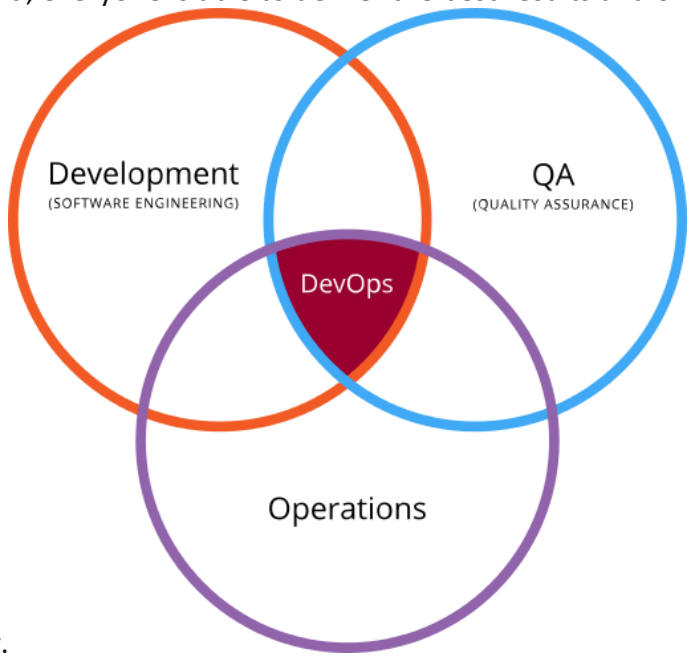
DevOps Tutorial

1. Introduction

“DevOps” as a term was first coined in 2009 by **Patrick Debois**, who became one of its chief proponents. Simply put, **DevOps is a combination of software development and operations**—and as its name suggests, it’s a melding of these two disciplines in order to emphasize communication, collaboration, and cohesion between the traditionally separate developer and IT operations teams.

Rather than seeing these as two distinct groups who are responsible for their specific tasks but don’t really work together, the DevOps methodology recognizes the interdependence of the two groups. By integrating these functions as one team or department, DevOps helps an organization deploy software more frequently, while maintaining service stability and gaining the speed necessary for more innovation.

And, in the end, everyone is able to deliver the best results and overall experience possible to



the customer.

DevOps is about breaking down the wall between Developers and Operations.Speeding up deployments, such that we manage it more like Pipeline with many, smaller updates flowing through to Production on a more frequent basis.

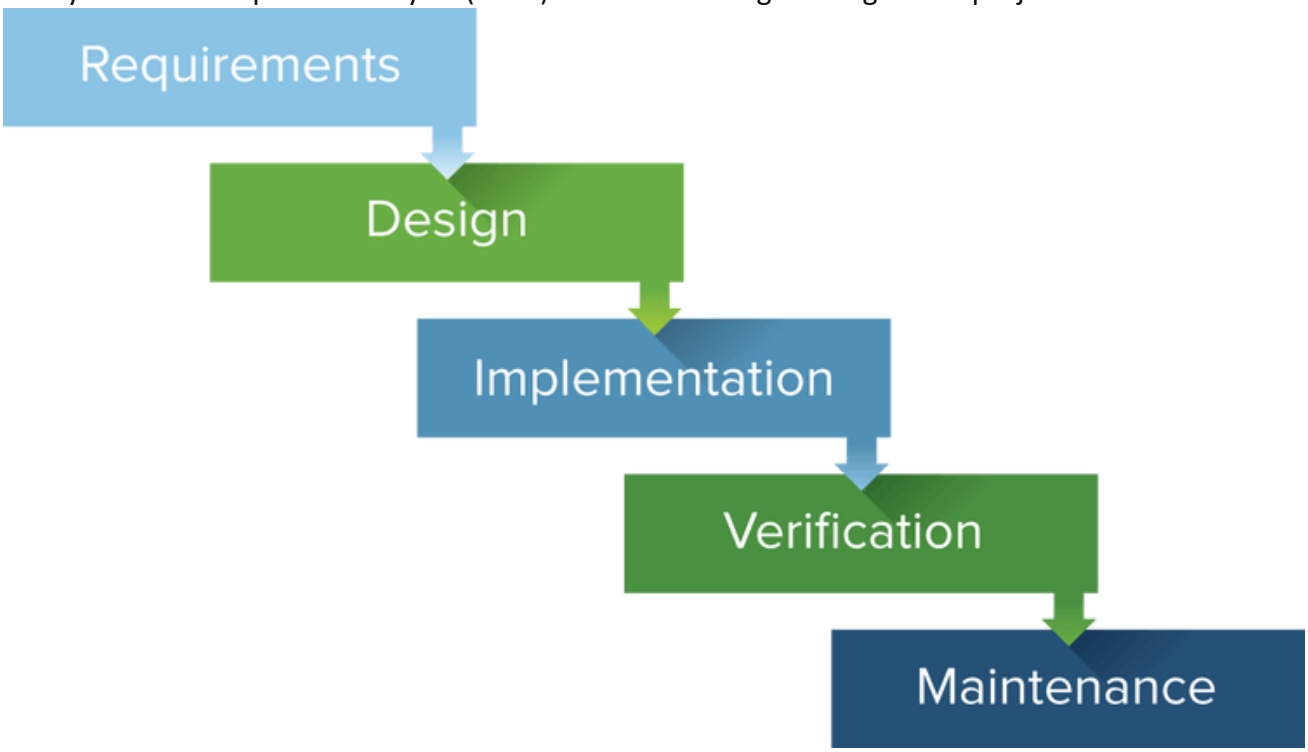
Benefits of using DevOps

- Improve deployment frequency
- Achieve faster time to market
- Lower failure rate of new releases
- Shorten lead time between fixes
- Improve mean time to recovery

2. Software Development Models

2.1 Waterfall Methodology

Waterfall methodology follows a sequential, linear process and is the most popular version of the systems development life cycle (SDLC) for software engineering and IT projects.



Advantages of Waterfall

Waterfall is best used for simple, unchanging projects. It's linear, rigid nature makes it easy to use and allows for in-depth documentation

- Easy to use and manage
- Discipline is enforced
- Requires a well-documented approach

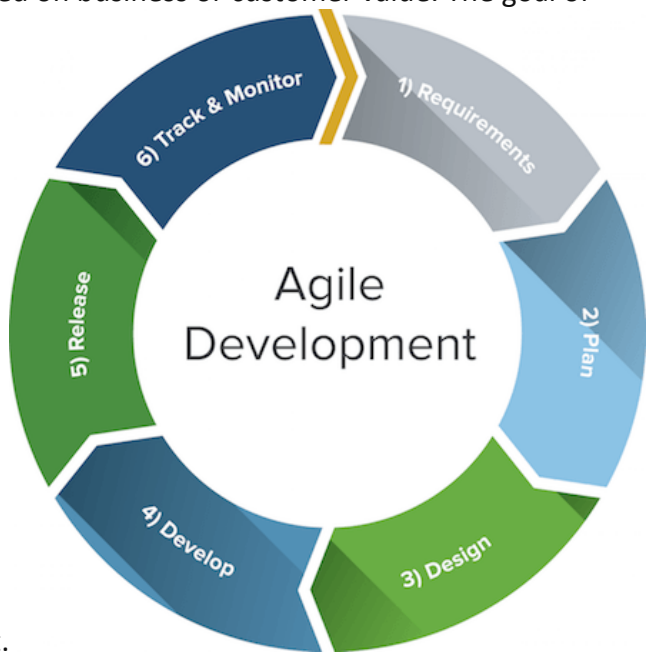
Disadvantages of Waterfall

The biggest drawback of Waterfall is how it handles change. Because Waterfall is a linear, sequential model, you can't bounce between phases, even if unexpected changes occur. Once you're done with a phase, that's it.

- Changes can't be easily accommodated
- Software isn't delivered until late
- Gathering accurate requirements can be challenging

2.2 Agile Methodology

Agile software development is based on an incremental, iterative approach. Instead of in-depth planning at the beginning of the project, agile methodologies are open to changing requirements over time and encourages constant feedback from the end users. Cross-functional teams work on iterations of a product over a period of time, and this work is organized into a backlog that is prioritized based on business or customer value. The goal of each iteration is to produce a working product.



Advantages of Agile

Agile evolved from different lightweight software approaches in the 1990s and is a response to some project managers' dislike of the rigid, linear Waterfall methodology. It focuses on flexibility, continuous improvement, and speed.

- Change is embraced
- End-goal can be unknown
- Faster, high-quality delivery
- Strong team interaction
- Customers are heard
- Continuous improvement

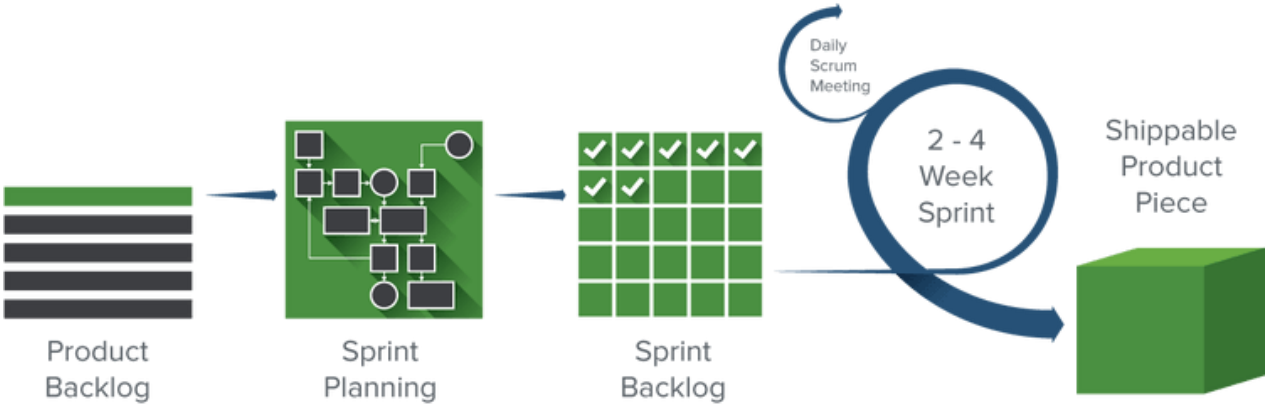
Disadvantages of Agile

While the level of flexibility in Agile is usually a positive, it also comes with some trade-offs. It can be hard to establish a solid delivery date, documentation can be neglected, or the final product can be very different than originally intended

- Planning can be less concrete
- Team must be knowledgeable
- Time commitment from developers
- Documentation can be neglected
- Final product can be very different

2.3 Scrum Methodology (subset of Agile)

Scrum is a **subset of Agile** and one of the most popular process frameworks for implementing Agile. It is an iterative software development model used to manage complex software and product development. Fixed-length iterations, called sprints lasting one to two weeks long, allow the team to ship software on a regular cadence. At the end of each sprint, stakeholders and team members meet to plan next
steps.



There are three specific roles in Scrum. They are:

- **Product Owner:** The Scrum Product Owner has the vision of what he or she wants to build and conveys that vision to the team.
- **Scrum Master:** Often considered the coach for the team, the Scrum Master helps the team do their best possible work. This means organizing meetings, dealing with roadblocks and challenges, and working with the Product Owner to ensure the product backlog is ready for the next sprint. The Scrum Master also makes sure the team follows the Scrum process
- **Scrum Team**: The Scrum Team is comprised of five to seven members. Everyone on the project works together, helps each other, and shares a deep sense of camaraderie

Advantages of Scrum

Scrum is a highly prescriptive framework with specific roles and ceremonies. While it can be a lot to learn, these rules have a lot of advantages. The benefits of Scrum include:

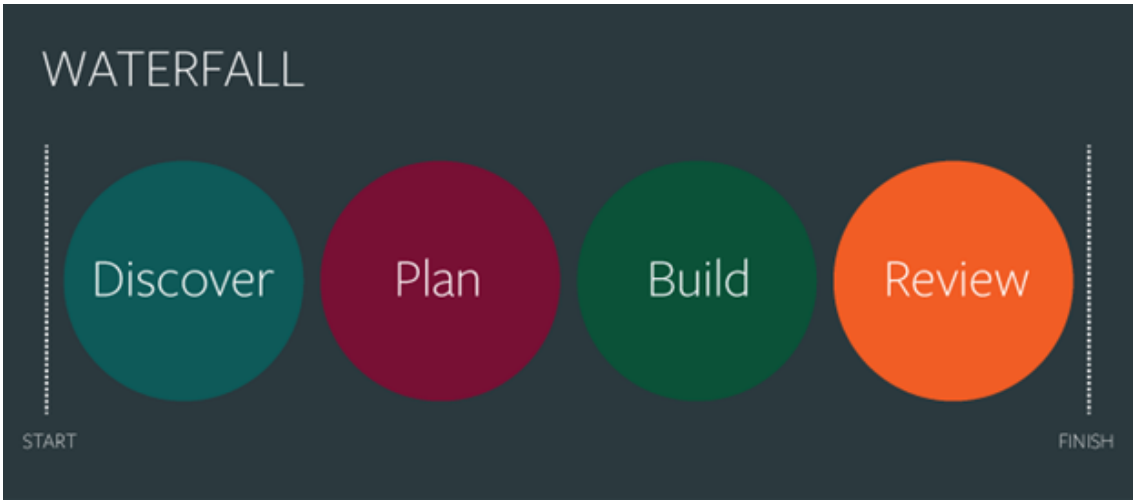
- More transparency and project visibility
- Increased team accountability
- Easy to accommodate
- Increased cost savings

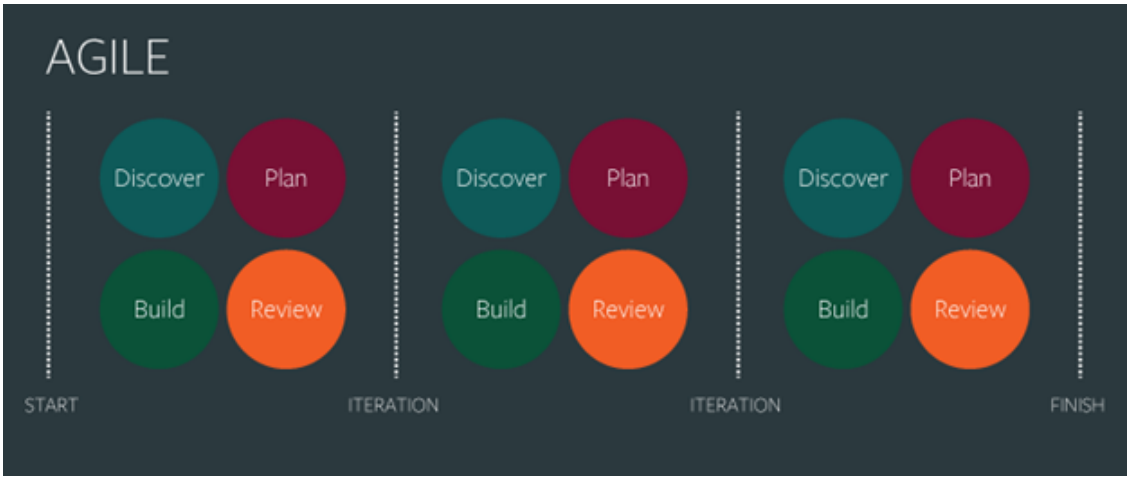
Disadvantages of Scrum

While Scrum offers some concrete benefits, it also has some downsides. Scrum requires a high level of experience and commitment from the team and projects can be at risk of scope creep.

- Risk of scope creep
- Team requires experience and commitment
- The wrong Scrum Master can ruin everything
- Poorly defined tasks can lead to inaccuracies

2.4 Waterfall VS Agile

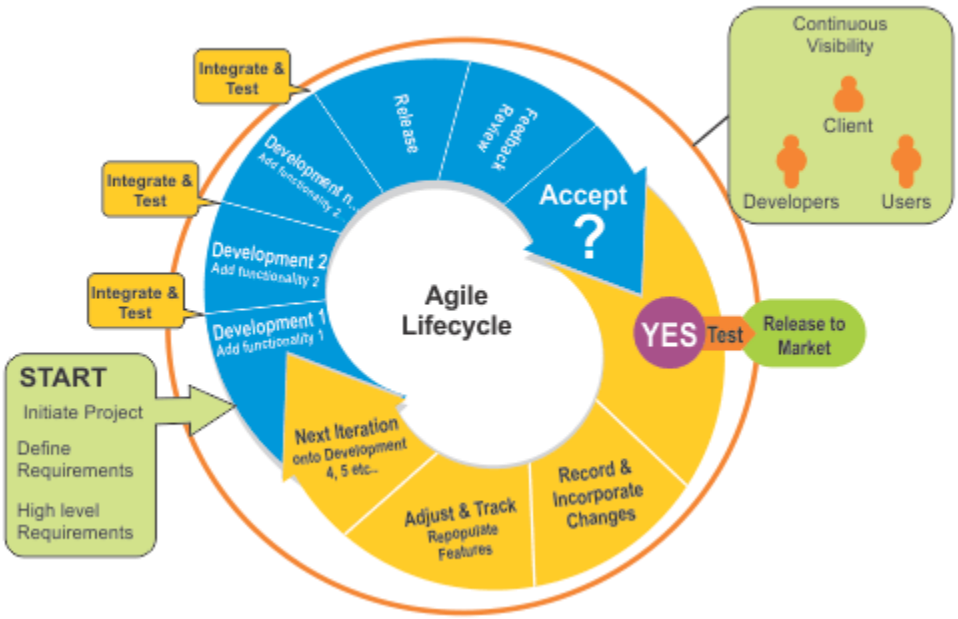




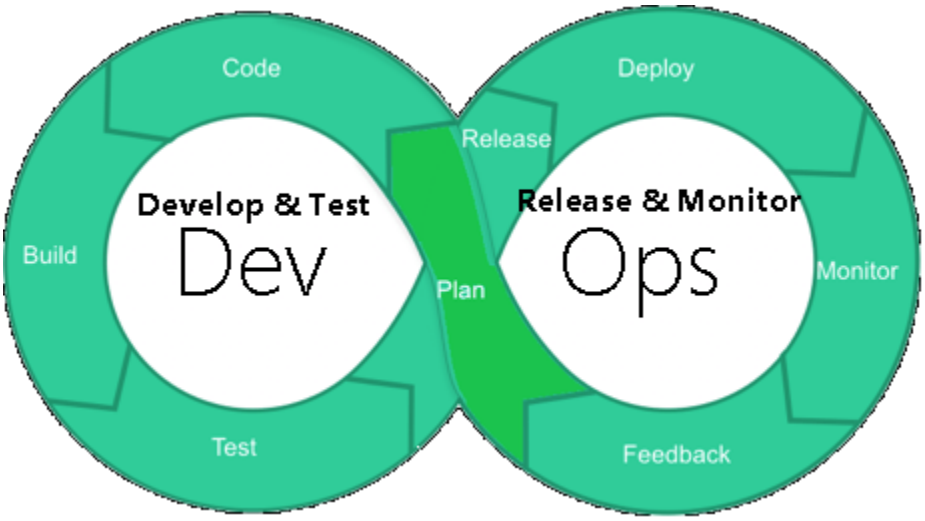
3. Agile Life Cycle VS DevOps Life Cycle

3.1 Agile Life cycle

In agile the tasks are divided to time boxes (small time frames) to deliver specific features for a release. Iterative approach is taken and working software build is delivered after each iteration. Each build is incremental in terms of features; the final build holds all the features required by the customer.



3.2 DevOps Life Cycle



- Implement code
- Run tests
- build repository
- Deploy and release
- Configure environment
- Ready to use, make tested updates available to users frequently

4. DevOps Implementation

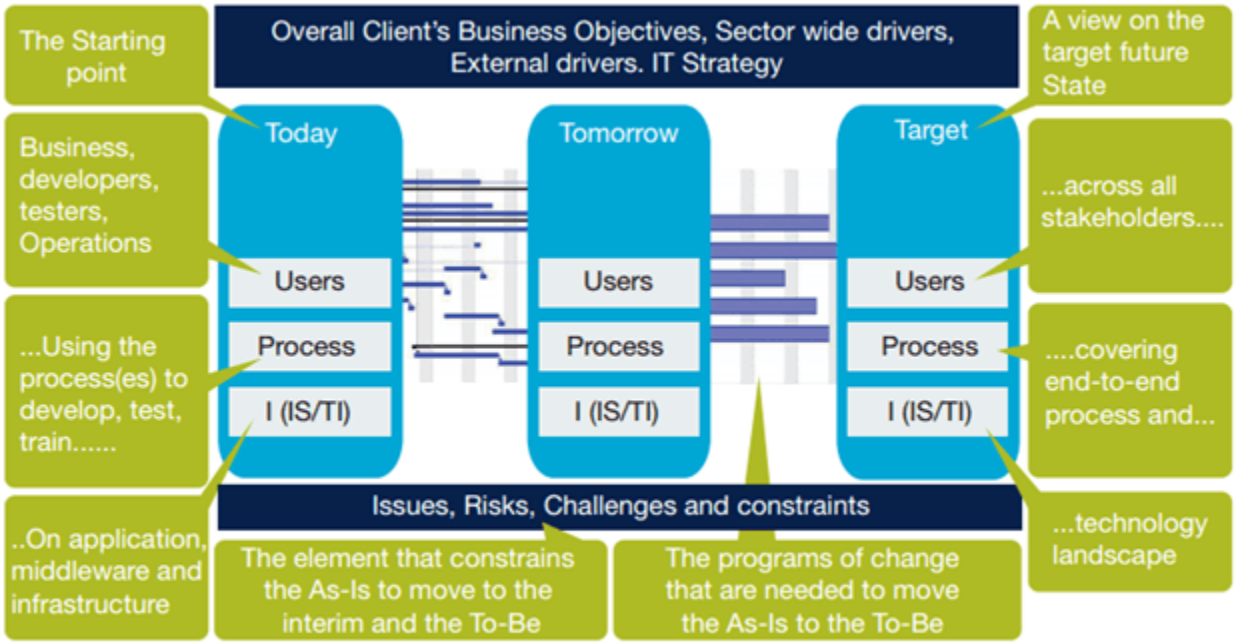
Three steps for a successful DevOps implementation:

1. **Define a target**
2. **Establish a plan**
3. **Manage plan execution**

4.1 Define a clear target

DevOps claims to reduce impact of changes to reduce cost and minimize impact to the live services. As applicable to every change project, the decision to change culture or processes and to deploy the right tools must be backed by a strong business case.

Many businesses struggle to take the right decisions at this stage. To estimate the benefits of DevOps implementation within their environment, they should analyze the existing situation — the existing tools, processes, resources, and their
skills.



DevOps Implementation Framework (DIF

4.2 Establish a clear transformation plan

Once a business case is created and approved, a detailed plan of actions is needed to manage the implementation of the changes needed to achieve the anticipated outcomes set out in the business case. Typically, three actions need to be followed:

1. **Change the culture**
2. **Establish one Development-to-Operations Lifecycle**
3. **Deploy common tooling**

The choice of activities that need to be executed for a solution depends on the actual context and needs to be established during the “define a clear target” step.

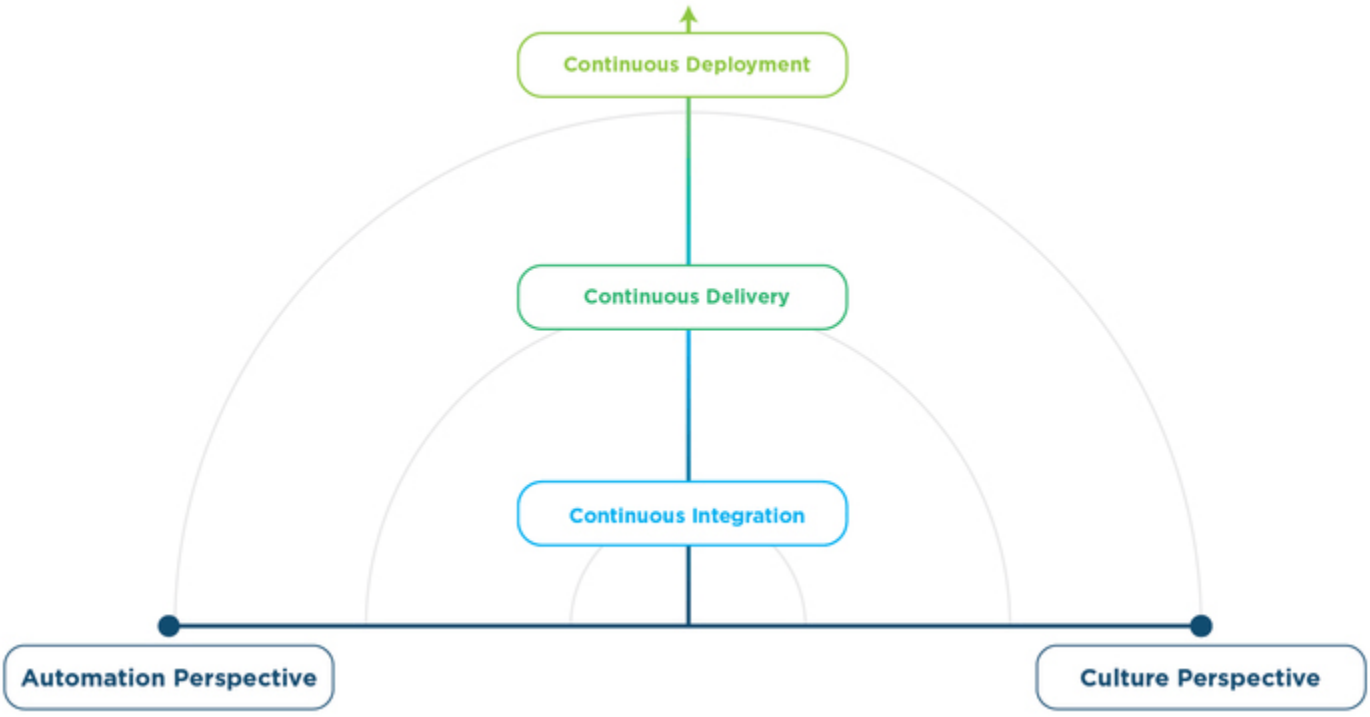
4.3 Actively manage the plan execution

In addition to careful formulation of the plan, it is important to carry out an efficient execution. **To successfully deploy DevOps a plan that covers people, process and tools is needed**

5. Phases of DevOps

The DevOps continuum is a helpful way to look at the different aspects of DevOps. The bottom horizontal axis represents what people perceive DevOps to fundamentally be focused on. Some people adamantly feel that DevOps should focus on culture more than tools, while on the other people tend to value tools over culture.

There are several phases to DevOps maturity; here are a few of the key phases you need to know.



Waterfall Development
Before continuous integration, development teams would write a bunch of code for three to four months. Then those teams would merge their code in order to release it. The different versions of code would be so different and have so many changes that the actual integration step could take months. This process was very unproductive.

5.1 Continuous Integration

Continuous integration is the practice of quickly integrating newly developed code with the main body of code that is to be released. Continuous integration saves a lot of time when the team is ready to release the code.

The continuous integration process from a DevOps perspective involves checking your code in, compiling it into usable (often binary executable) code and running some basic validation testing.

5.2 Continuous Delivery

Continuous delivery is an extension of continuous integration [DevOps stage 2]. It sits on top of continuous integration. When executing continuous delivery, you add additional automation and testing so that you don't just merge the code with the main code line frequently, but you get the code nearly ready to deploy with almost no human intervention. It's the practice of having the code base continuously in a ready-to-deploy state.

5.3 Continuous Deployment

Continuous deployment, not to be confused with continuous delivery is the most advanced evolution of continuous delivery**. It's the practice of deploying all the way into production without any human intervention.**

Teams that utilize continuous delivery don't deploy untested code; instead, newly created code runs through automated testing before it gets pushed out to production. The code release typically only goes to a small percentage of users and there's an automated feedback loop that monitors quality and usage before the code is propagated further.

There are a very small number of companies that are truly practicing continuous deployment. Netflix, Etsy, Amazon, Pinterest, Flicker, IMVU and Google are popular examples of companies doing continuous deployment.

While DevOps nirvana is often not the end goal for most enterprises, they often focus on moving towards continuous delivery.

6. DevOps Tools

Thinking like a Software Guy, and remembering it's about Speed & Agility, here's what the stack looks like Many choices for tool sets and point solutions, both Open Source & COTS

1.Project Plan & Team Management -Collaborate on vision and design

- **Confluence**
- **HipChat**
- **JIRA Software**
- **VersionOne**

2.Source Control Tools

- **Bitbucket**
- **GitLab**
- **GitHub**
- **CFEngine**
- **Subversion**
- **Mercurial**

3.Continuous Build Tools

- **Ant**
- **Maven**

4.Continuous Integration Tools

- **Jenkins**
- **Bamboo**
- **HipChat**
- **Cruise control**
- **Continua CI**
- **Buildbot**

5.Test Automation Tools

- **Cucumber**
- **Selenium**
- **Appium**
- **TestNG**

- **JUnit**
- **Tosca**
- **SonarQube**

6.Continuous Deployment Tools

- **Puppet**
- **Chef**
- **Docker**
- **Splunk**