

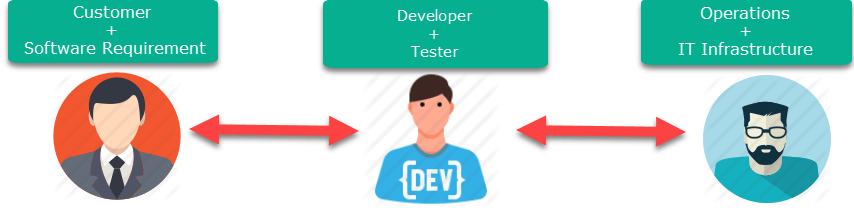
Simply put, Waterfall involves completing each project phase before moving on to the next. It emphasizes **planning upfront** the different phases of development, such as requirements gathering, design, implementation, testing, and maintenance.

In contrast, Agile involves breaking a project into smaller, more manageable chunks and continuously iterating and improving upon them. Instead of following a predetermined plan, Agile emphasizes **flexibility and collaboration** between team members throughout the entire project cycle to accommodate change.

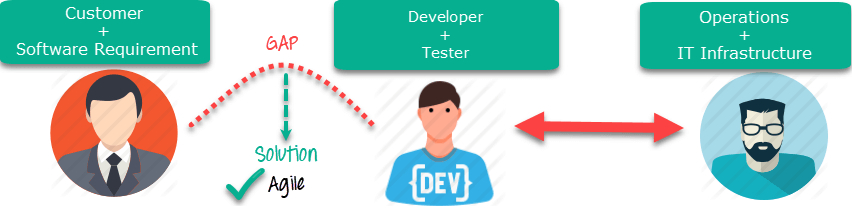
~~Generally, Waterfall is best suited for projects with a clear set of requirements that don’t change much over time and where risk and uncertainty are low. Agile is better for exploratory projects where conditions may evolve or change over time.~~

## How is DevOps different from Agile? DevOps Vs Agile

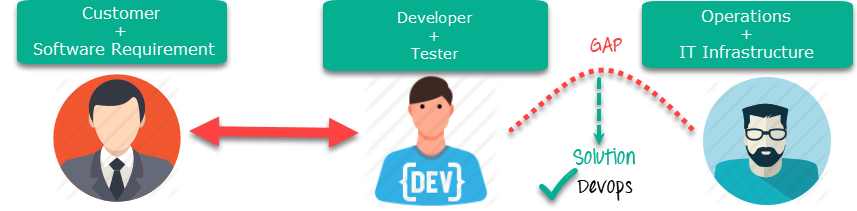
Stakeholders and communication chain a typical IT process.

[](https://www.guru99.com/images/2-2017/092917_0812_DevOpsTrain4.png)

Agile addresses gaps in Customer and Developer communications

[](https://www.guru99.com/images/2-2017/092917_0812_DevOpsTrain5.png)Agile Process

DevOps addresses gaps in Developer and IT Operations communications

[](https://www.guru99.com/images/2-2017/092917_0812_DevOpsTrain6.png)DevOps Process

DevOps incorporates a new collaborative culture that embraces numerous practices combined together for a continuous software development methodology that places significant emphasis on feedback loops, collaboration and continuous improvement. DevOps requires fundamental cultural changes and includes so many practices that can be overwhelming to those just starting their journey.

1. **DevOps is not simply combining Development & Operations teams**

We’ve all seen this one when the term DevOps comes up. “Let’s combine the Development & Operations team and, voilà!, we now have DevOps.” DevOps combines a set of processes and practices to be adopted throughout the entire delivery pipeline and spans multiple stakeholders. A couple of the key practices within DevOps adoptions include continuous integration (CI) and continuous delivery (CD). Simply combining two teams and calling it DevOps cannot accomplish those practices.

**2. DevOps is not a separate team**

Setting up a separate DevOps team is another trap many organizations fall into when beginning their DevOps journey. I believe having a separate DevOps team could leads to more silos in the end. I’ve also found the creation of these teams can lead to further confusion when the mission is not clearly defined.

There are some cases where a temporary DevOps team may make sense to help enable the processes and potential tooling that may be needed for adoption, but the key is making it temporary—which is often better in theory than in practice. There are several great blogs out there discussing DevOps teams such as Matthew Skelton’s blog, “[What Team Structure is Right for DevOps to Flourish?](http://blog.matthewskelton.net/2013/10/22/what-team-structure-is-right-for-devops-to-flourish/#more-839)”

**3. DevOps is not a tool**

Let me be the first to say I actually love the growing number of tools that enable us to continue maturing our DevOps adoptions, but I’ve noticed the use of one or two tools starts to lead to the perception that DevOps is a tool. How many times have you heard?

*“We’re already doing DevOps. We have Chef.”*

*“We do DevOps. We automatically deploy through Jenkins.”*

Keep in mind, I’m a big fan of both Chef and Jenkins, but I think the power of DevOps is drastically underutilized if you’re equating the utilization of a single automation tool to DevOps success. Adopting automation and tooling is, of course, a part of DevOps, but only when combined with end-to-end practices of increased collaboration with continuous integration/continuous delivery, amplified feedback loops and continuous improvement (to name a few)! DevOps is a journey and that journey may start with a tool, but the goal should be to first identify your DevOps strategy and then find a tool or tool chain that meets those goals.

**4. DevOps is not a one-size-fits-all strategy**

Because there are so many different business drivers and technologies to consider when setting up an overall DevOps adoption strategy as well as identifying your DevOps tool chain, it’s important to apply the same tenets of DevOps to your DevOps strategy. Embrace change, gather metrics, understand feedback, fail fast and correct your course quickly. As an example, if you initially identify a tool that no longer works for your technology or environment, abandon it and move on. Just because you used it on the last project doesn’t make it a silver-bullet fix for the next one. We need to first understand our current strategy and environment, then react accordingly.

**5. DevOps is not automation**

That one may have caught the eye of a few people, so I’ll clarify: DevOps is not solely automation. Automation is absolutely a huge part of DevOps; however, it is not the only part, and deploying some degree of automation is often used interchangeably with DevOps. I think understanding the key DevOps practices is a great precursor to ensuring DevOps is viewed as more than just automation. Understanding the core DevOps principles is key to understanding the true benefits of DevOps adoption.

There are a lot of things that DevOps is and I think I share a widely spread passion for DevOps. There are also a few things that DevOps isn’t—or isn’t solely. If you’re just starting your DevOps journey or still maturing your existing model, making sure everyone on your team has some basic DevOps education and is aware of the DevOps strategy for your project is key. It will go a long way in helping everyone understand that DevOps is about a lot of things. It’s also *not* about a lot of things.

**CALMS** is a framework that assesses a company’s ability to adopt DevOps processes and a way to measure success during a DevOps transformation. The acronym was created by Jez Humble, co-author of ‘The DevOps Handbook‘, and stands for **Culture, Automation, Lean, Measurement and Sharing**.

In summary, DevOps is about transforming the culture of the organisation, so that you can:

* **Move faster**
* **Be more efficient**
* **Collaborate more effectively**

**Culture**

C stands for Culture as cultural change can be considered the core of any DevOps transformation. The change must be embraced by top management and put into practice with consistent leadership that sets a good example.

[DevOps](https://www.qrpinternational.be/blog/glossary/devops-what-is/) is not simply a process or a different approach to development, but a *real cultural change*. **A fundamental part of the DevOps culture is collaboration**. Any tool or automation is useless if development and IT/Ops professionals do not work together: DevOps does not solve problems related to tools but solves problems related to professionals, to the human aspect of a job.

Because DevOps does not solve tool-related problems. It solves human problems.  
Sharing a common goal and a plan to achieve it together are two key factors in fostering collaboration within and between teams; the very goal of the DevOps

**Automation**

In addition to increasing speed and quality of delivery, automation is the most important part of DevOps: it allows value to be delivered faster and with higher quality, which is the main goal of the methodology.

Automation is fundamental and should be exploited as much as possible for several reasons:

* It helps to eliminate manual and repetitive tasks and to create reliable systems for performing them: computers, in fact, perform more rigorous and reliable tests than people, allowing them to detect bugs and security issues faster
* Automation allows simpler jobs to be done automatically, leaving time for higher value work for which human input is indispensable

**Lean**

[The Toyota Production System](https://theleanway.net/what-is-lean), developed around eliminating the three types of deviations that shows inefficient allocation of resources. The three types are Muda = waste), Mura = unevenness), and Muri = overburden).

Muda means wastefulness, uselessness and futility, which is contradicting value-addition. Value-added work is a process that adds value to the product or service that the customer is willing to pay for.

Mura means unevenness, non-uniformity, and irregularity. Mura drives and leads to Muda. For example, in a manufacturing line, products need to pass through several workstations during the assembly process. When the capacity of one station is greater than the other stations, you will see an accumulation of waste in the form of overproduction, waiting, etc. The goal of a Lean production system is to level out the workload so that there is no unevenness or waste accumulation.  
  
Mura can be avoided through the Just-In-Time ‘Kanban’ systems and other pull-based strategies that limits overproduction and excess inventory. The key concept of a Just-In-Time system is delivering and producing the right part, at the right amount, and at the right time.

Muri means overburden, beyond one’s power, excessiveness, impossible or unreasonableness. Muri can result from Mura and in some cases be caused by excessive removal of Muda (waste) from the process. Muri also exists when machines or operators are utilized for more than 100% capability to complete a task or in an unsustainable way. Muri over a period of time can result in employee absenteeism, illness, and breakdowns of machines. Standardize work can help avoid Muri by designing the work processes to evenly distribute the workload and not overburden any particular employee or equipment.

**Measurement**

Measuring data is generally a key aspect of making choices, and as far as DevOps is concerned, the focus is exclusively on certain metrics.

Rapid iteration only works if you know where you are going, and it is therefore necessary to be data-aware: measuring and keeping track of what is being done in the software development process is a non-negotiable prerequisite for making fact-based decisions.  
For this reason, disciplined and well-organised organisations collect data on everything.

Having a large amount of data will help your team make decisions, but this data will prove even more useful if it is also shared externally with other teams in the organisation to create clear roadmaps and receive support from all the teams involved in the process and, if necessary, from all stakeholders in general.

**Sharing**

Finally, we find the S of sharing:

[Knowledge sharing](https://www.qrpinternational.be/blog/glossary/the-importance-of-corporate-knowledge-sharing/) and best practices acquired during the entire process, is **a key aspect** in order to successfully conclude a project because it enables collaborative work.

<https://blog.sonatype.com/principle-based-devops-frameworks-three-ways>

**The First Way: Principles of Flow**

The First Way is mostly concerned with accelerating the “flow” of work throughout a process. Gene Kim also refers to the First Way as Systems Thinking in his article [The Three Ways: Principles Underpinning DevOps](http://itrevolution.com/the-three-ways-principles-underpinning-devops/). Whether you’re calling it Flow or Systems Thinking, the principles underpinning the First Way are working toward the same end: viewing the flow of work as one continuous system (unsiloed) that can be continually refined and optimized.

Some of the key principles of the First Way are:

* **Making work “visible”.** Unlike manufacturing processes, which are easily observable on a plant floor, the flow of software through its development lifecycle is not easily seen. Using methods such as Kanban boards can surface the activities going on behind the scenes, by showing the left-to-right movement of a user story through the development phases.
* **Limiting work-in-progress (WIP).** Keeping work-in-progress to a minimum has also been shown to accelerate work flow, because it minimizes multi-tasking and context-switching.
* **Reducing batch sizes.** “Chunking” work into smaller pieces like a two-week sprint can also help deliver features (albeit smaller ones) and bug fixes to the customer faster. Issues are often caught earlier when those updates and additions are released sooner.
* **Reducing hand-offs between teams.** The risk of “dropping the baton” increases as the hand-offs do. Although hand-offs can’t be completely minimized, the key is to keep the teams in tight communication with one another so that the hand-off itself is almost a non-event rather than a large ordeal with the potential for communication missteps along the way.
* **Identifying and removing constraints and waste.** Constraints might be bottlenecks in the process, such as environments, test setup, and overly tight architecture, while waste includes things like manual work, heroics, and context-switching.

**The Second Way: Principles of Feedback**

The Second Way works to enable fast and constant feedback cycles throughout all stages of a development cycle.

Some of the key principles of the Second Way are:

* **Swarming and solving problems to build new knowledge.** This principle fits into the “fail fast” mentality, so that teams can find issues with an implementation as soon as possible and address them early and often as iterations continue.
* **Pushing quality closer to source.** This principle is at the core of the DevSecOps movement, which is concerned with addressing security concerns during the development cycle, instead of at the end, when rework to remediate is more difficult and costly.
* **Optimizing for downstream work centers.** This principle works against the “throw it over the wall” mentality, by underscoring that development should be just as invested in their application being deployable, working with operations to bridge that gap (and vice versa).

**The Third Way: Principles of Continuous Learning**

The Third Way seeks to create a culture of continual learning and experimentation within the development organization.

Some of the key principles of the Third Way are:

* **Enabling organizational learning and a safety culture.** Leaders must help “set the tone” for the organization, making it okay to learn, make mistakes, and try again.
* **Institutionalizing the improvement of daily work.** Improving what you do and how you accomplish it should be part of everyone’s daily thinking and call to action.
* **Transforming local discoveries to global improvements.** Surfacing and sharing improvements at all levels will help enable a “bubble up” culture of continuous improvement.
* **Injecting resilience patterns into daily work.** Some examples might include rehearsing failures, and working toward improving key metrics for deployment.
* **Leaders enforcing a learning culture.** Organization-wide learning is unlikely to take hold and become pervasive unless it is sanctioned and exemplified by its leaders. So being intentional about communicating the value of learning and problem-solving is crucial to building that culture.