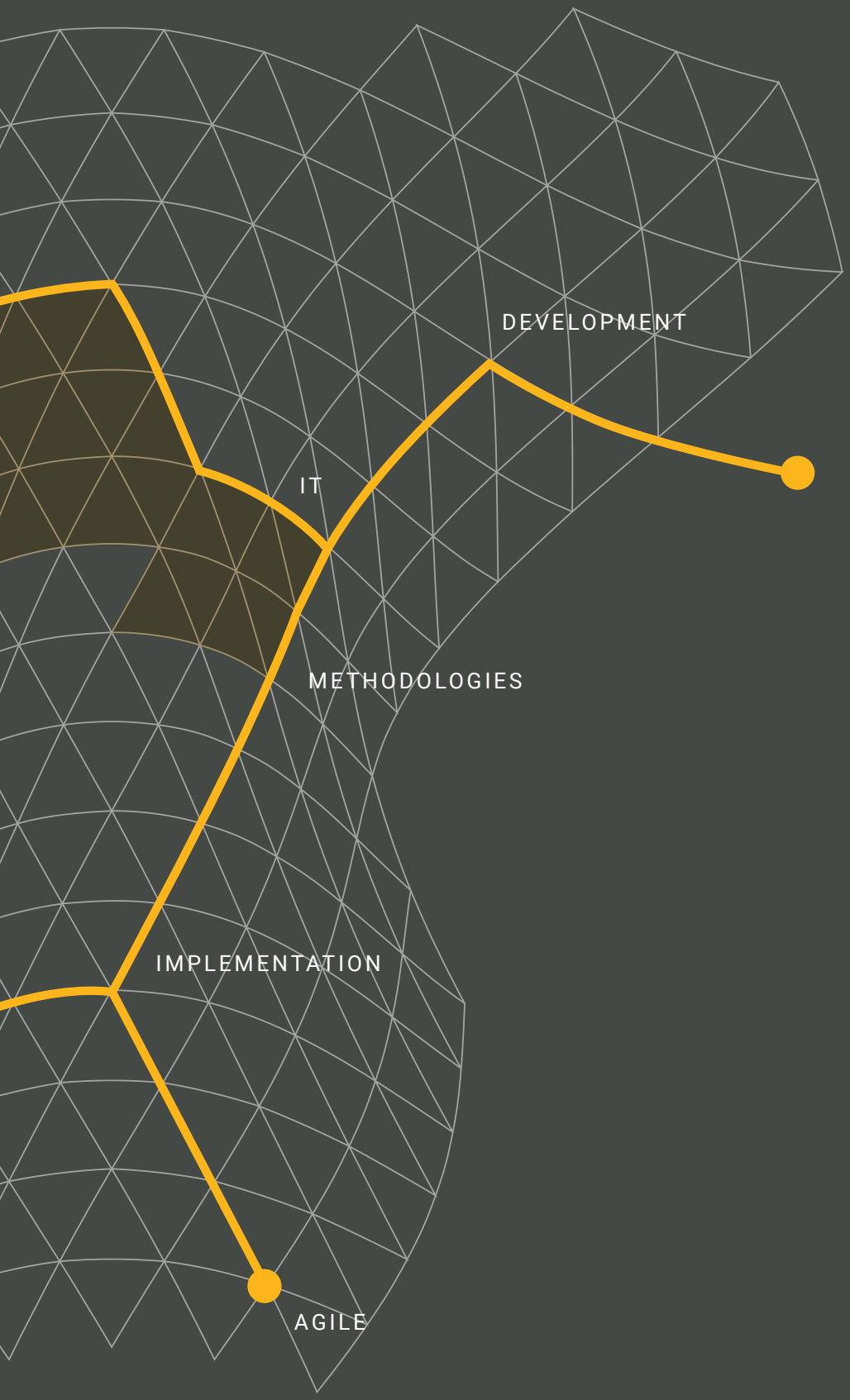
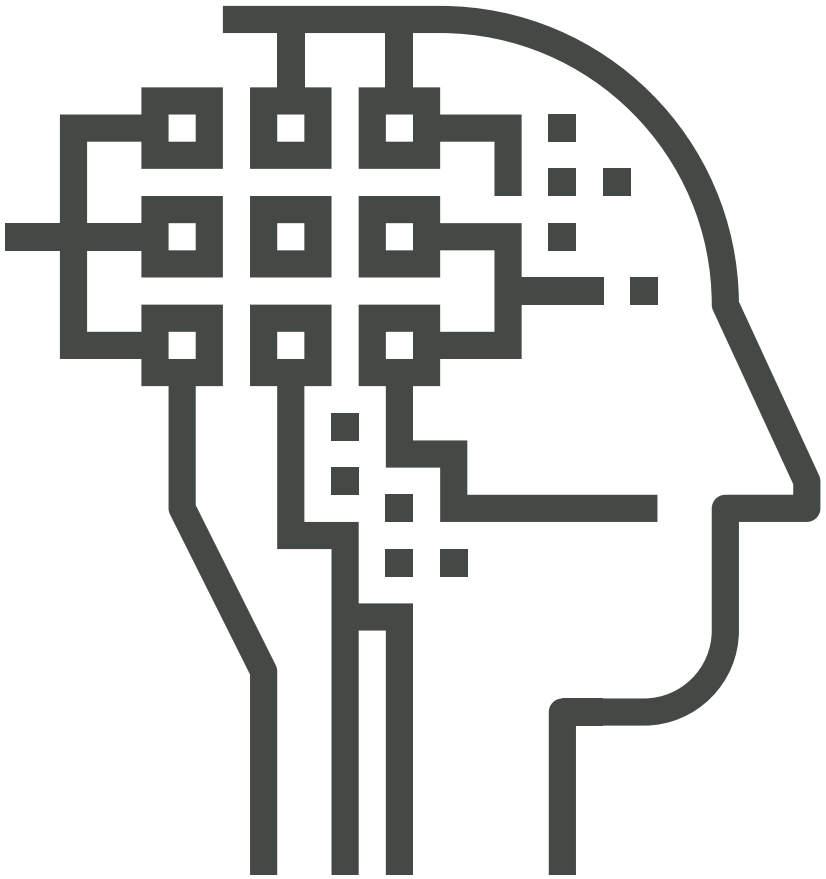
modis



DevOps

The manifestation of the Agile ideology.



How DevOps can bring productivity to the forefront of development.

DevOps is a manifestation of processes that further enhance Agile software development practices. In other words, the ideologies of Agile and DevOps are intertwined, yet are separate in execution. To better understand DevOps, one must have an understanding of the Agile philosophy.

While there are multiple Agile methodologies, each with its own approach, all share a common vision and the same core values. In essence, Agile dictates the practices and characteristics of software development with the goal of accelerating the development and deployment of software while supporting the processes of continuous improvement.

Agile methodologies incorporate iterations and continuous feedback cycles that successively refine and deliver a software system. Agile also involves continuous planning, continuous testing, continuous integration and other forms of continuous evolution of both the project and the software.

Agile brings lightweight processes that are inherently adaptable to developers, and more importantly, focus on empowering people to collaborate and make decisions together quickly and effectively.

The benefits of DevOps.

The DevOps ideology arose as a result of the increased software velocity and output Agile methods have realized. Its primary goal is to improve collaboration between development and operations to create an environment where all stakeholders participate in the planning and delivery processes. The DevOps ideology also serves to provide tangible benefits, including:

1

Shorter Development Cycles

DevOps promotes increased collaboration and communication between the development and operations teams. That shortens timeframes to move from engineering code into executable production code.

2

Increased Velocity for Software Releases

Shorter development cycles increases the frequency for release of code into production. Releases can be reduced to a daily, if not hourly, release build cycle. That nurtures continuous development and deployment, and increases the value of IT to the business. Increased release velocity also creates a competitive advantage by delivering market features that customers need.

3

Improved Defect and Bug Detection

DevOps builds on top of the Agile methodology and leverages several Agile principles such as collaboration, iterative development and modular programming, breaking larger codebases into smaller manageable features. That eases detection of code defects.

4

A Reduction in Deployment Failures

Software developed using a DevOps mindset takes into account operational viewpoints which, when combined with improved defect detection, significantly decreases the number of pre- and post-deployment issues.

5

A Reduced Need for Rollbacks

A reduction in pre- and post-deployment issues results in a reduction of rollbacks, saving time and further accelerating iterations.

6

Faster Recovery Times from Failures

If a failure occurs, the return to operational efficiency is reduced with a DevOps environment. A result of the efficiencies gained by tearing down the silos between development teams and operations teams is a cross team understanding of work methodologies, which allows for a more robust versioning process to accomplish fast rollbacks.

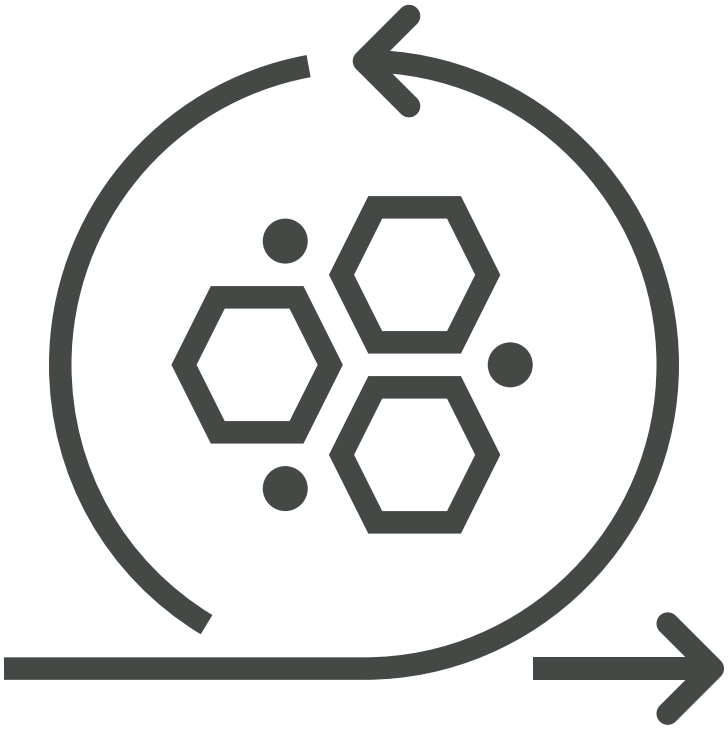
According to the Puppet State of DevOps Report 2017, respondents identified as high performers (having mastered DevOps) were showing significant achievements over low performers (those not fully committed to DevOps). High performers achieved 46 times more frequent code deployments, 440 times faster lead time from commit to deploy, 96 times faster mean time to recover from downtime, and a five times lower change failure rate than the low performers identified in the report. Research firm Gartner further makes the case for DevOps with a prediction that 60% of IT organizations will embrace DevOps initiatives by 2021, an increase from less than 5% today.

The iterations of DevOps.

Jumping on the DevOps bandwagon means embracing the ideologies associated with workflows. Conceptually, workflows act as a plan of execution, defining tasks along with the necessary requirements that indicate a task has been completed, then automatically moving onto the next task.

From a practical standpoint, workflows are usually scripted and executed by a workflow engine, which provides a visual representation of the workflow. In other words, those using the workflow have a map-like process diagram that makes processes and procedures clear to those responsible for completing the tasks defined in a workflow.

The workflow is the foundation of a successful Agile DevOps deployment.



Workflows at a glance.

A workflow offers the ability to build a complex set of repeatable tasks that can be executed automatically. The hallmarks of a DevOps workflow are velocity, scale, consistency and the ability to integrate feedback. An organization may have multiple workflows for a given project and those workflows may drive additional workflows based upon Boolean logic. In other words, workflows can quickly become complex, which to many is the antithesis of what an Agile powered DevOps deployment is all about.

Regardless of the workflow design tool used, all workflows start with adopting the best practices that lead to success. Most DevOps workflows can be broken down into achievable steps or processes, each of which is related to a best practice. Typically, a workflow should address:

Requirement Management

As the name implies, it all comes down to identifying the operational requirements and understanding the expectations of the stakeholders. Here, issues such as performance, SLAs, manageability, security and operational architectural constraints are identified and are attributed as requirements of the solution.

Architecture and Design

This is where the design of the application is defined and the resultant architecture is identified, creating a boilerplate of application expectations. Critical design cues should be incorporated at this point, such as the ability to perform automated testing as well as identifying the architecture specifications of the staging environment and production environments.

Coding

There is more to Agile development in a DevOps environment than just creating application source code. During the development of that source code, team members should also be creating testing scripts as well as the necessary infrastructure code. Since quality assurance and extensive testing are part of the Agile process, it is critical that any source code has corresponding test developed at the same time. This step in the workflow should also reflect everything that is required to build and deploy the application.

Build

Normally, the build process is integrated with whatever source code management system is in place. The idea here is to make sure that builds can be done automatically and can be readily repeated with each iteration of the application. The critical element to realize is that builds are logged and have some sort of accountability associated with them so that forensic processes used for build failures can be used to improve future

Deploy

For the most part, deployment should be automated, only requiring minimal administrative interaction to successfully deploy the application. Deployment and configuration should go hand in hand, meaning that the deployment engine in use should also handle automating configurations and eliminate the need to manually change any settings or other elements.

Test

While automation can help to speed workflows and bring additional velocity to the DevOps lifecycle, it is still critical to fully test everything, including the operational aspects of the application.

Release

Once operational and functional testing is completed, the next step is to release the application. The release process includes provisioning, deployment, additional testing, making the final deployment decision, and then rerouting production to use the new code (if necessary).

Build and deployment methodologies are a critical element of workflow design. While the goal of a workflow is to deliver an application, there are other considerations that must be taken into account:

Continuous Integration

A software development methodology where building and testing of code is achieved with every code commit. That gives teams instant feedback and helps to resolve problems before any code makes it to deployment.

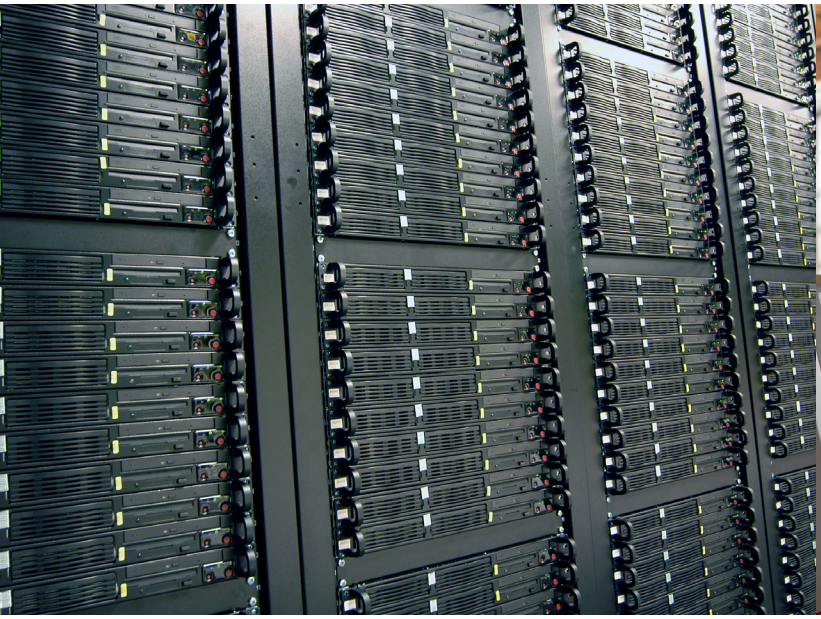
Continuous Delivery

A specific development practice that exemplifies a repeatable and reliable process for releasing quality software. This consists of continuous integration, automated testing and automated deployment capabilities functioning together to achieve Agile results.

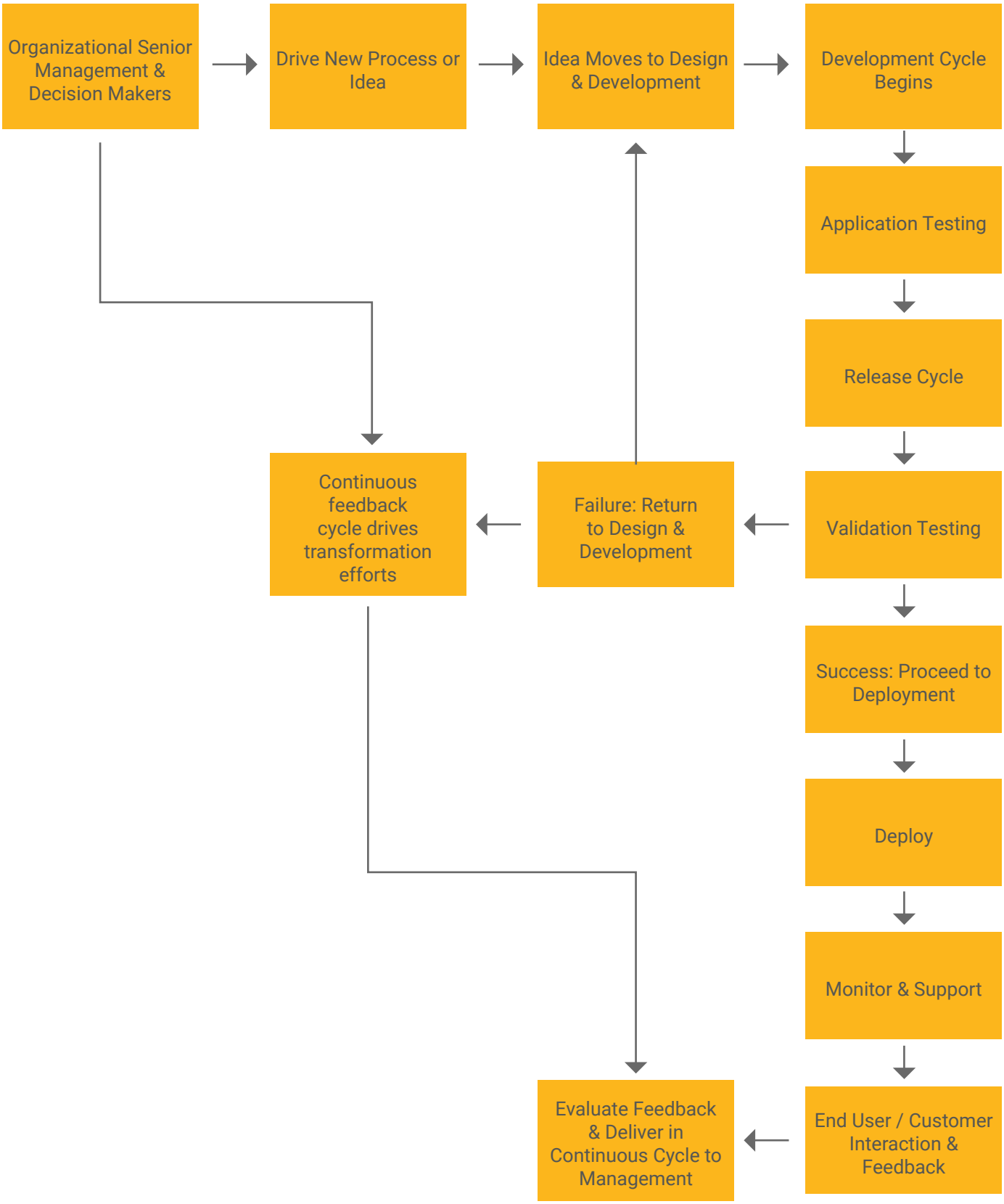
Continuous Deployment

Sometimes confused with continuous delivery, it is a methodology that creates an environment where code is built, tested and deployed with every commit.

Each of the above methodologies can impact how a workflow is constructed and processed. For many startups and those undergoing digital transformation, the practice of continuous delivery offers the best fit for those adopting DevOps and committing to the Agile ideology.



Example of Workflow Design



Quality assurance

A Key Component of the Process

The very nature of DevOps means that businesses must strive toward continuous improvement, indicating that quality assurance must be an essential part of the development and deployment cycle. As mentioned previously, when continuous feedback is monitored and logged, it should be included in any decisions surrounding overall application quality. This should be applied to more than just the actual end product–it should also include the processes and interactions of the associated teams, allowing problems to be identified before any major impact.

Deploying a DevOps ideology.

Successfully adopting a DevOps ideology may require a cultural shift in both operations and development departments: it takes a new set of commitments from staffers and aims to tear down the artificial silos that development and operations have come to inhabit. For example, developers have to extend their responsibility beyond their code and take full ownership of the development, testing, automation tools and, most importantly, building operational capabilities into the pipeline so the move to production is easier. Those on the operational side of the equation will have to foster open communications with developers and strive to support them with needed tools, being involved with planning from the outset. Achieving that level of culture change has proven challenging for many organizations, but it is far from impossible, as evidenced by the numerous success stories associated Agile DevOps teams.

Strategies for success.

1 Prepare for a Culture Change

DevOps and Agile methodologies require a different mindset and changes to established development and operations cultures. This is probably one of the most critical strategies to successfully move to DevOps. Luckily, the transition can be eased by investing in both time and training to accomplish the collaborative environment that DevOps demands. There are numerous avenues to get training and advice, further reducing what was once an insurmountable task.

2 Choose an Automation Platform

Volumes can be written on the best practices for choosing an automation platform simply because no two businesses are exactly alike. The trick here is to choose a platform that meets defined needs. For example, if the goal is continuous delivery, then the automation platform should include the ability to accomplish phased rollouts. Other features needed may be the ability to perform A/B testing with an automated rollout based upon test results, or support for advanced release management.

3 Architecture

With Agile and DevOps steeped so deeply into the cloud, it only makes sense to leverage the ideologies that make the cloud flexible, scalable and ultimately Agile. That may mean shifting to technologies such as microservices and containers, which will transform applications into cloud native solutions. The idea is to be able to independently and frequently update each service (or application) without disrupting operations. Cloud native applications using microservice architectures and advanced deployment techniques allow teams to bypass certain stages and testing in production, increasing deployment velocity.

4 Security

Automating code scanning, incorporating template checking and performing other security tests during development leads to more secure applications. Before DevOps, businesses using waterfall-based ideologies left security as one of the last steps before deployment—a practice that created numerous security problems. Security should be part of the DevOps pipeline itself and automated as part of the DevOps process. The key is to validate security without slowing down the process.

5 Visualization

Garnering insight from workflows, pipelines and test results is critical for team members to understand the impact of changes. Providing a common view of the variables associated with development and the results of test data improves both collaboration and traceability.

6 Develop Delivery Plans

Automated testing is a core element of the Agile DevOps process. It should include test cases and test suites that are mapped to the functionality and business outcomes desired to ensure that only properly vetted services are moved to production.

7 The Feedback Loop

Continuous feedback is important to create an environment of constant improvement. Deploying methodologies that can gather feedback based upon the user experience, response times, performance of key business services and the technical metrics from all stages of the life cycle is critical for reducing errors and increasing the frequency of releases.

8 Map Old Processes

The only way to introduce something new is to fully understand what is being replaced. For DevOps to succeed, old processes must be mapped and understood to ensure minimal disruption and that no previous functionality is lost.

Best practices.

Automate Everything

While automating testing and builds is a given in the world of DevOps, other automation opportunities exist, thanks to the cloud. Processes such as provisioning, VM or container deployment, middleware configuration, and application code processing can be automated, helping to ensure consistency and repeatability.

Test Everything

Quality can only be fully achieved if everything is tested. That means microservices, containers and VM configurations must be tested, as well as middleware installation, deployment scripts and the application.

Version Everything

This is where a Version Control System (VCS) goes a long way toward ensuring consistency. Everything should be kept in a VCS system, including code, deployment scripts, infrastructure definitions and test cases.

Plan and Track Everything

This includes OS changes, patches, updates to test cases, bug fixes and so on. The key here is to tie every action to a work item so that those collaborating can see the whole picture of what has been done and why.

Monitor and Audit Everything

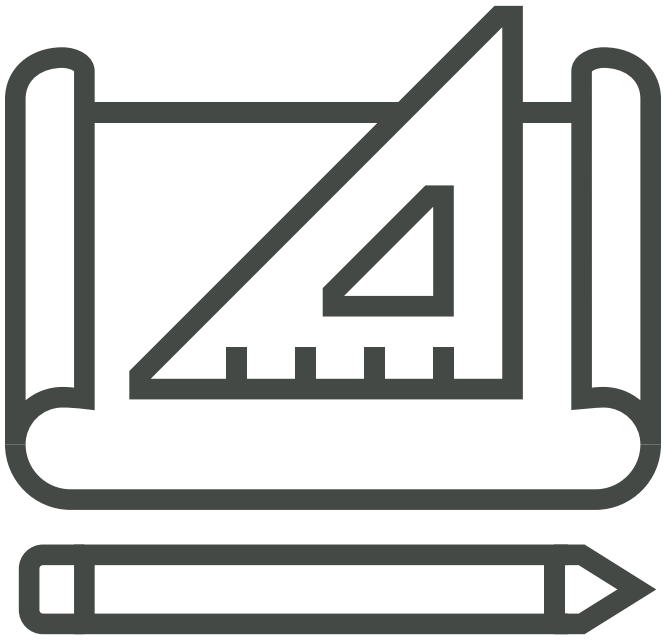
Applications and processes can be monitored using agents, which can deliver critical insights into usage, performance and other elements. Logs can be audited to capture deployment actions and other changes, helping to ensure accountability.

Dashboard and Report on Everything

Application and system performance can be relayed in real-time using dashboards, which helps to identify problems such as bottlenecks and errors. Reports can further narrow down those problems based upon time frames or filtered elements.

Infrastructure as Code (IaC) to Version Everything

Adjusting code creates opportunity for error. These can be prevented by implementing Infrastructure as Code (IaC). IaS is an infrastructure that protects your live assets by duplicating them in the developer's environment, giving them access to what's in development without user involvement. This allows developers to focus on advancements instead of core developments and creates even more efficiencies for the business.



Measure twice, cut once.

Measurement is one of the key tenants of DevOps: everything is measured and judged based upon set criteria. Judging the success of a DevOps operation can be challenging, but it is critical for a business's success to know how DevOps is performing. There are several Key Performance Indicators (KPIs) that can be used to judge how effective DevOps is:

- **Deployment Frequency:** How often does the DevOps team deploy new code?
- **Volume of Change:** How much new code is being deployed?
- **Change Complexity:** How complex are the new changes?
- **Lead Time:** How much time occurs from new code development to deployment?
- **Failed Deployments:** How frequently are negative outcomes encountered?
- **Recovery Time:** How long does it take to recover from a negative outcome?
- **Customer Support:** What is the frequency and volume of customer support requests?
- **User Volume:** How many users are accessing the system? Is it increasing?
- **Availability:** What is the system uptime?
- **Performance:** What is the latency, speed and usability of the applications?

Each of those metrics offer an insight into DevOps performance. For the most part, each element should decrease over time, except for user volume. Some elements should plateau over time, such as deployment frequency and availability.

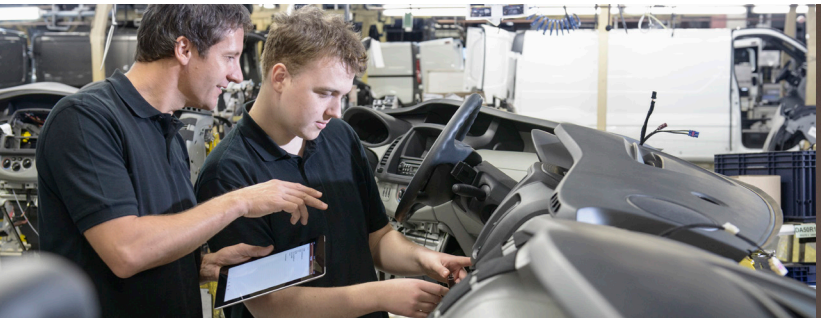
Ultimately the goal of DevOps is to improve on velocity, quality, productivity and security, each of which can be measured in its own right. A failure in any of those KPIs can indicate that there is a serious problem with DevOps adoption and should be addressed immediately.

The ROI conundrum.

Like any business process, DevOps falls under the auspices of measuring ROI (Return On Investment). While some may claim that DevOps incurs too many intangible elements, there are still plenty of indicators that can demonstrate if DevOps is worth the money spent.

Some key metrics to measure include:

- **Release Frequency:** How fast code makes it to production and the pain points it resolves
- **Infrastructure Recovery:** How quickly production systems can be restored after a failure
- **Infrastructure Resiliency:** Reduction in downtime caused by infrastructure failures
- **Infrastructure Efficiency:** Impact of production deployment issues
- **Automation:** Reduction in human intervention for deployment, problem resolution and support



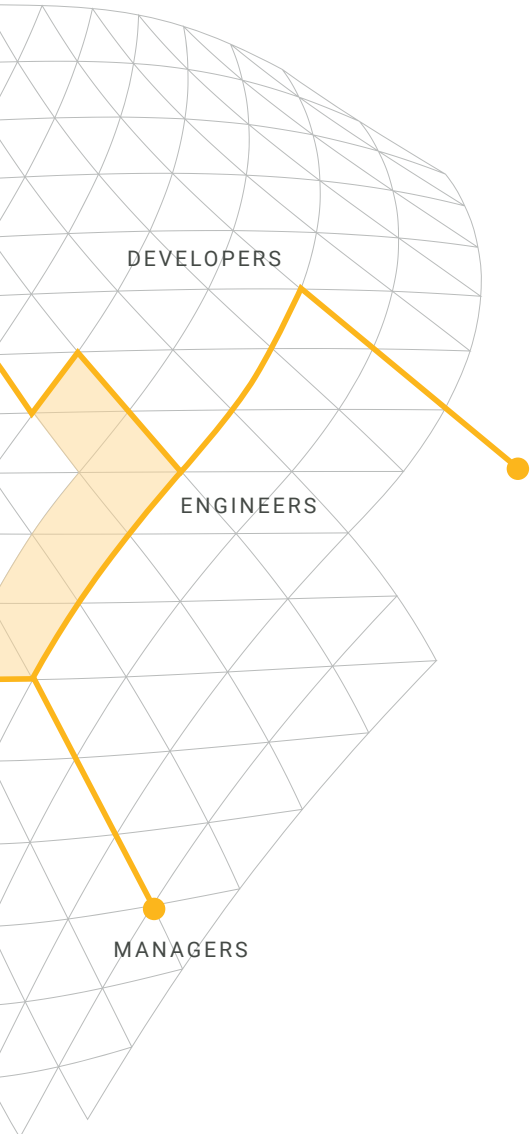
Finding the right fit.

Getting Started with DevOps

Select a Capable Partner: Outsourcing DevOps adoption can offer significant advantages to an organization. Choose a vendor that has specific experience with successfully integrating DevOps, and has experts on hand that will offer advice on the best way to grow. Outsourcing removes much of the trial-and-error guesswork that businesses would experience on their own. The insight offered will help to implement a strong strategy from the onset rather than searching for one that works. The business will then see higher returns with a capable plan and can reap benefits faster than ever.

Start Small with Automation: One of the first places to start with DevOps is automation, as it directly links into other Agile processes. Automation is also beneficial as it helps organizations accelerate and streamline their projects by executing repetitive tasks with software rather than manually. Automation frees team members from tedious activities, enabling them to address conflicts earlier and build a fully auditable trail that can be leveraged for future strategies.

Adopt Test-Driven Development: One element of the Agile ideology is to adopt strategies that lead to faster decisions and better operations. The goal is for developers to write the minimum amount of code in order to pass the test. They then refactor the code to eliminate redundancy. This process is repeated as the project evolves and offers an incremental way to add value while ensuring that it remains viable with each new iteration. If any code fails, risk is reduced and the time needed to correct is greatly reduced.



The DevOps team.

The typical DevOps team requires collaboration across multiple functions and shared responsibilities to ensure the best possible quality, a concept that can wreak havoc across an organizational structure. One of the first steps for building a DevOps team comes in the form of shredding the org chart and starting new. Ideally, roles and responsibilities will be defined for the team and will bring together a project-based team of professionals ready to collaborate and cooperate.

Most teams will consist of:

- **IT Manager:** Someone charged with creating a climate of continuous improvement and responsible for delegating authority to the team members
- **Dev Manager:** Works closely with Ops Manager to align goals and create plans
- **Ops Manager:** Brings ops expertise to the table to incorporate it with the dev process
- **Systems Engineer:** Focuses on the automation aspects of the DevOps platform
- **Quality Engineer:** Works to improve scale, performance and validate code
- **Developers:** The developers who code and move elements through the pipeline

Of course, team members can take on multiple roles or teams can grow in size where functions are shared or handed off to others.

There is no denying that when adopted correctly, DevOps can deliver on the promises set forth by Agile ideologies, bringing faster development cycles to fruition, with a reduction in errors, and the ability to react quickly to change. What's more, embracing a DevOps ideology can help improve both the importance of IT operations and development teams, while better serving the needs of the business.

Sources

https://blog.newrelic.com/2014/05/16/devops-name/ - Fredric Paul

https://www.linux.com/blog/what-devops-patrick-debois-explains - Dawn Foster

http://Agilemanifesto.org/ - Agile Team

https://puppet.com/resources/whitepaper/state-of-devops-report?pcnav=off&pctiles=off&ls=Campaigns&lsd=Search&cid=7010f000001eViJ&utm_medium=paid-search&utm_campaign=Q2FY18_AMER_All_CAMPGN_SER_ADWRDS_2016-DO-sal-rpt&utm_source=google&utm_content=devops-salary-report&gclid=CjwKCAjwqcHLBRAqEiwA-j4AyCucF6wNK2IGWzv-1wRQqNUSGkTUtcsP6W3o3b0HXJpj49jKJs9XARoCRJAQAvD_BwE

https://twitter.com/i/web/status/863815191317839872 - Gartner

http://www.drdobbs.com/architecture-and-design/top-10-practices-for-effective-devops/240149363 - Scott W. Ambler

http://www.techrepublic.com/blog/10-things/10-best-practices-for-devops/ - Mary Shacklett

https://devops.com/devops-best-practice/ - Jayne Groll

https://blog.newrelic.com/2014/06/02/devops-tools/ - Eric Wilinkski

https://devops.com/9%C2%BD-simple-steps-start-devops-today/ - Sven Malvik