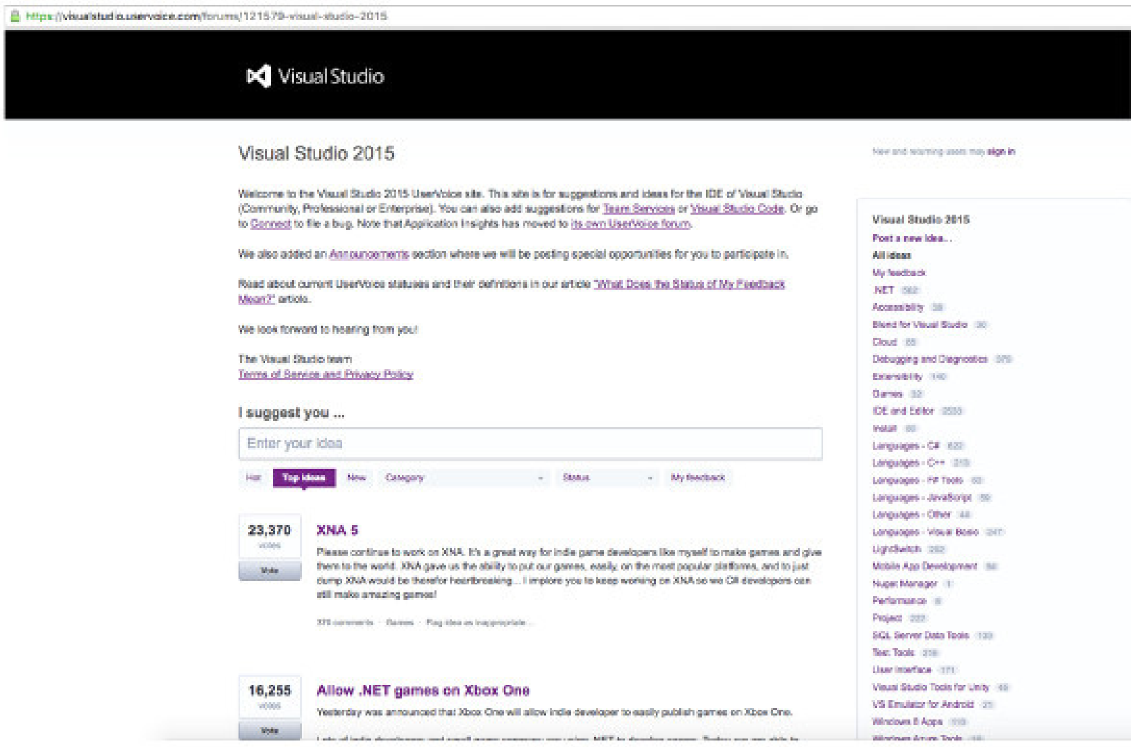
There is a lot of focus on instrumentation and telemetry to support the customer feedback loop, improving continuous delivery, and foster hypothesis-driven engineering (see the next section on idea generation).

But we have found that giving customers an easy and direct feedback form, like the "tell us what you like" and "tell us what you don't like" icons in the Microsoft Office apps, helps the feature teams develop a community, increase product quality, and bring customers closer to engineering teams. It shouldn't be a surprise that only relying upon people to tweet their problems about an app or service isn't the best mechanism for gathering direct customer feedback. So several Microsoft teams included a "send feedback" mechanism inside their applications and services, and we triage all feedback and send it to feature teams for their backlogs or support. For example, Microsoft Office, the Bing homepage, and the Azure Portal all have discreet feedback buttons.

This image is the feedback icon on Microsoft Office apps, for example.



A lot of teams also have implemented UserVoice or similar feedback venues to gather and group feedback. These become backlog items for the teams. UserVoice is used for suggestions and ideas, and not for bugs that are raised via support. Here is an image of the Visual Studio UserVoice page:



https://visualstudio.uservoice.com/forums/121579-visual-studio-2015

Visual Studio

Visual Studio 2015

Welcome to the Visual Studio 2015 UserVoice site. This site is for suggestions and ideas for the IDE of Visual Studio (Community, Professional or Enterprise). You can also add suggestions for Team Services or Visual Studio Code. Or go to Connect to file a bug. Note that Application Insights has moved to its own UserVoice forum.

We also added an Announcements section where we will be posting special opportunities for you to participate in.

Read about current UserVoice statuses and their definitions in our article "What Does the Status of My Feedback Mean?" article.

We look forward to hearing from you!

The Visual Studio team
Terms of Service and Privacy Policy

I suggest you ...

Enter your idea

Hot | Top ideas | New | Category | Status | My feedback

23,370 votes — Vote

XNA 5

Please continue to work on XNA. It's a great way for indie game developers like myself to make games and give them to the world. XNA gave us the ability to put our games, easily, on the most popular platforms, and to just dump XNA would be therefor heartbreaking... I implore you to keep working on XNA so we C# developers can still make amazing games!

375 comments · Games · Flag idea as inappropriate...

16,255 votes — Vote

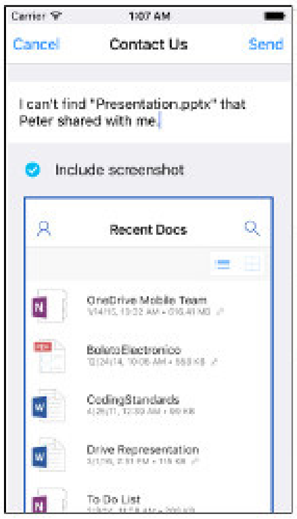
Allow .NET games on Xbox One

Yesterday was announced that Xbox One will allow indie developer to easily publish games on Xbox One.

New and returning users may sign in

Visual Studio 2015
Post a new idea...
All ideas
My feedback
.NET 582
Accessibility 38
Blend for Visual Studio 30
Cloud 85
Debugging and Diagnostics 375
Extensibility 140
Games 32
IDE and Editor 2533
Install 80
Languages - C# 622
Languages - C++ 213
Languages - F# Tools 62
Languages - JavaScript 59
Languages - Other 44
Languages - Visual Basic 247
LightSwitch 282
Mobile App Development 84
NuGet Manager 1
Performance 8
Project 222
SQL Server Data Tools 133
Test Tools 218
User Interface 171
Visual Studio Tools for Unity 45
VS Emulator for Android 21
Windows 8 Apps 113

Here's an example of a mobile app: the OneDrive team included a "contact us" feedback mechanism inside all their applications for every platform. To efficiently handle the volume of feedback, they previously used a product called Parature. The console gathers customer data and centralizes all feedback for the team to review.



Carrier 1:07 AM

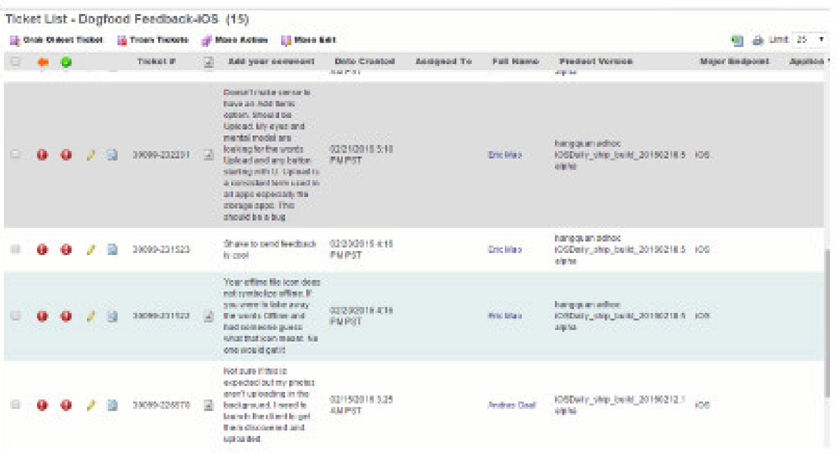
Cancel | Contact Us | Send

I can't find "Presentation.pptx" that Peter shared with me.

Include screenshot

Recent Docs

OneDrive Mobile Team
BoletoElectronico
CodingStandards
Drive Representation
To Do List



Ticket List - Dogfood Feedback-iOS (15)

Ticket #	Add your comment	Date Created	Assigned To	Full Name	Product Version	Major Endpoint
39099-232231	Doesn't make sense to have an Add items option. Should be Upload. My eyes and mental model are looking for the words Upload and any button starting with U. Upload is a consistent term used in all apps especially the storage apps. This should be a bug	02/21/2015 5:18 PM PST		Eric Max	hangquan adhoc iOSDaily_ship_build_20150218.5 alpha	iOS
39099-231523	Shake to send feedback is cool	02/20/2015 4:18 PM PST		Eric Max	hangquan adhoc iOSDaily_ship_build_20150218.5 alpha	iOS
39099-231522	Your offline file icon does not symbolize offline. If you were to take away the words Offline and had someone guess what that icon meant, no one would get it	02/20/2015 4:16 PM PST		Eric Max	hangquan adhoc iOSDaily_ship_build_20150218.5 alpha	iOS
39099-226070	Not sure if this is expected but my photos aren't uploading in the background. I need to launch the client to get them discovered and uploaded	02/15/2015 3:25 AM PST		Andrea Graf	iOSDaily_ship_build_20150212.1 alpha	iOS

Parature Console

This direct customer feedback approach brought the following benefits:

- increased quality,
- help in developing a community,
- feature teams understand customers better with direct feedback, and
- increased customer satisfaction

Idea velocity

Idea velocity has been a big focus at Microsoft. It is the speed of experimentation, all the way from back-of-the-napkin ideation to full-blown analysis of the feature's impact on user engagement.

Individual employees are encouraged to create ideas, implement them, and test them in production as often as they like. This is done mostly in teams with a mature continuous-delivery pipeline with a strong automated testing infrastructure that have instrumented telemetry at all levels. The most important guiding principle is that the idea for a feature can come from anywhere. While still receiving guidance from above, we built systems to allow ideas to come from anywhere.

Once Bing's continuous-delivery and testing-at-scale infrastructure was in place, they focused on allowing anyone in the team to experiment and test ideas. This empowers individuals, allows product decisions to be made from real customer data, and rewards creativity. And to say that test automation is central to this success would be a vast understatement.

We organize our engineering ecosystem into an efficient idea funnel, which lets us easily iterate on ideas with end users at the top so we can churn through as many ideas as possible. This affects everyone in the organization, and connects the engineer to the user in a visceral way. One part of this is done through events and incentives such as growth hacks, incubators, and hack days.

Growth hacks are tracked at VP level, and so allow you to influence the direction of the organization by, for example, improving the efficiency of our engineering systems or improving the user engagement on a major segment.

BingCubator is a forum where entrepreneurs can pitch an idea that is large enough for funding. Their ideas run through an incubation process under v-team (virtual-team) management before the entrepreneurs can present them to upper management for funding.

Hack days closely model our engineers' typical daily interactions, though are designed to allow them to shelve their normal deliverables temporarily and pursue something outside their areas of expertise.

Bing provides tooling to engineers that allows them to get feedback about their ideas from external users within a few minutes. Engineers submit their mock-up visual concepts and questions and select their target audience. Their experiments are then sent to hundreds of people for comment. Microsoft has its own crowd-sourcing platform with a pool consisting of several thousand external people on panel so feedback from the pool usually comes back within two hours. It allows engineers to experiment without any need to write code.

Do you want to know if your idea for a hack day is a good one? You can do a quick prototype, compose a brief survey, push it to the crowd, and evaluate the feasibility of your idea. This nearly real-time feedback lets our developers learn whether their ideas really can be scaled to production for our end users.

Once the idea has been proven, it is released through continuous delivery. Implementing idea velocity brings the following benefits:

- It decouples engineering and marketing.
- It supports early feedback and experimentation.
- It allows ideas to be generated at every level.
- It supports a shared metrics culture.
- It reduces HIPPO (highest-paid person override) decisions.
- It rewards creativity.
- It improves continuous delivery, testing at scale, and telemetry.

Conclusion

These changes are helping to bring improvements in code velocity, quality, and productivity, which are reflected in the quality of our products and customer satisfaction.

But another important outcome is that our engineers love it. We removed the parts of their job that were annoying and encouraged the best engineering practices, which resulted in better engineering and happier teams. We have seen an amazing improvement in work/life balance scores. Making these teams more efficient means they feel like less of their effort is wasted and it improves all facets of their work. ■

Q&A on Starting and Scaling DevOps in the Enterprise

by Ben Linders

Gary Gruver is an experienced executive with a proven track record of transforming software development and delivery processes in large organizations, first as the R&D director of the LaserJet FW group that completely transformed how they developed embedded firmware and then as VP of QA, Release, and Operations at Macys.com, leading the journey toward continuous delivery. He now consults with large organizations and runs workshops to help them transform their software-development and delivery processes.

Gary Gruver's Starting and Scaling DevOps in the Enterprise provides a DevOps-based approach for continuously improving development and delivery processes in large organizations. The book contains suggestions to use to optimize the deployment pipeline, release code frequently, and deliver to customers.

InfoQ interviewed Gary Gruver about what makes DevOps fundamentally different from other agile approaches.

InfoQ: Why did you write *Starting and Scaling DevOps in the Enterprise*?

Gary Gruver: As I started working with more and more different large organizations that wanted to transform their software-delivery processes, I discovered one of the biggest challenges was getting the continuous-improvement journey started and aligning everyone on implementing the changes. For this to work, I feel pretty strongly that you should start the continuous-improvement process with the changes that will have the most significant impact on the organization so you build positive momentum.

I found I was spending most of my time analyzing the processes in different organizations and helping them identify those areas. I saw a lot of common problems but I also found each organization had some unique challenges that resulted in different priorities for improvement. Over time, I started

KEY TAKEAWAYS

Coordinating work across different parts of your organization with DevOps depends on team size due to architectural coupling.

Transformations that start by addressing the biggest inefficiencies in development processes are more successful.

People need a common understanding of how their current approaches are causing inefficiencies for the overall software development and delivery system to change their way of working.

Mapping your current deployment pipeline, including metrics, with teams reveals the biggest inefficiencies in their software development processes.

Increasing the frequency of deployment while maintaining or improving quality and security using DevOps forces out inefficiencies that have existed in your processes for years.

refining the process I used for analyzing different businesses and felt it was important to document the approach to share as pre-work for my workshops. I would send this out ahead of time to a few key leaders in the organization to start mapping out the deployment pipeline so we would have a good straw dog for the workshops. The workshops would then really help get everyone aligned on the changes that would have the biggest impact and get everyone excited about starting the journey.

The reason I decided to publish the book is that I realized I can't do workshops for every company that needs to improve their processes and I thought others might find the approach helpful.

InfoQ: For whom is this book intended?

Gruver: The book is intended for large organizations that have tightly coupled architectures. Small organizations or large organizations like Amazon that have architected to enable small teams to work independently won't learn much by reading this book. They would be better served by reading a DevOps cookbook to identify some best practices that they have not implemented yet. This book is not intended for them. It is instead for larger organizations that have to coordinate the development, qualification, and release of software across lots of people. This book provides them with a systematic approach for analyzing their processes and identifying changes that will help them improve the effectiveness of their organizations.

InfoQ: What makes DevOps fundamentally different from other agile approaches?

Gruver: I try not to get too caught up in the names. As long as the changes are helping you improve your software development and delivery processes then who cares what they are called? To me it is more important to understand the inefficiencies you are trying to address and then identify the practice that will help the most. In a lot of respects, DevOps is just the agile principle of releasing code on a more frequent basis that got left behind when agile scaled to the enterprise. Releasing code in large organizations with tightly coupled architectures is hard. It requires coordinating different code, environment definitions, and deployment processes across lots of different teams. These are improvements that small agile teams in large organizations were not well equipped to address. Therefore, this basic agile principle of releasing code to the customer on a frequent basis got dropped in most enterprise agile implementations. These agile teams tended to focus on problems they could solve like getting signoff by the product owner in a dedicated environment that was ▶

isolated from the complexity of the larger organization.

The problem with that approach is that in my experience the biggest opportunities for improvement in most large organizations are not in how the individual teams work but more in how all the different teams come together to deliver value to the customer. This is where I believe the DevOps principle of releasing code on a more frequent basis while maintaining or improving quality really helps. You can hide a lot of inefficiencies with dedicated environments and branches but once you move to everyone working on a common trunk and more frequent releases, those problems will have to be addressed. When you are building and releasing the enterprise systems at a low frequency, your teams can brute-force their way through similar problems every release. Increasing the frequency will require people to address inefficiencies that have existed in your organization for years.

InfoQ: How can DevOps help to optimize requirements and planning activities?

Gruver: My view of DevOps is optimizing the flow of value through organizations all the way from a business idea to a solution in the hands of the customer. From this perspective, it requires analyzing waste and inefficiencies in your planning and requirements process. In fact, I see this as one of the biggest sources of waste in many large organizations because they build up way too much requirements inventory in front of developers. As lean manufacturing has taught us, this excess inventory leads to waste in terms of rework and expediting so it should be minimized as much as possible. There are others in the DevOps community that tend to look at DevOps as starting at the developer and moving outward because that is where a lot of the technical solutions of automation and infrastructure as code are implemented. Again, I would not get too caught up in the naming. This is not about doing DevOps. It is about addressing the biggest sources of waste and inefficiencies in your organization. If you can develop and release code in a day but it takes months for a new requirement to make it through your requirements backlog, you probably need to be taking a broader end-to-end view of your deployment pipeline that includes your planning process and move to a more just-in-time process for requirements and planning.

InfoQ: Which metrics do you recommend for continuous integration?

Gruver: The first step is understanding the types of issues you are finding with continuous integration. It is designed to provide quick feedback to developers on code issues. The problem that I frequently see, though, is that the continuous-integration builds are failing for lots of other reasons. The tests may be flaky. The environments may not be configured correctly. The data for running the test may not be available. These issues will have to be addressed before you can expect the developers to respond to the feedback from continuous integration.

Therefore, I tend to start by analyzing why the builds are failing. This is one of the first steps you use to prioritize improvements in your process. Next, it depends on what you are integrating. If you are integrating code from a small team, you probably want to measure how quickly the team is addressing and fixing build issues. If you have a complex integration of a large system, I am much more interested in keeping the build green and making sure the code base is stable by using quality gates to catch issues upstream, because failures here impact the productivity of large groups of people. There is a lot more detail on metrics in the book because it really depends on what you are integrating at which stage in the deployment pipeline.

InfoQ: In the book, you distinguish between scaling with tightly coupled architectures and with loosely coupled architectures. What makes it different and how does that impact scaling?

Gruver: From my perspective, DevOps is a lot about coordinating work across different people in the organization, and the number of people you have to coordinate depends on the size of your organization and the coupling of your architecture. If you have a small organization or a large organization with a loosely coupled architecture then you are working to coordinate the work across five to 10 people. This takes one type of approach. If, on the other hand, you are in a large organization with a tightly coupled architecture that requires hundreds of people to work together to develop, quality-test, and release a system, you need a different approach.

It is important to understand which problem you are trying to solve. If it is a small, team environment, then DevOps is more about giving them the resources they need, removing barriers, and empowering the team because they can own the system end to end. If it is a large, complex environment, it is more about designing and optimizing a large, complex deployment pipeline. This is not the type of challenges that a small empowered team can or will address. It takes a more structured approach with people looking across the entire deployment pipeline and optimizing the system.

InfoQ: Which types of waste do you often see in large organizations?

Gruver: Most large organizations with tightly coupled systems spend more time and energy creating, debugging, and fixing defects in their complex enterprise test environments than they spend writing the new code. A lot of times they don't even really want to do DevOps; they just need to fix their environments. They are hearing from all their agile teams that they are making progress but are limited in their ability to release new capabilities due to all the environment issues. I usually start there. How many environments do they have between development and production? What are the issues they are seeing in each new environment? Are they using the same processes and tools for defining the environments, configuring the databases, and deploying the code in all the different environments? Is it a matter of automating these processes with common tools to gain consistency or are the environment problems really code issues introduced by other teams that are impacting the ability of other groups to effectively use the environments for validating their new code? These are frequently some of the biggest sources of waste that need to be addressed.

InfoQ: What can be done to remove the waste?

Gruver: It depends a lot on the source of waste.

A lot of the waste is driven by the time it takes to do repetitive tasks and manual errors that occur by having different people implement the process in different ways. This waste is addressed through automation. This requires moving to infrastructure as code and automating deployment and testing processes. The problem is that this effort takes some time so you should start where the improvements will provide the most value for your organization. This is why we do the mapping to determine where to start.

There is waste that is associated with having developers working on code that won't work with other code in development, won't work in production, or doesn't meet the need of the customer. Reducing this waste requires improving the quality and frequency of feedback to developers. The developers want to write good code that is secure, performs well, and meets the need of the business but if it takes a long time for them to get that feedback you can't really expect them to improve. Therefore, the key to removing this waste is improving the quality and frequency of the feedback.

Lastly, a lot of organizations waste a lot of time triaging issues to find the source of the problem. Moving to smaller batch sizes and creating quality gates that capture issues at their source before they can impact the broader organization are designed to address this waste.

InfoQ: Why should executives lead continuous improvement? How can they do that?

Gruver: In large tightly coupled systems, somebody needs to be looking across the teams to optimize the flow through the deployment pipeline. As we discussed above, this is just not something that small empowered teams are in a position to drive. It requires getting all the different teams to agree that they are going to work differently.

I frequently see people start with grass-roots efforts but most of these initiatives start to lose momentum as they get frustrated trying to convince peers and superiors to support the changes. If you are going to release code on a more frequent basis with a tightly coupled system then all the teams need to be committed to keeping their code base more stable on a day-to-day basis. If nine of the 10 teams focus on stability, it won't work. All the teams need to be committed to working differently. This is where the executives can help. They can pull the teams together, analyze the process, and get everyone to agree on the changes they will be making. They can then hold people accountable for following through on their commitments. This can't be management by directive because before mapping out the process with the teams, the executives ►

typically don't have a good feel for all the inefficiencies in their processes. It needs to be more executive lead where the executive is responsible for pulling everyone together and getting them to agree on the changes and then leading the continuous-improvement process.

This advice for executive leads is for large tightly coupled system. For organizations that have small teams that can work independently, this is not as important.

InfoQ: Any final advice for organizations that want to adopt DevOps?

Gruver: Don't just go off and do DevOps, agile, lean, or any other of the latest fads you are hearing about. Focus on the principles, analyze the unique characteristics of your organization, and start your own continuous-improvement journey. Judgment is required. Understand what you are trying to fix by changes in your process and then hold regular checkpoints to evaluate if your changes are having the desired effect. Bring the team along with you on the journey. At each checkpoint, review what got done, what didn't get done, what you learned during that iteration, and agree on priorities for the next iteration. The important part is starting the continuous-improvement journey and taking everyone with you. You will make some mistakes along the way and discover that what you thought was the issue was just the first layer of the onion but if you come together as a team on the journey you will achieve breakthroughs you never thought were possible.

APPDYNAMICS

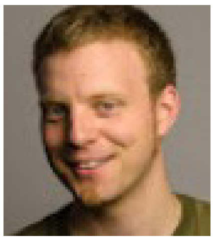
Secure your app's future.

Make better decisions with deep insights into performance.

Take a Tour

Read online on InfoQ

DevOps and Product Teams: Win or Fail?



Peter Neumark is a full-stack software engineer focused on building systems that solve real problems for real people.

To be honest, DevOps for me wasn't love at first sight.

My first encounter with it was in 2012 in an infrastructure team. Engineers in these teams work on building services such as authentication and image-metadata storage, which are used by product teams.

We decided to stop outsourcing the operation of our services and start going on call ourselves. It seemed like overkill: our Prezi was a hip new startup and we weren't struggling with the age-old enterprise problems of silos slowing down communication between developers and operation folks. With a newborn baby daughter, my biggest problem was trying to get enough sleep at night and the newly created on-call rotation was unlikely to help with that. What we hoped it *would* improve was availability.

Availability is a tricky thing to measure. In the fall of 2012, Prezi gave up on outsourced operations in an effort to stall our growing mean time to recovery (MTTR) as our user base exploded. MTTR is the average time it takes to repair an outage, which for us took around 40 minutes at the time. This may not seem like a big deal but it could be catastrophic for someone who could not present a thesis, startup pitch, or lecture for that time. The other widely used availability metric is mean time between failures (MTBF), which is the average length of time the system operates between outages. For us, this was around four days. ▶

Software engineers learned about the intricacies of debugging in production with strace. Sysadmins started writing webservices in Python. It was a love story of epic proportions.

These very similar four-letter abbreviations lead to very different engineering cultures. MTBF is maximised when few people make changes to the system in a highly controlled manner, carefully following well-proven processes. It's ideal for companies working on pacemakers or brake systems in trains.

MTTR, on the other hand, creates the perfect incentive for DevOps: the fastest way to find the cause of an outage and fix it is to bring together the engineers who wrote the component with those who operate it. Going one step further, Prezi put the software engineers writing the code on call. Why? In most cases, bugs or performance issues introduced during recent changes to services caused the back-end outages. Who would be better qualified to fix them than the people who introduced those changes in the first place?

DevOps delivered on our expectations. One year later, our user count doubled from 16 to 32 million, while MTTR and MTBF improved to around 30 minutes and one week, respectively. There were additional benefits: less specialisation in the team meant that any engineer could perform more tasks, leading to fewer bottlenecks and faster development. Software engineers learned about the intricacies of debugging in production with strace. Sysadmins started writing web services in Python. It was a love story of epic proportions. How did it end? I'm happy to say that infrastructure teams at Prezi are still well served by the DevOps way today. The DevOps nirvana ended for me personally when I joined a product team — a situation with very different rules and incentives at play as I soon found out.

What makes product teams special?

She said, "You ain't special, so who you fooling? Don't try to give me a line."
— Guns N' Roses, "Bad Obsession"

Infrastructure teams build services for product teams. Product teams build user-facing code on top of these services. Some product teams are responsible for the Prezi presentation editor and viewer, while others work on the Prezi.com website. Most teams at Prezi contain three to eight people. Unlike product teams, teams working on infrastructure contain engineers only. The most refreshing thing for me about adopting DevOps in such a team was the small set of axiomatic statements that guided our decisions:

- Everything we do is going to improve our primary metric (MTTR) at some point in the future.
- Specialisation is occasionally unavoidable but always frowned upon. Anyone in the team should be able to write the code and operate the resulting service.
- Every service has a clear owner. It's the owner's responsibility to guarantee availability. Teams should not modify services belonging to other teams.
- Write lots of tests to ensure bugs are caught before they are deployed.

Even if these directives aren't followed by all teams that claim to practice DevOps, they're fairly popular ideas in the DevOps community. Much to my surprise, none of these — to me, sacred — laws held true in their original form once I joined the product team.

My new team is responsible for the new-user experience at Prezi.com. We're happy when people follow a link to a presentation on Prezi.com and stick around because they like what they see. To achieve this, we're mostly using the playbook outlined in The Lean Startup: run cheap experiments to test ideas and don't build anything complex unless it's proven. It's a solid way to build a product, which incidentally invalidates all the DevOps rules I've grown to love on the infrastructure side.

Tests and ownership

Much to my initial surprise, a cheap experiment doesn't necessarily include tests. It's always a good idea to write unit tests but complex acceptance tests can often wait until the hypothesis of the experiment is validated (or invalidated, in which case we save the time we'd have used to write the tests). We learned this the hard way when we spent over four weeks building a content-recommendation experiment that brought disappointing results: none of the algorithms we tried was a hit with our users, who decided to ignore most of the recommended prezis. Prior to releasing the experiment, we spent almost a week writing top-notch Cucumber tests for this functionality. We decided to try something completely different, and the code, despite being wonderfully tested, was just bloat waiting to be removed, tests included.

I also had to seriously rethink my concept of service ownership. In the infrastructure teams, each critical service belongs to one team that is on call, reviews all pull requests from other teams, and generally assumes responsibility for the availability of the service. There is no such thing as an orphaned infrastructure service. Libraries shared between these services also have owners who oversee the overall architecture of the code and communicate API changes.

In our product team, some experiments require small modifications to many different services or libraries, none of them owned by us. Outside Prezi's infrastructure, community ownership rules have traditionally been more lax so some of these have no clear owner (or their owner has no plans to work on our feature requests). In these cases, it makes sense for us to do whatever is needed to get our experiment or feature out the door. As long as we're not causing any regressions that could negatively impact other teams' metrics, there's no harm in slightly loosening the rigid infrastructure definition of ownership. For example, one of the experiments we ran required us to log information in a way that the log-collection service didn't support. After countless redirects between different teams, we took matters into our own hands and decided to send logs to a deprecated but functional endpoint, which we patched to accommodate our needs.

Metrics

In a product team, making an impact on metrics is just as important as it is for infrastructure teams, but there can be several metrics. Choosing which to optimize for can be a daunting challenge (and may need to be periodically re-evaluated with shifting company priorities).

Outage times are relatively easy to measure, but the most relevant metrics for a product team can sometimes be quite difficult to calculate. For example, Prezi is a good match for marketing agencies who need to stand out and impress clients with their presentations. Our team would like to cater to their needs, providing an experience that wins their hearts. As a result, the most challenging part an experiment may be to guess whether a satisfied visitor was a marketing professional or not.

MTTR, despite all its advantages, is rarely a good measure of a product team's success. A product team that identifies an unfulfilled user need to address is victorious. Teams that achieve this by building insanely buggy experiments held together by luck and duct tape are no exception. In fact, as long as it's an experiment, the cheaper the better if the resulting data is sufficiently reliable. As long as it doesn't affect more than 2% of users and doesn't prevent customers from presenting their work, all is fair game.

Once the experiment is over and the product team is ready to release something, quality (and measurements like MTTR) become more important. Still, the amount of energy to pour into improving availability is primarily a product decision. I recently attended a talk by a senior engineer at Gilt, who mentioned that an outage outside of Gilt's peak hours is not considered a big problem because very few visitors use the site before its daily opening at noon. Gilt's back-end engineers are better served by a specialised metric that takes this aspect of their business into account than by the vanilla MTTR.

At Prezi, product teams clean up or sometimes even completely discard the scrappy experiment code prior to releasing a feature ▶

for all users. This is the right time to finish those acceptance tests nobody had time to write earlier. Since management evaluates product teams based on what they actually release for all users, product teams are highly motivated not to neglect this final step that is crucial to keeping the code base maintainable.

Specialisation

Specialisation (the archenemy of DevOps) is key in speeding up the experimental cycle. A daunting spectrum of skills is required to properly and efficiently execute experiments, which favours well-defined non-overlapping roles instead of generalists.

To start with, the team members need to identify where they can have the highest impact. For example, our team had to decide whether to concentrate on content recommendation or simplifying the user interface. This is generally the task of the product manager or product owner, often working together with a user-experience researcher, who conducts user interviews and gathers qualitative input from customers. Once the hypothesis for the experiment is ready (for example, "a larger sign-up button will increase registration"), a designer can create the necessary layout and assets. Finally, engineers build the minimum amount of software necessary to validate the hypothesis.

Depending on the product, it could make sense to have specialisation even among engineers. This doesn't make sense in our team but, for example, the Android team has engineers who specialise in writing the server-side features used by their client.

Despite striving to be cross-functional, one of the thornier problems product teams often face is lacking some necessary competence. For example, one of the product teams working on website personalization built a service to keep track of per-user settings such as the preferred language. This back-end service needs some kind of database but it probably doesn't make sense to have a dedicated DBA for the team. This won't be a problem until the database goes down because of an obscure leap-second bug at 4 AM. The traditional DevOps solution in this case would be to have the team's phones ringing off the hook until the problem is solved (hopefully, at least one team member would know what's going on). With a product team, this is not so clear: most iOS developers wouldn't know what hit the database and would have a hard time trying to figure out what's going on. One solution would be to have a dedicated team to operate databases for product teams, but this sounds a lot like the developers versus operations divide of the bad old days before DevOps.

The problem with dedicated operations teams

Every now and then, I'll meet someone who is nostalgic for the separate developer and operations teams that happened to get along just fine at their company. Even the classical DevOps novel The Phoenix Project ends this way, with separate teams on amicable terms. Be warned: even if this does occasionally happen, it's exceedingly rare. The reason is opposed incentives.

Developers are expected to deliver product features or user stories, preferably in a predictable way. When unforeseen problems cause delays, developers — keeping the release date in sight — struggle frantically to compensate, releasing incomplete features (although some would argue that there's no such thing as releasing too early).

Operations usually prizes availability. MTTR may be more DevOps-friendly than MTBF, but regardless of how it's measured, outages are more difficult to prevent in face of constant change. This can cause engineers in operations to be over-cautious and too conservative. Lots of new product features deployed to production are to the developers' merit, but if any of those shiny new features cause an outage, the operations team will be waking up to fix it.

Needless to say, the company needs both skill sets and accompanying perspectives. DevOps is useful because it unifies these opposing viewpoints in the team. But what if DevOps in the traditional sense is not practical? Should the product team that tracks user settings hire a DBA to resolve a once-a-year database-induced outage? If the incentives are correctly aligned, there's no reason a product team can't work with an operations team. The trick is to have well-defined interfaces between the teams.

Working with platform teams

I had a wonderful experience recently. A friend and former teammate who is working in one of the infrastructure teams told me that he was assessing what back-end pains product teams had. His team is responsible for software deployment and the staging environment. He had several project suggestions that could make our lives easier and he wanted to know if there was anything in these systems that was causing us trouble. His approach was nothing like that of a traditional operations engineer. Considering his team is a platform team, not an operations team, this is hardly surprising.

Operations teams assume complete responsibility for running an application or service once the handover from development ends. They have to try to make sense of whatever is thrown over the fence. The opposed incentives are almost guaranteed to be there. Platform teams, on the other hand, provide tools and services to ease the burden of operating software for their customers, the product teams. The responsibility for the user experience remains solely with the latter. For instance, as an engineer in a product team, I need to deploy my code into production somehow. I don't have the luxury of outsourcing this problem to an operations team: it's my responsibility, but I do have a number options. I could build my own custom deployment solution but this takes away valuable time from experimentation and product development. I could use an open-source or commercially available product but chances are that this would still require a significant time investment on my part due to the learning curve and possibly the need for customization. My friend's platform team offers a third solution: as long as I produce a tarball with a specific format, the service they provide deploys the code on our nodes. Needless to say, I am still responsible for what I deploy: if I end up using their service to deploy buggy code, that's my fault. All they guarantee is that the deployment mechanism will work as expected. Still, the service saves me a lot of time.

Many larger companies have complex internal billing systems, where the APIs of services operated by platform teams "bill" the product teams based on their usage (sometimes in terms of dollars). While this may not be an option for smaller organizations, the main message is that platform teams provide a service to product teams (who are still responsible for operating the product they build). A platform team's value can be measured by the value of the services they provide to product teams.

To solve the user-settings service problem, Prezi could establish a platform team that provides a database as a service. They could provide guarantees to their clients (for example, that they will take care of obscure leap-second-induced bugs). In exchange, when the product team's feature turns out to be a hit with users, they may humbly say that they couldn't have done it without the database team. And they would be right.

Third-party platform teams

I love Prezi's in-house deployment system but if I were the CTO of a startup, I wouldn't think about building my own. I would go straight to Heroku.

Heroku is the perfect example of a platform team that is not part of the company. It does an excellent job of documenting its services and its product is easy to use. If I'm not satisfied with Heroku's offering, I could go to a dozen competitors who offer various solutions to the problem of deploying and running my code.

Even larger companies are seeing the benefit of renting platform teams. Amazon Web Services' (AWS) Redshift database ▶

Platform teams provide tools and services which ease the burden of operating software for their customers, the product teams. The responsibility for the user experience remains solely with the latter.

is a great example. Redshift is a database service that is operated entirely by AWS. Its use requires no traditional DBA skills. It's not suitable for every kind of workload but if it proves to be a good match for your use case, it may be much easier for product teams to use than a data warehouse of their own.

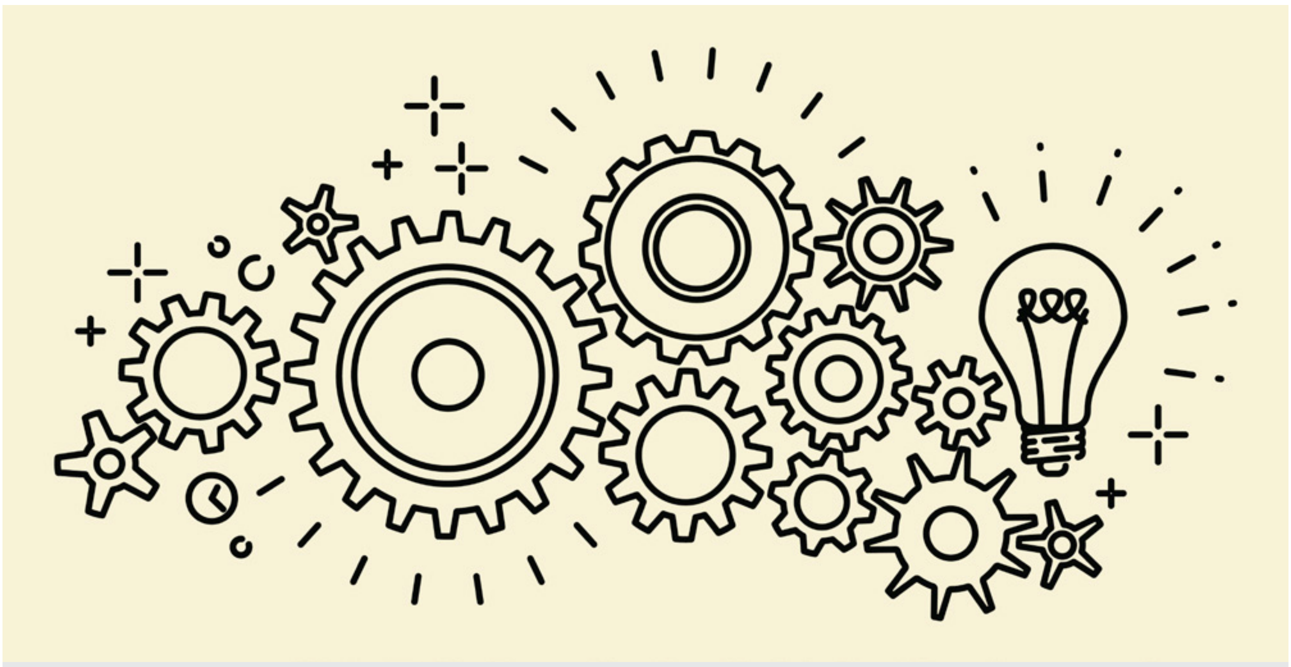
Using third-party services makes it easier to align the incentives of the service provider and the consuming product team. The services offered compete with similar services in an open market so it's in their interest to keep their offering stable, up to date, and easy to use. At the same time, services can choose their customers. They have the right not to address some customer needs. For example, AWS designed Redshift to be used as a data warehouse, not as a low-latency application database. The AWS ELB load balancer does not support URL rewriting: it's tailored towards relatively simple use cases and excels precisely because it is reliable and easily configured. In theory, the market forces both providers and consumers to compete, which is difficult to achieve with in-house platform teams.

Conclusion

Coming from an infrastructure team dedicated to writing backend services — the traditional DevOps stronghold — I was surprised to learn how different life can be in a product team. The need for a host of skills outside of engineering (product management, UX, design) requires specialization within the team. Ironically, product teams also lack certain engineering skills necessary to get the product out the door and keep it running in operation.

Companies like Prezi are experimenting with solutions to these problems in the form of platform teams. Despite the seemingly similar mission to the more traditional operations team, platform teams do not take over responsibility from product teams. Instead, they offer services to make the product team's work more effective. This setup remedies the ancient problem of misaligned incentives between teams that build software and teams responsible for system infrastructure. And if that's not the spirit of DevOps, I don't know what is! ■

Deploying at Scale: Chef, Puppet, Ansible, Fabric and SaltStack



By Omed Habib

The manageability, reliability and powerful technology of remote servers — cloud computing — allows IT managers to deploy hundreds, even thousands of machines. At the same time, the cloud creates a new challenge for sys admins and ops teams: how to maintain and configure all these machines. How do you apply patches, maintain updates and fix security gaps?

The answer is to use powerful tools like Chef, Puppet, Ansible, Fabric or SaltStack for managing Infrastructure As Code (IaC) automation. IaC means deploying and managing infrastructure for computing, including virtual servers and bare-metal servers. Definition files are used instead of physical hardware management. Here is a bit of the history, background, advantages and disadvantages for each of these infrastructure configuration management tools currently on the market.

Puppet

Puppet was founded in 2005 by Luke Kanies, making it one of the earliest infrastructure configuration management tools. It is free software written in Ruby and made available under the Apache Software License 2.0, although it was released using the GNU General Public License up to version 2.7.0. It operates declaratively on Microsoft Windows and UNIX-based systems like AIX, Solaris and Mac OS X. Puppet uses a declarative language to define system configuration. To begin, you set up system resources and relevant state that are stored in files called Puppet Manifests. A resource abstraction layer then lets you use higher-level terms such as packages and services to define configuration.

Because Puppet is model driven, you don't need an extensive programming background to use it. In a model-driven approach, you can set up how you want the infrastructure and applications to operate. With the model in place, you can then test and evaluate changes you want to deploy across the system. Constant reporting and feedback allows you to improve processes, show compliance and tweak the results as you go. Puppet is perhaps the most popular infrastructure configuration and management tool among those described here, used by a variety of organizations including:

- Mozilla
- PayPal
- Spotify
- Oracle
- Rackspace
- Wikimedia Foundation

Chef

Chef is a configuration management tool Adam Jacob devel- ►

oped to use in his consulting company. Seeing a broader use for managing Amazon Web Services operations, he joined with Nathan Haneysmith, Barry Steinglass and Joshua Timberman to found a firm called Chef to manage the tool.

Chef is based on "recipes" that describe how the software will configure and manage utilities and server apps like MySQL or Hadoop. These recipes can be combined to form a "cookbook." Each recipe defines resources used in a state such as what services should be operating, what packages need to be installed and what files need to be created. The resources can be modified to make sure programs are installed in a specific order based on dependencies. Industry commenters often suggest that DevOps and developers usually choose Chef while SysAdmin's prefer Puppet.

There are two versions of Chef: an open-source basic version and a premium enterprise edition. The enterprise offering has both on-premise and hosted versions. Open-source Chef is available at no charge but lacks many of the add-ons in the enterprise edition as well as ongoing support.

Chef began as a Linux product but later added support for Microsoft Windows. It runs on major platforms including

- Solaris
- Ubuntu
- Microsoft Windows
- FreeBSD

It is used by companies and websites such as:

- Facebook
- Airbnb
- Expedia
- Citi
- Disney

Chef and Puppet are two of the largest infrastructure management tools available to you. They both continue to respond to the needs of enterprise companies by providing new features, and they are also busy creating partnerships with major vendors like Microsoft to better integrate with their platforms. Puppet has also aligned with software defined networking (SDN) vendors to stay in the forefront of that technology. Choosing between the two is a matter of determining the core advantages of each and figuring out which align with your requirements.

Ansible

Ansible is an open-source software framework for managing and configuring infrastructure. It offers configuration management, software deployment for multiple nodes and ad hoc task execution. You can manage it using PowerShell or through a secure shell (SSH). This software framework was developed by Michael DeHaan, who was also one of the original developers over the Func framework used for administering systems remotely. Ansible is included in distributions of Fedora, and is also available if you use CentOS, Red Hat Enterprise Linux, Scientific Linux and other operating systems. A company of the same name was created to support the software product and help it grow in business markets. Red Hat acquired the company in 2015.

The name Ansible is derived from a communications system in "Ender's Game," a 1985 novel by Orson Scott Card. The fictional system was first invented for the 1966 novel "Rocannon's World" by Ursula K. Le Guin.

Ansible controls two kinds of servers: nodes and controlling machines. The system is based on a single controlling machine, which configures and manages nodes using SSH. Modules are deployed over SSH to orchestrate notes, which then communicate to the controlling machine using a JSON protocol. Ansible is light on resources because when it is not managing nodes, it does not run any programs or daemons waiting for utilization.

Unlike Puppet and Chef, Ansible has an agentless architecture where nodes need a daemon to talk to the controlling machine. Under this system, nodes do not need to install and operate daemons in the background to communicate. This set-up significantly reduces network overhead because it stops nodes from constantly polling the controlling machine.

Ansible was designed with a minimalist approach, with a focus on making sure managing the system does not create additional dependencies on the system itself. It is secure because it requires OpenSSH. In addition, Ansible playbooks are written in an easy-to-learn, descriptive language. It is used in a variety of private and public clouds including:

- Google Cloud Platform
- OpenStack
- SoftLayer
- Amazon Web Services
- XenServer

Ansible works well with Aerospike, Riak and Hadoop, monitoring resource consumption by every node while using few CPU and memory resources. Organizations and companies deploying Ansible include:

- NASA
- Weight Watchers
- Juniper
- Apple

Its agentless model makes it a popular choice for government divisions such as NASA because it is very secure, a quality highly valued in federal and state governments.

Fabric

Fabric is an open-source command line tool and Python library used to smooth out SSH utilization for system administration and application deployment. It consists of a suite of operations for launching shell commands, either locally or remotely, via sudo or normally; downloading and uploading files; and asking for input from users, stopping execution and other auxiliary functions. While products like Puppet and Chef focus on organizing and handling system libraries and servers, Fabric is more concerned with deployment and other application-level functions.

Developers like Fabric because it is simple, easy to maintain and you can add any type of job quickly. You can execute Python functions using the command line, and launching shell commands on SSH is simplified due to the extensive library of subroutines. Companies using fabric include:

- Snap
- Coursera
- Instagram
- Sosh
- FlightAware
- The Orchard

Fabric development is managed by Jeff Forcier. He is assisted by open-source developers who add suggestions and patches through the Fabric mailing list, on IRC chats or via GitHub.

SaltStack

SaltStack is an open-source platform based on Python, and it is used for managing and configuring cloud infrastructure. It was developed by Thomas S. Hatch using ZeroMQ to create a better tool for collecting and executing data at high speeds. Initially released in 2011, Reliable Queuing Transport (RAET) was added in 2014. The project has subsequently been developed through a partnership that includes several large enterprises. SaltStack was built from the ground up to be highly modular and flexible, and able to adapt to diverse applications. It creates Python modules that each manage a different part of the Salt system. You can detach and modify the modules to fit the needs of your project. Each module is designed to handle a specific action. The six types of modules include:

- Execution modules which offer functions for directly executing the remote execution engine as well as help manage portability and core API functions.
- Grains detect system static information and keep it in RAM for fast access.
- State modules represent the back end, executing code to configure or change a target system.
- Renderer modules pass information to the state system.
- Returners modules manage the return locations associated with remote execution calls.
- Runners are convenience apps.

SaltStack created a buzz early on by capturing the 2014 InfoWorld Technology of the Year Award as well as the 2013 TechCrunch Award for Most Exciting Project. Organizations and companies using SaltStack include Adobe, Jobspring Partners, Dealertrack Holdings, JumpCloud and International Game Technology.

This article covered five of the top infrastructure configuration and management tools available. It's a highly dynamic area of enterprise computing, with new tools constantly evolving to solve various challenges. Each of these solutions gives you lots of ways configure your infrastructure, allowing you to manage digital transformation at scale easily and efficiently. ■

Internal Tech Conferences: How and Why

Matthew Skelton has been building, deploying, and operating commercial software systems since 1998. Co-founder and principal consultant at Skelton Thatcher Consulting, he specialises in helping organisations adopt and sustain good practices for building and operating software systems: continuous delivery, DevOps, aspects of ITIL, and software operability. Matthew curates the well-known DevOps team topologies patterns and is co-author of the books Software Operability (Skelton Thatcher Publications) and Continuous Delivery with Windows and .NET (O'Reilly). He tweets at @matthewpskelton.

Victoria Morgan-Smith is an agile delivery coach at the Financial Times, where she has been helping teams succeed since 2009. Before this, she was a developer for nine years, a background which fuels her interest in finding fun ways to coach, energise, and motivate teams into self-organising units. She is passionate about collaboration beyond the team, adopting agile principles to get under the skin of what will deliver measurable business value around the organisation. She tweets at @VictoriaJMS. https://www.linkedin.com/in/victoriamorgansmith

Matthew Skelton and Victoria Morgan-Smith discuss how to use internal conferences to boost your organisation's social capital, the currency by which relationships flourish and businesses thrive.

An internal 'tech' conference can be a powerful way of communicating and celebrating technology teams that build and operate the software systems that are increasingly essential to many organisations. In this article, we share our experiences of organising and running internal tech conferences: getting buy-in, preparing speakers, promoting the day (before and after), and running the day itself. We hope that you'll be inspired to help run a tech conference within your own organisation!

Why run an internal tech conference?

Building and operating modern software systems is a challenging task: the pace of delivery is rapid; the opportunities (and threats) from a global, web-connected population are immense; and we

KEY TAKEAWAYS

Software engineering is as much about people as the technology itself: an internal tech conference can greatly **boost your organisation's social capital** — that currency by which relationships flourish.

The format you choose for your internal tech conference depends on **what you want to achieve**: it can be "by the people for the people" or a showcase to celebrate achievement. You can limit the audience or speakers to a single department, invite other divisions, or even invite external speakers or audiences.

Making the event a success takes effort: **choose your speakers well and mentor them** as they prepare their talks. **Work on the logistics** because it's the little things that count.

Remember to **have fun**. Death by PowerPoint will mean people remember the event for the wrong reasons!

Follow through: for a lasting impact, keep sight of the outcomes you seek and be ready to **work with others to keep the momentum going**.

need to draw on a wide range of skills and experience in a coordinated, joined-up way. The world of work is now highly complex, meaning that instead of streamlining our organisations to continue producing the same output more efficiently ad infinitum, we need to enable them to be agile, to respond to constant change around them.

Internal tech conferences can help people to build relationships and discover more about things that are going on in a friendly environment and non-threatening context, so that they have the confidence to wholly participate and know that others will be able to get in step with them to help make new ideas happen.

In this article we draw on our personal experience of running internal tech events companies we've worked with, along with reflections and advice from people at Paddy Power Betfair, Callcredit, ING and others. You'll find further reading & listening material at the end of the article - there is so much inspirational work happening in so many organisations.

What types of events work well?

The Financial Times, the world's leading business news provider, decided to run a full day event with a mixture of content and debate from a diverse set of people across the department, joined solely by an internal audience. Their conference consisted of Lightning Talks, Panel Debates, Open Spaces and a game, rounded off with beer, chatter, and more beer, writes Victoria Morgan-Smith.

When planning the first internal tech conference at a ticketing company based in London, writes Matthew Skelton, we realised it would be a good idea to invite teams outside of technology/engineering to learn what the tech teams actually did, as that was a mystery to many people; the sessions were known as 'Engineering Day'. All staff in the London office were invited. The first installment was a full day session with many different speakers from the engineering teams (and some from other departments too). Subsequently, the team ran a focussed half-day Engineering Day events every six months which allowed more people to attend as they found it easier to spare a half day rather than a full day. Teams in India were included via video conference and eventually the show went 'on the road' to the teams in Edinburgh.

Other organisations have found different formats that work well for them. For instance, Rich ►

A panel at the Financial Times' Engine Room Live event get ready for a question from the audience - complete with soft blue, throw-able microphone!

Haigh of online betting and gaming company Paddy Power Betfair explains that they hold an annual DevOps Community conference for everyone in Product and Technology. *"On the morning we invite vendors to come and talk about how we have used their products – what's coming up in their roadmaps, etc. In the afternoon we open the floor to talk about projects they have been working on, interesting tech, R&D they are doing etc. We then have a social event in the evening.* [...] *We hire an external venue – have twitter walls and a stage – and we video everything so we can share the knowledge further post the event."*

At the Dutch bank ING, an internal change from older, silo-based ways of working to a more fluid, DevOps-inspired approach was accelerated by running an internal conference based directly on the DevOpsDays conference format, combining external invited speakers, internal talks, and short 'lightning' talks of 5 mins each. This event helped to "stir up the discussion" around new ways of working and inspired people to attend and help organise a public conference (DevOpsDays Amsterdam). Furthermore, after the ING people blogged about their conference and shared the slides, a team at US retailer Target were inspired to run their own conference! The recently-published DevOps Handbook by Gene Kim et al has more on this.

Callcredit, a major provider of financial data in the UK, has run a public-facing Testing conference called the Leeds Free Testing Atelier. It was a one day event, with a mixture of Testing related workshops, talks and games. Stephen Mounsey explains that their ambition behind it was to *"kick start a vibrant testing community. After seeing the remarkable community spirit of the Ministry of Testing and Test Bash conferences, we wanted to replicate this on a smaller scale for Leeds. We also wanted to get to know other testers, get ourselves some social connections, someone to share our problems with."* They were keen that the event was "by testers for testers", but actually attracted a much wider audience of DevOps engineers, UX, BA's and developers.

There is no 'right way' to run an internal tech conference - it depends on what your team, department or organisation needs. An important thing to consider early on is the audience: who should we invite? Who would benefit most from the conference? The answers to those questions should help to frame your conference planning: as the attendee list grows the focal point of discussions stretch to fit the audience, whereas a compact group allows the focus and aims of the conference to remain tight and on track.

Another factor to consider is whether you bring in external speakers and the impact this could have; it can have the effect of feeling like an industry conference without the cost of flying everyone to another country!

However you choose to run your internal conference, it's important to give people enough time and space to immerse themselves in the event - help them clear their calendars so they can 'shake the every-day out of their hair'. This is a chance for people to give themselves permission to pause - they need to be ready to get as much out of the day as possible.

Typical goals or outcomes - 'why'

An event like this has many immediate, measurable benefits, as well as some hidden, long term ones. One major goal is to shape and promote 'good' culture, where people are encouraged to challenge the status quo, to get excited about new possibilities by having the space to experiment without the fear of failure. Psychologically people are braver surrounded by people they know and trust; an internal tech conference can bring people together - help them to get to know each other in an open context, allowing opinions to flow ideas to happen.

The FT identified these things upfront when they defined their vision for the day. They sought to "*create an opportunity for the FT Technology department to make connections between teams, learning together to increase communication and innovation. Participants would get a chance to take part in conversations that matter to them, and test out their public speaking voices in a friendly space. Attendees would discover good ideas and new channels of communication. The FT expected to see as a result better team working practices, happier staff, stronger communication, and ultimately improved job satisfaction; productivity and efficiency.*"

What do speakers and presenters gain?

An internal tech conference is a great opportunity to identify and celebrate 'unsung' people, teams, and achievements; it's a way to showcase team projects that might be forgotten or that do less exciting but crucial work. It's a great way to make people feel that they are being appreciated. Matthew's team deliberately looked for people who had done interesting, foundational, or transformative work: database upgrades that allowed them to shift platforms, deployment automation that reduced outages, and what it's like to be on the 1st-line support team.

Speaking at an internal event will force people to challenge themselves. Those opinions they've been expressing on a daily basis to their neighbours? Well, if they want to express them on stage, they might want to check some facts, reflect on what they've heard and read elsewhere, and then do a bit more research to be sure they know what they're talking about. A team can easily become a small echo chamber, so it's healthy for people to poke their head out of it occasionally. Then they can say "hey, it's not just me - other people are saying this stuff too!". Many conference speakers will tell you that the biggest thing they get out of talking at an event is the extra learning they do in order to test and flesh out their message.

To many, public speaking is the most terrifying experience in the world, consequently many great ideas and opinions remain unheard. If we can help people tackle that fear, that's a huge gift. Several people from the our internally run events have now been talking at external conferences as a direct result of gaining confidence to do so at internal events. We're now hearing from people whose voices were not heard before.

What do attendees gain?

One of the big benefits for attendees is the buzz of sharing insight and successes; this makes an organisation feel like a great place to work. The FT was able to get energy going early by starting the day with 'Lightning Talks' where speakers shared thought-provoking ideas about technical challenges and about human behaviour in the space of 4-5 minutes. The events at the ticketing company had two parallel streams of talks along with tutorials for non-techies on HTML and simple website design, delivering something for everyone. In both cases, attendees felt a strong sense that they were worth investing in.

Another benefit is the validation that comes from hearing colleagues talk about practices used elsewhere in the industry too. By bringing a piece of the outside world into a company, everyone gains better understanding of how their shared work makes up a bigger picture. Often this involves reinforcing they're doing the right things already - a big motivational factor.

How to 'sell' an internal tech conference

The main benefits of an internal tech conference are ultimately social: improved interaction, trust and understanding between people and teams. Selling the value of the event may simply be a matter of selling the importance of these aspects to your work environment.

The FT's event started with some engineers chatting over drinks with their CTO. They suggested an internal event to him as a way to get ideas discussed and experience shared, getting more than they bargained for when he then asked them to organise it! The successes of the event was sufficient to sell the idea of a second one the next year.

Specific benefits we have seen across our events and those at other organisations include:

- **Smoother collaboration:** Developers and Operations staff organising rotations in each others' teams
- **Improved learning:** a significant increase in lunchtime/evening tech talks; engineers self-organising regular open discussion sessions.
- **Increased engagement:** working groups set up by groups of developers to delve into particular challenges (e.g. alerts overload on microservices!, smarter cloud man-

►

agement, new approaches to quality assurance).

- **Inter-team communication:** improved communication between the technology department and the rest of the organisation, opening up dialogue that was not possible (or not happening) before the events.

- **Inspiration:** a Product Owner, inspired by what he'd heard, taking back to the Product team a long list of ideas where they could collaborate better with the tech team on tackling some of the next big challenges.

- **Staff retention and recruitment:** speakers taking to the stage at external events, which in turn has brought good candidates applying for jobs.

Jess Lancaster of TechSmith [8] makes a great point about the impact of the safe environment we're striving to create: "*It makes it likely the person, who may be very shy, will come out of their comfort zone and do something they would not otherwise. That breaks open information, but it also gets people used to pushing their boundaries.*" So, inspire people to be proud of their successes, and they will do even more.

To help 'top and tail' the event with a buzz beforehand and lasting memories afterwards, make sure you sort out attractive posters and good quality photos and video footage.

Logistics - making the day happen

Stage-management is crucial

A day like this can have many moving parts, so the FT had someone in place to "stage manage" the day. Each talk had scribes, mic throwers (for ease *and* entertainment!), a timekeeper and a "Slack channel watcher" in case of questions or problems from remote team members. Making sure everything runs to schedule and logistical transitions are smooth (including ensuring people have clip-mics switched *off* if they go for a last minute bathroom "visit" before going on stage, ahem…). Having an in-house "Chief Worrier" is a blessing!

You have to consider having one or two 'plants' (people prepared) in the audience who are primed with interesting questions. People can be shy and can find it hard to be the first to ask a question, so if the plant asks the first question, that can open up the discussion. On top of that, it can take people a moment to absorb what they've been listening to and formulate questions, as well as planning how to articulate this without stuttering or tripping up verbally. Having a confident ally planted ready to ask a worthwhile question if there is silence can really help others to follow suit.

Conference organisers need to be prepared for when one or two speakers drop out for whatever reason (emergencies, illnesses, nerves). Having a couple of backup talks ready with willing participants who don't mind not getting to speak if disaster doesn't strike is essential!

If your organisation is distributed, then consider what level of involvement you want from smaller off-site teams. The FT sought input in terms of what topics and questions would be asked, but on the day itself it was a one-way broadcast from the main office. Local posters, and local pizza, with live streamed YouTube content was as far as they went. Matthew's team arranged to incorporate sessions from India via video conference, and also took a small team to a branch office in Edinburgh to replay talks and answer questions.

The "catch box" microphone at the FT tech events was a big success, leading to more interactive and spontaneous questions than handheld microphones.

Choose the right format

During the day, having a variety of sessions keeps things lively. Lightning Talks and presentations can inspire, Open Spaces can stimulate broader discussions, and panels can enable lively debates. A good game towards the end of the day can set everyone up for the all important drinks at the pub after where real networking takes place.

The FT were keen to keep all of the speakers internal for two reasons. The first was about maintaining the safety and security of the space - not only in terms of feeling safe to fail, but also in terms of confidentiality. Attendees and speakers were allowed the freedom to say anything they liked. Rich Haigh from Paddy Power Betfair has the same view about the need to be able to present "*stuff that might be too sensitive to discuss externally*". The second reason was related to the objective of giving people a voice, and showing them their opinion is respected - bringing in external speakers for the FT's event was considered to undermine and distract from this.

Mentor less experienced speakers

Less experienced speakers often wildly underestimate how long their talk will take and end up writing too many slides and rushing the talk, trying to fit in all the material they prepared, or overrunning by 15 or 20 minutes. Avoid this by mentoring speakers: help them to prepare a slide deck (one slide per minute of talking is a good guide) and ask them to present the talk to you a few days before the event so that they can practice the timings. Make sure the material is suitable for the audience, too; there is little worse than someone talking 'tech' in a way that is incomprehensible to the audience. If the talk is too long or too ambitious, help them to trim the unnecessary bits - be kind but firm!

Keep the lights on and maintain service

Rich Haigh at Paddy Power Betfair found that "*The biggest problem was that I was intending to take a large number of the tech staff out of the business for a day. In order to mitigate this, we always set up a war-room.*" Because they were off-site for the full day, they needed to create a specific set-up to allow essential support to continue. Holding your event on-site could enable people to opt into specific parts of the event they are interested in and also means their availability in case of disaster isn't an issue. This is definitely one to bear in mind when designing your event.

Choose speakers based on enthusiasm and demographics

It's important to be inclusive - you're trying to inspire people by amplifying good ideas and discussions, and so your panels and speakers need to be representative. Start with diverse organising panels who selected speakers accordingly. We found this to be the best approach:

- Start with who's keen and pick from these
- Consider the demographic mix and see who's missing that you really want to have there
- Revise the list - then go and do some sweet-talking!

Some might say that this is artificial, but in technology departments across the industry we find ourselves in a bit of a bootstrap situation of a less-diverse-than-we'd-like workforce, and it takes conscious effort to get out of it - it's worth going the extra mile!.

Crucially, make sure you have speakers from teams or departments across the spectrum of the attendees; specifically, if you have only extravert developers talking, then more reserved DBAs, support people, Ops and security people will feel left out! Find ways to include a range of subjects and people from different backgrounds - you will likely be pleasantly surprised by the result.

What benefits result from a successful internal conference?

"*An internal conference can be a fantastic opportunity.*", says Cait O'Riordan, CIO of the Financial Times. "*We go to external conferences to learn from other companies and an internal conference ensures that we also learn from the brilliant people inside our company. Ideally that would happen anyway in the course of our daily lives - but putting some time in the calendar makes sure that sharing actually happens and happens at scale*". An extra element for O'Riordan was the timing of it: "*Our conference happened to coincide with when I first joined the FT - so on a personal level it was a brilliant opportunity to find out about the great work the team is doing.*"

Some of the observed benefits from our own experiences have already been described above. But there will be others we've not even thought of yet. Extra benefits witnessed at Callcredit from their public event include:

- We gained exposure in the local marketplace.
- We got awesome speakers to come and talk to our people for free.
- We have enthused our staff, given them the confidence to get out there and speak.
- We have created an external support community for our people.

It seems to be particularly useful when the events occur regularly ▶

and people start to look forward to the next one. Capture the good ideas being thrown about during the event, and follow up afterwards to encourage the people who raised them to take their ideas forward. Send out a feedback request form immediately after - ask "what will you do differently?" - encourage attendees to think about what the day has meant to them and generate some intention to act. Finally, write up a summary of the day on a public tech blog. It acts as a great reminder to those who are tempted to quickly forget the event!

Conclusion

You could choose to run your event "by the people, for the people", like the Financial Times have done. In this scenario, all talks and events are done by people internal to the department with their colleagues as their audience. They charged people up and let them loose on topics designed to generate heated debate.

You could also turn the event into a real showcase and invite the rest of the company to come and meet your engineers and hear about some of the exciting things they've been doing. Here you could be uncovering some of the challenges of delivering modern software systems and building a dialogue between the technical departments and the rest of the organisation.

Onlignment hit the nail on the head when they say conferences are "*at their best when lively and interactive, highly relevant, varied, primarily bottom-up but with some inspiring top-down input;*".

Whatever format you choose, remember that people will get out what they put in - so the more you can involve everyone in the up front contribution of ideas, organisation and actually taking part in the day, the more they will feel it is "for them" and the more they will take away from the event.

Acknowledgements

We'd like to thank the people we interviewed for this article: Stephen Mounsey of Callcredit, Jan-Joost Bouwman of ING, Rich Haigh of Paddy Power Betfair and Heather Mickman of Target. We'd also like to thank our past and present colleagues for their huge contributions to making the tech conferences there a success - these events are always a team effort. ■

Further reading

[1] Towards a Social Theory of Rhythm by Peter Nelson, pp 149-155 in Topicality Of Musical Universals / Actualites Des Univers Musicaux, ed. Jean-Luc Leroy, published 2013 by Éditions des archives contemporaines, ISBN 9782813000613 http://www.research.ed.ac.uk/portal/en/publications/towards-a-social-theory-of-rhythm(424b5f79-9c31-45bd-80e2-a24aaaae9846).html

[2] Principles of Sociotechnical Design Revisited by Albert Cherns, in Human Relations March 1987 vol. 40 no. 3 153-161 http://hum.sagepub.com/content/40/3/153.short

[3] Social capital in the growth of science-and-technology-based SMEs by Jukka Partanen, Kristian Möller, Mika Westerlund, Risto Rajala, Arto Rajala http://www.sciencedirect.com/science/article/pii/S0019850108000655

[4] Conferences – jumped up classrooms? By Donald Clark http://donaldclarkplanb.blogspot.co.uk/2008/11/conferences-jumped-up-classrooms.html

[5] How to run an internal unconference by Henrik Kniberg http://blog.crisp.se/2013/06/30/henrikkniberg/how-to-run-an-internal-unconference

[6] Running Internal Events – The Goat Farm – Episode 5 (Michael Ducy / @mfdii) https://goatcan.do/2015/03/25/running-internal-events-the-goat-farm-episode-5/

[7] DOCCON1 – organising a conference (Rich Haigh / Betfair) https://betsandbits.com/2014/09/23/doccon1-organising-a-conference/

[8] How TechSmith Rocks Its Internal Developer Conference (CIO.com) http://www.cio.com/article/2380063/continuing-education/how-techsmith-rocks-its-internal-developer-conference.html

[9] Engine Room Live internal conference 2016 (Victoria Morgan-Smith / FT) http://engineroom.ft.com/2016/05/03/engine-room-live-internal-conference-2016/

[10] Leeds Testing Community unConference (Ash Winter) http://testingisbelieving.blogspot.co.uk/2015/10/leeds-testing-community-unconference.html

[11] DOES15 - Heather Mickman & Ross Clanton - (Re)building an Engineering Culture: DevOps at Target https://www.youtube.com/watch?v=7s-VbB1fG5o

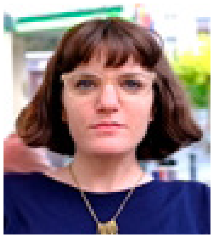
[12] Reinventing organizations by Frederic Laloux (Nelson Parker, 2014) ISBN 978-2960133509 https://www.amazon.co.uk/Reinventing-Organizations-Creating-Inspired-Consciousness/dp/2960133501

[13] The Future of Management Is Teal (Frederic Laloux) http://www.strategy-business.com/article/00344?gko=10921

[14] DevOps Handbook by Gene Kim, Jez Humble, Patrick Debois & John Willis (IT Revolution Press, 2016) ISBN

Read online on InfoQ

Communities of Practice: The Missing Piece of Your Agile Organisation



Emily Webber is an independent agile coach, consultant, and trainer. She helps organisations in public and private sectors to develop their agile capability with a focus on sustainable change. Her focus is on people, teams, communication, communities, and creating a culture of learning. She is the author of Building Successful Communities of Practice: Discover How Connecting People Makes Better Organisations and has ongoing research into the subject. She is a regular conference speaker and supports new speakers through her (co-run) Agile on the Bench meet-up and work on diversity at conferences. She blogs at emilywebber.co.uk and tweets at: @ewebber

Communities of practice regularly bring together people who share areas of interest or concerns. They are loose structures that support their members and the organisation's development of those areas. They often form around a specific job role but can also centre on a specific area of interest.

A community of practice is not a new concept: cognitive anthropologists Jean Lave and Etienne Wenger first published the term in 1991. But communities of practice have a specific application in agile organisations and because of this they are increasingly popular.

I attribute this increase in interest to two factors: organisations seeking to scale agile development either using scaled frameworks or referring to concepts introduced in the popular Spotify articles (2014) and individuals seeking to connect with others who share similar concerns.

Agile frameworks champion the multidisciplinary team, which is a great way to focus people on a specific outcome. This is different from the traditional role-based team setup or functional silo, where people who perform the same role sit together. The downside to the multidisciplinary team is that people who perform the same role in different teams have less interaction with each other. The community of practice brings these people back together to regain the benefits of regular contact while keeping the value of the multidisciplinary team.

The message I hear from organisations who don't have communities of practice in place is that work is often duplicated and ▶

KEY TAKEAWAYS

Communities of practice help connect people in organisations that are scaling their agile delivery.

Communities of practice can support people, build capability, reduce the duplication of work, and build better practices.

A mature community will benefit members and the organization.

Communities of practice take time and effort to create.

A community goes through a number of stages as it develops and a self-sustaining community has the best chance of survival.

people are not benefitting from each other's experience. People in organisations without communities of practice don't know who else in the organisation performs the same role as they do and they lack the support they need.

This is what makes communities of practice a vital part of an agile organisation.

I have been helping organisations in the public and private sector to establish communities of practice and moving existing communities to the next level.

Benefits

There are many benefits to having a community of practice, both for the individuals involved and for the organisation in which they work.

A community of practice is a support network for people who share a common role. It will create opportunities for learning, building capability, sharing knowledge, and reducing duplication of work. A mature community of practice will take ownership of its area of interest and create value for its members and for its organisation.

Creating a support network for members

What underpin any organisation are the people within it. People naturally want to connect with other people to find support — this is as true inside the workplace as it is outside of it. This is why communities of practice and small groups tend to form without any encouragement from the organisation.

People who don't feel supported at work quickly lose motivation and may leave. Even if they stay, they are not giving their best to the organisation.

I have seen a number of examples of this where people who have moved from the role of project manager to agile team lead. Some senior managers assume this is a natural move and that an experienced project manager can become an agile team lead with little training or ongoing support — what I would call a "sink or swim" scenario. I've seen those agile team leads become frustrated, overwhelmed, and unhappy, which leads to a drop in team productivity that isn't good for them, for the team, or for the organisation.

On the other hand, I've seen people join organisations with strong communities of practice in place and how those communities help them come to grips with the organisations and excel in their roles.

The types of activities that can foster a supporting environment are regular face-to-face time with the community, buddy systems (with one or two other people), shadowing, observation opportunities, and mentoring schemes.

Accelerating professional development across the organisation

Organisations all too often consider professional development as an individual activity and promote it with personal training budgets or solo learning objectives. While it's useful to think about professional development for individuals, it's also valuable to think about the professional development of a group of people and the organisational capability it may bring.

We learn better when we learn together because we benefit from collaboration and building on top of each other's ideas.

People in groups are also able to learn from the successes and challenges of others in the group.

“Copying other people’s successes, when combined with individual learning, is dramatically better than individual learning alone”.
— Alex “Sandy” Pentland

To boost the value of collaboration, create a safe environment in which people are free to ask questions and try out new ideas. It helps to validate ideas and accelerates learning.

Organisations that invest in communities of practice can get a greater return on investment than those that only invest in individuals because the groups will contribute to members’ professional development.

Individuals gain access to more learning opportunities and will discover things that they hadn’t yet realised they needed to know.

I’ve seen real value when members of communities of practice spend a portion of their time together on learning activities. These can include presentations by internal and external speakers, practicing new skills in a safe environment, games and workshops, exercises to embed learning, and visits to other organisations with similar challenges.

Breaking down organisational silos

Most organisations have silos. Silos can form when a group of people feel a deeper loyalty to each other than to other groups of people. They can form naturally or arise as a result of an organisation’s structure. Silos can spring up between organisations or within organisations and between functions, departments, programmes, teams, and other groups. Silos can hamper communication, causing duplication of work and frustration for those isolated inside them. They can damage an organisation. You can recognise silo behaviour when people stop referring to others by name and start referring to them as an activity — for example, “procurement are slowing us down”.

Supporting cross-team communities of practice will go a long way toward breaking down those barriers and fostering better appreciation for how others work. This can improve communication, reduce duplication, increase knowledge sharing across the organisation, and make it easier for work to flow.

Sharing knowledge and building better practice

When a community starts to meet, members will start to share stories and challenges. This is a great place for them to start building trust, which is crucial for a community of practice to thrive.

Sharing stories has a number of advantages for members and the organisation. It will reduce duplications of effort within the organisation as people discover the similar things they are working on. It will improve problem solving as a wider range of people will be working to solve each issue. It will lead to better working practices for the organisation.

As the community matures and members continue to share, they will identify common issues and become motivated to fix them. This often leads to valuable outputs with tangible benefits to community members and the organisation.

But this is only possible if the community is empowered to make ▶

Supporting cross-team communities of practice will go a long way toward breaking down silos and fostering better appreciation for how others work. This can improve communication, reduce duplication, increase knowledge sharing across the organisation and make it easier for work to flow.

APPDYNAMICS

Find & fix app performance issues, faster.

Discover how APM keeps your customers happier.

Take a Tour at **AppDynamics.com**

changes and if the members are allowed to own its vision and goals.

Positive examples I have seen are designer communities creating design patterns and assets for others to use, product owners creating courses for other parts of the organisation, developers creating code tests for interviews and even agile practitioners creating an agile transformation strategy for the organisation.

Hiring and building capability

If you have established communities of practice, bringing new skills into an organisation becomes easier.

When a community of practice is formed of people who share a particular role, it can help the organisation by being responsible for hiring staff for that role and developing new employees once they arrive. Having a community of practice in place is attractive to potential hires. Knowing that you are joining a ready-made support network is an additional bonus in a new job.

If the community has an identity and a set of values, those values can help identify who is a good fit for the role. If the community can articulate skills gaps and development needs, they can look for a person with the skills to fill those gaps. The community can then support the new hires once they arrive.

Whilst working with a London agency, I worked with the agile team leads' community of practice to collectively agree on the skills needed for the role. They then used this set of skills to refine the existing job description, to help members set objectives and to create career paths for that role. This meant that community members agreed on what good professional development looked like for that role, rather than having it defined for them by someone else, leading to greater buy-in.

Other communities I know of have created learning and development frameworks for members and taken ownership of training budgets to support people through it.

Starting out

Communities often emerge from a need. When people need support, they often reach out to other people like them who they can connect with. A person with a particular unique role on a team or with a particular challenges that no one near them shares is more likely to reach out in this manner.

If you are looking for a place to start communities of practice, find these small pockets in your organisation. If these pockets don't exist, cultural or organisational challenges may be inhibiting them.

I have encountered organisations where time sheets and utilisation rates prevent staff from explicitly spending time on learning activities. I've worked with organisations with members who are geographically distributed, preventing them from meeting regularly. I've experienced lack of buy-in from senior or line managers, so members aren't encouraged or empowered to spend time on community activities. You will eventually need to address these barriers for the community to succeed — although with enough motivation, communities can start up despite these challenges.

Starting a community before dealing with challenges can be a great way to establish momentum and prove its value before seeking buy-in from an organisation — which in turn will help remove organisational barriers. This is a path that I have seen succeed.

A large organisation I spoke to used this approach, which eventually led to a mature community ecosystem. One of the first communities of practice that the staff formed was the NoSQL community as a number of groups were already looking at that challenge. They took the opportunity to unite isolated groups that were working on the same problem in order to learn from each other. The community intentionally stayed under the radar so that it could be a safe-to-fail experiment, one that proved successful. They repeated the experiment with two more successful communities before pitching the concept to the CIO, who approved of the idea, leading to more formalised communities of practice.

Meeting regularly

I usually recommend that a new community's members try to meet weekly. Meeting every week more quickly builds trust and allows any member who misses a session to easily catch up the following week. I also recommend that members meet face to face; where this isn't always possible, members should aim for some initial face-to-face time and follow up with digital communication.

Just getting people together to talk is a great place to start. Look for enthusiastic people as they will get others enthused too. As the members start to form into a community, they will start to work out what they need.

Create a community charter

In the early days of the community, it can be useful to create a charter that covers why the community exists, who the members are, the community's vision and goals, and principles for how it will run.

I recently worked with the business-analyst community of practice at the Department for Work and Pensions (DWP) to help members with the community's direction while they were forming. I ran them through some workshops to give them the chance to think about where they were and question what they should do next. This motivated them to define their community vision, which became

"Create a community for business analysts to enable them to engage, support, and share working practices within a safe environment."

The goals they have are:

- define the business-analyst role and objectives;
- share knowledge, tools, and techniques;
- connect with other business analysts;
- provide the opportunity to interact with a larger community in a safe environment; and
- hold meaningful engagement events in a range of formats to meet the needs of the community.

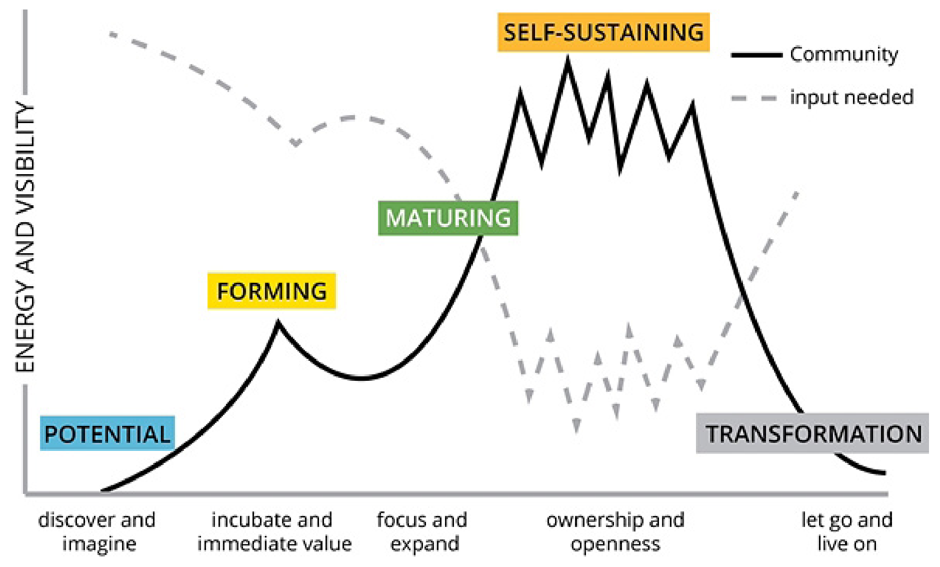
We worked on turning these goals into a backlog on which they work together as a community.

Let the community evolve

It's important that the community meets the needs of its members, so what it does together needs to come from within the community. Organisations that want to support the growth of communities should offer them support in the form of time, money, and people — but let the members decide how it grows.

Community maturity stages

A community takes time and dedication to grow and passes through a number of stages during its lifetime. At each stage, ▶



Graph adapted from Cultivating Communities of Practice: Wenger, McDermott and Snyder (2002)

it has different needs and energy levels.

The stages of a community of practice are: potential, forming, maturing, self-sustaining, and transformation.

As a community moves from potential into forming, members will start building trust and exploring opportunities, and there will be an increase of energy. As a community progresses through the stages, energy levels change and the initial dip after the excitement of forming stage is normal and not something to be disheartened about. When the community moves into maturing, it will grow in membership, commitment, and depth of knowledge that members share. The community will start to form strong bonds and trust and create real value for its members and the organisation. The final active stage for a community is self-sustaining, where it will have its own momentum and be an integral part of the organisation.

There are times when a community transforms to become something else and others when it never makes it past the forming stage. Many factors can contribute to this. In order for the community to thrive, it needs the right safe environment, the right leadership, the ability to meet regularly, and support from the organisation.

Communities of practice only exist as long as the members are interested in maintaining it. This is why the practice of regularly inspecting how the well the community is meeting needs — and adapting it to ensure it does — is crucial to a community's survival.

A community will best survive when it becomes self-sustaining. Self-organisation is supported by a clear understanding of the community's goals and by a community's autonomy to achieve those goals in the way that best fits its members. On regular occasions, the community will need to revisit its vision to check that it is still relevant. It will need to review its goals as they are achieved or to verify that they are still appropriate.

The key elements of a self-sustaining community are:

- good leadership and clear vision and goals,
- engaged members,
- sharing of knowledge and practices,
- support of skills development, and
- visibility and support from the organisation.

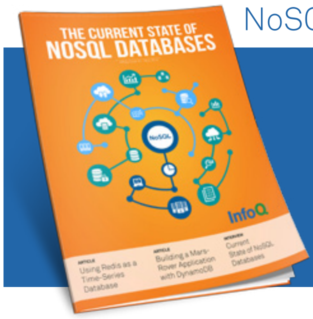
Communities of practice are an essential part of any agile organisation. Supporting the creation of healthy communities will benefit an organisation and community members in many ways. They can support organisational learning, accelerate the personal development of members, improve knowledge management, improve communication, build better practices, break down silos, facilitate staff hiring and retention, and create value for the organisation.

Are communities of practice the missing piece of your agile organisation? ■

AN INTRODUCTION TO
MACHINE LEARNING
InfoQ
ARTICLE Introduction to Machine Learning with Python
ARTICLE Real-World, Man-Machine Algorithms
BOOK REVIEW Andrew McAfee and Erik Brynjolfsson's "The Second Machine Age"

GETTING A HANDLE ON
DATA SCIENCE

REACTIVE PROGRAMMING IN JAVA
InfoQ


THE CURRENT STATE OF
NOSQL DATABASES
InfoQ