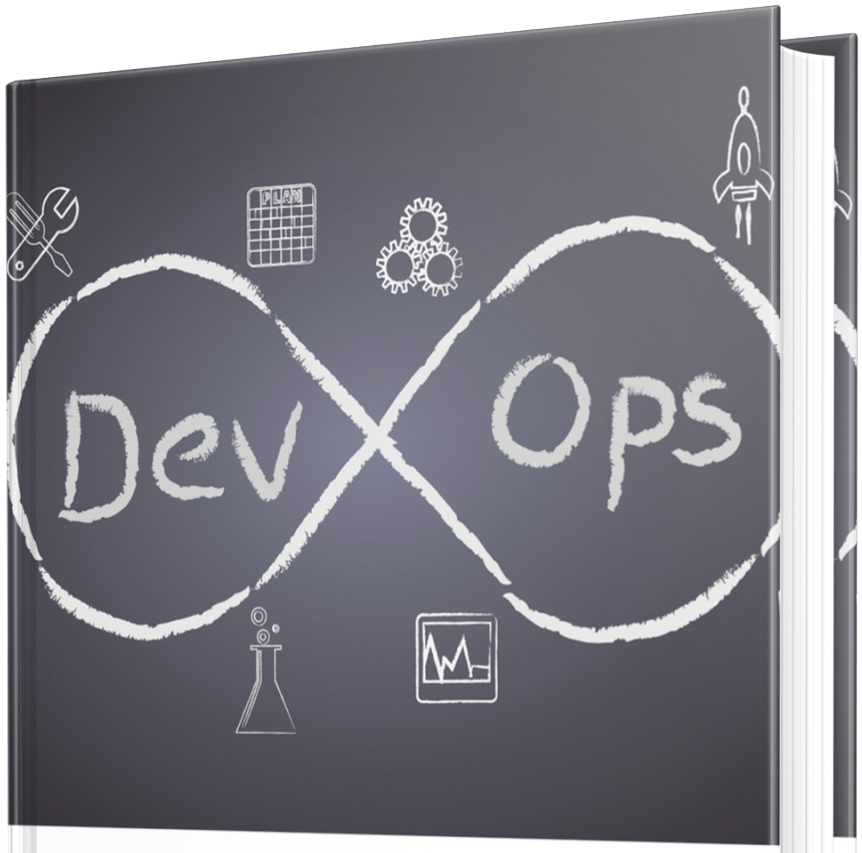


DEVOPS REVEALED

INTERNATIONAL DEVOPS CERTIFICATION ACADEMY™

THE ONLY BOOK, YOU CAN SIMPLY LEARN DEVOPS
3RD EDITION

DEVOPS REVEALED

TRAINING BOOK

THIRD EDITION

BY INTERNATIONAL DEVOPS CERTIFICATION ACADEMY™

WWW.DEVOPS-CERTIFICATION.ORG

Dedication 

To all of the International DevOps Certification Academy™ students, thank you for inspiring us, keeping us focused, and making sure we do our best to help you grow in your career with your DevOps skills and knowhow. Without you, your engagement and your loyal support, International DevOps Certification Academy™ could not come where it is today.

TABLE OF CONTENTS CLICKABLE

WELCOME

Hi! I'm Jenny.

I love that you are taking your time to read your DevOps book. I want to briefly share with you why we wanted to write this book for you and how you can get the best use out of it.

Within the context of our Official DevOps certification programs we made a thorough research in DevOps education space.

The conclusion was: We failed to find one single textbook, we could sincerely recommend to our students!

We talked to our successful students and found out that, almost none of the DevOps books in the market could really help them make a smooth entry to DevOps Methodology. Significant number of DevOps books in the marketplace claim that they cover all details of DevOps, but what they are not telling is that, they don't have understandable, clear and logical content to help their readers comprehend and most importantly love DevOps!

Therefore, we wrote for you **DevOps Revealed** and brought it for your service!

We are absolutely confident that DevOps Revealed will make you proficient in DevOps Methodology, so that you will have an outstanding opportunity to love DevOps and keep on taking the tangible benefits of being a DevOps professional.

Take some coffee to enjoy and some paper to take your notes, and spend some quiet time to read your DevOps book!

Afterwards you will have a great understanding about DevOps framework and be prepared to pass your Official DevOps certification exam. You will be ready to deliver great products and services to your clients and employers and to build your bright career and future!

Jenny Evans
Chief Operations Lead
International DevOps Certification Academy™

ABOUT INTERNATIONAL DEVOPS CERTIFICATION ACADEMY™

Have you ever wondered why your IT department falls often short of its goals, **your organization is dissatisfied with the performance of your IT, resulting in frustrated customers and unhappy IT professionals who feel powerless?** This challenge you and most of IT departments face perfectly illustrates why DevOps had to emerge. **DevOps is a methodology and as well as a culture to develop and deliver software.** DevOps enables you and your organization to develop great products and services that your customers love.

International DevOps Certification Academy™ is an independent Institute which helps IT Organizations and Professionals get accredited with worldwide renowned and recognized Official DevOps Certifications and prove their competence in DevOps domain. We empower DevOps Professionals worldwide to build their Careers, and Companies to become High Performers in their industries by engineering Outstanding Products and Services.

Your Official Certified DevOps Generalist™, DevOps Executive™, DevOps Project Manager™, DevOps Product Owner™, DevOps Architect™, DevOps Developer™, DevOps Operations Engineer™, DevOps Quality Assurance Engineer™, DevOps Information Security Engineer™, DevOps Release Manager™, DevOps Trainer™ and DevOps Coach™ **Certification Programs have proven their worldwide Acceptance and Reputation by being the choice of more than 961'000 DevOps Professionals in 143 Countries.**

Your Official DevOps certifications resonate and work very well in the market, and they create value for its students which is at least as important as having an official certification. The best proof for this global recognition and acceptance is that: **Every single day we receive success stories from our students who found new jobs or secured promotions. Beside their focus and willingness to succeed, these women and men demonstrate confidence with DevOps skills and knowhow we have been helping them to learn.**

DevOps is an open Software Development and Delivery Framework, and yet before International DevOps Certification Academy™ was

founded for you, there has been no reasonable way for DevOps Professionals like yourself to obtain DevOps Certifications and to prove your competence in DevOps domain. DevOps Professionals had to pay expensive fees for the one way profit-driven DevOps Certification Programs of other Certification Entities.

International DevOps Certification Academy™ aims to remove these barriers set in front of the DevOps Professionals in developed and emerging markets by saving them from paying unreasonable fees for DevOps Classroom Trainings and DevOps Certification Examinations before they certify their knowhow in DevOps. Moreover, feel free to check out "What makes Your DevOps Certifications Best of the Industry?" section on our **www.devops-certification.org** web portal to read why we perform and serve you far more better than our competition.

IT Professionals all over the world are passionate about DevOps. However, **DevOps is not another hype which will soon fade away. DevOps is a natural and logical continuation of agility** to manage your work beyond the goal of "potentially shippable code", extending it to have your software in "deployable state" in production-like environments, or even further in "deployed state" in your production environments.

DevOps is the process to build their engineering horse powers for companies like Google, Amazon, Apple, Netflix, Facebook, Etsy and many others. Every single day hundreds of loosely coupled, but tightly collaborative teams at these companies perform thousands of production deployments. And they do these deployments as you enjoy their services. **We want you and your organization to become a high performer like these companies and we want you to succeed!** Bear in your mind that your only sustainable competitive advantage is your ability to learn and execute faster than your competition. **And here we are to help you learn and execute DevOps!..**

International DevOps Certification Academy™ provides 12 major Official Online DevOps Certification Programs which are designed by our consortium of renowned Business and People Leaders, DevOps Coaches, Mentors, Experts and Authorities from all major Industries. You can check your DevOps Certification Programs from this **List of Official DevOps Certifications**.

WHAT IS DEVOPS?

DevOps is a process to develop, deliver and operate Software. As simple as that you shouldn't really underestimate understanding what DevOps really is. There are many DevOps definitions out there all over bookstore shelves and in the Internet which are either self-serving or at best missing and confusing.

Some commercially-driven definitions of DevOps tightly couple DevOps to some certain build and delivery tools or cloud infrastructure platforms. Although these tools and platforms could be really helpful to accomplish your IT and organizational goals with DevOps, you can't really plug-in yet another tool (which you usually don't have much control over it because you haven't written it) or you can't migrate your software applications to cloud computing platforms and then announce that you and your organization now deliver with DevOps. As some old but little known saying goes ravens would laugh at this.

Another misleading, but still better variant to define DevOps is to see DevOps as an intersection of work and people in an IT organization which brings software developers, software testers and software production operations engineers together. Having said that, once you reach to the end of this book, you can't help but see how dry this definition is.

The best approach to define DevOps is to resemble DevOps in iterative agile software development and process improvement frameworks such as Scrum, Lean, ITIL, IDEAL (Initiate, Define, Establish, Act and Learn) and Six Sigma DMAIC (Define, Measure, Analyze, Improve and Control).

Although Agile Scrum, Lean, ITIL, IDEAL and Six Sigma DMAIC can serve as efficient enablers for DevOps, DevOps isn't meant to be seen as an improved and combined superset of these methodologies and techniques. The very simple reasoning behind this fact is that none of these methodologies and frameworks except Agile Scrum have been introduced to specifically solve the problems and serve for software industry.

Many DevOps practices and principles have been derived from Lean Movement after they have been adapted, experimented, validated and fine-tuned for software development, delivery and operations. Moreover, DevOps borrowed and adapted many

techniques and philosophies from Continuous Delivery Movement, Toyota Improvement Kata, Theory of Constraints, Agile Manifesto and Agile Infrastructure.

Lean Movement:
Lean movement can be summarized by seven principles:

1. You see the whole and you create customer value with systems thinking.
2. You build the quality in, you create flow and pull (instead of push).
3. You deliver as fast as possible (short lead times to convert raw materials into finished products, or in software terms to convert ideas into running benefits and features in production systems).
4. You amplify learning and you embrace scientific thinking.
5. You empower your team, you lead with humility and you respect every individual.
6. You eliminate waste (tools, systems, most importantly your time you may spend for unproductive activities).
7. You decide as late as possible (You give just in time decisions).

Continuous Delivery Movement:
You enable continuous delivery through the deployment pipeline. Deployment pipeline has three important missions: visibility, feedback, and continuous deployment.

- **Visibility -** You make all aspects of the delivery system including building, deploying, testing, and releasing visible to every member of your team and other teams, so you promote collaboration in your team, and you offer transparency to your clients and stakeholders.
- **Feedback -** You and your team members learn about problems as soon as they occur so that you're able to quickly fix them before you put another brick onto an already collapsed block.
- **Continuous Deployment -** You deploy and release any version of the software to any environment including test and production environments through a fully automated process.

Toyota Improvement Kata:
You use Toyota improvement kata as a routine to move from the current situation to a new situation in a creative, directed, meaningful way. It's based on a four-part model:

1. In your consideration of a greater vision or improved direction…
2. You grasp the current conditions.
3. Then you define the next target conditions.
4. Ultimately, you continuously and iteratively attempt to move toward that target condition. Along this route you uncover obstacles that need to be worked on and you sort them out.

In contrast to other improvement approaches that you attempted to predict the path and focus on implementation, with Toyota improvement kata you learn and build on discovery that occurs along the way. You and your teams use the Toyota improvement kata to learn as you strive to reach a target condition, and you adapt based on what you're learning.

Theory of Constraints:

In an organization including yours which delivers any kind of software, there is always an ongoing conflict between your IT and your business. Quickly responding business needs whereas still ensuring stable and predictable production environments is a continuous tradeoff faced by your IT. Assuming that you're already fully aware that there isn't no risk production release, in a complex system like yours you can't be quick enough if you want to ensure a low risk production release. And you can't ensure a low risk release if you want to quickly release your code to your production systems.

You cope with this conflict when:

- You divide your IT organization into many small, autonomous, independent, self-sufficient and highly collaborative teams (You should start to see your teams as "intelligent units" with mission).
- You reduce your batch size (the size of your work in progress) given to each team.
- You reduce your lead time (time required to convert ideas and requirements into running benefits and features in production systems).
- You build faster, reliable and continuous feedback loops to ensure readiness for deployment.

Agile Manifesto:

With DevOps you adhere to the cornerstones defined by the agile manifesto:

- You respect individuals and interactions over processes and tools.

- You respect working software over comprehensive documentation.
- You respect customer collaboration over contract terms and negotiation.
- You respect responding to change over following a plan.

Agile Infrastructure:

Our organizations large and small including probably yours are relying on hybrid cloud infrastructures, combining public and private cloud with dedicated servers. Therefore, you collect the rewards of colocation security in combination with the flexibility and scalability of public cloud.

Here's some of major drivers why an agile infrastructure is beneficial for DevOps:

- You pay for what you use with transparent and simple pricing.
- You can quickly provision and de-provision your test and production environments from your code base (IaC - Infrastructure as Code).
- You have architectural flexibility.
- You can easily expand other geographies.
- You probably have better security.
- You stay compliant without audits and additional costs.

CONCLUSION

In this section we explained you what DevOps is and what it isn't. For you we have also covered the major movements and principles which DevOps was derived from.

A small caveat: Like many agile practitioners have been unfortunately doing in their agile practices since years, you shouldn't stick to any stone-grained definition of DevOps to dictate how your particular organization, your product, your service and most importantly your team should use DevOps. There is no one-size-fits-all solution. Neither in agile nor in DevOps.

Never forget that it is all about learning, experimentation and adaptation. As DevOps will do a lot of good work for us, it is our role biggest duty as human minds to identify our correct version and tone of DevOps. This is what each and every successful DevOps organization including Google, Amazon, Facebook, Netflix and Apple have been doing.

WHAT ARE YOUR PROBLEMS IN IT WITHOUT DEVOPS?

In your world of IT without DevOps, development and operations teams do usually conflict. End-to-end quality assurance of required client features and benefits first start after development team begins to work on a completely different project. Information security and production readiness audits are conducted the day before the planned production roll-out. Solving software problems at this critical project stage, especially design and security flaws that must have been already identified and rectified 8 months ago do not only yield challenges for the successful continuity of your IT, but also for the continuity of your business. Tired developers, uncooperative operations, testing and information security silos, demanding but desperate project and product managers, angry executives and frustrated clients and IT stakeholders do not really add much added value.

Too many hands-off of activities and dependencies in every possible imaginable direction, countless uncoordinated, manual, nontransparent work in your IT department is just another day in business.

Despite of long lead times to get any work done, there is no better option available, at least not for now, and you know that you still need this job to earn money.

Finally the big day comes. After a problematic and chaotic production deployment which caused significant downtime at your business, you wonder if your deployment broke anything in the existing production functionality. You start thinking of the lack of clarity and tangible evidence about what really have happened and what have been really impacted in your production systems.

And yet Go-Live celebration already starts in the corridors of your large IT room. It's also time for you to have a drink and mentally recover from the tough and sleepless night you left behind. What you broke last night in your production will be anyway the problem of somebody else. This is not a big deal...

Having said that you must be now wondering how it's possible at all that these tough projects can finish in one or another way although they once have been really in an ugly and desperate shape. The answer lies in **The Infinite Monkey Theorem**. In a nutshell, the infinite monkey theorem states that a monkey

hitting keys at random on a typewriter keyboard for an infinite amount of time will surely type a given text, such as the complete works of William Shakespeare. This is no joke.

Similar to this, beyond any predictability about when the work can finish, provided that your IT team sufficiently delays and gains time for a given project, it will surely deliver the project.

Chronic Conflict Between Development and Operations

Among many other key responsibilities, there are two major missions in your IT organization:

1. Your development team has to quickly deliver to respond changing demands from your clients and from the market your business serves for.
2. Your operations team has to guarantee stable, predictable, secure and continuous service to serve your clients and market.

Given that your IT organization is structured in this way, your development and operations teams have conflicting missions and motivations. This conflict reduces productivity, quality of service, client satisfaction and outcomes of your IT department and finally your overall business performance.

Now You're in Technical Debt

Continuous firefighting, workarounds and unplanned activities to save your day will only result in accumulation of more technical debt. This not only impacts the performance of your development and operations, but it also negatively impacts all other units including your architecture, information security, testing, product management, release management and your business stakeholders.

You can resemble technical debt in financial debt. The more financial debt you have, the less options you have got to reach your desired goals. Ultimately it becomes almost impossible to give correct decision to reach the desired outcomes and you would be more likely to make additional financial debt.

Technical debt has no difference. The more technical debt you have, the less options you have to build and deliver correct solutions. Finally, you end up building workarounds on the top of workarounds, house of cards on house of cards. And it is a matter of time before they all collapse. You already know this.

Your Business Earns From Most Fragile Systems

A non-surprising but concerning fact about your business is that your most critical and revenue-generating systems are most prone to errors, crashes and downtimes.

These systems are usually in the center of your business and you have yet to participate a meeting that they are not somehow part of discussions. They are the most critical systems impacted by almost all projects. Due to the their severity in your business, they deserve continuous work, urgent changes, unplanned fixes, and yet these systems are never tired of producing Priority-1 incidents.

You Promise Bigger If You Break One

As your IT organization constantly breaks promises and frustrates its stakeholders, your stakeholders become more demanding to compensate the past. You and your leaders have often say nothing, but "why not" unless there is a better job offer lined up.

Then your software development and delivery organizations are assigned a larger challenge than usual. As your critical systems and IT architecture are not even in a close proximity to cover these requirements in given "urgent" timelines, you set out designing your new workarounds on the top of your existing set of workarounds.

You Have No More Fun At Work

And finally everyone gets little bit busier. Your calendar have no more free space to perform an hour of uninterrupted work for the position you are hired and paid for. Emails rain to your inbox, so you are now using mostly unproductive time at your meetings to clear and scan your mostly unclear and pointless incoming emails. You are curious why there is no mandatory training at your organization to teach how to use and write emails.

You're usually dependent on other teams to be able to continue your own work. And yet the expectations you communicated a couple of weeks ago are not even in the radar of the other team yet because they just started their new "Horizon 2400" project while they had to solve 9 Priority-1 incidents caused by their fragile software applications.

You are slowly little bit impatient because you need some answers from your designated single point of

contact person at the other team to get your own work going.

You couldn't stop and sent another email to remind your request. This time you put her boss in CC and you thought you should have done this when you sent your request for the first time.

A couple of seconds haven't passed yet and two out-of-office notifications popped up on your screen. Your single point of contact at the other team will be at "Horizon 2400" workshops until the end of this week. And her boss is sick at home today.

CONCLUSION

In this chapter we explained the challenges most of businesses and IT professionals face. The downward spirals and vicious cycles almost every business and every professional live with.

Gartner estimates that companies worldwide waste yearly about 600 billion USD for non-budgeted and non-scheduled IT maintenance work to keep revenue-generating IT systems up and running. Just to express this number with digits to see how it looks like: $600,000,000,000.-

As it might have been clear for you until this point, this level of waste in a highly cognitive profession such as information technology is not trivial to comprehend.

This is a good challenge to tackle with. DevOps have some answers for some companies and may be for your company too. Of course, if your company is ready to experiment, learn and adapt.

HOW DOES DEVOPS SOLVE YOUR PROBLEMS IN IT?

You Continuously Deploy

Companies from fresh start-ups to Fortune 100 and any other size in between, companies with any type of software product and service portfolios have already proven that: With DevOps your independent, small and self-sufficient teams deliver tens of production deployments each and every day. You design, build, deliver and test on production-like environments. You no longer release your code to your production systems in every third month after midnight while everyone else is sleeping. You release your code many times every day, and most importantly during your typical working hours, while your clients and market you are serving for still enjoy and get the benefits from your software.

DevOps does not only enable you to frequently deliver tiny software features, but also with the help from DevOps Dark Launch techniques you can frequently deliver small pieces of high-volume giant software features to your production. These small pieces wait in dormant (inactive) mode in your production until you switch the giant feature on from a configuration setting. Even only this one small technique can significantly improve the life quality of your team by avoiding big bang code roll-outs and their unpredictable adverse effects in your existing production systems, chaos and finger-pointing in your organization.

You Organize Your Teams around Your Mission

To serve your market, as you and your organization have a bigger mission than your competitors do, you organize your IT department with long-term teams around your client-serving (either internal or external clients) products, services, capabilities and micro-services. Instead of temporary project teams which dissolve as soon as or even before projects are done.

With the model of temporary project teams, individual project members neither have proper ownership of their own contributions because they rarely know where their project stands in the big mission of their organization nor they properly receive feedback about their work. Their work is constantly pushed to them in their area of expertise (database developer, front-end designer, translator, etc.) from various different projects, and they live in a world of continuous interruptions.

With the model of long-term teams, your teams willingly have the ownership of their work, IT and business performance enabled by their throughput. They are totally aware where their contributions stand and how important they are in the big mission of their organization.

You Build Systems to Achieve Business Goals

You and your teams are fully aware that you are getting paid to serve your clients and to accomplish the mission of your organization. Therefore, you are respectful to everyone's time and resources from your organization and clients. Before you commit any long-term project, you test drive and validate proposed and envisioned business value you expect from your IT solution. You show your clients the models, prototypes, various A/B variations or lightweight versions of what you intend to build. Then you measure and record their reactions and feedback to understand how you fine-tune your product and service, and if your clients like it at all.

You don't add features to your product just because you have a good feeling about them. You test your ideas and build tangible, demonstrable and reproducible evidence that what you are building and investing the resources from your organization will most probably add the bottomline and positively differentiate your employer from its competitors.

You Create with Built-in Quality and Monitoring

You build fast, reliable and amplified feedback loops in all stages of your software delivery and operations lifecycle. Because quality is not a monopoly which belongs to a certain team, everyone strives for built-in quality. In order to make sure you have correctly done your job, you don't wait for feedback from another team. Or you don't ask somebody else's permission to deploy your code in production. You trust your team with peer reviews of your design, code, test and infrastructure.

You always build test automation and monitoring (telemetry) for every possibly measurable and testable feature. You are conscious that if a feature deserves your time to be coded and delivered, it also deserves continuous, reliable, fast and consistent testing with test automation. Moreover, it also deserves continuous monitoring and built-in analytics in your software to validate what you win from this feature matches why you built it in the first

place. In all environments including production and non-production.

Every check-in in your code repository automatically adapts, restructures or if necessary rebuilds your operational environments, automatically rebuilds impacted applications and ultimately executes all automated tests to validate existing features and the purpose of your latest check-in. Once validation is successful, the same changes are automatically adapted to your production.

Your built-in analytics and telemetry in your applications continuously monitor and record key events in your software and in its operational environments. The key metrics (such as number of orders, number of log-ins, CPU usage, RAM usage, CPU load, number of errors, length of database queries and many others) are continuously recorded and presented in real-time, so it is a matter of minutes, if not seconds before your team discovers a negative impact triggered by a deployment. This is why fast feedback loops are vital to get your job done fast.

Thanks to Errors You Learn and Teach

If an error occurs, you solve it calm and confidently. You identify root cause of an incident by relying on scientific evidence obtained and observed from your automated tests and telemetry. If you identify a missing test case or missing telemetry after the resolution of the issue, you add them to your code repository together with your fix.

In a complex system like yours, no single person can know and foresee everything. Therefore, everyone in your organization believe that errors are not bad surprises, but they are part of maturity process of your products and services. Errors are opportunities to learn and grow. You and your organization conceive errors as a mean of new personal and organizational learnings.

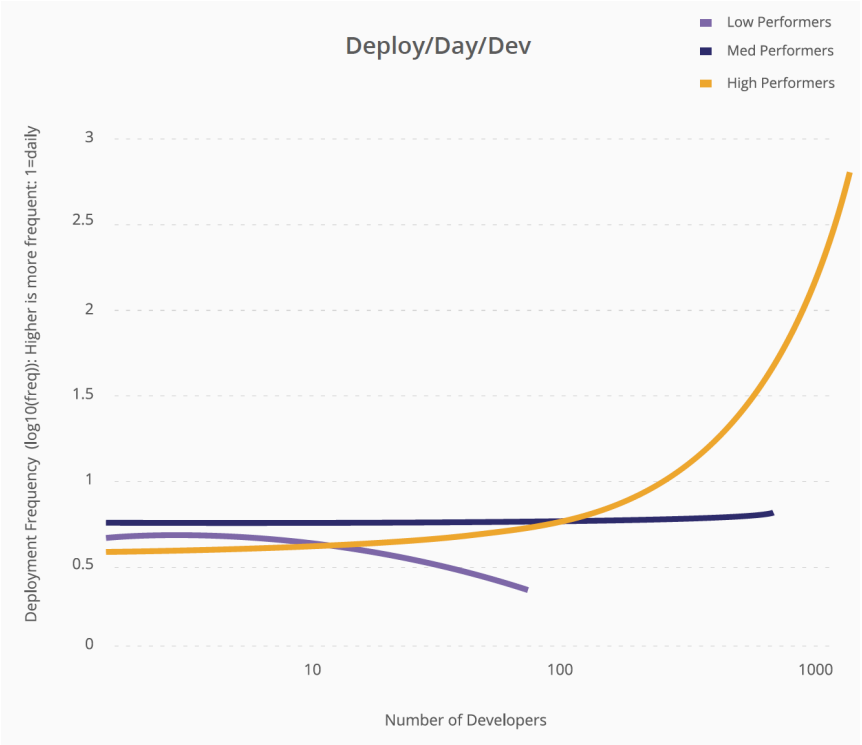
Blameless post-mortems of errors will also enable you to even better identify root causes of incidents and discover techniques to avoid same and similar errors in the future. Thanks to your incident, you now convert your local learnings from your team and software into organizational memory which will serve everyone in your organization.

You and your team are never punished because of this error, but you are awarded due to the lessons you learnt and shared after this incident. Thanks to your contribution, many teams in your organization know how to avoid such an incident in the future.

Your Teams Scale Even Better while Your Business Gets Bigger

Now imagine an average organization which evolves from 1 to 1,000 developers. As the organization was small, developers were continuously building and deploying their code to production. However, as the organization gets bigger, there is now a much larger organizational and bureaucratic hierarchy, more fear from mistakes. There is now more coordination and communication effort required to deploy your code in production. Therefore, average number of code deployments per day per developer usually reduces and at best remains same while organizations grow.

However, DevOps high performers play in a different league and they manage to increase average number of code deployments per day per developer as they have more developers. With correct software architecture, correct teams, correct mission, correct leadership and correct culture, DevOps high performers such as Google, Amazon, Facebook, Netflix and Apple have already proven that this is possible. Your organization can be the next one to become one of these DevOps High Performers.



Number of Code Deployments per Day per Developer (Source: DevOps Handbook)

CONCLUSION

High Performer DevOps organizations outcompete their rivals in following dimensions.

- 46 times more frequent code deployments.
- 440 times faster lead time from commit to deploy.
- 96 times faster mean time to recover from downtime.
- 5 times lower change failure rate (meaning changes are 1/5 as likely to fail).
- Improved productivity and throughput.
- Improved code and operational reliability.
- Improved organizational performance and client satisfaction.
- Improved market penetration, market share and profitability of organization (2 times more likely to accomplish).
- Improved market capitalisation growth.

IT performance metrics	2016	2017
Deployment frequency	200x more frequent	46x more frequent
Lead time for changes	2,555x faster	440x faster
Mean time to recover (MTTR)	24x faster	96x faster
Change failure rate	3x lower (1/3 as likely)	5x lower (1/5 as likely)

DevOps High Performers IT Performance Metrics (Source: P-Labs)

HOW SHOULD YOU START DEVOPS IN YOUR ORGANIZATION?

How you start your DevOps journey in your organization will dictate how successful your DevOps initiative will become. Therefore, let's get the best out of your first and probably one single shot.

You Avoid Big Bang and Start Small

As you already know you can't really shake the status quo in one go and announce an organizational DevOps transformation on day one. You should first identify teams which continuously tackle with issues and unmet goals. The teams which are definitely conscious about the necessity of improvements and who are the most receptive to consider a different way of working. And teams which are unconditionally willing to serve their organization and clients.

Demonstrate Quick and Early Wins

Your mission is to demonstrate quick and early wins from your DevOps initiative and prove to skeptics and naysayers the problems you solve and the value you create with DevOps. And then you will step-by-step replicate these improvements in the other areas of your organization.

This is pretty much how a leader ERP, CRM and SaaS manufacturer from Germany set out its DevOps journey. Their maintenance team was doing one single delivery per year which was quickly resulting in an accumulation of non-released code in various different incident code branches. This made for them extremely difficult to merge their code and to release fully tested and reliable maintenance releases.

Their maintenance development team was first combined with maintenance delivery planning, quality assurance and operational environments management teams. They also started using a joint task and ticket management system to simplify coordination and communication efforts. Then they set out monthly and shortly after two-weekly releases.

Within the first year of this new practice in maintenance group, the team managed to resolve 6 times more incidents compared to the year before. As important as that, the ratio of reopened incidents and associated rework reduced from 39% to 4%.

Having had this significant improvement and demonstrable improvement metrics in the pocket, a small DevOps initiative won major support and credibility in the organization and executive sponsorship for the further expansion to other product and service teams.

Identify an Appropriate Greenfield or BrownField Product and Service

Another consideration to get started with your DevOps initiative is whether you should choose a Greenfield or Brownfield product and service.

A greenfield product and service is new in your organization. Therefore, it is for your best interest to build a new, separate team which works in isolation, especially while you are getting started. Thanks to this isolation, your team members will have better chance to break the existing status quo and think out of the box. Furthermore, your team will be hopefully less frequently interrupted due to their former responsibilities in the organization. You task your new DevOps team with tangible IT and business performance metrics and outcomes such as improved sales, improved client engagement, or reduced time to deliver etc. As your greenfield DevOps initiative don't rely on legacy code and dependencies, it poses less risk for you and your team to start a successful DevOps initiative. However, your success will be less impactful in your organization as your DevOps initiative in greenfield will not prove whether DevOps can solve existing problems in your organization and whether your success can be replicated in the other areas as well.

On the other hand, you can start your DevOps journey with a brownfield product and service too. Many brownfield products and services do already run in your production systems, and they serve your business and clients. If you are from the same planet as we are, your brownfield software application is most likely to have significant technical debt and it doesn't have much reliable test automation. Due to this nature, starting your DevOps initiative with a brownfield product and service could be more challenging and yet positive throughput from a brownfield DevOps initiative will be more awarding, convincing and impactful at your organization.

DevOps is for Large Organizations with Lots of Running Legacy Systems too

Many people wrongly believe that DevOps is not tailored for brownfield products and services. Their wrong belief goes even further to state that DevOps is only for start-up companies. And yet some of the most exemplary DevOps initiatives started in companies with giant and mature IT organizations which have countless business-critical legacy systems and thousands of employees. Amazon, Easy and Facebook are a few examples to name. As described with the SaaS manufacturer case study above, you can initiate DevOps in an already existing organization with brownfield products and services too. In fact, because most brownfield products and services have major gap between their performance throughput and client expectations, they are usually the best candidates to take benefits from your DevOps transformation. For you a good piece of advice to get started with your brownfield DevOps initiative is: To begin with, invest your time to remove or at least reduce the technical debt and then build missing test automation for existing product features and benefits. Otherwise, your brownfield product and service will never allow you to do reliable production deployments in shorter cycles.

Identify an Appropriate System of Records or System of Engagement

During your first DevOps initiative, you can feel free to choose either a system of records or a system of engagement. Systems of records are centralized systems to manage single source of truth in your organization. They heavily master on management of your data including clients, transactions, security and privacy. Systems of engagement are mostly decentralized applications to enable and encourage your clients and users to interact with your organization via omni-channel responsive portals or mobile, tablet and desktop applications.

With your preference of a system of record, your advantage will be that you are going to demonstrate how your DevOps team competently handles time consuming and highly bureaucratic tasks such as compliance, security and privacy. You and your team accomplish this by generating scientific and tangible evidence and by using test automation and telemetry during the whole development and operations life cycle of your product and service. A typical example can be a migration program of your geographically distributed data from your on-premise database servers to cloud by ensuring that your single source

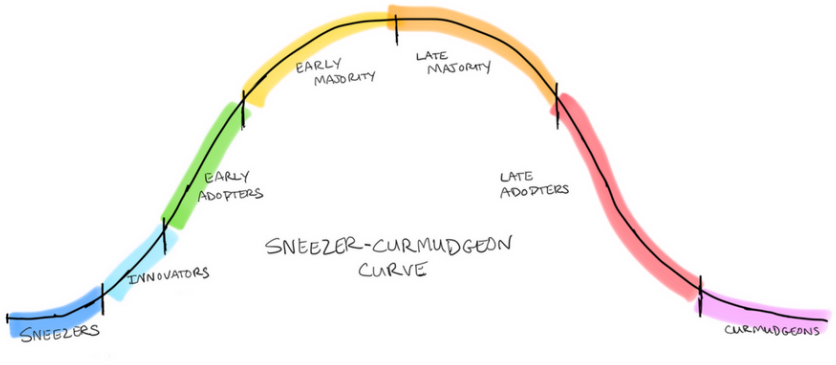
of truth still remains compliant to legal, industrial, local and international rules and regulations.

If you prefer a system of engagement, you are going to demonstrate how rapidly and reliably you innovate with quick delivery cycles. This is because most of the visible innovations which result in commercial successes today are originated from systems of engagement. A typical example can be a more intuitive and simplified check-out flow in your e-commerce portal which improves your sales 25%. The moment you accomplish this, your executives will not only adore DevOps, but they will adore you too.

Your Recipe to Successful DevOps Transformation

In the beginning of your DevOps journey, don't waste your time to win conservative teams for your DevOps initiative. Identify the most innovative, open-minded, well-connected and influential teams which believe in the necessity of DevOps methodology and principles. Start small with one or two teams. By using the tips provided above, identify the most appropriate greenfield or brownfield product and service... And appropriate system of records or system of engagement to get started.

After your first success stories these teams you have on board will be your biggest supporters to spread DevOps across your entire organization. Then you extend your coalition to other groups and you increase the number of success stories. These successes will create more and more social proof and trust for your DevOps initiative. Once they see the measurable benefits, you will have more support, sponsorship and attention from your management too.



DevOps Adoption Curve illustrated as Sneezer-Curmudgeon Curve (Source: write/repeat)

In the final stage you identify influential destroyers and strong naysayers (Curmudgeons). You can

probably beat them by using their own technique which is spreading around risk and fear. You explain them how bad their teams and products will appear and become the last weak chain in your organization if they deny learning and using DevOps.

CONCLUSION

It is not an easy task to drive any organizational IT transformation. DevOps is no exception. However, by starting small, you won't be taking much risk. On the other hand never forget that it is one of the most profound duties of any young or seasoned leader to be able to motivate and mobilize her or his team to act for the sake of improved business performance... And to take some calculated risks for the best professional satisfaction.

HOW SHOULD YOU BUILD YOUR DEVOPS ORGANIZATION AND DESIGN YOUR SOFTWARE ARCHITECTURE?

According to Conway's law, organizations which design systems are constrained to produce systems which are copies of their own communication structures. In other words, your software cannot do any better than how efficiently your teams communicate and interact. Therefore, how you structure your teams will surely impact your software architecture, IT and finally business performance as well.

How You Configure Your Organization Defines What Kind of Software You Get

Most companies, probably including your company too, compartmentalize their software delivery organizations in a number of teams, and they end up producing their software architected with the very same number of layers. The controlled experiments have also proven that when an organization of 6 teams was asked to build a software, their teams came up with an architecture of 6 layers. When another organization with 3 teams was asked to build the very same software, they came up with an architecture of 3 layers. In fact, there are many similar examples.

A renowned insurance company with 86,000 employees worldwide was coincidentally structured in 2 major functional teams for one of their IT organizations which delivers the backend to manage their customers, contracts, invoices and services. One of these functional teams was oriented in Java programming language and the other functional team was oriented in PL/SQL stored procedures. Furthermore, they were dependent on organization-wide and centralized IT operations, database admins, IT infrastructure, IT networking, IT security and IT quality assurance service desks which made their lives even more challenging to be able to safely deliver their software.

The other challenge of this organization was that: Although the teams were separate, in their architecture they couldn't competently separate business orchestration logic in Java from data access logic in PL/SQL. They delivered hundreds of features, each feature being a spaghetti of Java and PL/SQL codes. Therefore, every change and new feature

request required tens of handovers from one team to another, and they resulted in long waiting queues and lead times to get even minor changes done. Moreover, in this complex environment, as overall impact of even minor changes was very difficult to foresee, they continuously broke existing features in production and caused downtimes which also required long lead times, rework, escalations and stress to resolve.

To recover from this modus operandi, these two functional teams merged into one single product team. They gradually redesigned their software by converting their data access layer into a set of API functions. In addition, they built a new business system completely decoupled from the internal dynamics of their data access API. Even in its early stage, this initiative improved the team morale because both Java and PL/SQL experts started working for the success of their joint product team instead of motives of their past functional silos. As they built a loosely-coupled architecture, now the impact of changes are easier to identify, changes are easier and quicker to implement and defects are more straightforward to locate and fix. As a result, average lead time of new features reduced from 4 months to 3 weeks, and incident queue of the team is now almost empty, so they profit from this free capacity by further reviewing, refactoring and improving their codebase.

Matrix Organizations, Functional Teams and Illusion of Cost Optimization

Your companies are organized with a number of functional teams around various different subject matter expertises such as databases, networking, operations, information security, quality assurance, release management, project management, product management and so on. Project teams are built by people from different functional silos. In these matrix project organizations which attempt to deliver business products and services, project managers are usually responsible for getting something delivered, and product managers are responsible for ensuring that correct thing is being delivered.

Your problem in this organizational configuration is that functional teams have no to little understanding about the the extent of the work they contribute. In extreme but often typical cases, your functional teams neither care the big picture nor the overall IT and business throughput of the product and service they contribute. What they care is to make sure that

none of their doors are left open after projects will go nasty and everyone starts to finger-point.

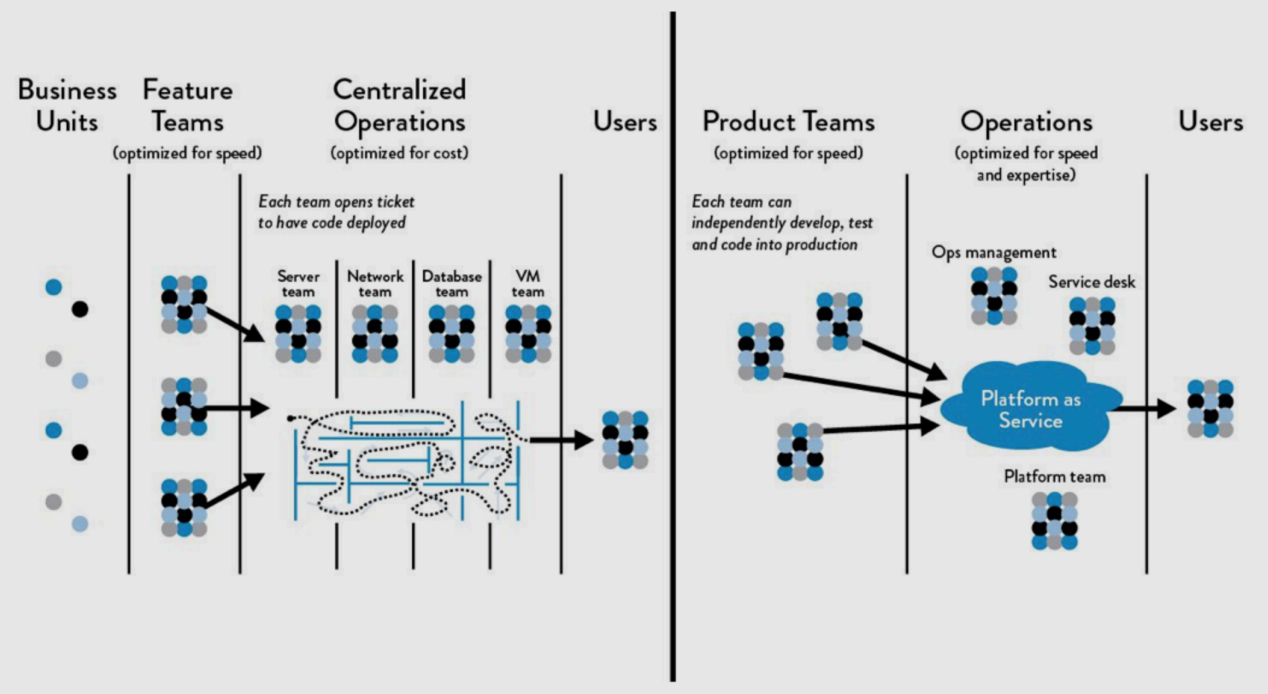
As your functional teams usually have to manage long queues of tickets, they usually require long lead times to support your project. Because projects fight for functional resources, escalations are the only way to get quick attention for your project. Escalations over escalations obviously pollute the working climate and trust between your teams.

Multiple handovers from one team to another, delays, quality issues, reworks, bottlenecks and stress are now part of your daily job. This is because your matrix organizations are not meant to do any better than that, as long they continue focusing on a opaque and fake illusion of cost optimization. In fact, due to quality issues, reworks and delays, functional organizations are probably even more expensive than any other random reorganization you can ever imagine. In addition to this, if your organization has already managed to outsource some of its vital functional skills such as quality assurance and IT operations to geographically remote locations, then your IT organization must be now barely surviving to safely deliver and fulfill demands which are critical for your business continuity.

This is indeed a Big Problem. How Does DevOps Solve This? Answer: With Product and Service Oriented Teams

In order to solve this problem, DevOps suggests you to switch gears from cost optimization illusion of functional teams to DevOps' valid and proven speed optimization. In fact, done correctly, DevOps will anyway enable you to save costs while you and your team quickly and continuously deliver.

For you DevOps' suggestion is product and service oriented teams. Product and service oriented teams are independent, cross-functional, autonomous, self-sufficient and usually small teams which are able to cover all phases of software engineering life cycle of a given product and service (but not a temporary project) including architecting, designing, coding, testing, experimenting, deploying, operating and maintaining software. Yes. You read it right. Who else can operate and maintain software better than their own creators? The answer is of course: No one. Therefore, it is no surprise that the most successful DevOps teams at Amazon and Netflix do not abandon their products and services once they release them. They are also in charge of operating and maintaining their software in their own production systems.



Functional Teams vs Product and Service Teams (Source: Lean Enterprise, DevOps Handbook)

Your Organization Does No Longer Finance Temporary Projects, But It Finances Products and Services

To make its business performance flourish, your organization does no longer finance temporary projects whose tangible business outcomes are not trivial to evaluate in long run. Now with DevOps, your organization finances its own mission, its own purpose and its own products and services associated with this mission and purpose. The success of your teams are now assessed and evaluated based on their IT and business performance. Based on return of investment within their particular business domains where they serve

their products, services and micro-services to their internal and/or external clients. Your teams now act like owners of products and services they create and provide, instead of merely being members of functional silos who don't pay much attention to business outcomes.

How Can You Build Your DevOps Teams and Software Architecture?

DevOps does not of course suggest you to break and reorganize all ongoing projects at your organization in one go. A non-disruptive, but still impactful way of adapting your teams for DevOps methodology is to inject functional experts into projects teams. Once your teams get functional experts in their desired domains such as quality assurance, information security, databases, networking, servers, operations and so on, then you can expect these teams to independently and self-sufficiently design, deploy and maintain their software products and services.

In these new product and service oriented DevOps teams, availability, quality, performance, information security and compliance are everyone's daily job. How can your teams build highly available, secure and high quality software applications which are able to cope with stress and load demanded by your clients if the teams never pay attention to these non-functional requirements until they finish their design and development? How good can external experts judge and validate the security and quality of your software applications without being involved at any software engineering stage of your products and services? This is why high performer DevOps teams rely on external subject matter experts only to get consultancy, but they still fully own all non-functional requirements at every stage of their software engineering lifecycle.

There is an Alternative You Can Consider

One exceptionally successful DevOps organization which still heavily uses functional teams in combination with product and service oriented DevOps teams is Google. At Google DevOps teams and functional teams see their organizational mission with their products and services as a shared goal. Functional teams build blueprints (such as Templates, Checklists, Application Skeletons, etc.) and self-service systems (one example: to independently provision and configure servers and networking) to make sure that they never become bottlenecks which slow down DevOps product and

service teams. Functional teams at Google collaborate and support product and service teams at early stages of software engineering life cycle, they constructively perform peer-reviews of designs, codes and tests, and they continuously provide consultancy to DevOps product and service teams.

How Big Should Be Your DevOps Team?

The ideal size for a DevOps team is 5 to 10 people. Such a limited team size reduces complexity of communication and alignment within your team. Furthermore, your team lead and team members do not spend and waste much time with errands and overhead. This also keeps the size of product and service your team is responsible for up to a certain limit which further reduces the complexity, maintenance and operations difficulty of software applications. Every team member in such small teams sees the big picture, and everyone collects little bit leadership experience by becoming part of a crucial mission for their organization. Your team lead works with upper management to understand goals and translate them to your team members.

What If Your Product and Service Can No Longer Be Handled by a Team of up to 10 People?

Then your solution is to spin out a new product and service, and to build another DevOps team which takes it over. Here you shouldn't conceive product and service concepts only as entities served and provided to external clients who pay for them. But also you can freely build internal products, services or so called "micro-service APIs" and their respective DevOps teams for your internal clients. For instance if your billing system becomes too big for a team up to 10 people, then you should spin out another DevOps team which takes over database access API. May be another one to take over online payment API. And then may be another one to take over batch processes. Of course, all these teams should be using a common code repository and a joint deployment pipeline to ensure continuous integration, quick delivery and success of their organizations.

As you already know in a tightly-coupled architecture, small changes in one application can eventually cause many adverse effects for numerous workflows. Therefore, products, services and micro-service APIs in your architecture must be loosely-coupled. Each DevOps team must be only responsible for one piece

of an loosely-coupled architecture. Each DevOps team can independently design, develop and deploy their software. Early alert mechanism built in the deployment pipeline should automatically and rapidly inform DevOps teams about potential adverse effects any code check-in causes.

How Can Such Small Teams up to 10 People Can Self-sufficiently Handle All Phases in Software Engineering Life Cycle from Architecture to Operations by themselves?

To manage this, you should encourage everyone in your team to become a generalist. You should encourage and enable them to continuously build new skills.

You should only hire team members who are eager to learn and grow regardless their effective level of knowhow and experience. You should strictly avoid people who expect to be evaluated in a fixed set of roles and responsibilities. You already know that neither your organization, nor your products and services remain fixed.

The demand will change tomorrow, if not today.

CONCLUSION

In this chapter, we explained suboptimal outcomes of matrix organizations, functional silos associated with them and DevOps' solution to address this.

DevOps' suggestion for you is to build product, service or micro-service API oriented small teams up to 10 people.

These DevOps teams should constitute generalist full-stack software engineers which are able to self-sufficiently cover all phases of software engineering life cycle from design to maintenance.

DevOps relies on loosely-coupled service oriented architecture (SOA) in which every DevOps team owns and operates one piece of your loosely-coupled architecture.

WHAT ARE THE ROLES IN YOUR DEVOPS ORGANIZATION?

DevOps Generalist

The main role of a DevOps Generalist is to ensure smooth establishment, efficient and healthy progress and continuous improvement of DevOps Practices in a DevOps organization and in its DevOps teams. Therefore, competence and perspective of every single employee in a DevOps organization to be able to act with DevOps teams is a fundamental factor which determines the success level and lifetime of DevOps organizations.

Whether you are part of a DevOps organization or you just collaborate and work together with other DevOps teams, it is profoundly important for you to have a clear understanding about how and what makes DevOps methodology far more successful, efficient and delightful to work with than other software development and delivery methodologies. Therefore, regardless you're an IT, software and technology practitioner, leader or manager or not, every professional at this current digitalization age (when software and everything around it are king) are highly recommended to be a DevOps Generalist.

DevOps Executive

DevOps Executives are responsible for successful utilization and application of DevOps knowhow within their organizations. They are key players for DevOps teams to enable cultural shift of doing and thinking in DevOps way. They have proven ability to constructively influence teams and management to get things done. DevOps Executives support the alignment of DevOps teams with business strategies, and they are responsible for identification, selection, scoping, prioritization and ultimately managing the penetration of DevOps into their organizations.

DevOps Executives select key members of their DevOps organizations, and they ensure all DevOps Professionals have been adequately trained, tasked and deployed to DevOps projects which best fit to their skill and experience levels. They work hard to coach and mentor DevOps teams, support resource planning, and they remove issues and obstacles to make the entire DevOps organization successful.

DevOps Executives report to executive management in terms of preset DevOps business mission objectives and identified business throughput metrics. They promote best practice solutions, achieved improvements, success stories and

leverage them towards their entire DevOps organization. They work very closely with all DevOps teams, but with particular emphasis on DevOps Project Managers, DevOps Product Owners, DevOps Release Managers, DevOps Trainers and DevOps Coaches due to their particular organizational mission they strive to accomplish.

DevOps Project Manager

DevOps Project Manager is the person responsible for accomplishing the stated project objectives. Key DevOps Project Manager responsibilities include creating clear and attainable project objectives, building the project requirements, and managing the constraints of the project management triangle, which are cost, schedule, scope, and quality.

DevOps Project Manager is often a client representative and has to determine and implement the exact needs of the client, based on knowledge of the firm he is representing. DevOps Project Manager is the bridging gap between the development/ delivery team and client. Therefore, DevOps Project Manager has a fair knowledge of the industry he is in, so that he is capable of understanding and discussing the problems with the delivery team and client. The ability to adapt to the various internal procedures of the contracting party, and to form close links with the nominated representatives, is essential in ensuring that the key issues related to cost, schedule, scope and quality can be efficiently resolved, and above all client satisfaction can be realized.

The term and title “DevOps Project Manager” describes the person who is given the responsibility to complete a project. DevOps Project Manager is the person with full responsibility and he has the required level of accountability and authority to deliver the desired project objectives within project budget, on time and with the highest possible quality.

DevOps Product Owner

DevOps Product Owner role is a very unique and broad role in DevOps which combines all of the challenging aspects of traditional Project Manager and Product Manager roles. Moreover, DevOps Product Owner represents the customer point of view in a DevOps team by ensuring that the right work is done at the right time. It needs to be tightly integrated with the overall DevOps Software

Development and Delivery Teams and Processes to ensure maximum added value for each and every Product Release.

The DevOps Product Owners have a number of key responsibilities. Some of them are:

- Managing the DevOps Product Backlog.
- Support Creations of Product Release Roadmaps, and Release Plans.
- Identify Product Dependencies and appropriate Prioritization driven by Organizational Mission.
- Stakeholder Management and Communication.

Whether, you act as a DevOps Product Owner or not in your DevOps team, as long as you're directly working with your customers it is fundamentally important for you to comprehend the role of DevOps Product Owner in order to be utmost helpful for your customers and to create the maximum added value for them.

DevOps Architect

A Devops Architect owns architecture, design and development of product deployment tools and processes. In this role, the DevOps Architect is expected to architect and develop innovative solutions to build and maintain product architecture, its related tools and processes for continuous integration and continuous deployment pipeline.

DevOps architects are in charge of defining loosely coupled set of services whose consumers are minimally impacted by changes to these service or their environments. Some coupling is obviously inevitable since the consumer has to make use of the service but DevOps Architects minimize these dependencies by means of separation of interface and implementation, versioning policies, runtime contracts, platform independence and location transparency.

DevOps Architects become extremely critical for the success of DevOps organizations and teams. On the top of building an optimal loosely coupled architecture for products and services, DevOps organizations must possess an extremely reliable environment that is fully automated and free from obstacles. With waterfall, everybody had to have 4x4 cars to drive off-road on tough terrain. The DevOps Architect role is tasked with building the highway so the rest of us can use faster cars.

DevOps Developer

As a DevOps Developer, you transform the business goals of your customers into Software Solutions and Systems. The main difference between an ordinary developer and a DevOps Developer is that: As a DevOps Developer, during the full course of your work, you are conscious of the business goals and business demands of your customers. You are fully aware of why your employer has hired you and what your customers need and expect from you.

Moreover, as a DevOps Developer, you are the trusted technology partner of your customers. You give your employers and customers the full confidence and every reason to keep you on board. You are the one person who connects business goals and business requirements to software designs and lines of runnable software code. You are no longer a sole designer or programmer. You own the end-to-end engineering life-cycle of your entire Software from its requirements analysis, architectural vision and test automation until its user experience, extensions, operations, maintenance and end of life.

As a DevOps Developer, you are in charge of deploying running Software Solutions in small and frequent iterations. You excel and ensure the continuous delivery of your Designs and Software to your clients to create rapid and uninterrupted value for their and your businesses. You always remember that: Your mission is to have happy customers while building software that you and your customers love!

If you are a passionate Developer and a dedicated DevOps Software Engineering Practitioner to build efficient, world-class and high-quality systems, it is highly recommended to be a DevOps Developer.

DevOps Operations Engineer

DevOps Operations Engineers are responsible for monitoring, maintaining and deploying the state-of-the-art software and infrastructure behind the technology of their Products. They deploy and maintain Network Infrastructures and Servers at Data Centers. They also participate in DevOps Delivery and Deployment Teams on installations and develop product delivery and deployment contingency plans.

In this role, duties range from the physical deployment of data center-related technology to working closely with the various stakeholders, especially with DevOps Developers to ensure that

availability, maintainability, monitorability and analytics are embedded into the cores of products.

Behind everything the clients of your organization see is the architecture built by DevOps Operations Engineers. And DevOps Operations Engineers are in charge of keeping these systems up and running. From developing and maintaining products and services to building the next generation of your platforms, DevOps Operations Engineers make product portfolio of their organization possible.

DevOps Operations Engineers are proud to be engineers' engineers and love voiding warranties by taking things apart so they can rebuild them. They're always prepared to be on call to keep their systems and networks up and running, ensuring their clients have the best and fastest experience possible.

DevOps Quality Assurance Engineer

DevOps Quality Assurance Engineers play proactive role for the processing of Unique Selling Points of their Product, Requirements, Use Cases, Software Architecture and various other software design material to find out desired test types to validate the quality of Product under Test. They work with DevOps Developers and DevOps Project Managers to determine Test Implementation Methodologies and Tools to run and operate the Testing Work.

DevOps Quality Assurance Engineers write Lists of creative Test Cases with strong “Break it” attitude. They also classify Priorities and Difficulty Levels of their Test Cases. They create and review detailed Documentation of Test Cases to make sure their Test Cases correctly perceived by the rest of their DevOps team.

DevOps Quality Assurance Engineers continuously follow up and review Test Execution Phases to ensure that implemented Test Cases realize their goals. They act as self-confident and dependable Subject Matter Experts to emphasize the objective and importance of their Test Cases. They raise and review defects, and they make sure that there is no compromise from planned Product Features.

Furthermore, DevOps Quality Assurance Engineers nurture the entire DevOps Team to review their Test Designs, and process their Feedbacks to add additional creative Test Cases and to reduce redundancies in their Test Designs. Last but not least, during a significant portion of their times, they

automate, execute tests and they re-execute tests to validate bug fixes which are coming from Software Development Teams.

DevOps Information Security Engineer

In traditional configurations of IT organizations information security is largely an afterthought. It is yet another nonfunctional requirement that is often taken care when it is most difficult, expensive and hectic to identify and fix the problems.

DevOps Information Security Engineers design big picture security strategy of their organizations while laying out the details of an implementation plan. They understand the constant need to balance the benefits of incremental security measures with the potential burdens on the business. They proactively find and fix security problems in designs at early engineering stages of products, and in running software systems. They monitor networks, software telemetry and prioritize efforts based on risk.

DevOps organizations have DevOps Information Security Engineers working side by side with DevOps Developers and DevOps Operations Engineers. They embed their recommendations and subject matter expertises much earlier on into software development and delivery process. DevOps Information Security Engineers enable their organizations to build security into product during its entire end-to-end delivery life cycle.

DevOps Release Manager

DevOps Release managers work to address the management and coordination of the product from development through production. Typically they work on more of the technical details and hurdles in which a traditional project manager cannot be involved. DevOps Release managers oversee the coordination, integration, and flow of development, testing, and deployment to support continuous delivery. They're focused not just on creating, but also maintaining the end-to-end application delivery tool chain.

DevOps Release Managers closely work with DevOps Project Managers and DevOps Product Owners to create product release roadmaps, release plans, identify dependencies, make them visible, ensure prioritization of dependent tasks by DevOps teams, support management of DevOps Product Backlog, stakeholder management and communication.

DevOps Trainer

Like all other hyper growth trends in our IT industry, adoption of DevOps Methodology is also not immune to potential misunderstandings and misconceptions. Significant number of DevOps teams and companies make organizational, behavioural and operational mistakes which negatively impact the performance of the DevOps teams and their fit to the overall -and usually not yet really agile- organizations. Unfortunately these inconsistencies sometimes end up with the abolishment of DevOps practices from organizations which hurt our industry and the supporters of DevOps.

DevOps Trainers are talented DevOps supporters like you to ensure correct training and education of DevOps practices within organizations.

DevOps Trainers can be independent people from the DevOps teams in the organizations and they can be directly sponsored by the executives to enable top to bottom organisational training of DevOps. Alternatively, DevOps Trainers can be part of DevOps teams and work with executives to get the required support and to consistently train other parts of organizations to fit to the DevOps teams. If you would like to help your DevOps teams, business teams and executive sponsors to properly learn and implement DevOps practices, it is highly suggested for you to be a DevOps Trainer.

DevOps Trainers provide training services usually to their external clients to teach them DevOps in an educational environment setup. DevOps Coaches coach DevOps team members usually within their client organizations and make sure that they properly understand and operate with DevOps. They provide tips and tricks to teams as DevOps teams do their work in the real work environment.

DevOps Coach

Like all other hyper growth trends in our IT industry, adoption of DevOps Methodology is also not immune to potential misunderstandings and misconceptions. Significant number of DevOps teams and companies make organizational, behavioural and operational mistakes which negatively impact the performance of the DevOps teams and their fit to the overall -and usually not yet really agile- organizations. Unfortunately these inconsistencies sometimes end up with the abolishment of DevOps practices from organizations

which hurt our industry and the supporters of DevOps.

Similar to DevOps Trainers, DevOps Coaches are also talented DevOps supporters like you to ensure correct understanding, penetration and adoption of DevOps practices within organizations.

DevOps Coaches can be independent people from the DevOps teams in the organizations and they can be directly sponsored by the executives to enable top to bottom organizational and cultural adoption of DevOps. Alternatively, DevOps Coaches can be part of DevOps teams and work with executives to get the required support and to consistently evolve other parts of organizations to fit to the DevOps teams. If you would like to help your DevOps teams, business teams and executive sponsors to properly understand, adopt and implement DevOps practices, it is highly suggested for you to be a DevOps Coach.

CONCLUSION

In this chapter we covered the most frequently used roles in a DevOps organization. Let's keep in mind that not every DevOps team in your organization must have someone with each of these roles. Less is more. Ensure your team focuses on significant few and eliminates trivial many to achieve IT and business performance your organization aims.

The best rule of thumb is your team should have roles and skills to enable best possible continuous flow of work. Flow is the subject we will start to cover from next chapter and onwards.

HOW SHOULD YOU ENABLE YOUR DEVOPS FLOW?

In DevOps, Flow means end-to-end manufacturing chain of software from idea to running lines of codes in your production systems. You and your DevOps teams are in charge of building and sustaining a reliable, consistent and fast flow to meet and exceed your organizational goals and to outcompete other products and services in your particular market.

Define a Mission for Your DevOps Transformation

After you have your DevOps team in place, the next step is to define a mission. Only by having a common mission, your team will be equipped to correctly function. Without a mission your team will never be able to prioritize the critical work and distinguish showstoppers from errands. The mission should be challenging and impressive, but it still needs to be achievable in a given timeframe.

A mission should be tailored for your own DevOps team. It is based on organizational characteristics, challenges and objectives to serve internal and external clients interacting with your DevOps team.

Some example mission statements to get your DevOps transformation started can be:

- 75% reduction of average lead time from code check-in to live production systems in 3 months.
- 33% reduction of average lead time from request to live production systems in 6 months.
- 50% reduction of number of production incidents in 12 months.

What is Value Stream and Who are Involved in Your Value Stream?

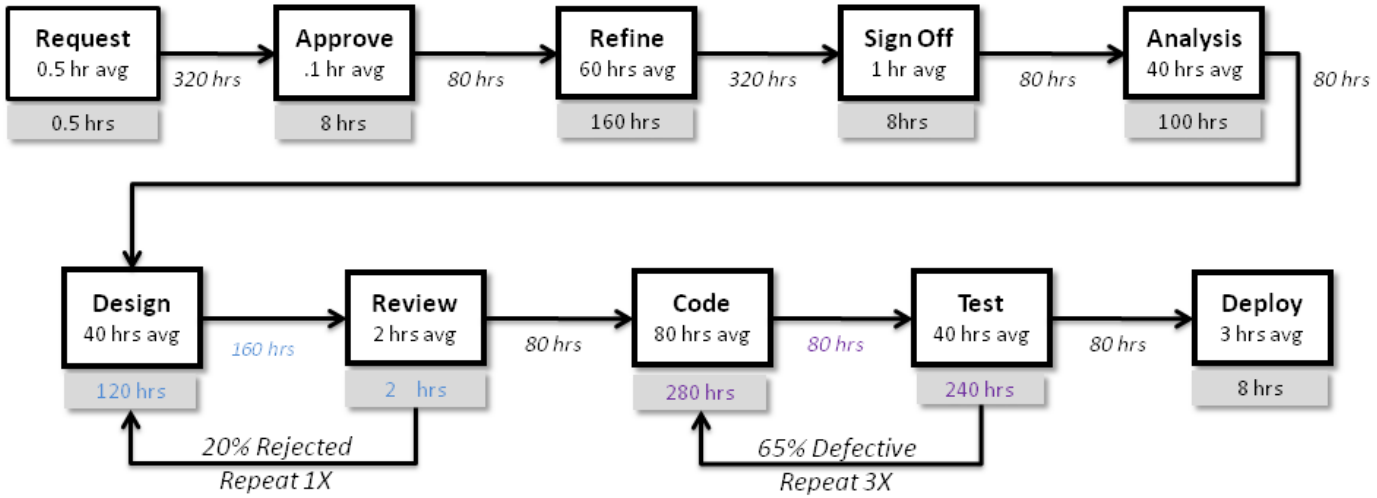
In order to improve your work, you need to know your value stream. A value stream in information technology systems is a sequence of activities required to design, build, and operate a specific software product and service. Furthermore, a value stream defines human resources, knowhow, utilities and materials which enable the value stream to flow.

In a complex organizational structure, no one is fully capable of identifying an end-to-end value stream. Therefore, it is important that all members of your DevOps team, stakeholders, providers, client representatives, if possible your clients themselves should participate to identify your value stream.

Build Your Value Stream Map to Find Out the Improvement Potentials

A value stream map enables you and your DevOps team to visualize how a typical work in your software development and delivery organization is performed. By building a value stream map your goal is not to make a comprehensive documentation about each and every step to get your work done. Your goal is to gain sufficient insight about your typical workflow to identify where major inefficiencies, constraints, issues and waste of time and resources lie.

When software development and operations engineers for the first time sit together and build a value stream map, it is usually the first time for an operations engineer to comprehend the negative effects of misconfigured database servers without table spaces on development teams. Similarly, it is usually the first time for a software development engineer to see the consequences of delivering software without built-in monitoring, availability, testability and configuration features.



DevOps Value Stream Mapping Example for Software Engineering Process (which presents significant improvement opportunities to enable faster flow)

Eliminate Handoffs, Constraints and Waste As Much As You Can

Some of the most significant, but overlooked factors of efficiency are handoffs, waste and constraints. When an activity is handed off from one team to another it requires signalling, requesting, scheduling, prioritizing, deprioritizing and resolving conflicts.

When a work is passed from one team to another, each handoff will not only result in loss of information, but it also negatively impacts overall lead time of your value stream.

To overcome these problems:

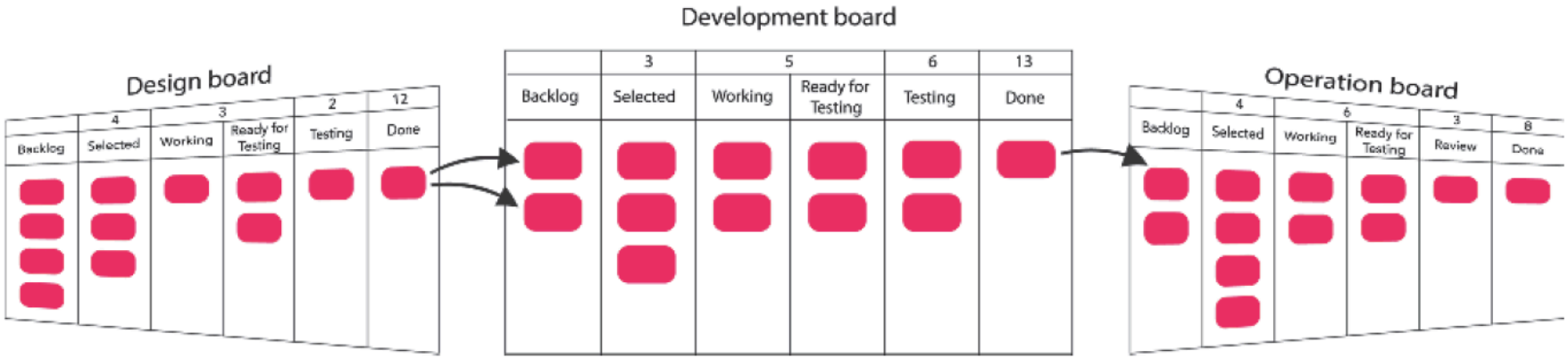
- **You need to challenge the duration required for each handoff.** For instance according to above value stream mapping example, it takes 80 hours until a successfully tested code is picked by deployment team. Why is this like that? Can't we automate this process? Of course we can.
- **You need to eliminate the waste caused by repetitive handoffs.** According to above value stream mapping example, test team hands off 65% of features back to the development. Fixes for some defects need to be delivered up to 3 times until they are approved by testers. Couldn't we speed up these iterations by enabling better integration between development and testing teams? And high number of defects show that something does not optimally function for your development team. The reasons of these quality issues need to be identified and sorted out.
- **You need to challenge and remove handoffs as much as you can.** According to value stream mapping example above, once a request is refined, it takes 320 hours until it is processed by signed-off authority. Why do you need such a sign-off authority in your organization? What is the value of this authority for your value stream? Has this authority got sufficient level of information and vision to give a good decision? Why doesn't this sign-off authority become part of request refinement team, so it knows far more better about what is being signed-off and there will one less time-consuming handoff which takes 320 hours. Unless you remove constraints and barriers which slow down your flow, you can't really expect much from DevOps.

DevOps expects you to ask such questions to break the status quo. Not everyone in your organization will be of course happy to hear such questions. But you are already prepared for this.

Make Your Flow Visible for Everyone

In order to make sure that your work flows from left to right, and your DevOps organization accomplishes its goals, you need to have tools and mechanisms in place which make your flow visible. In information technology business, it is a matter of a mouse click to assign a work from one team to another in your value stream. However, due to incomplete work, inconsistent dependencies and misunderstandings, it is part of your daily business that your work bounces from one team to another (so called "one step forward, two steps back") and it flows too slowly if it flows at all.

Therefore, it is important to make sure that your flow is visible. Not only for your team, but for everyone. DevOps relies on Product Backlog, Sprint Planning Backlog and Kanban boards to visualise flows. These boards do not only involve the works (tasks) which belong to your own DevOps team, but they should also make the entire flow visible from the idea conception of your products and services to operational maintenance and end of life. In this way, whenever a work doesn't flow, it will be quickly visible for everyone. And it will be the joint responsibility of everyone to remove roadblocks and impediments to enable continuous and successful flow of your work.



DevOps Kanban Board Example
(from Design to Development and Operations)

Limit Batch Size and Work in Progress

Research conducted at **Stanford University** found that multitasking is less productive than doing a single thing at a time. The researchers also found that people who are regularly bombarded with several streams of electronic information cannot pay attention, recall information, or switch from one job to another as well as those who complete one task at a time.

What does this mean for your DevOps team? It means that you need to reduce work in progress and limit the batch sizes of your code deliveries. As an illustrative example: Your team can have maximum 6 work in progress tasks in development to avoid continuous context switching and enable full focus on work at hand. Alternatively, you can estimate each task with a story point weighted by Fibonacci numbers (0, 1, 2, 3, 5, 8, 13, 21, 34, ..) and your DevOps team processes up to a certain total number of story points (velocity) in a given timeframe (sprint).

Beside increasing quality and productivity, by limiting batch sizes of deliveries, you will be quicker to identify root causes of issues and resolve them. Once a task is finished, you will check it in your common code repository, validate it with your continuous integration platform and subsequently deploy it in your production. As the batch size is small, identification of production issues due to code deliveries will be easier, potentially required rollbacks will be less cumbersome. Furthermore, by continuously delivering in production, your team will have the constant pride of contributing your organizational mission. We will leave this to you to compare this mode of working with morale, motivation and technical challenges of teams who build their codes for 8 months without delivering one single line of code to their production systems.

Use 20% of Your Time to Reduce Technical Debt

When your organization doesn't reserve time to reduce its technical debt, but continues building workarounds on the top of workarounds, it will come to a point where all engineers spend all their times to fix issues. In financial analogy of debt concept: Your organization will be only paying debt interests.

CONCLUSION

In this chapter we covered for you what value stream and flow in DevOps are, why they are important and what you do to make your work flow in value stream.

HOW SHOULD YOU DESIGN YOUR DEVOPS CONTINUOUS DELIVERY AND DEPLOYMENT PIPELINE?

In many organizations, probably including yours, developers do coding work in isolation in separate code branches. Although this mode of operation deceivingly seems to foster individual productivity of your developers, it makes your team productivity suffer.

Separate code branches make it difficult for you to merge codes from multiple developers. To integrate various different code pieces, make them work in harmony and delivering benefits and features to your clients do usually require involvement of multiple developers, significant reworks, solving conflicts that had to be solved long time ago.

Your Challenge Gets Bigger if You Don't Continuously Integrate

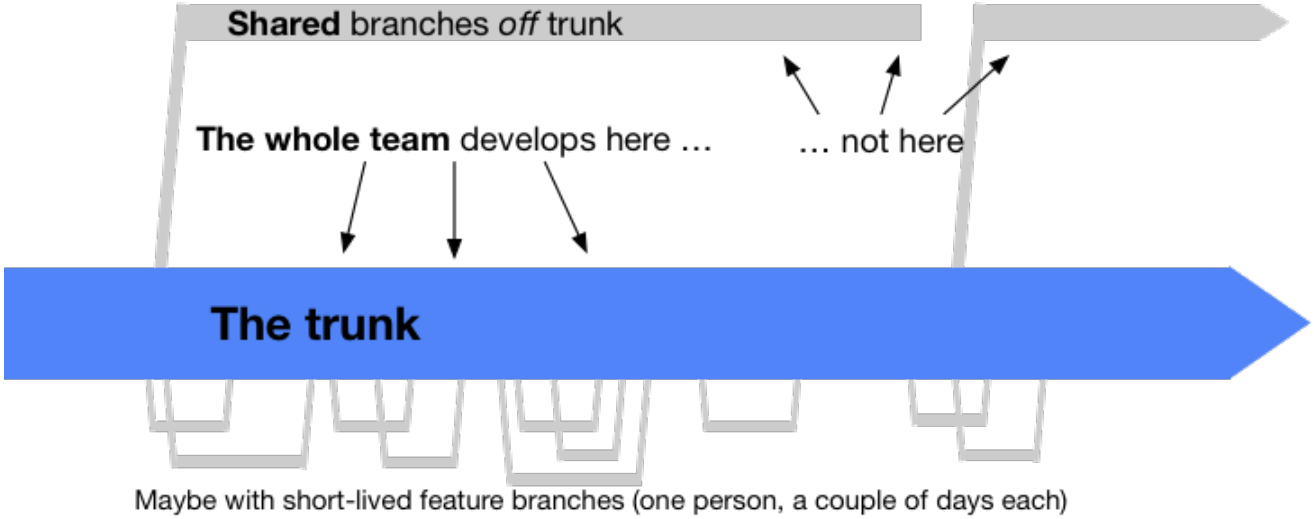
Ironically enough, as merging codes is a challenging and tiring activity within the code building cycles, most development teams unfortunately merge their code before they deliver their application to integration or user acceptance testers, which makes the problem even more impactful. They make successful integration and quality of their joint coding work sole responsibility of quality assurance teams. This is not only ridiculous, but this also perfectly explains why projects delay, project budget deficits skyrocket, team morale and motivation suffer, organizational goals and promises fail, and companies worldwide waste yearly about 600 billion USD for non-budgeted and non-scheduled IT work. Or let's say IT rework or IT workarounds work...

How Can Trunk (Main Code Branch) Based Development Help You?

Your solution for this challenge is trunk (main code branch) based development in combination with continuous integration and test automation. Trunk based development is a source-control branching model, where developers collaborate on code in a single branch called "trunk". They resist any pressure to create other long-lived development branches.

Trunk-Based Development is your key enabler for Continuous Integration and by extension Continuous Delivery. When your developers commit their

changes to the trunk multiple times a day it becomes easy to satisfy the core requirement of Continuous Integration that all of your team members commit to trunk at least once every 24 hours. This ensures your codebase is always releasable on demand and helps to make Continuous Delivery a reality.



Source Control Illustration of DevOps Trunk Based Development (Source: Trunk Based Development)

Continuous Integration Enables You To Deliver Your Code In Smaller Chunks In Deployable State

Furthermore, frequent check-ins from your developers enable small development and delivery batch sizes which increase quality, make it easier to resolve problems and conflicts. Continuous integration with daily check-ins motivates your team to build deployable, but still meaningful codes in smaller chunks without breaking integrity of your continuous integration & continuous delivery frame-

works. Updates in this common source code repository also provides a mean and medium to you and to your team members for quick alignments, decisions and feedback.

Each time a code is checked-in, your continuous integration framework runs automated test cases for the functions and features dependent on, impacted from that changed module. If anything is broken, the owner of this particular check-in is now responsible for fixing the check-in. With top-priority. Nothing else is more important at this moment to ensure that the rest of your team can continue integrating their work to trunk. Until the erroneous check-in will be quickly fixed or rolled-back, no other check-ins to this module of your trunk are allowed to protect the stability and reliability of your trunk.

If you think different from one of these major DevOps principles, the question you need to answer is: Why would you add other new code to a module which already has errors?

You may have legitimate reasons, but let's make sure that you avoid technical debt and workarounds on the top of workarounds.

Gated Commits as an Additional Later of Safety Net for Your Trunk Based Development

Alternatively, your DevOps team can also rely on gated commits which further increase the quality and build another layer of safety net for your trunk.

Each time a developer attempts to check-in a code, test automation runs first to validate the code. Only after successful validation, the code is formally checked-in to your trunk.

Use Production-Like Environments In Every Stage Of Your Software Engineering Value Stream

One other typical challenge in many IT organizations is inconsistencies between production and pre-production (development, integration and testing) environments. Inconsistencies between your configurations, versions and patch levels of operating systems, databases, third party tools and APIs. Or different versions of your own code scattered all over in various different production and pre-production environments. Even very minor differences in JVM versions between your pre-production and production environments can make or break your entire production deployment. Unless you have the full control of your environments and

you know what they have got inside, you can never know what you will get out of them.

Embrace Environment Management Part Of Your Daily Software Development and Delivery Work

Therefore, it is no brainer that your pre-production systems should be the replications of your production systems. You need to enable your team to create self-service environments on demand in automated manner in order to safely do their daily software engineering tasks in production-like environments.

Your IT value stream in your organization does not only offer value with code of your software, but also with environments these codes are running. Both your development and operations specialist need to embrace the fact that, your IT value stream constitutes two equally important building blocks:

1. Codes to build your software which serve value to your internal and/or external clients.
2. Codes to build your pre-production and production environments to host your software.

Use Your Single Source of Truth (Trunk) to Control Your Software and Environments (aka Adopt Infrastructure as Code (IaC))

With DevOps, tasks to build your pre-production and production environments are now part of your trunk which is the purest definition of Infrastructure as Code (IaC). Instead of your isolated operations teams which hardly know the specifics of software your environments supposed to host, your development and operations specialist work together to build your Infrastructure as Code.

In order to ensure code of your software and code of your environments are treated equally important, you use your single source of truth (trunk) to store the code of your software and as well as your environments.

To be able to build your entire pre-production and production systems and your deployment pipeline including its continuous integration and delivery frameworks, you and your team do not only store code of your software in version control system, but also you store required configurations, scripts and anything required to build your end-to-end delivery chain. If you want to permanently change a configuration setting in an environment you do this

change in your environment building scripts, so next time a DevOps team member of yours creates an environment, he or she wouldn't have to reinvent the wheel.

Most of detailed documentations become anyway very quickly obsolete, so for you it will be obviously a better investment to spend your energy to keep your environment building scripts up-to-date. This is mainly because without having self-service and on-demand environments, your cloud infrastructure will remain yet another underutilized hosting platform and your DevOps initiative will be yet another underutilized process improvement attempt. You should be able to create entire production and pre-production environments from your version control system. With DevOps it is easier and quicker to rebuild environments than fixing them. Manuel changes in your environments are no longer allowed. Only code and configuration should rebuild, configure and launch your environments.

You Have a New Definition of Done (DoD)

Having your continuous delivery framework and deployment pipeline up and running, now you need a new definition of "Definition of Done (DoD)".

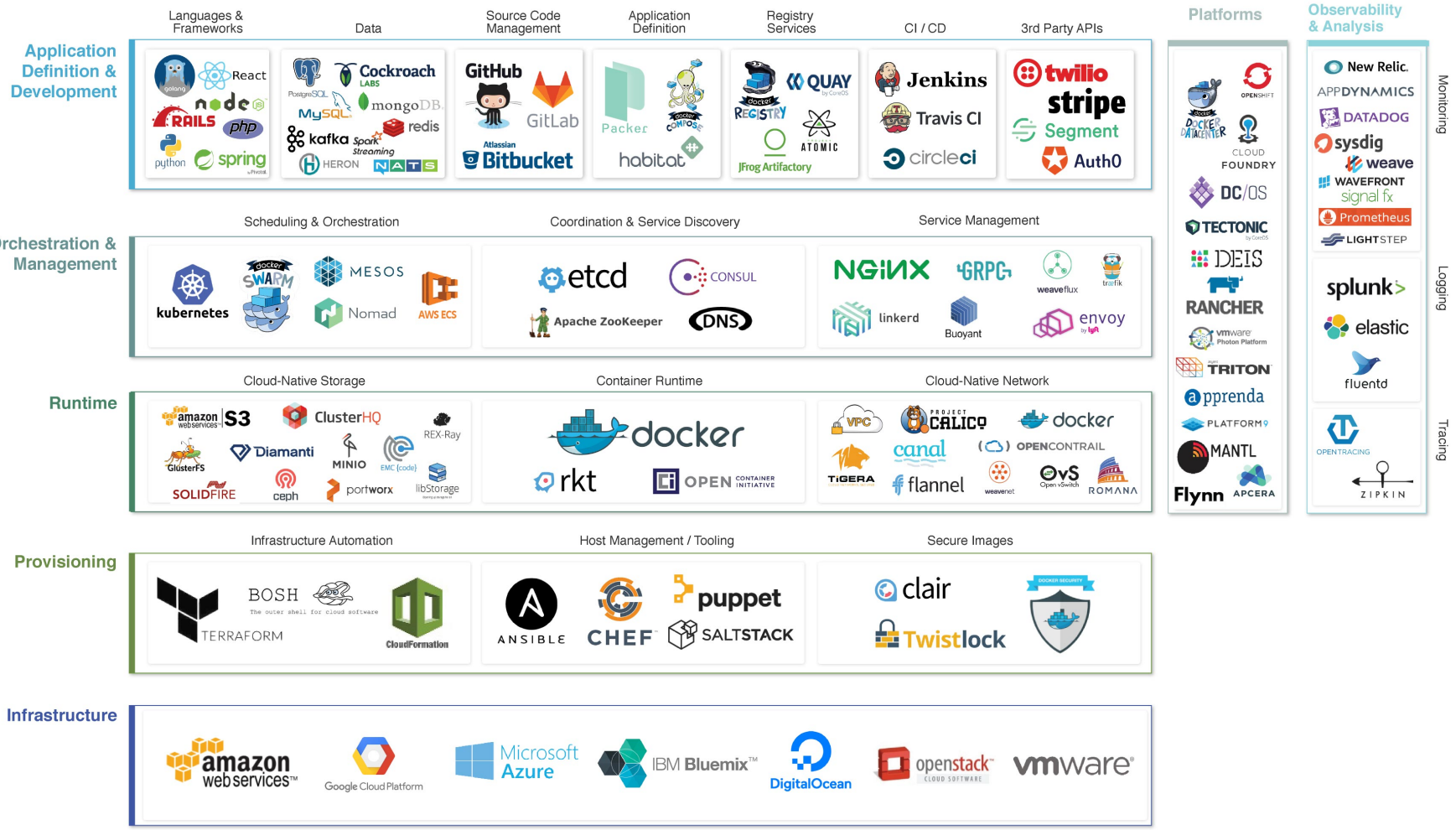
A task should be now classified as "Done" when its associated client features and benefits are:

- created from your trunk either automatically or with a one-click process,
- tested in production like environments with automated tests,
- demonstrated in production like environments,
- ready to deploy or even better already deployed in running production environments again either automatically or with a one-click process.

CONCLUSION

With DevOps, integration of your code happens part of your daily work instead of in the end of your projects. Developers, operations teams, information security specialists endless times rehearse deployment of their work already during the course of software engineering lifecycle which undoubtably improves the chance of smooth live deployments.

Higher business throughput, improved software and environment stability, higher job satisfaction, improved quality of service to your internal and external clients will be also your other benefits as soon as you adopt continuous integration.



DevOps Tools Landscape to Build Your Continuous Delivery and Deployment Pipeline (Source: StackOverDrive)

WHY DO YOU NEED TEST AUTOMATION IN YOUR DEVOPS ORGANIZATION?

If your software delivery department doesn't have decent test automation yet, then there must be a clear disconnect between your development and testing efforts. Your developers are just getting late feedback about the quality of their work while they are already busy with other tasks. They usually miss the cause and effect relationship between code they have written (or they haven't written) and defects. The efforts your developers put for this context switch to turn back to times when they introduced defects clearly reduce productivity. Because the connection between systems and defects are complex and fuzzy after late feedback, getting quality fixes without workaround flavour is neither realistic nor expected.

It is very clear that without test automation, your DevOps organization would need more time and money to deliver quality products and services. This is no longer a scalable business model for any technology organization in 21st century. No matter what business your organization is in.

You Can Automate Your Tests,
But You Can't Automate Creating Quality

Your DevOps team should automate as many test cases as it is possible. Because you can automate your tests, but you can't automate the cognitive ability of your DevOps team to create quality, you shouldn't waste your human capital for any repetitive and repeatable work that software can do for you.

DevOps Principles For Good Test Automation

- Test Automation should give quick and early feedback about your quality of work.
- Tests should generate consistent, deterministic and repeatable results provided same conditions for different test runs.
- Tests shouldn't generate false positives.
- Small number of automated and reliable test cases are better than high number of non-automated, unreliable tests which can give inconsistent results even though same conditions for different test runs are provided.
- Test Automation should first focus on the validation of key features and benefits your systems provide to their internal and external

clients. Even this level of test automation is far more better than no test automation.

- Afterwards, focus on automating as many test cases as it is possible to get the best use out of your human capital in your DevOps teams.
- With your test automation, avoid slow and periodic feedback. What you need is fast feedback whenever you or your developer attempts to check-in code to your trunk.
- If your automated tests are not fast enough, find ways to speed your tests and/or your software. This is not only critical to have a reliable continuous delivery platform and deployment pipeline, but also to have a great quality of service for your clients. To get even faster parallelize your automated tests.
- Consider Test Driven Development. First write your automated tests, then build your software.

Test Automation Is Your Major Enabler For Continuous Delivery and Deployment Pipeline

As we have covered before, DevOps methodology gives your developers the ability to deploy their code to your production systems. And yet observations with early DevOps teams in organizations have proven that: Although developers are quick enough to check-in their code to trunk and close tasks and tickets, they are quite reluctant to push the button to release their code to production systems. They are kind of afraid to break production and get blame for it.

Only after DevOps teams have built comprehensive and reliable test coverage with test automation, developers could overcome this fear.

As an example: In Google, when a code is checked-in to trunk by a developer, after initial checks such as static code analysis, duplication analysis and test coverage analysis, this code is validated with almost a million automated test cases. If all checks and test cases pass, the build is generated from trunk, put it to production and replicated to all other production servers. This is pretty much how 5,000 small and independent Google teams stay productive and make ten thousands of production deployment every day. Without test automation this couldn't be an option.

Fast and reliable feedback is the only way to have a safe deployment pipeline and to protect its integrity. When deployment pipeline is broken due to failed automated test cases, your DevOps team fixes

erroneous codes and configuration before they check-in any other work to your trunk. Furthermore, whenever you find an error in your deployment pipeline that could have been identified beforehand and prevented with an automated test case, you also create and add this test case to your test automation repository.

Types of Tests You Can Automate

- **Unit Tests:** Validate single functions, classes or modules of your applications.
- **Acceptance Tests:** Validate features and benefits your applications provide to their clients.
- **Integration (Service) Tests:** Validate interplay of your applications and their end to end service flows.
- **Performance and Stress Test:** Validate how your services scale (or don't scale) with expected and unexpected client loads. This is particularly important for the most frequently used services.
- **Non-Functional Tests:** To validate security, availability, capacity and scalability.

If you and your team find it difficult to automate unit and acceptance tests, but heavily rely on automation of integration (service) tests, probably you have tightly couple architecture. You should identify ways to decouple elements of your architecture.

CONCLUSION

In this chapter we covered the importance of test automation in DevOps. Test automation is key element to deliver safely and fast.

Regardless your organization adopts DevOps or not, without test automation there is no way to have a world class successful, productive and profitable IT organization which manages to serve well to its internal and external clients.

HOW DO YOU ENABLE LOW RISK DEVOPS CODE DEPLOYMENTS IN YOUR PRODUCTION?

In many IT organizations you can observe, production deployments are cumbersome and stressful. To stay in peace this results to a tendency to reduce frequency of production deployments as much as it is possible. Then organizations inevitably face deployments with larger batch sizes which cause even larger problems. This vicious destructive cycle gets only worse and it unfortunately represents status of most of the largest organizations including the ones whose core businesses are software and technology.

Your DevOps Team Has Built-In Control and Risk Mitigation Mechanisms

In a typical IT organization, development team builds software and operations team takes care of its deployment. In contrast, DevOps methodology shifts reliance of control and risk mitigation mechanisms from other independent teams to your own self-sufficient and competent team. To automated deployment and peer review processes, again within your own team.

As all production and non-environments are assets which are as important as software your DevOps team produces, your DevOps team embraces these three important principles:

1. Ensure consistency of all environments in terms of operating systems, components, interfaces, patch levels, all other dependencies and of course your own software and configurations.
2. It is a priority number one activity to fix an impediment which breaks consistency of your environments.
3. Deploy with the same techniques to all environments, so rehearsal of production deployments become one of your daily tasks and habits.

Your Built-In Self-Service Deployment Mechanism

Your automated self-service deployment mechanism follows the following pattern.

1. Code and check-in to trunk.
2. Build deployable packages.

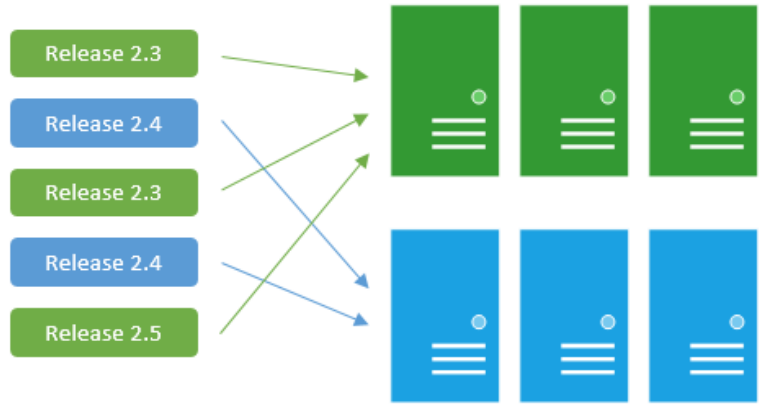
3. Validate deployable packages, dependencies and configuration readiness.
4. Validate environment readiness.
5. Run automated testing.
6. Record all validation and automated test results for audit and compliance reasons.
7. Deploy packages to target environment.
8. Run automated tests in target environment to validate deployment.
9. Monitor system, performance and activity metrics of target environment.
10. Provide fast feedback, correct or roll-back the deployment if anything goes wrong.

Your Deployments Are NOT Identical To Your Releases

With automated self-service mechanism to deploy your code to non-production and production environments, your DevOps team is now empowered and enabled to do daily deployments to your systems. Because every single deployment cannot be categorized as a feature or benefit release for your clients your team is serving for, your DevOps team needs to architect your applications and/or your environments in a way that releases do not require code changes and further associated deployments.

Blue-Green Deployment Pattern

One of the challenges with automating deployment is the cut-over itself, taking software from the final stage of testing to live production. You usually need to do this quickly in order to minimize downtime. The blue-green deployment approach does this by ensuring you have two production environments, as identical as possible. At any time one of them, let's say blue for the example, is live. As you prepare a new release of your software you do your final stage of testing in the green environment. Once the software is working in the green environment, you switch the router so that all incoming requests go to the green environment - the blue one is now idle.

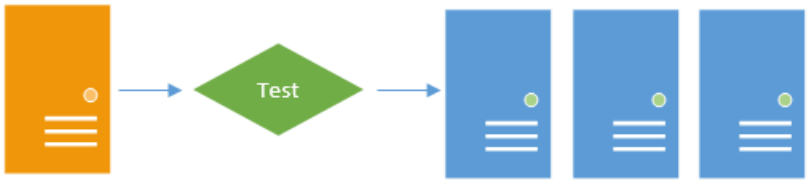


Blue-Green Deployment Pattern
(Source: Octopus Deploy)

Canary Deployment Pattern
(aka The Dark Launch)

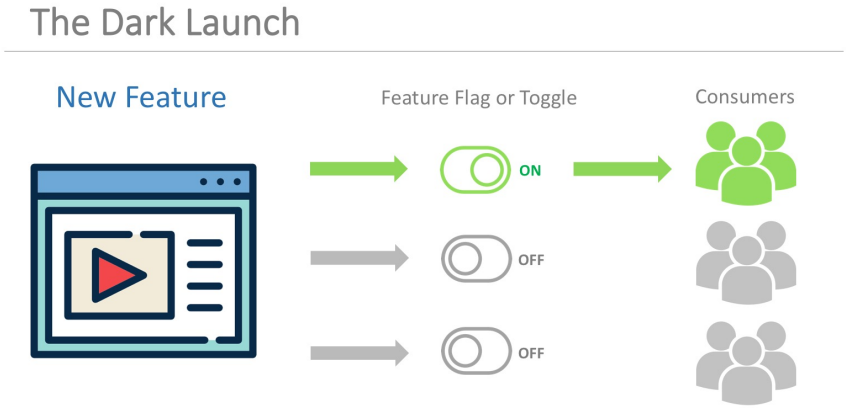
Canaries were once regularly used in coal mining as an early warning system. Toxic gases such as methane or carbon dioxide in the mine would kill the bird before affecting the miners. Signs of distress from the bird indicated to the miners that conditions were unsafe.

Inspired from canaries in mining industry, Canary deployment is a pattern for rolling out releases to a subset of users or servers. The idea is to first deploy the change to a small subset of servers, test it, and then roll the change out to the rest of the servers. The canary deployment serves as an early warning indicator with less impact on downtime: if the canary deployment fails, the rest of the servers aren't impacted.



Canary Deployment Pattern
(Source: Octopus Deploy)

Facebook and Google, along with many leading tech giants, use canary deployment pattern called "The Dark Launch". They gradually release and test new features to a small set of their users before releasing to everyone. This lets them see if you love it or hate it and assess how it impacts their system's performance. Facebook calls their dark launching tool "Gatekeeper" because it controls consumer access to each new feature.



The Dark Launch
(Source: TechCo Media)

It is called a dark launch because these feature launches are typically not publicized, but rather, they are stealthily rolled out to 1 percent, then 5 percent,

then 30 percent of users and so on. Sometimes, a new feature will dark launch for a few days and then you will never see it again. Likely, this is because it did not perform well or the company just wanted to get some initial feedback to guide development.

Cluster Immune System Release Pattern

This release pattern is an extension of canary deployment pattern. It requires performance and software activity monitoring systems tightly integrated with your release process.

In the canary deployment pattern depicted above, once the initial deployment on orange colored server is done, your monitoring systems record all system, performance and software activity metrics and generate early alerts if problems increase over certain thresholds. This results in automated roll-back of installed code from orange colored server.

If automated tests on orange colored server and monitoring system provide positive outcomes, your DevOps team is now far more confident to deploy their code to all other blue colored server clusters too. In summary, Cluster Immune System Release Pattern offers for your DevOps team:

- Additional safeguard for the issues that could be missed by test automation.
- Quick feedback and automated roll-back action for production issues.

Feature Toggles

With feature toggles you can switch on and off features of your application. Therefore, once all codes required for a certain feature are deployed in your production system, release of this feature is nothing but switching on your toggle. Feature toggles are usually settings in runtime system configuration files or system configuration databases.

Thanks to Feature Toggles, your DevOps team is now able to switch off a feature if a canary deployment results in errors or suboptimal user experience. With feature toggles, you also have the ability to disable resource intensive, but relatively less important features if your system has challenges to scale. Therefore, even tough your system may have issues to scale with all of its features, you can still enable your clients to get the best throughput from the most critical features of your software. As an example, in an e-commerce portal, you can temporarily switch off the toggle of viewing invoices,

so your checkout flow has more CPU and memory resources to consume.

Furthermore, in a service oriented architecture, your DevOps teams can deploy different versions of services without switching them on. Once all new versions of services required for a new feature or service flow are deployed, you can switch on toggles of these services to enable the new service flow.

Architect for Safer Releases

There is no perfect one size fits all architecture for all software products and services in all scales.

When IT platform of a startup organization is initially built, a monolithic architecture is most of the times the first choice to ensure the quickest and cheapest entry to the market and to validate the business case. The problem with monolithic architectures is that functionally distinguishable aspects (core functions, supplementary functions, user and systems interfaces and integrations) are all interwoven, rather than containing architecturally separate components. During the lifespan of many large organizations including Amazon, Google, Facebook and Ebay, they need to abandon their monolithic architectures in order to rapidly scale and enable low risk releases and expansions of their features they serve for their clients. And their next direction was service (or micro-service) oriented architectures with the adoption of Strangler Application Pattern.

Strangler Application Pattern To Enable Low Risk Migrations to Micro-Services

Completely replacing your complex system can be a huge undertaking. Often, you will need a gradual migration to a new system, while keeping the old system to handle features that haven't been migrated yet. However, running two separate versions of an application means that clients have to know where particular features are located. Every time a feature or service is migrated, clients need to be updated to point to the new location.

Strangler Application Pattern incrementally replaces specific pieces of functionality with new applications and services. Create a façade that intercepts requests going to the backend legacy system. The façade routes these requests either to the legacy application or the new services. Existing features can be migrated to the new system gradually, and

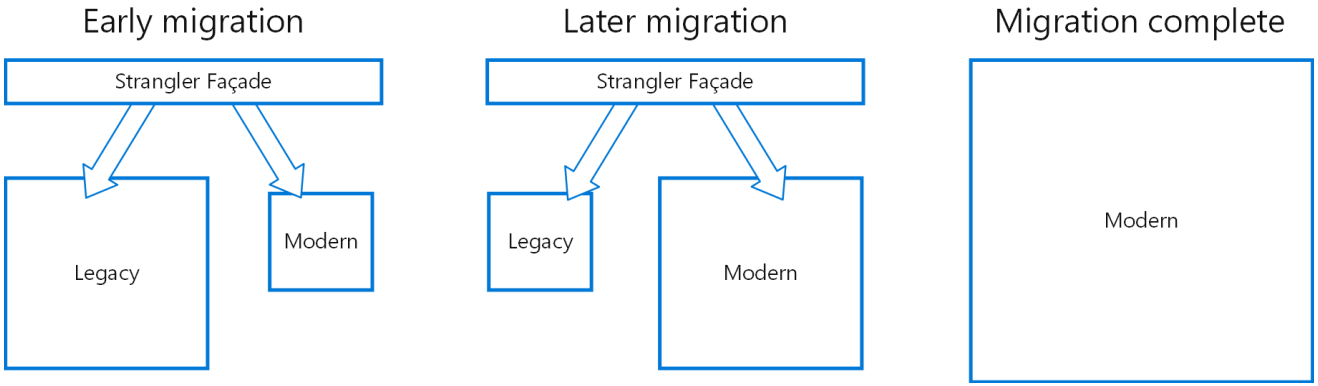
consumers can continue using the same interface, unaware that any migration has taken place.

Ebay is one of the first organizations which have used Strangler Application Pattern to migrate their legacy monolithic architecture into micro-services. They started their migration by merely putting their applications behind well defined universal APIs in Ebay ecosystem, so they didn't have to rewrite all existing code in one single migration attempt. They reorganized their engineering groups with small teams from 6 to 10 engineers. A team as big as 10 people is now able to handle universal API for the auction platform which is the most frequently used auction API in the world.

CONCLUSION

In most of the IT departments, teams are not held responsible for building future-proof, scalable and safer deployment patterns and their associated architectures. And yet, building an architecture and application features which enable your teams to do faster and safer deployments is a major prerequisite to succeed with DevOps.

Only by enabling your teams to do low risk, faster and daily deployments, you and your teams can reap the benefits of continuous deployment and smooth releases.



Strangler Application Pattern
(Source: Microsoft Azure Design Patterns)

HOW DO YOU PROTECT YOUR DEVOPS DEPLOYMENT PIPELINE?

After you and your DevOps team build a working deployment pipeline for the continuous delivery of your IT organization, by adhering the principles you have learnt before, you are now empowered to do low risk production deployments without external approvals. Now majority of your production deployments don't need to go through a change approval process. This is because you put your reliance on proper design of your software delivery methodology, automated testing, automated monitoring and intelligent alerts from your environments instead of merely relying on external authorities. And yet, bear in mind that security and compliance is still an important duty of your DevOps team. This is not only to fulfil requirements from lawmakers, but also to take good care of your clients. Of course your DevOps team will still rely on and collaborate with security and compliance subject matter experts in your organization, and yet designing secure systems compliant within the legislation ecosystem your organization navigates should be now a daily task and habit of yours.

Type of Changes in Your Production Systems

- **Standard Changes:** Low risk changes, require no approvals, quick deployments because they are completely automated and logged. Updates of database look-up tables, content updates, styling changes, standard operating system, database or other external component patches are some of standard changes.
- **Normal Changes:** High risk changes, require approval from Change Advisory Board (CAB), CAB expects Request for Change Form (RFC) to assess changes, require long lead times because most of the times CAB members are not knowledgable enough to assess changes. They hold changes to give decisions ultimately based on their experience, intuition and bias on who requested changes.
- **Urgent Changes:** High risk changes, require approval from senior management. Critical errors in production systems which impact clients, fixes of problematic deployments, security patches, service restorations are some of urgent changes.

Regardless types of changes, you and your DevOps team need to document all changes in your change management and work planning systems (such as Remedy and Jira), so the work will be visible to everyone within and outside your DevOps team.

Use Your Track Record of Successful Automated Deployments History to Convert Normal Changes into Standard Changes

Only by increasing the ratio of standard changes over normal changes, you can avoid Change Advisory Board (CAB) approvals and you can enable faster flow and higher quality deployment pipeline.

You and your DevOps team now use your history of successful automated deployments to convert as many normal changes as it is possible into standard changes. You show CAB your track record of deployments, list of production incidents and prove them your automated production deployments cause none or only negligible incidents in your production systems.

Speed up Deployments of Normal Changes

For the changes which must still remain normal, automate composition process of Request for Change Form (RFC). Link all automated and non-automated test results, resolved and non-resolved incidents, monitoring records and logs from non-production systems to RFC. Try to simplify CAB's job and provide all information they are looking for in the first version of RFC, so you speed up CAB approvals as much as it is possible.

Once approved, enable one-click deployments of normal changes. Regularly review types of normal changes with CAB, identify and define expectations which would convince them to convert normal changes into standard changes, so that you can automatically deploy majority of your changes in your DevOps organization.

Reduce Bureaucracy and Reliance on Others in Your DevOps Organization

The bigger an organization gets, the more food bureaucracy finds for itself. Bureaucracy has invisible ability to grow exponentially and live forever. Once you rely on authority and control mechanisms one level above you to approve your own job, you'll no longer feel empowered and responsible for the outcomes of your own job. This will make the authority one level above you regularly fail.

Finally this authority would also require another authority to rely on and get audited. This downward spiral gets only longer until the next reorganization in your company. After reorganization, chances are

almost 100% that a new downward spiral with slightly other form will replace your actual downward spiral.

Your company needs to remember that separation of duties, and reliance and control mechanisms outside your own team will not only slow down and reduce the quality of your software delivery flow, but also they make your organization less secure. A major US finance organization became victim of an ATM fraud due to a backdoor a software developer had built in ATM software. None of external auditors, external security experts, external compliance officers and external CAB authority figures could manage to identify the fraudulent source code deployed to production systems. This fraudulent could have been only identified within the team it was developed by deploying fundamental DevOps techniques such as inspection of code check-ins and code reviews.

In order to deliver quickly and securely, you need to reduce reliance on others and separation of duties because they would only prevent your DevOps team from taking responsibility. In your DevOps team you need to deploy other control mechanisms such as inspection of code check-ins, pair programming, peer reviews, automated testing and monitoring. If separation of duties are mandatory due to legal reasons, your built-in controls in your DevOps team will still empower auditors and compliance officers to give better, informed and faster decisions.

CONCLUSION

DevOps brings a new and dynamic dimension for compliance and information security. Where your infrastructure is code, when code makes your systems appear and disappear, and where your code is automatically deployed, it is not a trivial process for auditors to understand what is really going on. This is a new challenge, but as well as an opportunity to create and deploy leaner and smarter auditing and compliance mechanisms and officers.

In your DevOps team information security, compliance, quicker deployment pipeline are everyone's job. It is a major goal for your DevOps team to enable compliance officers to access self-service information, logs, reports and metrics which prove high quality delivery of your DevOps team. These will simplify bureaucratic approval processes or even better to altogether remove them.

HOW DO YOU ENSURE YOUR DEVOPS INFORMATION SECURITY?

In most organizations, information security concerns are one of the most frequent objections against DevOps adoption. And yet, DevOps methodology is one of the best techniques to deliver world's most secure systems.

In many organizations, perhaps in your organization too, the ratio of information security specialists over entire software engineering team is 1/100. In other words, in a software engineering team with 100 people you usually find only one single information security specialist. This results in long lead times to get any software security related problems resolved, delays of software deliveries and even worse sub-optimal level of information security for your clients.

If you have learnt one single thing from your software delivery experience, this must be that showstoppers at the end of projects are bad, but showstoppers related to security issues are even worse. Therefore, every single member of your DevOps team should embrace information security part of daily engineering work, rather a checkbox ticked (or unticked) in the end of your projects.

Involve Information Security Specialists In Early Stages Of Software Engineering Process

In order to ensure an information security issue does not become a showstopper and bottleneck just before your software deployment, involve information security specialists in early stages of your software engineering process. You invite them to demonstrations, early planning and review sessions, so they get a feeling about business your software is associated with. In this way they can better judge potential information security risks and issues, so they support your DevOps team to define information security and compliance goals that must be handled during the course of your software engineering process.

Information Security Is Part Of Daily Work

You and your DevOps team need to track security features as well as security incidents with your standard task planning and incident management tools instead of dumping them to compliance management tools which your DevOps team doesn't

pay much attention to. Whenever there is an information security related issue in your software architecture, design or running systems, educate your DevOps team about these issues. Make them comprehend root causes of these problems and how they should think and approach similar situations in the future in order not to recreate the same issue.

In terms of information security, tactical approach of your DevOps team is:

- To prevent security mistakes from being repeated.
- To integrate security objectives into project goals, planning and tracking tools.
- To make security tests part of automated tests in your deployment pipeline.
- To define reusable self-service tools and software libraries which combine information security best practices of your organization, after making comprehensive information security analysis from all angles of certain product and service features.
- To educate and trust DevOps Developers and DevOps Operations Engineers whose core competences are not necessarily information security.

Build Secure Libraries, Procedures, Blueprints, Architectures and Designs for Your Software

By building such reusable assets, your DevOps team should standardize information security aspects of your software in various critical dimensions such as:

- Communication and data transfer between clients and software.
- Data storage.
- Secure environments.
- Operating systems, databases and configurations of 3rd party tools, components and other interfaces to avoid vulnerabilities.
- Password storage.
- Handling of forgotten passwords.
- Handling the logging of sensitive client information.
- Avoiding cross-site scripting (XSS).
- SQL Injections.
- Other information security vulnerabilities specific to your business and legislation ecosystem your business operates in.

Recommended Techniques for Information Security With DevOps Methodology

- **Static Analysis:** Code analysis to identify backdoors and security vulnerabilities.
- **Dynamic Analysis:** Backdoor and vulnerability analysis while the system is running. Continuous monitoring and analysis of CPU, RAM, Network I/O and Disk I/O operations in non-production as well as in production environments.
- **Dependency Analysis:** Static and dynamic analysis for external tools and dependencies which contain source code you cannot control. When your organization uses a third party tools, libraries or services, you also inherit their information security issues. Don't forget to review open security incidents of third party components and vendor's track record of how quickly they rectify such issues.
- **Security For Source Code Access:** Your DevOps team should use a Public Key Infrastructure where everyone should possess one public and one private key. In this way, all check-ins/check-outs and reads from your code repository are authorized, monitored, and all changes are signed by their respective performers.
- **Integrate Security Monitoring (Telemetry) in Your Environments:** To check system usage details to identify security breaches. These breaches are usually indicated by excessive number of some critical user-generated events such as failed log-in attempts, number of password recovery requests and purchase transactions from same user with various different credit cards and so on.

CONCLUSION

In this chapter you have been provided some recommendations about DevOps' way of information security. Information security by itself is an art and science, so the approach articulated in this chapter doesn't meant to make you an information security expert, but to explain you how DevOps software development and delivery methodology approaches information security.

It is evident that DevOps empowers and very well integrates information security and compliance goals of your organization to the daily work of your DevOps engineers by making information security everyone's job in your organization. World's most dynamic companies have already proven that this is a safe way to securely serve your clients.

HOW SHOULD YOU ENABLE YOUR DEVOPS FEEDBACK?

In your IT ecosystem where demands and requests from your clients constantly become more challenging, you and your DevOps team need to strive for continuous quality improvement of the work you are doing. In order to achieve this, you enable fast, continuous and reliable feedback loops from right to left in your value stream.

These feedback loops will help you quickly deal with impediments while they are small, cheap and easy to remove. They will help you create an organizational learning culture while you do your work. When problems occur, you and your DevOps team treat them as opportunities to learn and improve the quality of your software products and services.

Why Do You Need Built-In Quality Enabled By DevOps Feedback Loops?

From your experience in software engineering industry, you must be so far pretty clear that in a complex system, no one is able to know everything. Even doing the exactly same work twice does not yield the same outcomes. Because this level of uncertainty of outcomes is not tolerable in any business, organizations tend to build control and reliance with quality assurance mechanisms by deploying checklists, audits, compliance, quality assurance professionals and micromanagement. And yet, all these measures are not sometimes sufficient enough to avoid mistakes and errors.

Therefore, building an organization which builds built-in quality starts with accepting mistakes and errors part of your daily job.

Design A Safe System Of Work Culture

You design a safe system of work culture. Nobody in your DevOps team is afraid of making mistakes because you and your DevOps team know that errors are quickly detected and fixed while they are small and before they cause catastrophes such as significant defects, downtimes of services and negative client reviews.

By building a safe system of work culture you, your DevOps team and your entire IT and business organization will enjoy following benefits:

1. Complexity of your systems will be managed, so problems in designs and operations will be quickly detected.
2. Problems are quickly resolved while they are small. Resolving problems will result in spontaneous construction of new organizational knowledge and experience.
3. New knowledge and experience are spread around and distributed towards your entire organization.
4. Leaders in your DevOps organization develop other leaders who create and continuously improve safe systems of work.

Identify Problems While They Occur

If feedback mechanism in your organization is slow, infrequent and late it also late and expensive to prevent undesirable outcomes. Like good old waterfall software delivery days when applications are built for a year before they are for the first time shown to clients, and then rewritten for another year, none of businesses today have luxury to work with this modus operandi.

Your goal is to create fast feedback loops. When your work moves from left to right in your value stream, it should continuously provide feedback from right to left. Building quality in your DevOps organization is all about building quick feedback cycles. When an issue is introduced, your DevOps team identifies it about while it is first time occurring in your value stream. You and your DevOps team quickly fix issues and you constantly validate correlation between client expectations (internal clients, external clients and all other stakeholders in value stream who are impacted from your work) and your implementation to fulfil these expectations. Quick feedback cycles do not only enable you to quickly fix issues, but they also enable you to learn from them and prevent from doing same errors again in the future.

Undo Your Errors When They Are Easier, Quicker and Cheaper To Fix

Quickly finding problems enable you to quickly fix them too. Otherwise,

- Cost and effort to fix them exponentially grow and you allow technical debt (workarounds on the top of workarounds) to accumulate.
- Errors proceed to downstream work centers in your value stream which most likely contribute construction of other errors.

- Errors happen again and again far away from the work center they were first introduced and they will require more fixes and work from you to be undone.
- Memories about why errors did happen in the first place fade and circumstances contributed construction of errors change. If errors are first identified months after they were introduced, you can't really find out true root cause and worst of all you can't learn from them. You would do a workaround to save the day and move to the next workaround. You already know that this is exactly what you want to avoid with DevOps.

Encourage your DevOps team and yourself to raise your voice and to build continuous feedback mechanisms to identify and quickly fix errors. Do NOT introduce erroneous work on the top of erroneous work. In other words, don't let your DevOps team build new features before fixing errors which would negatively impact construction of these new features. Continuously improving quality of your work and continuously removing errors while they are happening will ensure that you build best software products and services in your particular market.

Don't Push Quality Control Decisions Further Away From Where The Actual Work Is Performed

Pushing decisions about quality controls further away from where the work is performed lowers quality, increases delivery lead times, decreases the strength of feedback between cause and effect and reduces your ability to learn from your mistakes.

In a nutshell:

- Don't require your teams to do manual quality control work that can be automated.
- Don't require approvals from busy people who have no to limited knowhow about the work that is being performed.
- Don't create large documentations for approvals which will soon become obsolete due to nature of your work.
- Don't push large batch of work to authorities for approvals.

Instead, every single member of your DevOps team should be finding, fixing, sharing, talking and teaching about errors in her or his own area of control. Pair programming, peer reviews, automated testing, inspection of code check-ins, internal checkpoints, very frequent demonstrations should

make quality assurance responsibility of everyone instead of responsibility of a dedicated quality assurance department.

CONCLUSION

You and your DevOps team are always conscious that you are getting paid to serve your clients. Therefore, you should be always working for the best interest of your internal and external clients.

And yet, the next work center in your value stream is particularly important. You should optimize your work for them with empathy. You should build fast and reliable feedback flows with them to enable fast, smooth and high quality flow in your value stream, so that your DevOps team is able to identify and resolve problems as quick as it is possible.

HOW DO YOU CREATE MONITORING (TELEMETRY) TO MANAGE YOUR DEVOPS SOFTWARE LIFE CYCLE?

From your experience in IT industry you can already easily tell that when things go wrong it is not trivial to identify root causes of issues. The problem can be in your software applications, in your environments or in other components your applications and environments are integrated to.

When you look at the development of relatively complex client and server platforms during last 50 years, one type of event which has been frequently occurring is that: When a server starts behaving suboptimally, throwing errors and it doesn't deliver, you simply restart it by hoping that restart will resolve issues. Indeed a restart can sometimes temporarily undo a problem that you have never understood. And yet, if this doesn't help, your next stop is developers and testers who didn't properly deliver and who didn't properly identify issues while the software was tested. All this chaos will not only impact client satisfaction from your services, but it will also pollute working climate in your organization.

Champion DevOps organizations do barely restart their servers during rectification of their issues. They deploy systematic approaches to identify and resolve problems. They rely on production telemetry to understand root causes and contributing factors to problems instead of blindly restarting servers. They have 96 times better MTTR (Mean time to recover) than other organizations. In other words they solve their production issues 96 times faster than an average company. Top technical practice of champion DevOps organizations is they deploy telemetry in their software and in their applications.

What is Telemetry?

In pretty simple terms telemetry is the process of recording the behaviour of your systems.

To make this happen you need to design your software, your production and pre-production environments and your deployment pipeline in a way that they continuously generate records for telemetry. Your goal is to deploy enough telemetry, so that you can confirm that your services do correctly function in your production environments. When a problem occurs, thanks to viewing your telemetry records, you can quickly understand what

the problem is and take informed decisions to rectify it.

Furthermore, telemetry helps you validate your understanding of what is happening in your production systems compared to what is happening in your production systems in reality, so you can easily see if they correlate.

Build Your Telemetry Infrastructure

In order to have telemetry you need to have two major components in place:

1. **Recording of Telemetry Metrics:** All High Performer DevOps organizations constantly record hundreds of thousands of metrics at every layer of their applications, environments and deployment pipeline. A few examples are events in business logic such as number of sales or server platform health checks such as monitoring of operating systems, databases, disk I/O operations, network I/O operations, RAM, CPU and Security.
2. **A Central Platform to Manage Telemetry Metrics:** This platform stores metrics and events. It enables visualization, trending, sampling, alerting, anomaly detection. It converts logs into metrics such as number of fatal exceptions in a software application. Furthermore, events in your deployment pipeline such as commits, rollbacks, installations, uninstallations, automated test results in production and pre-production environments should be also stored in the same telemetry infrastructure.

You, your DevOps team and all other stakeholders working together with your DevOps team should be able to retrieve information from your telemetry Platform via self-service APIs and GUIs, instead of opening tickets to send requests to access telemetry information.

All of your telemetry information must be 100% accessible to your entire organization except telemetry metrics which may violate privacy and jeopardize security of your clients.

Types of Telemetry Metrics

1. **Business Layer Metrics:** Such as A/B testing results, profit, revenue, number of new users, average session durations, number of completed orders and number of abandoned checkouts.

2. **Application Layer Metrics:** Such as application response times, transaction durations, number of core dumps and number of fatal exceptions.
3. **Infrastructure Layer Metrics:** Such as server traffic, disk I/O operations, network I/O operations, RAM, CPU and disk usage.
4. **Client Layer Metrics:** Such as client application response times and client errors on web, mobile, JavaScript and other client applications.
5. **Deployment Pipeline Layer Metrics:** Such as check-ins, deployment lead times, frequencies, status of environments and green/amber/red status results after execution of automated tests.

It is profoundly important to build your metrics within hierarchies under various categories and nested sub-categories, so you and your DevOps team can easily interpret them.

Make it easy to understand log entries of your applications which will be later converted into telemetry metrics. Just like you group logs under various event categories, do a similar a grouping for your telemetry metrics too. An example of such a grouping is: Debug, Info, Warn, Error and Fatal levels of telemetry metrics.

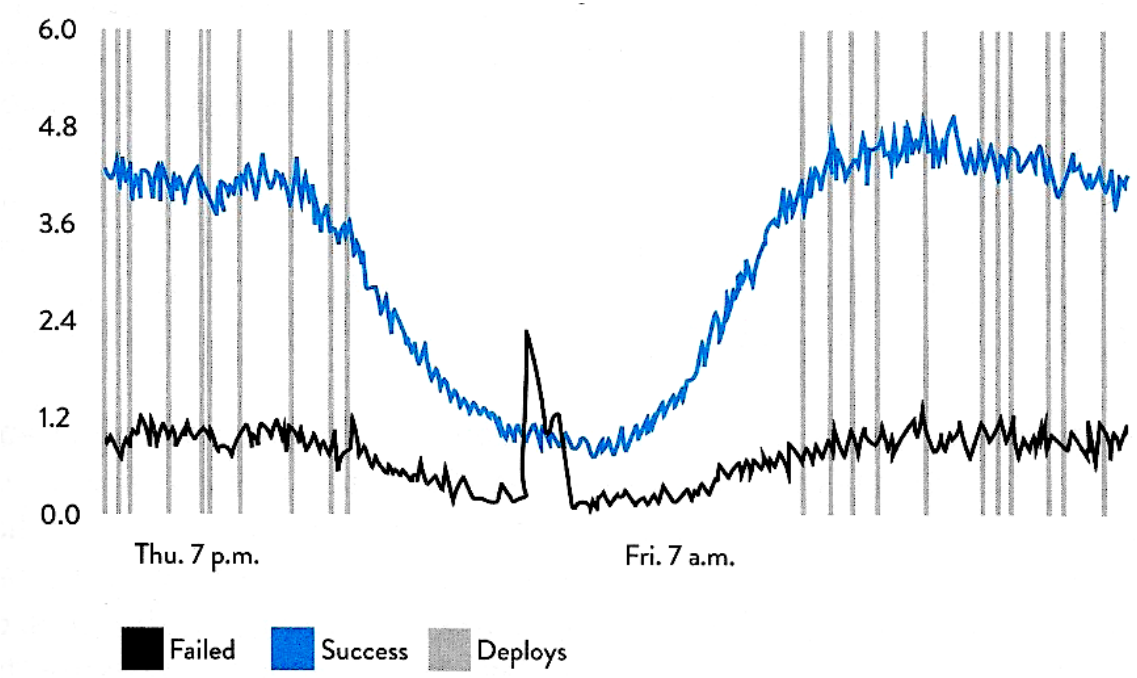
Use Your Telemetry Information To Guide Problem Solving

If your organization has a culture of blame, nobody wants to make changes in production systems fully visible and nobody is willing to display telemetry. In this atmosphere, root causes of issues are barely correctly identified and worst of all no new organizational learnings happen.

In order to ensure you can use your telemetry to guide problem solving, make sure creation of telemetry becomes a daily job for your entire DevOps team. Create easy to use libraries, so that one line fo code easily creates a telemetry record.

Furthermore, create telemetry records for write events of your version controlling system and running environments, so from your telemetry monitoring, it will be very clear and easy to visualize the correlation between changes you do in your systems and their associated impact on your clients.

As an example: From the below chart it is very clear to see that one of the last deployments on Thursday evening is a probable root cause which increased the failed purchase events from your checkout flow.



Deployments vs Key Business Events Chart
Increased Failure and Reduced Success Rates of Purchase Events between Two Deployments
(Source: Measure Anything, Measure Everything, DevOps Handbook)

Telemetry will help you communicate about issues in detail. You and your DevOps teams have nothing to hide from yourselves and from your stakeholders. Therefore, you constantly monitor and present charts like above to your stakeholders in realtime to support quick identification of issues and to see potential cause & effect relationships between your deployments and key business events. Business people become a better understanding and transparency about the work you and your team perform. Furthermore, DevOps Developers and DevOps Operations Engineers see the correlation between incidents and deployments.

Telemetry enables you to see the problems while are easy and cheap to fix, so you undo them before they spread and you build other problems onto them.

With telemetry you and your DevOps team can identify patterns of key business and technology metrics and create alerts if anomalies happen. Your alert thresholds in the beginning can be false and they may generate false positives. This is totally normal. Don't panic. And don't let anyone undermine your effort and investment to build your telemetry infrastructure. Like everything else in your complex systems, you will figure this out too and fine-tune acceptable thresholds for your alerts too, so they will work for you, your clients and business.

CONCLUSION

Champion DevOps organizations identify impact of problems as measurable business metrics such as number of lost clients or lost revenue. So everyone in their DevOps organizations become more sensible about telemetry. Not only in production, but also in pre-production environments. They invest time and resources to build and use telemetry, and they rely on using telemetry to quickly identify and undo their errors.

WHY SHOULD YOU ENABLE FEEDBACK FOR YOUR SAFER PRODUCTION DEPLOYMENTS?

In competitive markets like yours, having a separate quality assurance and operations departments from your development team is not really acceptable if you want to rapidly serve your clients and constantly fulfil their demands. And yet, although most of developers complain about bureaucracy in their organizations, when they are given a chance to collaborate quality assurance and operations specialists in their owns teams, they are still afraid of doing non-supervised production deployments on their own. This brings us to techniques and measures we should put in place in order to ensure safer production deployments.

How to Enable Continuous DevOps Flow

To enable smooth and continuous flow, the secret sauce of most successful DevOps organizations is frequent deployments in small batch sizes of changes in their production systems. Therefore, everyone in your DevOps teams can assess and understand changes, and fix them when necessary.

Building an automated deployment pipeline is not fully sufficient. You need to integrate operational telemetry into your deployment pipeline to quickly get feedback about the results of your changes in production and pre-production environments. Furthermore, in your organization you need to create a common cultural understanding about: Everyone in your DevOps team is responsible for the health and successful continuity of entire value stream and deployment pipeline.

Rely On Telemetry For Your Safer Deployments

You never consider a change and deployment marked as “done” until you prove that it operates what it was designed and coded for. After your deployments you closely monitor metrics of changed modules, newly created metrics if any and the metrics of other components in your system which may be impacted from your change.

Although you use your pre-production environments to run automated tests and you monitor your system under test with your telemetry infrastructure, there

will be still issues in your production systems. You can't prevent all problems from happening, but you can be very well prepared to rectify them when they happen. If a change breaks your deployment pipeline, you bring all subject matter experts required to undo the problem and make your deployment pipeline healthy again. Following are three of frequently used methods to solve issues:

1. **Switch off the Feature:** This is easiest way to fix a problem. It doesn't require an urgent code deployment. You just switch off the feature with feature toggles. You don't have to quickly correct erroneous pieces in your deployment, so your DevOps team can take time to properly identify root cause of this issue and improve their techniques to ensure such a problem will not likely happen again in the future.
2. **Fix forward the problem:** Deploy a new code to fix the issue. Although fix forward is a dangerous way to address production issues in traditional IT organizations, in DevOps organizations it can work very well, efficiently and safely if there are test automation, automated deployment and comprehensive production telemetry in place.
3. **Rollback the Change:** You remove erroneous code, so the problem (hopefully) disappears.

Make Your Developers Get Feedback From Real Life Of Operations

Rotate your people in your DevOps team to handle responsibilities of operations teams, so they handle and deal with operational incidents. In this way everyone in your value stream wins a sense of challenges and responsibilities of downstream work centers. Put your developers, testers, architects, designers, managers and directors on operational non-scheduled duties, so they get incident alert calls at 3am in the morning. This makes everyone in your value stream to build a solid opinion about the consequences of decisions they are giving during their daily jobs.

Such a rotation encourages operations specialists not to feel isolated and alone. Everyone in your DevOps team supports to build a proper balance between fixing production incidents, reducing technical debt and developing new features. It is quite clear that when you wake up architects and developers at 3am in the morning, incidents will be fixed faster than ever.

When developers are asked to observe their clients while clients use their software, they have lots of aha moments to discover what they should immediately

improve. This is also true when architects, designers, developers and testers internally monitor other downstream work centers in software engineering lifecycle. When they comprehend the impact of their work on downstream work centers, they gain a new angle to improve the quality of their work and fine-tune the outcomes in order to help downstream work centers perform better. Everyone in your DevOps team starts to take over non-functional operational requirements part of their daily work within their backlogs. And this is only possible by enabling quick and continuous feedback loops within your DevOps organization.

Make Your Developers Operate Their Own System

It is very difficult to transfer learning experiences from real production systems to development teams. Therefore, some prominent DevOps organizations including Google make their development teams be responsible for operations of software during and after initial product launches. In this developer managed state of a product, operations engineers act as consultants. After it is proven that the product is stable enough in production for about 6 months, it is handed off to operations teams. This hand off can only happen if the product in production already fulfils a number of checks such as past and ongoing defects, telemetry coverage, out of work hour incidents, loosely coupled architectural design and change and deployment safety.

If the product in operations managed state ended up having uncovered significant design and coding issues, it can be handed off back to developers. In developer managed stage, developers are in charge of stabilizing software whereas operations engineers act as consultants.

CONCLUSION

In this chapter, various techniques to ensure successful and safer flow of deployment pipelines in your DevOps organization are covered.

These techniques demonstrate exemplary mutual respect and collaboration between developers and operations engineers in your DevOps teams.

HOW DO YOU IMPROVE YOUR HYPOTHESES WITH DEVOPS AND EMPOWER YOUR EXPERIMENT AND LEARNING-DRIVEN DEVOPS ORGANIZATION?

In typical IT organizations you used to build, most likely you are still building multiple releases of products and product features without validating if the desired business outcomes from these products and features are fulfilled or even if they are being used by some living human beings at all. The most inefficient way to test a business model or a product and feature idea is that: You build a complete software product and service to see whether the predicted business demand already exists and your idea can meet this demand.

Before you build a full-blown product and service you need to ask: "**Why should I build this?**", "**Is this really worth of my time and resources?**". You need to create and run fastest and cheapest experiments possible to validate if your ideas, products and features meet your desired business outcomes.

Long time statistical experiments and measurements with various enterprise product suits have shown that: Only 1/3 of all performed changes can considerably improve a key business metric such as revenues, conversions, orders, subscriptions, new customer acquisition and so on. In other words 2/3 of changes make from no to negligible improvement or they make results of the metric even worse. For these 2/3 of changes, your organization would be better off giving entire team a vacation, instead of building these non-value adding features.

Your counter measure to overcome this challenge is user research by conducting A/B testing. You test results of your features before you build them. You integrate feedback from user research to your software engineering process to make sure that you are not only correctly building with DevOps methodology, but also you are building the correct thing. The faster you can experiment, learn and integrate client feedback into your software engineering life cycle, the better ability we will have to outperform your competitors in your particular market.

A/B testing technique became first popular with direct response marketing via postal mail. By changing wording, color schema, design layout, headline, copy text and so on, marketers have been testing numerous variations of flyers and post cards in order to find out the versions which perform and sell more than others. British government has been doing A/B testing to identify the best performing letter to collect overdue tax revenue from its citizen. A/B testing is nowadays a mainstream method not only to test marketing and communication elements, but also to identify what ideas, products and features work and what don't.

In order to adopt A/B testing in your DevOps team, you need to be able to quickly and easily deploy multiple versions of product features and present them to various different user segments. With your comprehensive telemetry and business layer metrics you identify if the feature produces better business results. If yes, which version of this feature performs best.

An example: Your DevOps organization wants to release a new checkout flow for an e-commerce portal. Before this new checkout flow becomes a major new work for your DevOps team, it is only a hypothesis for better business results. Only after this hypothesis is investigated and validated, your new checkout flow can turn into a real project which will consume significant time, resource and energy from your team. The first work package of your DevOps team with this checkout flow is:

Validate Hypothesis: **We Believe** that a new version of checkout flow, **Can Result In** improved conversion rates. **We Will Be Confident** to fully build and deliver this feature when we see that one of experimental check-out flow versions presented to 1% of visitors increases sales at least 5% during next 7 days.

CONCLUSION

Instead of merely relying on your gut feeling and best practices you and your DevOps team have been learning and observing so far, the focus must be getting real people in the real world to really perform in your experiments.

DevOps Product Owners should see their product and feature ideas as hypothesises to be validated. These hypothesises can be comprehensively designed, built and tested only after they are rationally proven to be good ideas.

WHY DO YOU ESTABLISH YOUR CONTINUOUS REVIEW PROCESS TO ENSURE QUALITY?

Your organizations heavily rely on reviews, audits, inspections and approvals just before production deployments. These inspections are usually conducted by people who are only remotely involved in your work, if any. Worst of all these inspection and approval authorities sometimes have incorrect understanding about your work, so you need to educate them just before you want to deploy your code in production.

Problems With External Control Mechanisms

In most organizations building control mechanisms is easier than building mutual trust. Therefore, adding more questions to change control forms, adding additional approval layers in the hierarchy and requiring extra lead times to understand and approve changes are typical problems with external control mechanisms.

Low trust environments of command and control cultures are doomed to live with a lot incidents which continuously repeat themselves. A vicious negative cycle emerges in these organizations because there is no clear transparency about real work performed, because there is no transparency about recurring problems, and because work is performed in large batches to keep number of painful deployments as low as it is possible. Longer lead times to deliver hurts the time-to-market competitiveness. Slow, late and non-actionable feedback to your teams makes learning from mistakes impossible.

A competent DevOps organization like yours know that: People who are closest to real work performed and its associated problems know the most about them. Therefore, having additional external control and approval mechanisms do not bring added value to your value stream. Which brings us to the largest management and leadership challenge of our time. Building high trust cultures where the work performed is transparent, where genuine respect and mutual trust are essence of daily work.

You need to shift reliance away from periodic inspections and start relying on peer reviews part of your daily work. Make your developers, operations

engineers, quality assurance and information security specialists in your DevOps team constantly collaborate and review work of each other to make safe deployments possible.

Coordination Of Complex Work Without External Control Mechanisms

It is already complex when one single team works on a project, but work can become even more complex when multiple teams work on different components of the same system. Some measures to keep you and your DevOps team on the top of your game are:

- Change boards who are subject matter expertises themselves, not only managers, can help you identify dependencies of various teams before changes happen.
- Loosely-coupled architectures based on SOA and micro services reduce dependencies and communication effort.
- Technical representatives of DevOps teams should identify dependencies, development and deployment sequences.
- Continuous communication with common chat platforms to synchronize changes and comprehend and review the understanding of others.
- More impactful and risky changes such as migrations or infrastructure changes require rehearsals on test environments and technical countermeasures such as failover and full-rollback mechanisms.

High Performer DevOps Organizations Rely On Peer Reviews And Less On External Approvals

Many prominent DevOps organizations including Amazon, Netflix, Etsy and GitHub use "**Pull Request**" Review & Deployment Process. Here is how it works:

1. The developer creates a separate development branch from trunk. This branch should not live more than a few business days and it should have a clear, descriptive name such as "login-screen-v2". The developer works on this branch locally on her workstation and regularly checks-in this branch to version controlling system.
2. When the developer thinks that the development branch is ready for merging trunk, she opens a "pull request" to ask review feedback.
3. After the developer gets review feedback and approval from other members of her DevOps team, she merges her code to the trunk. Then she deploys her code into production systems.

For high profile changes such as database schemas or changes that may impact information security of applications, the developer can also send additional "pull requests" to get feedback from other subject matter experts in your organization such as database administrators or information security specialists.

Small batch size principle ought to be used again while codes are being viewed. Otherwise reviewing code takes longer and this puts a lot of burden on reviewing engineers. When the change gets bigger, the risk of this change gets exponentially larger and the review becomes less reliable. If a change is too big and difficult to understand, the developer can be asked to split her change into multiple smaller and understandable chunks.

Everyone in your DevOps team regardless seniority and experience level must have someone review his or her own changes. Everyone should monitor commit streams, so potential conflicts can be quickly identified and solved before they cause larger issues in deployment pipeline, even worse during production deployments.

Pair programming by itself and pair programming in combination with test-driven development (TDD) (one developer writes code of application and another developers writes automated test) can enable your DevOps team to conduct peer reviews while the code is being written. Given the same team size and project requirements, it is measured and proven that pair programming takes 15% more time, but it reduces 85% of coding errors. Therefore, pair programming is an important practice you and your DevOps team should evaluate to use.

Furthermore, over-the-shoulder reviews, automated notifications of check-in steams and tools to assist code reviews and increased code quality are other alternatives you should be considering to implement.

CONCLUSION

Make every DevOps team in your organization be responsible for own quality of its deliverables. Build a culture that values continuous and high quality code reviews as much as writing high quality code.

HOW SHOULD YOU ENABLE YOUR DEVOPS CONTINUOUS LEARNING?

From your software projects you already know that: Even though you have checklists, peer reviews, control, audit and compliance mechanisms, you still have problems. This is inevitable. It is time for your DevOps team and organization to build a self-diagnostics, self-learning and self-improvement culture. Your culture accepts problems and your teams are ready when problems occur. Solving problems is not an exceptional state of work. But they must be part of your daily work to contribute on continuous learning and improvement journey of your organization. And you multiply the effects of these solutions for the problems you solve, by making them transparent, available and easily accessible within your entire DevOps organization.

One of the prominent DevOps organizations, Netflix, has built an in-house software (Chaos Monkey) to simulate catastrophic events in their cloud-based data centers. Chaos Monkey randomly destroys servers in production systems, so Netflix team can build additional assurance on their operational ability for resilience, stability and uninterrupted service quality for their clients. From each failure they learn new lessons and they exploit these lessons to make their systems even more stable and resilient.

Understanding Importance of Building a Learning Culture

In your organization if there is finger-pointing after incidents, this will create a fear culture for engineers. Thus, your organization simply becomes slow, bureaucratic and a political slippery landscape. Instead of consciously learning from errors, being organically more resistant and resilient against errors and being more mindful and careful to prevent errors. Everyone in such organizations care about self-protection. Work, problems and even solutions themselves are never fully transparent.

Because problems are inevitable in complex systems, instead of finger-pointing, blaming and shaming the ones who cause problems, your organization should value actions to make problems visible in your daily work. It should encourage organizational learnings from errors and inefficiencies, so everyone in your DevOps organization can also learn and profit from these problems, solutions and knowledge.

When engineers in your DevOps organization feel safe about giving details about mistakes, they voluntarily go extra mile and spend a lot of energy to make sure that a similar problem will not happen again in their own work center and in other work centers in your organizational value stream. If engineers are punished or even if they feel that they are punished when they do mistakes, then they will be afraid of making mistakes, so

1. They produce less work to do less mistakes.
2. They are not transparent about work, problems and solutions.
3. They are not incentivized to convert solutions of problems into organizational learnings.
4. It is guaranteed that the same or very similar problem will happen again because nobody ever spends time and energy to learn, share and teach about problems/solutions and make them visible.

Run Post-Mortems As Soon As Incidents Happen, Before Memories About Problem Causes Fade

The goals of a post-mortem review are very simple:

- To identify the things you did right, so that you can remember to try them again in similar situations.
- To note the things that should have been done differently, so that you can refine your techniques in the future.
- To note the things that you did wrong, and to suggest alternative approaches or safety measures that you should employ the next time you face a similar problem.
- To find out why it did make sense to take (or not to take) the action which caused in the incident.

Exploring what you did wrong is frightening and in some organizations it is dangerous. If admitting having made mistakes opens you to criticism or discipline, you are unlikely to make such admissions. This strategy is ultimately self-defeating, since failing to understand a past mistake usually condemns you to repeating it again in the future. Organizations that are serious about improvement understand this, and take trouble to create a process and culture wherein it is safe to explore mistakes.

When you enter into a post-mortem review process, you must accept a few basic premises:

- Everybody tries to do their best, as best they understand it.

- You make our decisions in stressed situations, with imperfect information.
- You are often called upon to carry out tasks for which you have not been trained, with whatever tools and resources happened to be at hand.
- Mistakes are inevitable in such situations.
- The goal of this process is not to find fault with any individual or their actions. Rather it is to look at what happened and see what lessons you can learn from it.
- The output of this process will not be an assessment of any person or group of people, but rather an assessment of our processes, and how they can be improved.

It is absolutely essential that everyone involved completely accept this "**No blame, We are here to learn model**". Many organizations go to great trouble to create such safe environments. The FAA, for instance, has an Aviation Safety Reporting System, whereby pilots who make "mistakes" can gain immunity from regulatory discipline if they report those incidents.

Post-mortem reviews must always define actionable measures to prevent the incident from happening again in the future. New Telemetry metrics, new automated test cases, identification of type of changes that require additional code reviews, refactoring code or decoupling complex system components which cause frequent problems can be examples of such preventative measures.

Publish post-mortem review protocols and lessons learnt widely in your organization. This will help you convert your local learnings from one work center in your value stream into organization-wide global learnings. And this will be a clear message in your DevOps organization to nurture transparency, openness and learning culture.

Organize Game Days To Improve Your Systems

A game day is not one of your typical boring team events where extraverts enjoy the show and introverts play with their mobile phones to speed up the flow of time.

In a game day catastrophic failures are simulated in your test systems. And DevOps teams work towards fixing and learning from these failures.

For instance, a critical server is terminated to validate the successful operation of failover mechanism

without service interruptions. Then your DevOps team validates if/how your recovery mechanism from backups or from your Infrastructure as Code (IaC) works. Identifying problems in these fail scenarios helps your DevOps team build resilient, fault-tolerant systems and create learnings.

During the process of solving problem, your DevOps team builds relationship with other departments while they rehearse fail events in non-stress conditions. You will test and have a visible chance to improve communication and troubleshooting processes within your larger global organization.

Furthermore, you will have the ability to observe weaker signals for potential larger issues that may reveal themselves in the future. Frequently happening low priority incidents during these fail scenarios, or a small side effect that may have come close to crash another critical component in your architecture are important week signals that you should take into account and work out to improve your systems.

CONCLUSION

In your DevOps team encourage calculated risk taking. High performer DevOps organizations like yours do more often errors. This is not only OK, but this is also what your organization needs. To learn and perform better.

Over typical organizations, high performers have 80% less critical failures in their production systems. In other words they have 5 times less incidents which impact their clients. This is why your engineers in your DevOps organization needs to feel free to do errors and learn from them.

WHY DOES YOUR DEVOPS TEAM NEED TO BLOCK TIME TO ENHANCE THE WORK?

It is not a new technique from DevOps to block your time to enhance your work. Lean Manufacturing decades ago embraced the motto: "**Improvement of work is more important than the work itself.**"

In other words, this implies, without improving the way you work today, you work will be obsolete tomorrow. And ultimately this will in return hinder your competitive strength in your market. This is why DevOps methodology borrowed this technique from Lean Manufacturing to make your teams and your organization continuously learn.

Here are some ideas from DevOps software development and delivery methodology about how you and your DevOps team can use your time to enhance your work and continuously learn:

1. **Feedback from External People:** Concentrated support of people outside your business processes to the individuals inside your own business processes. Feedback from outsiders can be really helpful and eye opening to improve processes and remove inefficiencies.
2. **Regularly Reduce Technical Debt:** You regularly bring your team together to stop working on new features and work towards reducing technical debt. Engineers whose skills spanning in your entire value stream work on (not only discuss) code, environments, tools, components and architectures to reduce technical debt, so your new features will be built on a stabler and stronger foundation.
3. **Regularly Reserve Time for Innovation:** Free a certain percent of work time of your engineers, so they can use this time to innovate your technology and business. Google is one of prominent DevOps organizations which use this practice since years. Google employees have 20% unallocated time which they use to develop prototypes, products, features, test suites and solutions. Gmail, Google Maps and AdSense are a few of major products innovated within this 20%.
4. **Foster Teaching and Learning:** Dedicate time for teaching. Organize internal conferences, workshops, mentoring, coaching, counselling to regularly teach your teams. Developers learn about operational work and their challenges and

operations engineers learn about development. This will help create and maintain a stronger mutual understanding about work of each other. In return your developers and operations engineers build an informed and personally connected foundation of cooperation. In addition, encourage your DevOps teams to go and join conferences to better understand how other organizations solve similar engineering and operational challenges like yours.

5. **Make Teachers out of SMEs:** Make sure your subject matter experts have free hours when anyone can go and ask questions to discuss matters which may not be even related to an ongoing project. We are life time learners. Your team does not only respect this, but also they know this because they are life time learners too.
6. **Organize Hackathons for Innovation:** Here is how Facebook's Marc Zuckerberg explains this idea: "Every few months we have a hackathon, where everyone builds prototypes for new ideas they have. At the end, the whole team gets together and looks at everything that has been built. Many of our most successful products came out of hackathons, including Timeline, chat, video, our mobile development framework and some of our most important infrastructure like the HipHop compiler."

CONCLUSION

By regularly scheduling buffers for improvement, you enable everyone in your organizational value stream to get ownership of innovation and quality. Therefore, all engineers in your DevOps team continuously integrate safety, quality, learning, reliability and improvement into the daily work which are in the end integrated into software products and services your organizations offer to your clients.

By helping each other in your DevOps teams, your organization will overtake its competitors and your team members develop their utmost potentials as humans and engineers.

HOW DO YOU ENABLE ORGANIZATIONAL LEARNINGS FROM DAILY DEVOPS WORK?

One of main principles you and your team capture from DevOps is: You learn continuously from your daily work, and you convert these learnings into reusable global assets for your entire organization.

Use Chatrooms To Spread Knowledge

Use chatrooms to capture and create organizational knowledge. Build bots which are integrated to your chatrooms to quickly perform operational tasks for you. For instance the following message you address to your bot in the chatroom can install FrontEndApp application in TestServer-9 environment, and print the outcome of this command back in the chatroom as if you run the command from your command line interface.

@bot install FrontEndApp TestServer-9

Thanks to this mechanism everyone in your team can quickly synchronize on progress of your work. No information is lost in private instant messages and emails. This nurtures a culture of collaboration and transparency with the work you do. And this is a very effective way to convert local knowledge into organizational learnings.

Last but not least, using common chartrooms speeds up onboarding process of new joiners to your DevOps team or to your company, so they can easily monitor a substantial part of how the work in their teams is performed. You can consider using separate chat rooms for different products, services, components, tools, libraries or integrations, so it will be easier to search and find past activities associated with a certain element in your system.

Put Your Focus On Executables Rather Than Documentations

Instead of putting your knowhow and expertise into documents or Wikis, build executable standards and processes as much as it is possible. One of the best ways to build and share knowledge in your DevOps organization is to create a reusable tool and store it in your code version controlling system, so anyone who needs something like this can find it. When you express a standard and the way you work as

executable code instead of static documents which can wrongly and differently interpreted based on who reads it, this code will not only speed up and simplify your business, but it will also support your DevOps organization to pass various audit and compliance checks given that the code has at least a lightweight document which explains how well the code handles the process which requires audit and compliance.

For the work, processes and standards which cannot be automated, create reusable users stories with clear and distinguishable steps and checklist, so everyone in your DevOps organization can reuse and take benefits of such assets. Thanks to these reusable assets, different teams can consistently do similar work. And they improve these assets just like they continuously review, reuse and improve code libraries.

Create Reusable Software Blueprints

Create ready to use blueprints for components, applications, services and micro-services which handle non-functional requirements built-in by design. In this way, developers will not have to worry about non-functional requirements each time they write a new code.

Furthermore, your organization will have a consistent set of non-functional requirements implemented across your entire application stack. In addition, these built-in implementation of non-functional features will simplify deployment, monitoring, trouble-shooting and operations of software in general while it is in running in test and production platforms. Examples of such non-functional requirements are:

- Telemetry metrics.
- Runtime configurations.
- Feature toggles.
- Ability to track dependencies.
- Ability to trace requests initiated by users.
- Resilient and fault-tolerant services.
- Services which can be gracefully downgraded on demand.
- Ability to search log messages.
- Compatibility to backward and forward versions.
- Data archiving and management of production data sets.
- Management and archiving of logs.

Comprehensive automated testing needs to be part of your coded blueprints. Automated tests clarify purpose of your code, they clarify how to use reusable code most efficiently and they speed up composition of automated tests for your custom applications inherited from reusable code blueprints.

Use One Single Code Repository

To make sure that all of your learnings and work are shared with everyone, you do not only store source code in your one single repository, but you also store tools for deployment, code to build and execute deployment pipeline, automated tests, code to build and integrate telemetry, standards, tutorials, documents, wikis, helper tools and configuration standards for the platforms.

In this way, all assets are transparently shared with everyone in your DevOps organization. Duplications, multiple versions of same artefacts, components and 3rd party tools are prevented. One repository also enables you to compile, link and build everything at runtime and to use latest version of everything when you develop, test and deploy your software.

Have Technology Standards, But Stay Flexible

If you want everyone in your DevOps team to read, review and fix each other's code, define your main stream technology standards which everyone in your development and operations teams can learn and work with. What you need is: One compile language, one scripting language and one GUI language. But still let everyone to explore whatever technologies they are interested in, to enable your DevOps team to continuously learn and experiment. How are you ever going to win with your next innovation if you are not exploring and testing at the edges?

CONCLUSION

In this chapter we summarized for you some of widely accepted DevOps techniques which will enable your DevOps organization to learn and create assets while you are performing your daily work. By leveraging these methods, the cumulative experience and expertise of your entire DevOps organization will backup and empower professional expertise and work quality of each individual in your DevOps team.

Good luck with your DevOps journey!..

THANK YOU

I would like to thank you again for taking the time to read DevOps Revealed. We hope that you enjoyed reading this book as much as we had enjoyed while we were writing it. It is our biggest pleasure if we by any means manage to help you build a strong DevOps foundation for yourself.

We know that it's a very complex, overwhelming and overcrowded world with all DevOps certifications out there in the market.

And yet we managed to build your Official DevOps certification programs more concrete, attractive, helpful, useful and simpler than our competition did. This is why we believe our valuable students choose International DevOps Certification Academy™ over bureaucratic, complex, expensive and half-baked solutions of our competitors.

Our one-of-a-kind industry leading registration, examination and certification process is very simple, quick and completely online. Click on this link to read all details: **Official DevOps Certification Programs**.

Jenny Evans,
International DevOps Certification Academy™