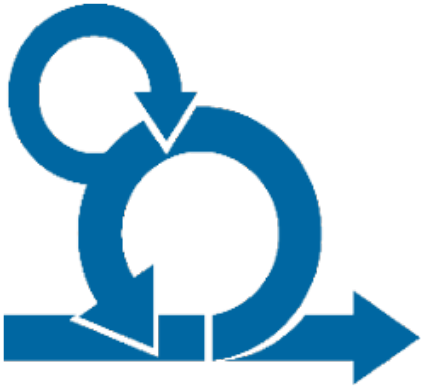
Terminology matters: Agile/Scrum vs CD/CI vs DevOps

Chances are you were triggered by the enumeration of buzz words in this blog's title. Without effort you could probably squeeze in a few more, like Lean or another flavor of 'Continuous x'. A lot of managers and teams struggle explaining the company's meaning regarding these terms. Striking is the absence of a clear and concise understanding of the vocabulary. DevOps is translated into Business involvement, CD/CI is equated to Scrum and Agile is basically a term covering everything. The point is, you need to make sure everyone around you understands what you mean and how the terms relate to your organization's context. During sessions, I use the diagram above to explain the different angles that Agile / Scrum – Continuous Delivery – DevOps have. In this blog, I will dive in a little deeper and share my thoughts on the terminology and how I explain the nuances and differences.



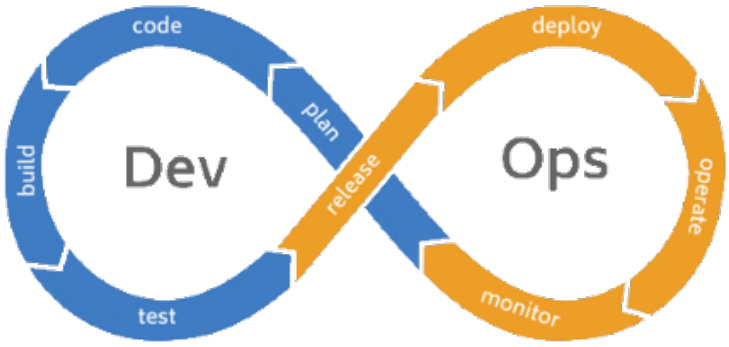
Agile/Scrum and Continuous Delivery

Agile/Scrum focuses on the ability of teams and organizations to absorb the array of changes that happen while making software. The setup of the Scrum rituals ensures that transparency is present at all times, for the team itself but also for the stakeholders. When functioning well, the Product Owner helps the stakeholders in guiding the product development in the right direction. Adapting change and changing course now and then is part of the normal process. The main aim of Scrum is to deliver as much value as possible working in a sustainable pace [1]. The framework says nothing about how this should be achieved from a technical perspective. Not what tools to use or what platform is appropriate, nor does it mention test automation, infra structure as code or deployment pipelines.

The power of short cycled feedback (highly valued within Scrum) is strengthened enormously when teams can focus on developing the properties of the systems (functional and non-functional) instead of spending time on manual tasks. Manual tasks can include everything from testing code by hand, setting up new environments or executing tedious deployment scripts following a step by step manual. This is the area where Continuous Delivery steps in.

Continuous Delivery empowers teams by giving them the tools and methods to automate their process as much as possible and minimize manual labor. Not only does it save time, it also decreases the error rate by eliminating the human factor [2].

Simply stated: by combining Scrum and Continuous Delivery, teams can create a new feature, run automated tests and deploy the new software to environments where validation by business can be done. All covered by a framework supporting transparency, inspection and empiricism at all times.

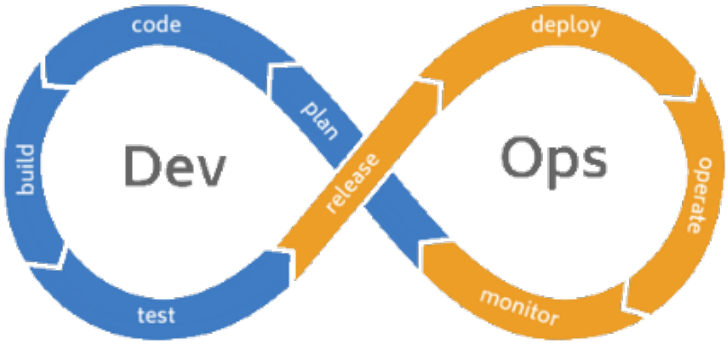


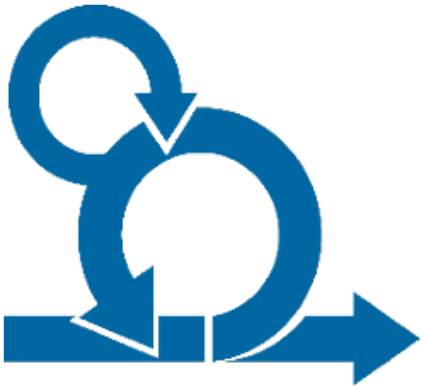
Continuous Delivery and DevOps

Continuous Delivery is largely based on the technology that enables teams to automate as much of the development process as possible. Traditionally, a handover moment between development and operations was in place when software was finished and accepted. The deployment to the production environment and the responsibility for the quality of service once running was in hands of the operation teams. In DevOps, teams become responsible for the entire chain all the way to production [3]. Technically, the tools and actual deployment on production are not so different than those used on other stages. Having the explicit mandates and authorizations marks the distinction, these include access to production monitoring and logging information.

Often the biggest chasm to cross in a journey towards DevOps is organizing the rights and responsibilities for newly formed DevOps-teams (see “DevOps understanding the Evolution” to have a brief chronological overview of DevOps). This becomes more evident when discussing quality of service. Who is allowed to see how a system is running, how to deal with production data and what information has been logged? Especially large organizations with a history of separated departments and teams for operations and development are discussion prone. If you find yourself in a mandate-discussion going in circles, what can help is the next simple question: Who would be responsible if we only had one team? Clearly, it is only logical that this one team should be in the lead. Barricades often dissolve after discussing why it should be different just because you have multiple teams.

A well-aged mantra reads as follows: ‘if you want to have a stable environment, don’t change it’. This contrasts with the nowadays popular one: ‘fail fast and fail often’. The technology now present – based on Continuous Delivery – enables teams to develop and deploy new software with no or hardly any downtime. Combined with DevOps, teams can take ownership for the entire life cycle of the product. Responsibility only ends when a system is decommissioned.





DevOps and Agile/Scrum

While DevOps ensures that teams are able to take technical and organizational ownership for the quality of service of the products they develop and maintain, the term says nothing about the team dynamics that should be in place to support this responsibility. The other way around is also true, Agile/Scrum makes no mention of different environments or how to address major production incidents. Of course, it is possible to have for example Scrum without DevOps. However, this means you will have a delay in your feedback loop and miss valuable information about the production environment. More importantly, teams that have production responsibility have more insights in the behavior – intended or not – of their system.

Naturally, DevOps requires some additional agreements in a team’s Scrum process. How is monitoring approached (not technical but process wise) or how much time can be spent on improving ‘the run’. By discussing the time spend on monitoring and analyzing the software in production, a more in-depth analysis can be made based on value. These Lean influences from DevOps provide the structure to improve operational excellence as well. For example: ownership and code-logging hygiene go quickly hand in hand when a team has to fix its own production problems. Decision making and prioritization by Product Owners is more coherent in DevOps. The total costs of running a system become more transparent and are influenced more directly by the Product Owner’s team. See “Product Owners in DevOps” to read more on backlog prioritization in DevOps.

Combining the flow and value stream ideas from DevOps with the sustainable pace and delivery of Done increments from Scrum is powerful. Feedback from clients or the business on the latest software can continuously be weighted up against other aspects of running the product.

Conclusion

While it is theoretically possible for teams to use exactly one of the ‘buzz terms’ without the others, it is not common. What I usually see in organizations is a mix of all three. That makes sense, the terms and related concepts focus on team responsibility and share the desire for short feedback loops, build-in quality and transparency towards stakeholders and teams internally. They re-enforce each other and offer opportunities to improve specific parts of an organization’s way of working. That is one of the reasons why you should be able to describe in a concise way what you mean when discussing improvements to your development process.

Cheers,
- Sjors Meekels

Disclaimer: the description above is not complete and it surely is not one hundred percent perfect. It is not meant to be. It is a personal simplification to address the consistency of a set of popular tech-buzz words.

References & recommended reading:

[1] SCRUM – a Smart Travel Companion, Gunther Verheyen
[2] Continuous Delivery, Jez Humble & David Farley
[3] The DevOps Handbook, Gene Kim, Jez Humble i.a.
[4] DevOps: understanding the Evolution: https://www.agitma.nl/devops-understanding-the-evolution
[5] Product Owners in DevOps: https://www.agitma.nl/product-owners-in-devops-what-dominates-your-backlog-urgent-or-important-matters
[6] https://www.linkedin.com/pulse/differences-between-continuous-integration-delivery-versteijnen Pieter Versteijnen (last checked 19-11-2019)