DevOps World

The concept of DevOps has grown and evolved into a conceptual and working model for more effective and efficient software development and implementation. That said, there are some differences of opinion on the real-world value of any DevOps approach to date, and on the best way to create and implement a real-world DevOps environment. This two-part article will focus on what an agile DevOps approach is meant to address, and what it is not meant to address.

DevOps sits at the nexus of three essential business technology functions: software development, quality assurance and operations. A short and concise definition of DevOps proposed i seems as appropriate as any:

DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into regular production while ensuring high quality.

The definition was suggested in the book, "DevOps: A Software Architect's Perspective," and the authors have hit upon the essence of the practice. The key, of course, is how to put that concept into practice.

Perhaps the first step on the journey to effective DevOps is the recognition that the concept is the result of the rise of the Lean and Agile software development methodologies. Those methodologies, among other things, emphasize the following:

- ✓ A Focus on Customer Value
- ✓ The elimination of waste.
- ✓ Reduced cycle time (accomplishing work faster, releasing faster).
- ✓ Shared learning.
- ✓ Avoiding batching (don't do things until required).
- ✓ Theory of constraints (break things up, focus on individual issues).
- ✓ Continuous integration, testing and delivery.
- ✓ Faster time to market.

DevOps in Practice

Adherence to the principles above meant that something had to be invented to accomplish them and that something was DevOps. Over time, an effective DevOps practice should address any number of business technology pain points. The following short list of those pain points and the DevOps response should prove instructive.

System Downtime

The developers' curse since systems were developed, system outages are inevitable as long as systems are designed, tested and implemented—even with increased automation—by imperfect beings. DevOps acknowledges that by changing the focus from trying to create applications that never fail to designing systems that can recover quickly, thus decreasing aggregate systems outage time over the life cycle of any application or system.

Stagnation

This was a staple of traditional systems development and is most closely associated with the waterfall methodology for systems development. After requirements were created, the development team would be locked away for weeks, months or, in some cases, years before emerging with "fully" working software that inevitably no longer satisfied rapidly evolving business requirements. DevOps is designed to fit hand-in-glove with the Agile practice of short windows of incremental changes instead of long release cycles, putting working software in the hands of customers as quickly as possible.

Team Conflict

Having been borne from the cultural combination of Agile and Lean, DevOps has taken on the problem of functional silos that are often erected between development, operations and the business customers. It follows the methodological approaches of collaboration and teamwork first to understand what others know and to leverage the best of it to solve business problems more rapidly. There is also a cultural bent toward experimentation, continual learning and constant improvement. This leads to blameless post-mortems, where instead of finger pointing when something goes wrong there is collaborative discussion and learning to correct and prevent the problem from occurring again.

Knowledge Silos

Functional silos have led to compartmentalized knowledge. If the old game was that knowledge is power, the new game in the DevOps world is that knowledge is freely exchanged as an enabler to solving business problems. DevOps addresses the problem of information being lost in translation between the development and operations functions by eliminating the functional barricades and making knowledge sharing the highest form of collaboration.

Inefficiency

Waiting for things to happen used to be a standard operating procedure in the pre-DevOps world. Project plans were created and managed to account for the time it might

take for new code to be moved into a testing, quality or even production environment. This was a momentum killer for projects and at times a morale killer for developers waiting to see what changes they might need to make to their code set. The combined Agile and DevOps approach has rewritten the traditional approach to code migration, smoothing and eliminating wait times so that projects flow more seamlessly from start to finish. It also has the benefit of keeping business resources—testers, approvers, etc.—more engaged as a result of a constant flow of new functions and features to test and use. And lest this be viewed as simply a way to keep the technical stuff moving along, it's important to remember that there is a financial aspect to this as well. Reducing speed to market with new functionality, reducing or eliminating idle hands—be they technical or business—and delighting customers with a steady stream of enhancements and features all go directly to an organization's top and bottom lines.

That, after all, is in many ways what the DevOps approach is all about. All of these critical areas become the means to accomplish it.

DevOps was born out of the Lean and Agile software development methodologies when it became clear that, while those methodologies did indeed speed up the development process, a bottleneck still occurred when push came to shove and new code had to be moved to quality assurance and production environments.

DevOps was created to more seamlessly connect the development and operations processes, making code migration, testing and promotion to production a more efficient process. To accomplish this, the DevOps approach had to find solutions for some of the issues that caused operational delays, and create new ways to organize, implement and continuously optimize the operations process.

Overproduction/Overprocessing

For those who have been in development and/or operations for any length of time, it quickly becomes clear that there is a multitude of operational safety checks that serve to protect a production environment. While that is vitally important, it was also clear that there had grown an "over" problem around many operational procedures, and in many cases that manifested itself in the development process. That includes overproduction, when making or requesting more than was needed from requirements and/or operations perspective to clear arbitrary operations process hurdles.

Alternatively, overprocessing, when development and operations resources do more work (as opposed to just enough, as Lean and Agile would suggest) than required to smooth the transition of code and functions from development to operations. This created waste regarding time, resources and budgets that were not proportional to the benefits derived from following the operations process.

Motion and Transportation

Similarly, DevOps also sought to solve the operational problems of both motion and transportation. That is, the amount of excess work required to deliver new code to meet the operational requirements for code migration. The friction caused by such requirements slowed the motion and momentum of the development process. The same is true of transportation, or the difficulty in moving code between environments such as testing, quality assurance and production.

In both cases, development and project momentum was sacrificed for what often turned out to be a series of artificial hurdles that had long since become less effective or even obsolete parts of the operations process.

Correction and Inventory

In most instances, all of the above resulted in the final maladies of the pre-DevOps development and operational ways. The first was the number of in-flight corrections required when timelines were squeezed, and the rush was on to get to production. Unfortunately, this went hand in hand with the ultimate problem of good code being sacrificed for expedient delivery, often resulting in inadequate functionality, system outages and, in the end, lost market opportunity and revenue.

3 Keys to DevOps Success

Any successful DevOps implementation must address three critical factors in this order: culture, organization and tools.

Culture

It's critically important to connect an organization's values to the DevOps process. Valuing quality, timeliness and organizational alignment of goals and objectives is the first step toward DevOps success. Such cultural values translate directly into a DevOps organization.

Providing empowerment and accountability to DevOps team members helps to build ownership among the team, and trust from their customers in the rest of the organization. It also helps to provide a physical environment that fosters collaboration, teamwork and continued learning. Co-working spaces and collaboration tools such as Slack are a good start. Attending external conferences to broaden perspectives and to bring new ideas

back to the team is often beneficial. From there, brown bag lunch sessions where ideas and experiences can be shared, frequent post-mortems on implementations to hone best practices, and even internal mini-conferences where several departments come together for a day to discuss DevOps practices are all effective ways to build a strong DevOps culture.

Organization

Any good DevOps organization is two-sided; that is it has to work from the top down and from the bottom up at the same time.

The top-down part is in the ability to "see the system" from a macro level, allowing for process understanding and insights from a business workflow perspective. This helps to identify the pain points and bottlenecks in the current process that can be optimized through the DevOps process.

Once that's accomplished, the bottom-up work begins. Identifying things such as inconsistencies in code deployment environments that cause delivery issues, elimination of manual and custom built deployment processes and quarantining inefficient and poorly written code until it can be redone or eliminated are all part of optimizing the time, quality, resources and success factors for deploying production systems on schedule. It's also important here to continually audit the current processes with an eye toward eliminating the processes that are no longer required or useful but have been kept in place out of the fear of "breaking something we don't understand." If nobody understands it, then it shouldn't be in production software.

Automation Tools

The final factor for DevOps success is to have the right toolset.

Communication: Any DevOps team requires the ability to quickly and directly communicate with other team members sans meetings. For this purpose, tools such Slack (real-time chat), Skype (video chat), and Confluence (for storing persistent information) are pretty good options.

Planning, Monitoring & Consistency: For the team's planning needs, a tool such as Trello that can provide Kanban board functionality is worth a look. For issue tracking and monitoring of any system's overall health, tools such as Jira and NewRelic respectively provide some good functionality. Likewise, consistency is vital in a DevOps world, and using automation to ensure that all systems are configured as desired across different environments is a crucial best practice. For this, a tool such as Ansible is worth a review.

Integration & Deployment: For continuous integration of systems in development and as a way to tighten the feedback loop for developers to determine if the central build used for deployment to production is working as intended, the Jenkins toolset might be a good fit. And finally, when it comes making any deployment process as painless as possible, a tool such as Docker that can handle created containers for an application that includes all dependencies, reducing the complexity of deployment to multiple environments, is a solid way to go.

The point of all of this is to create an environment—culturally, technically and physically—where DevOps can succeed, grow and thrive. Organizations that can create an effective and efficient DevOps environment have also created a competitive advantage for themselves.