AGILE VS DEVOPS: A FULL COMPARISON



Agile and DevOps are the two most popular software development lifecycle (SDLC) methodologies currently in practice.

One survey indicates that 97% of organizations use Agile, whereas the most innovative startup firms as well as large enterprises take advantage of DevOps to deploy new code features really fast:

- AWS deploys new code every 11.7 seconds.
- Netflix deploys new code thousands of times every day!

As more organizations are eager to follow suit, it's important to carefully understand the similarities and differences between Agile and DevOps—which is exactly what this article will help you do. We will:

- Briefly review the history of both SDLC models
- Understand the driving factors behind Agile and DevOps
- Highlight the key differences in the two

Agile origins: From Waterfall to Agile

Let's with a recap of software development history. As software development projects grew in scale and complexity, IT organizations needed a systematic approach to consistently deliver high quality software at speed, while minimizing risk and cost overruns.

In the 1970s, the IT industry and academia formally adopted the Waterfall SDLC model: a linear and sequential model that flows through various stages of a standard software development project in the following order:

- Requirements Gathering and Analysis
- System Design
- Implementation
- Integration and Testing
- Deployment
- Maintenance

Originating in the 1950s manufacturing industry, the Waterfall model worked well enough until most organizations identified a few critical flaws when they actually implemented it. Common flaws of the Waterfall model include:

- **Rigidity.** Requirements cannot be changed once the development process starts.
- **Risk.** Any flaw or inadequacy in the product is identified only at the end of the SDLC pipeline when the project takes its final shape.
- **Waste.** The sequential approach is slow and channels the bottleneck across the SDLC.
- **Scope.** In practice, the cost and time spent on waterfall projects frequently exceeds the expected limitations.

In 2001, a team of professional developers released the Agile Manifesto: a set of values and guiding principles that can be used as a philosophy or a mindset to develop high quality software components iteratively—small but frequent releases with small improvements.

Consider how the values of Agile (left) differentiate from the traditional SDLC practices and priorities (right):

- **Individuals and interactions** over processes and tools
- **Working software** over comprehensive documentation
- **Customer collaboration** over contract negotiation
- **Responding to change** over following a plan

Agile also destroys the idea of a "finished product", which was the goal of the Waterfall approach. Instead, Agile believes that software development is iterative and incremental. With each new release of software, the customer is able to either:

- Perform new functions
- Improve upon existing functions

Agile methodologies encourage developers to break down software development into small pieces known as "user stories". This highlights the value Agile places on the customer, which helps the developers by both:

- Providing faster feedback loops
- Ensuring product alignment with market need

Agile further advocates for adaptive planning, evolving development, early and continuous delivery, and continuous improvement—these all enable developers to rapidly and flexibly respond to change in client needs, software, or other external factors.

(Read our in-depth Waterfall vs Agile explainer.)

bmc

Agile	Waterfall
Iterative development in short sprints	Sequential development process in pre-defined phases
Flexible and adaptive methodology	The process is documented and follows the fixed structure and requirements agreed in the beginning of the process
Feedback-based approach: Sprints lead to short build updates that are evaluated on and guide the future direction of the development process.	Limited and delayed feedback: The software quality and requirements fulfilment isn't evaluated until the final phase of the development processes when testers and customer feedback is requested.
A provision for adaptability: Project development requirements and scope is expected to change over the course of the iterative development process.	The requirements and scope are definitive once agreed upon.
The SDLC phases overlap and begin early in the SDLC: planning, requirements, designing, developing, testing, and maintenance.	The SDLC phases are followed in order, with no overlap. Members of one functional group are not involved in another phase that doesn't belong to their job responsibilities.
Follows a mindset of collaboration and communication. The requirements, challenges, progress, and changes are discussed between all stakeholders on a continuous basis.	Follows a project-focused mindset with the aim of fully completing the SDLC process.
Responsibilities and hierarchical structure can be interchangeable between team members.	Fixed individual responsibilities, particularly in management positions.
All team members focused on end-to-end completion (achieved sequentially) for the projects.	Team members focused on their responsibilities only during their respective SDLC phases.
Suitable for short projects in high-risk situations.	Suitable for straightforward projects in predictable circumstances.
Limited dependencies as the focus is less on implementation specifics, and more toward the mindset.	Strict dependencies in technologies, processes, projects and people.

From Agile to DevOps

Agile sounds good in theory.

In fact, Agile is easy to plan. It's easy to consider as a philosophy for organizational culture and communication among development teams. Frameworks such as Scrum make it easier to adopt Agile principles.

In practice, however, Agile lacks in execution and delivery. Organizations often agree to follow rapid release cycles and conduct regular Scrum meetings, but find it challenging to adopt Agile.

One of the reasons? Agile as a guiding manifesto brings little practical advice as an SDLC process framework in itself. Slow and tedious governance process, inadequate communication and collaboration, lack of automation and, most importantly, the expanding divide between Devs and Ops personnel keeps organizations from becoming truly Agile.

Instead, developers end up practicing sprints of fast Waterfall: siloed, sequential, discontinuous development sprints that fail to iteratively improve on customer feedback.

As IT became essential to businesses in the 21st century, two imperative areas emerged: IT Operations (ITOps) and Development Operations (DevOps):

- ITOps responsibilities include ensuring security, compliance, and reliability.
- DevOps is responsible for developing and deploying new products to the end user.

While ITOps ensures safety and security for all business needs using the network, DevOps walks a line between flexibility and the rigorous testing and communication that comes with deploying new software.

DevOps is a theory rooted in communication, both within itself—as the developers and operators have to coordinate—and also across other departments. DevOps frequently communicates with ITOps to ensure secure and stable environments for testing. Their crossover to other teams like marketing and customer service makes sense as they deploy new software.

(Explore our multi-part DevOps Guide.)



Using DevOps & Agile together

Proponents of using both theories in appropriate business needs believe that DevOps can be seen as an extension of Agile. Agile relies on cross-functional teams that typically include:

- A designer
- A tester
- A developer

DevOps takes this one step further by adding an operations person who can ease the transition from software to deployment. Because of DevOps' inherent communication with other teams, DevOps can help automate processes and improve transparency for all teams.

(Learn about various IT teams.)

Consider these similarities between Agile and DevOps:

- **Business focus.** Aligning the software development process with user and market-centric products helps drive business value.
- **Collaboration.** Teams at an individual and group level must communicate regularly, actively breaking silos.
- **Lean philosophy.** Focus on removing waste processes, a Lean derivative, and driving value at every stage of the SDLC pipeline.
- **Continuous release cycles.** Short, iterative sprints that lead to a continuous release process. Adopt the mindset and technology capabilities that can help achieve this flexibility.
- **Approach.** Both Agile and DevOps are approaches—they are not hard-coded playbooks for IT organizations to follow.

Considering these similarities, it's easy to see how many practices that results from the Agile manifesto can be considered a subset of DevOps: collaboration, continuous improvement, and culture.

Agile vs DevOps: Contrasting points

While we are proponents of using Agile and DevOps theories together, it is important to understand where they clearly differ. Let's look at a few contrasting points.

Speed

Agile is all about rapid and frequent deployment, but this is rarely the goal—or even part of the goal—for DevOps.

Creating vs deploying software

Developing software is inherent to Agile, but DevOps is concerned with the appropriate deployment of said software.

For the record, DevOps can deploy software that was developed in any number of approaches, including Agile and non-Agile theories, like the Waterfall approach, which is still appropriate for certain projects.

(Explore the differences in deploying & releasing software.)

Specialization

Agile is an equal opportunity team: every member of the scrum can do every job within the team, which prevents slowdowns and bottlenecks.

DevOps, on the other hand, assumes separate teams for development and operations. People stay within their teams, but they all communicate frequently.

Communication

Daily, informal meetings are at the heart of Agile approaches, so each team member can share progress, daily goals, and indicate help when needed. These scrums are not meant to go over documentation or milestones and metrics; instead they look solely at progress and any blockers to progress.

DevOps meeting are not daily.

Documentation

Agile teams don't codify their meeting minutes or other communications, often preferring lo-fi methods of simple pen and paper.

DevOps takes documentation seriously, requiring design documents and specs in order to fully understand a software release.

Team size

Staying small is the core of Agile: the smaller the team, the fewer people on it, the faster they can move, even if they are contributing to a larger effort. DevOps will have many teams that work together and each team can realistically practice different theories.

Scheduling

Agile teams work in short, predetermined amounts of time, known as sprints. Sprints rarely last longer than a month, and often can be as short as a week.

DevOps values maximum reliability, so they focus on a long-term schedule that minimizes business disruptions.

Automation

Automation is the heart of DevOps, as the overall goal is to minimize disruptions and maximize efficiency, especially when deploying software. Agile doesn't require automation.

Infrastructure as Code is another example of how DevOps streamlines the collective efforts of Devs and Ops complying to organizational policies and governance without compromising efficiency in the SDLC pipeline performance.

These stark differences remind us that Agile and DevOps, at their roots, are not the same.

Culture of Agile and DevOps

While Agile does not necessarily lead to DevOps, both can have profound culture shifts within an organization.

An Agile approach encourages a change in how we think about development. Instead of thinking of development as cumbersome, Agile thinking promotes small, manageable changes quickly that, over time, lead to large changes. Companies of all sizes have experimented with how working in an Agile way can boost many departments, not only IT. Today some enterprises consider themselves fully Agile.

DevOps can also bring its own cultural shifts within an organization, including enhancing communication and balancing stability with change and flexibility.

Choosing to use both theories is an active decision that many industry experts believe can lead to more rational decision making, thus improving the company culture.

Related reading

- BMC DevOps Blog
- Intro to Agile with Scrum: 4 Tips for Getting Started
- The Complete DevOps Certifications Guide 2021-2022
- Deployment Pipelines (CI/CD) in Software Engineering
- How & Why To Become a Software Factory