

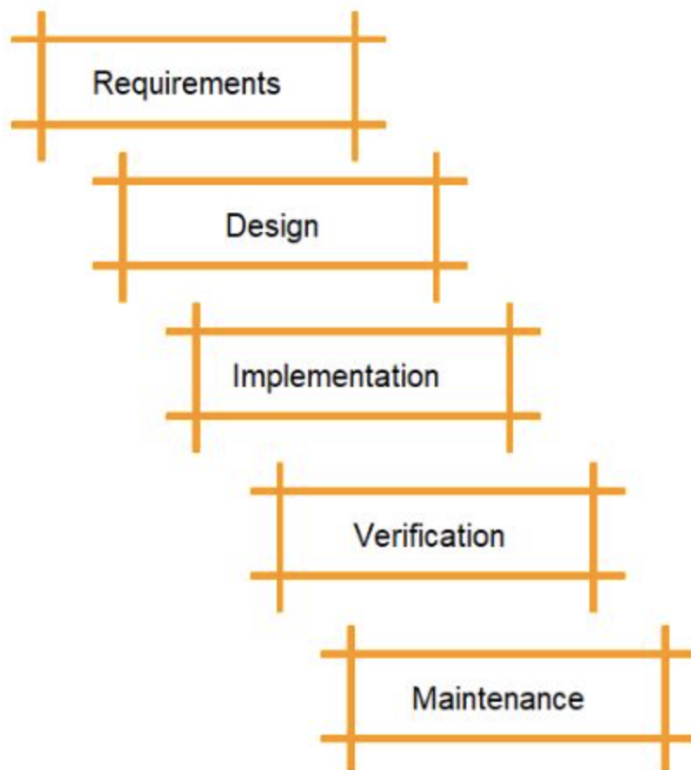
## Why DevOps?

Before we get deep into what DevOps is and all the revolutions it brought with us, first understand why DevOps in the first place. and Before DevOps, there were two development models: Waterfall and Agile Methods.

### 1. Waterfall Model

The waterfall model is the first model to be introduced in software development. It is a sequential process and very easy to understand. In this approach, software development is divided into several phases, and the output of one phase becomes the input for the next phase. This model is similar to a waterfall when the water flows off from the cliff; it cannot go back to its previous state.

The phases are; Requirements, Design, Implementation, Verification, and Maintenance.



Drawbacks of the waterfall model:

- It's difficult to make changes to the previous stage
- Not recommended for large-sized projects
- Developers and testers don't work together (which can result in a lot of bugs at the end)
- Not recommended for projects that will likely have changing requirements

From the figure below, we can see the issues with the waterfall model:

- The developer took a very long time to deploy code
- On the operations side, the tester found it challenging to identify problems and give useful feedback

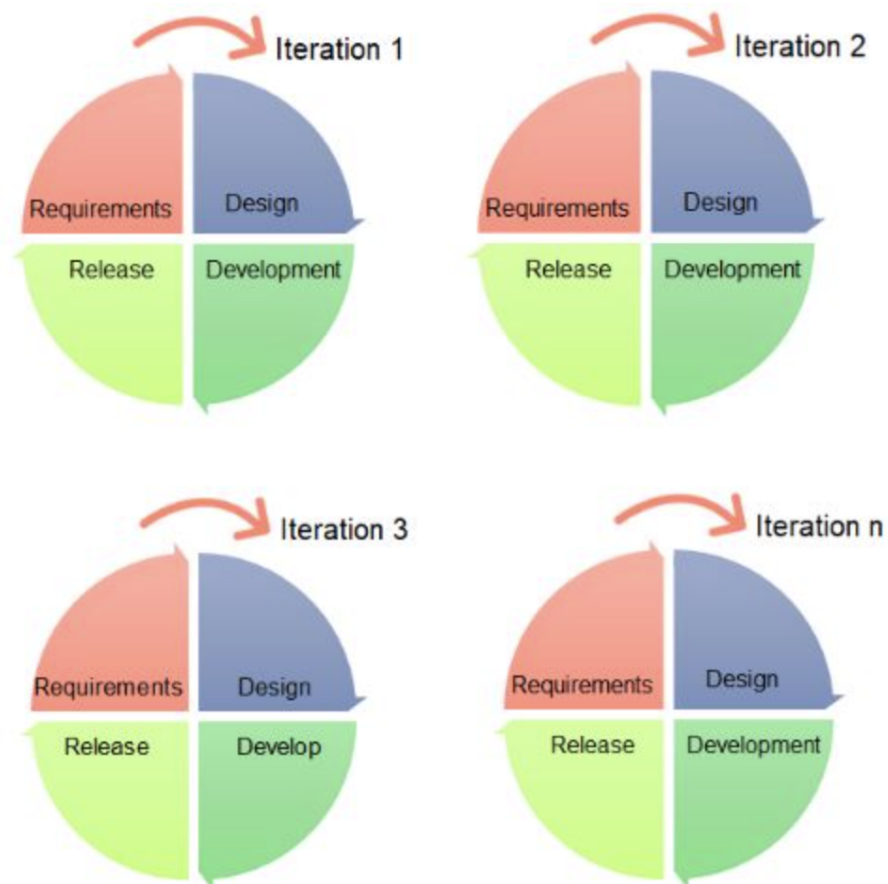


## 2. Agile Model

Agile is an approach in software development where each project splits into multiple iterations. As a result, at the end of each iteration, a software product is delivered. Each iteration lasts about one to three weeks. Every iteration involves functional teams working simultaneously on various areas, such as:

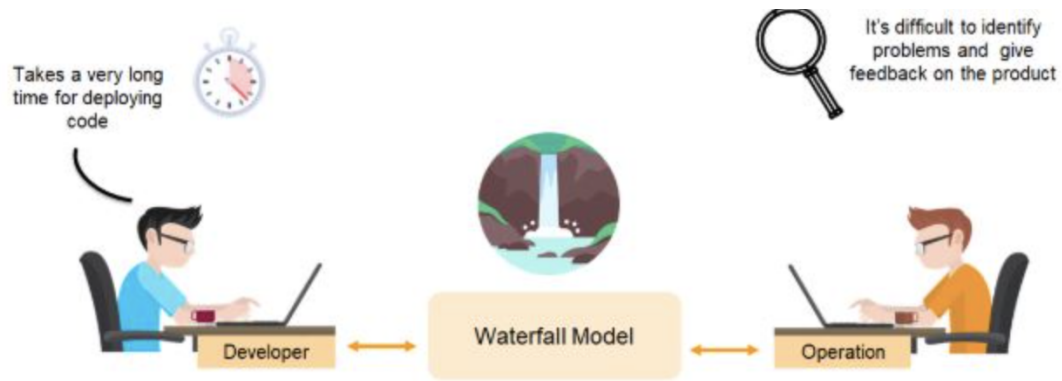
- Requirements
- Design
- Development
- Release

The figure below indicates that there can be a number of iterations needed to deliver a final product in the agile method.



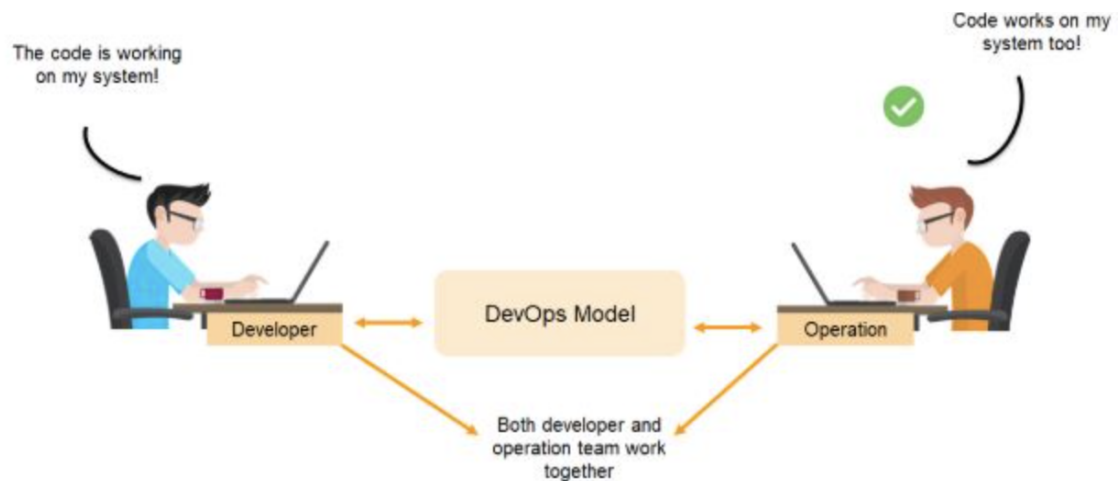
Using the agile method, the code that works for the developer may not work for the operations team.

So how can this issue be solved?



With DevOps, there is continuous integration between deployment of code and the testing of it. Near real-time monitoring and immediate feedback through a DevOps continuous monitoring tool enables both the developer and operations team work together.

The figure below shows how well the software is handled using DevOps.



## What is DevOps?

DevOps is a collaboration between development and operation teams, which enables continuous delivery of applications and services to our end users.

## Benefits of DevOps:

- Continuous delivery of software
- Better collaboration between teams
- Easy deployment
- Better efficiency and scalability
- Errors are fixed at the initial stage
- More security
- Less manual intervention (which means fewer chances of error)

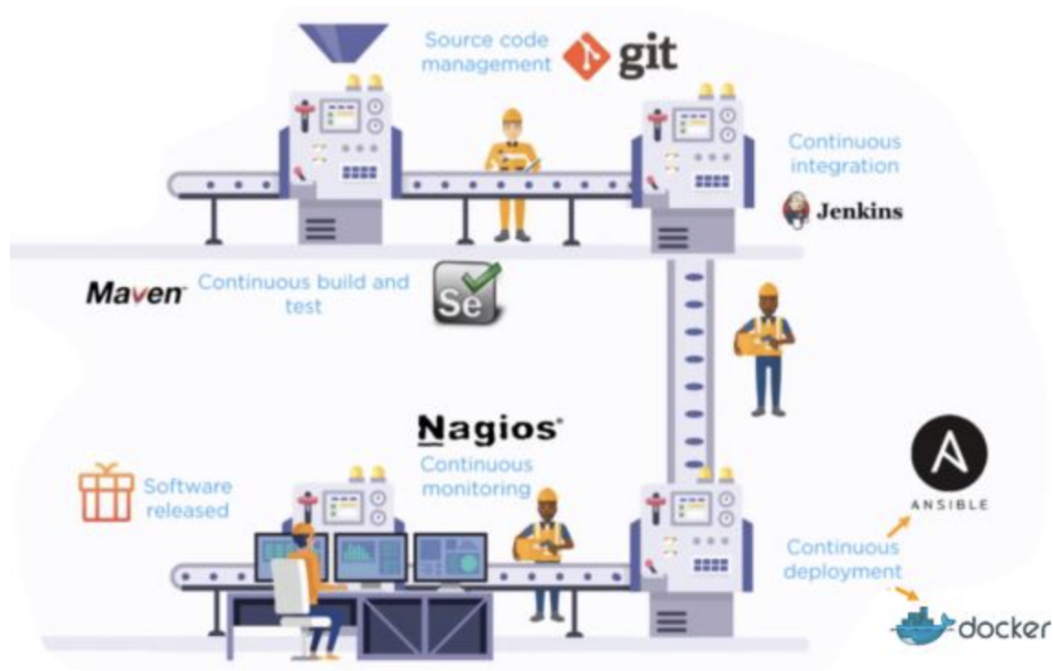
## DevOps Lifecycle

Now that you know why DevOps and what it is exactly, we will learn all about the DevOps Lifecycle which will give you a clarity on why DevOps, that is divided into six different phases which will give a clear idea on why DevOps:

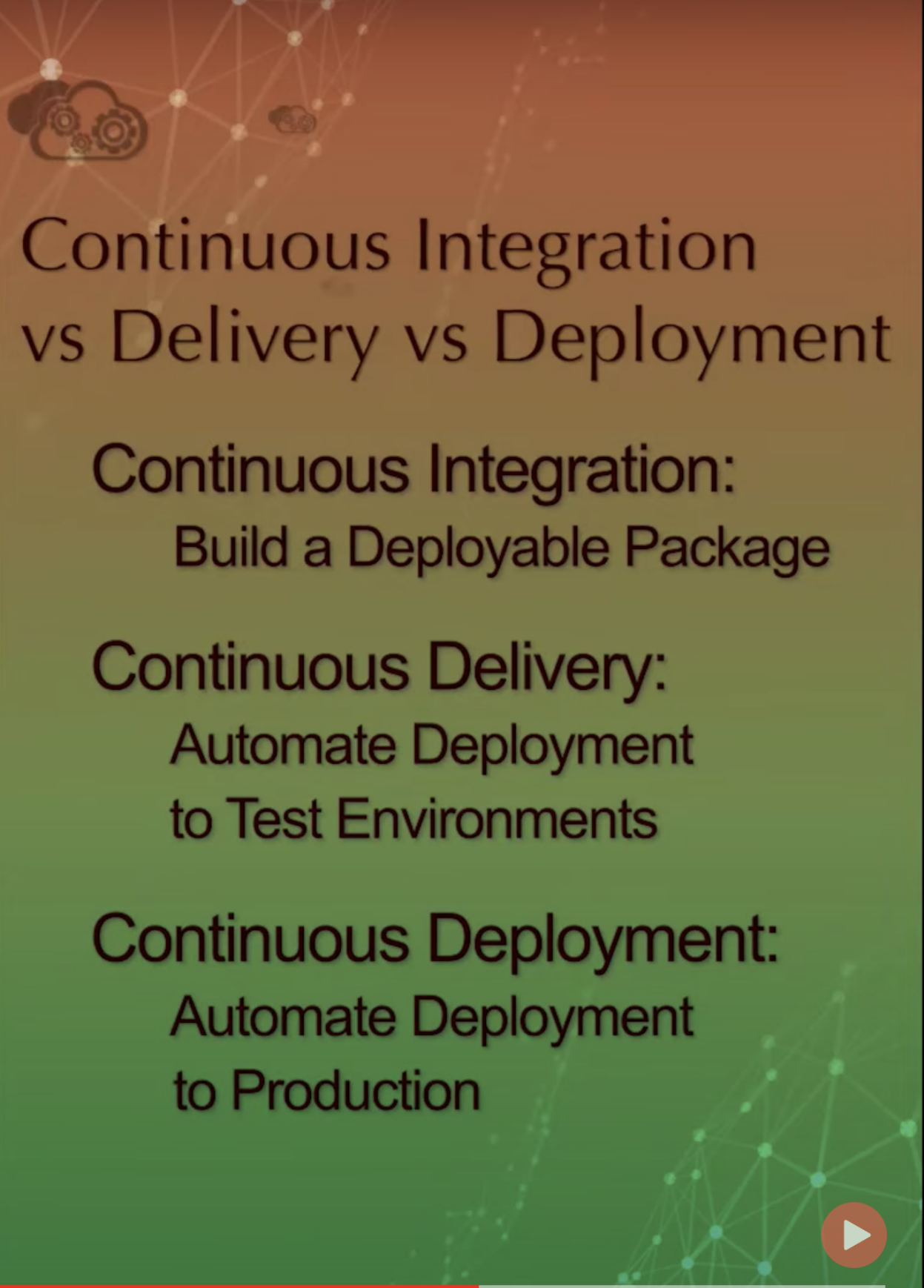
- Source Code Management - In this phase, the business owners and software development team discuss project goals and create a plan. Programmers then design and code the application, using tools like Git to store the application code.
- Continuous Build and Test - This phase deals with building tools, like Maven and Gradle, then taking code from different repositories and combining them to build the complete application. The application is then tested using automation testing tools, like Selenium and JUnit, to ensure software quality.
- Continuous Integration - When the testing is complete, new features are integrated automatically to the existing codebase.
- Continuous Deployment - Here, the application is packaged after being released and deployed from the development server to the production server. Once the software is deployed, operations teams perform tasks, such as configuring servers and provisioning them with the required resources.
- Continuous Monitoring - Monitoring allows IT organizations to identify issues of specific releases and understand the impact on end-users.
- Software Released - After all the phases are completed and the software meets the user's requirement, it is released into the market.

## Tools in DevOps

After why DevOps and its lifecycle, let us look at the various tools used in DevOps.



What are the differences between continuous integration, continuous delivery, and continuous deployment (CI/CD)?



# Continuous Integration vs Delivery vs Deployment

**Continuous Integration:**  
Build a Deployable Package

**Continuous Delivery:**  
Automate Deployment  
to Test Environments

**Continuous Deployment:**  
Automate Deployment  
to Production



Test environments[separate AWS Account] can be staging, sanity ,dev etc

What are the differences between continuous integration, continuous delivery, and continuous deployment (CI/CD)?

Continuous integration

**Continuous Integration** is the practice where developers merge the changes to the code base to the main branch as often as possible. These changes are validated by creating a build and then running automated tests against the build. If these tests don't pass, the changes aren't merged, and developers avoid integration challenges that can happen

## Benefits of Continuous Integration

This process also causes fewer bugs to be shipped to production as the issues are caught early and integration issues are solved before release.

## Continuous Delivery (CD)

**Continuous Delivery** is an extension of CI since it enables automation to deploy all the code changes to an environment (dev, qa, stage etc) after the changes have been merged. The artifact may be built as part of CI or as part of this process since the source of truth (your repository) is reliable



given your CI process. In simple terms, this means that there is an automated release process on top of the automated testing process and that developers can deploy their changes at any time by simply clicking a button or at the completion of CI.

## Benefits of Continuous Delivery

Since developers can deploy their changes at any time, it's recommended to deploy the changes to production as often as possible, making troubleshooting easier and providing your users with access to the best your product has to offer as soon as possible. Often times, the release to production may be managed by a Release Manager and governed by a compliance process to ensure organizational processes are being met. By enabling non-technical team members to control this process, you can reduce the burden on the development team so they may continue to execute on further application improvements

## Continuous Deployment (CD)

Continuous Deployment takes the process one step further than continuous delivery. Here, all changes that pass the verification steps at each stage in the pipeline are released to production. This process is completely automated and only a failed verification step will prevent pushing the changes to production.

# Benefits of Continuous Deployment

Apart from the fact that customers get updates quicker, developers also get feedback faster which means there is less pressure as small changes are pushed incrementally compared to big updates not that often. In order to successfully accomplish Continuous Deployment, tracking metrics around Mean Time to Repair and Change Failure Rate is critical to the success of fully automated deployments.

Now, that we've recapped what these all mean, let's look at the specific difference, which to some extent may already be apparent.

## **Difference between Continuous Integration vs Continuous Delivery vs Continuous Deployment**

Here is an important difference between Continuous Integration vs Continuous Delivery vs Continuous Deployment.

<b>Continuous Integration</b>	<b>Continuous Delivery</b>	<b>Continuous Deployment</b>
<b>CI is an approach of testing each change to codebase automatically.</b>	<b>CD is an approach to obtain changes of new features, configuration, and bug fixes.</b>	<b>CD is an approach to develop software in a short cycle.</b>
<b>CI refers to the versioning of source code.</b>	<b>CD refers to the logical evolution of CI.</b>	<b>CD refers to automated implementations of the source code.</b>
<b>CI focuses on automation testing to determine that the software has no errors or bugs.</b>	<b>Focuses on releasing new changes to your clients properly.</b>	<b>Emphasis on the change in all stages of your production pipeline.</b>
<b>CI is performed immediately after the developer checks-in.</b>	<b>In CD, developed code is continuously delivered until the programmer considers it is ready to ship.</b>	<b>In CD, developers deploy the code directly to the production stage when it is developed.</b>
<b>It helps you to identify and rectify issues early.</b>	<b>It allows developers to check software updates.</b>	<b>It enables you to rapidly deploy and validate new features and ideas.</b>

It uses unit tests.	It uses business logic tests.	Any testing strategy is performed.
Development team sends continuous code merging requests even when the testing process is running.	You deliver code for review that can be batched for release.	Deploy code using an automated process.
You require a continuous integration server to monitor the main repository.	You require a strong foundation in continuous integration.	You need a good testing culture

