Agile and DevOps: friends or foes?

DevOps is agile applied beyond the software team. But the real question is, in a match, which one wins?

BY IAN BUCHANAN

In one corner, we have the Certified Scrum Master, known to his friends as the Extreme Programmer, and to his children as the LeSS SAFe DAD... agile!

In the other corner, we have the Lean culture machine, who Continuously Delivers his Infrastructure as Code, he's named his left arm dev and his right arm ops... DevOps!

Although I've taken the hype to an extreme, the sound bites about agile and DevOps can make them sound like very different ideas. Compounding the confusion, both concepts seem to defy definition, even as they have their own jargon and slogans. As the elder, agile may be less vague, but it's certainly common for people to become frustrated with the myriad of definitions for DevOps. The lack of definition has lead to some common conflation.

> **Many people think agile means scrum and DevOps means continuous delivery. This oversimplification creates unnecessary tension between agile and DevOps so you may be surprised to find that they are best friends!**

There's no denying the historical connection between DevOps and agile. When Patrick DuBois and Andrew Clay Schafer tried to connect at the Agile 2008 Conference about "agile Infrastructure", the connection to DevOps was born. Although Patrick later coined the term "DevOps", the Agile Conference continues to honor this connection with a DevOps track. But let's dive deeper than history and consider the practical connections between agile and DevOps, when we look below the surface of scrum and continuous delivery.

There's more to agile than scrum

In some teams, scrum is the difference between a constant, frustrating struggle and productive, rewarding teamwork. In others, scrum replaces politics and overcommitment with objectivity and focus. It can also be embraced as if it were dogma. When the constraints of the business or the work itself demand something different, an agile team will leverage the underlying principles of scrum, then inspect their practices, and adapt to become more effective. This is particularly important when scrum is applied outside the context of software development.

Planning for unplanned work

In the DevOps community, those with agile experience acknowledge that scrum is useful for tracking planned work. Some work in operations can be planned: releasing a big system change, moving between data centers, or performing system upgrades. But much of the work of operations is unplanned: performance spikes, system outages, and compromised security. These events demand an immediate response. There's no time to wait for the items to be prioritized in a backlog or for the next sprint

items to be prioritized in a backlog or for the next sprint planning session. For this reason, many teams that have come to embrace DevOps thinking, look beyond scrum to kanban. This helps them track both kinds of work, and helps them understand the interplay between them. Or, they adopt a hybrid approach, often called Scrumban or Kanplan (kanban with a backlog).

In many ways, the key to scrum's wide adoption may be that it prescribes no technical practices. Scrum's lightweight management practices often make a big difference for a team. Where there were once competing priorities from multiple sources, there is now a single set of priorities in the backlog. And where there was once too much work in progress, there is now a plan constrained by what time has shown is really possible. In combination, these can take a team to new levels of productivity. However, the team may find themselves constrained by the lack of technical practices, such as coding reviews, automated acceptance tests, and continuous integration.

Other agile processes like Extreme Programming have strong opinions about how technical practices support the team's ability to maintain a sustainable pace and provide transparency and visibility to management and stakeholders. Some scrum teams resort to putting technical tasks in the backlog. While that fits well within the guidance of scrum, it quickly hits the practical problem of product owner bias towards features. Unless the product owner is quite technical, she or he may not have the skills to evaluate the cost/benefit of technical practices. That gets even harder for a product owner as the technical tasks stretch into operations to support reliability, performance, and security.

Product owners and service owners

At Atlassian, we have recognized that it helps to have two different roles for products we operate. Our product owners are good at understanding the features that users need but they are not so good at weighing those features against non-functional capabilities like performance, reliability, and security. So some SaaS products at Atlassian also have a service owner, responsible for prioritizing those non-functional capabilities. From time to time the two owners may have to do some "horse trading" but most of the time, these can be worked by independent teams. This might not be the only way to "amplify feedback" from operations, but it does help overcome an all-too-common bias in Product Owners about features. This "two-owner" approach isn't the only path to DevOps. What's important is understanding these non-functional characteristics as "features" and being able to plan and prioritize them just like any functional user story.

> Until DevOps is mainstream, it will not be an organic outcome of scrum. #DoDevOps

Ultimately, none of these criticisms of scrum are entirely inherent to scrum itself. As with all agile methods, scrum has a built-in "process improvement" mechanism called retrospectives. Hence, it is reasonable to believe that some scrum teams will draw on DevOps as a source of inspiration and use scrum retrospective as the opportunity to tune and adjust towards DevOps. However, simply practical to realize that most teams need an injection of outside ideas. Until DevOps is mainstream (perhaps even taught in school), DevOps will not be an organic outcome of scrum. Whether the team engages an agile or DevOps coach is probably immaterial, as long as that person can bring experience in automation across building, testing, and deploying software.

There's more to DevOps than continuous delivery

When done well, the discipline of continuous delivery (CD) helps to limit work in progress, while the automation of deployment helps to elevate constraints. In this way, CD helps a software team deliver more frequently and with higher quality, instead of having to choose between the two. However, just as teams focusing only on scrum can miss the broader context of agile, so too can teams focusing on

continuous delivery miss the broader context of DevOps.

The practice of CD does not directly address problems in communication between the business and a software team. If the business has a year-long, budget-driven planning cycle, then a team delivering every commit into production may still have to wait months before the business can react. All too often those reactions come as step-functions, like canceling the project, or worse doubling the project team (because a large influx of new people is disruptive).

The Agile Fluency Model indicates the first level of fluency as "Focus on Value", where teams focus on transparency and alignment. Without this fluency, CD can easily devolve into an endless cycle of technical improvement that yields no appreciable value to the business. A team might get good at delivering fast with high quality, but for a product that has low value for end-users or the business. Even when there are many users who say good things, that assessment of low value might only be possible at a larger business portfolio level. Without this important fluency, it is hard to weigh technical practices against features. This is particularly important for a team with a legacy codebase, that may not have automated tests or a design suitable for frequent deployment. In a legacy context, a CD transformation may take years. So it's that much more important to be able to demonstrate business benefit.

The three ways

DevOps is more than just automating the deployment pipeline. In the words of John Allspaw, DevOps is about, "Ops who think like devs. Devs who think like ops." Elaborating on that thought, Gene Kim explains The Three Ways as principles of DevOps:

The first way	System Thinking	emphasizes the performance of the entire system, as opposed to the performance of a specific silo of work or department – this can be as large as a division or as small as an individual contributor.
The second way	Amplify Feedback Loops	creating the right to left feedback loops. The goal of almost any process improvement initiative is to shorten and amplify feedback loops so necessary corrections can be continually made.
The third way	Culture of Continual Experimentation and Learning	creating a culture that fosters two things: continual experimentation, taking risks and learning from failure; and understanding that repetition and practice is the prerequisite to excellence.

Continuous Delivery (CD) focuses on The First Way: automating the flow from dev to ops. Automation plays an obvious role in helping to accelerate a deployment system. But there is more to systems thinking than just automation.

The Second Way is characterized by the practice, "Devs wear pagers too." Although the literal use of pagers may not be necessary, it means pulling developers into operational issues. This helps developers understand the consequences of their development choices. For example, that can inspire developers to put log messages in better places and to make those messages more meaningful. It's not just about awareness. Developers also bring their internal understanding of the system to troubleshooting efforts, so a resolution can be found and implemented faster.

The Third Way is about making incremental experiments in the system as a whole, not just as changes to the application moving through the pipeline. In other words, it's relatively easy to see how long automation takes and to use increasingly powerful infrastructure to keep improving it. It's harder to understand how the hand-offs between roles or organizations introduces delays. This means teams "inspect and adapt" across the whole delivery workflow, looking for opportunities to improve human collaboration. For that matter, CD requires a habit of adapting and improving. If the team doesn't reflect on how to become more effective, and then tune and adjust its behavior on anything else, then CD will not grow and thrive either. The team needs to feel empowered to solve their own problems.

In scrum, each retrospective is an opportunity to improve the practices and tooling. But if the team isn't taking

advantage of those opportunities to solve both short-term and long-term technical problems, then they will just wait for the product owner to put CD tasks into the backlog, which will never happen.

DevOps is agile applied beyond the software team

Scrum mainly maps to the agile principle, "Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage."

Continuous delivery mainly maps to the agile principle, "Our highest priority is to satisfy the customer through early and continuous delivery of valuable software."

That means agile is more about embracing incoming and outgoing change than about ceremonies like standups and sprint planning. Indeed, there are 10 other principles in the Agile Manifesto. Rather than trying to choose among the principles, they should be considered as a whole. Together these principles represent an attitude towards change that is common for both agile and DevOps.

> DevOps seeks to bring that agile attitude toward change to a new audience: IT operations. #DoDevOps

These folks have been stuck trying to run fragile systems that are also the most important for the business. Because they are most important for the business, they are also where the most urgent changes are needed. As such, this agile idea of embracing change isn't "change for the sake of change". It's about holding development accountable for the quality of their changes, while improving the overall capacity to deliver business value. This focus on business value is another aspect shared by agile and DevOps

Finally, neither agile nor DevOps are business goals in and of themselves. Both are cultural movements that can inspire your organization with better means for achieving your goals. Agile and DevOps work better in combination than as adversaries. The trick to avoiding confrontation between these two ideas is to understand the deeper values and principles upon which they are formed. Quick, but narrow, definitions lead to siloed thinking. Now that you know there's more to agile than scrum, and there's more to DevOps than CD, you're ready to try the powerful agile + DevOps combination.

Connect your tools with automation

Perhaps the biggest challenge working across multiple tools is the constant change of context and the interruption that brings. It can take hours to get back into flow if you get interrupted by a non-code task. For this reason, more and more folks are automating across their git providers and work management tools. Here are three of the most common automation rules

1. When a commit is created and the status is 'To Do' then transition this issue to 'In Progress'. Go to rule.
2. Transition to 'done' when the Pull Request is merged. Go to rule.
3. If a build fails in Jenkins, add a comment to Jira and Slack the team. Go to rule.

See these automation rules and 100s more in the Jira Automation Template Library.

Go to library

IAN BUCHANAN

While Ian has broad and deep experience with both Java and .NET, he's best known as a champion of agile methods in large enterprises.

He's currently focused on the emerging DevOps culture and the tools for enabling better continuous integration, continuous delivery, and data analysis. During his career, he's successfully managed enterprise software development tools in all phases of their lifecycle. He has driven organization-wide process improvement with results of greater productivity, higher quality, and improved customer satisfaction. He has built multi-national teams that value self-direction and self-organization. When not speaking or coding, you can find Ian indulging his passions in parsers, meta-programming, and domain-specific languages. Follow Ian at @devpartisan.

TUTORIAL

Learn kanban with Jira Software

Step-by-step instructions on how to drive a kanban project, prioritize your work, visualize your workflow, and minimize work-in-progress with Jira Software

Try this tutorial →

ARTICLE

Get started with agile project management

Agile project management is an iterative approach to managing software development projects that focuses on continuous releases and customer feedback.

Read this article →