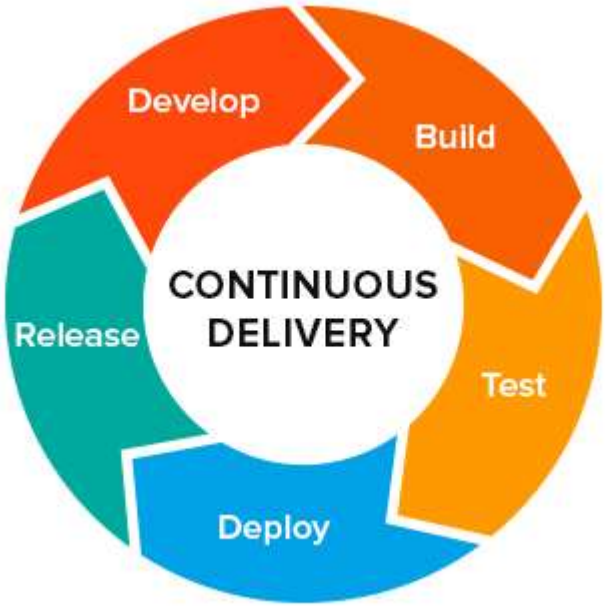
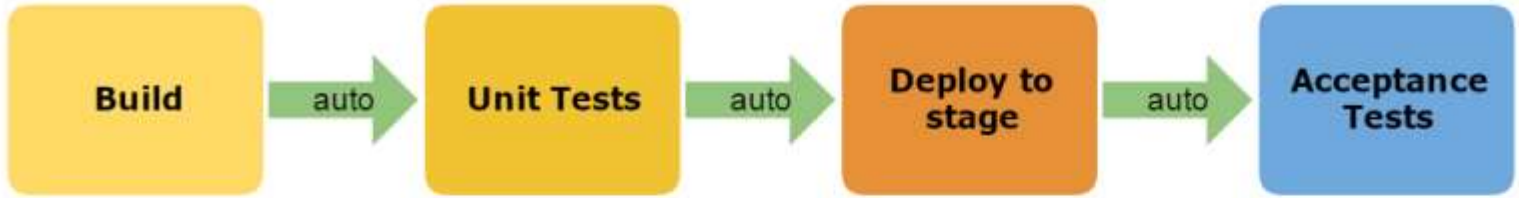
CONTINUOUS DELIVERY



- **Continuous Integration** usually refers to integrating, building, and testing code within the development environment. Continuous Delivery builds on this, dealing with the final stages required for production deployment.
- **Continuous Delivery** just means that you are able to do frequent deployments but may choose not to do it, usually due to businesses preferring a slower rate of deployment. In order to do Continuous Deployment you must be doing Continuous Delivery.
- **Continuous Deployment** means that every change goes through the pipeline and automatically gets put into production, resulting in many production deployments every day.

Continuous Integration



Continuous Delivery

Continuous Deployment

Continuous Delivery

- Continuous Delivery is a software engineering approach in which teams produce software in short cycles, ensuring that software can be reliably released at any time faster and more frequently.
- It aims to build, test and release software faster and more frequently.
- It reduces the cost, time, and risk of delivering changes by allowing for more incremental updates to production.

- Continuous delivery provides many benefits, including:
  - It encourages Infrastructure as Code and Configuration as Code.
  - It enables automated testing throughout the pipeline.
  - It provides visibility and fast feedback cycles.
  - It makes going to production a low stress activity.

- A deployment pipeline is an automated implementation of your application's build, deploy, test, and release process.
- Every organization will have differences in the implementation of their deployment pipelines, depending on their value stream for releasing software, but the principles that govern them do not vary.
- An example of a deployment pipeline is given below:-

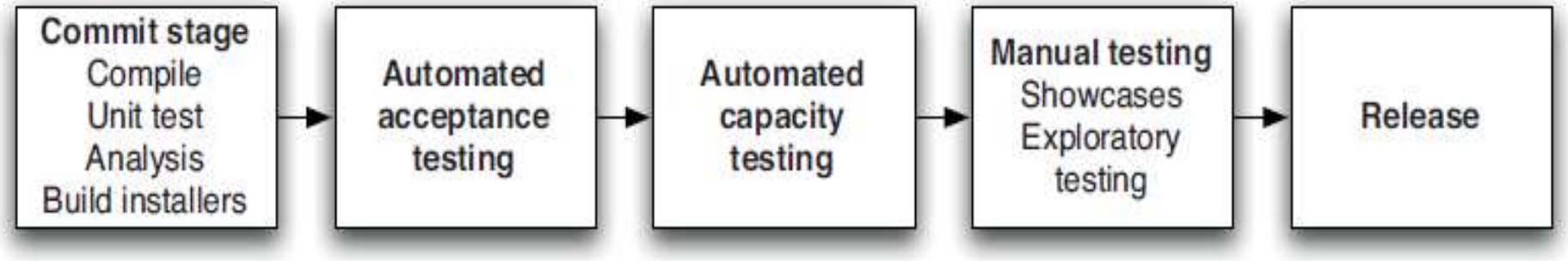
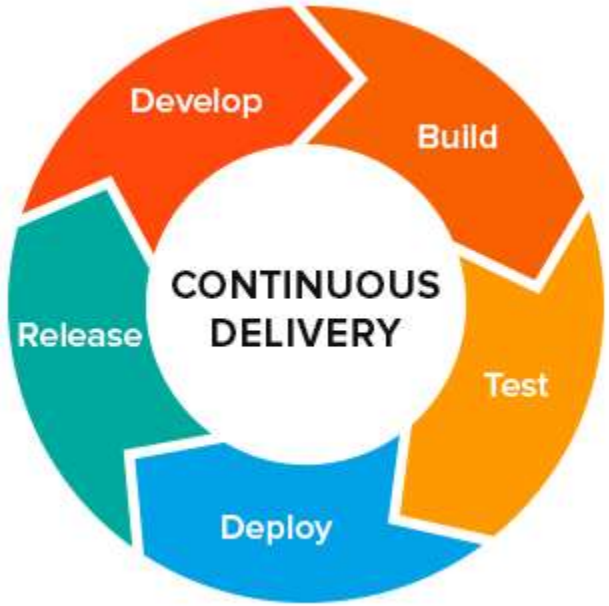
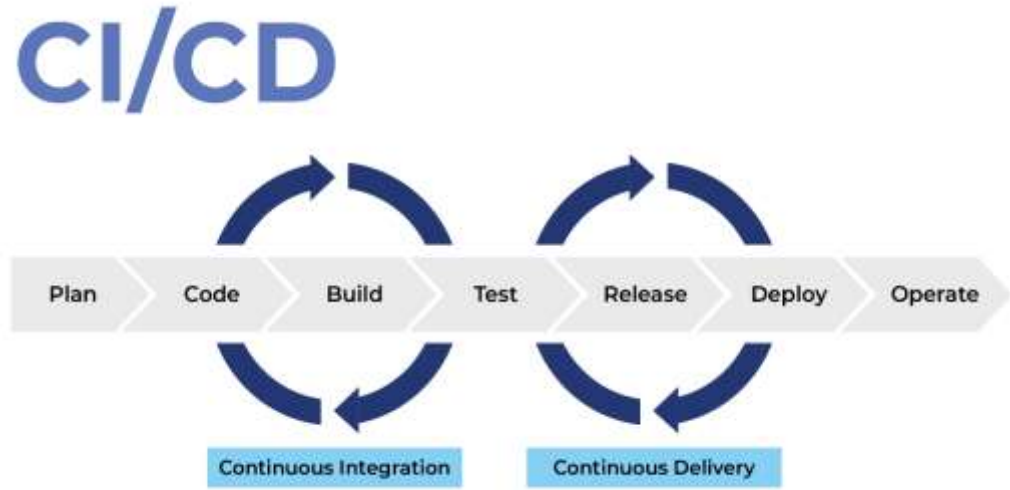
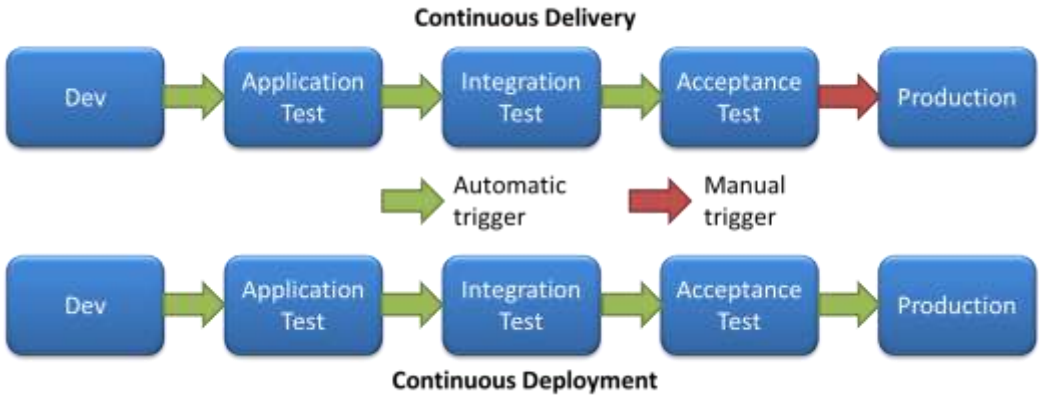


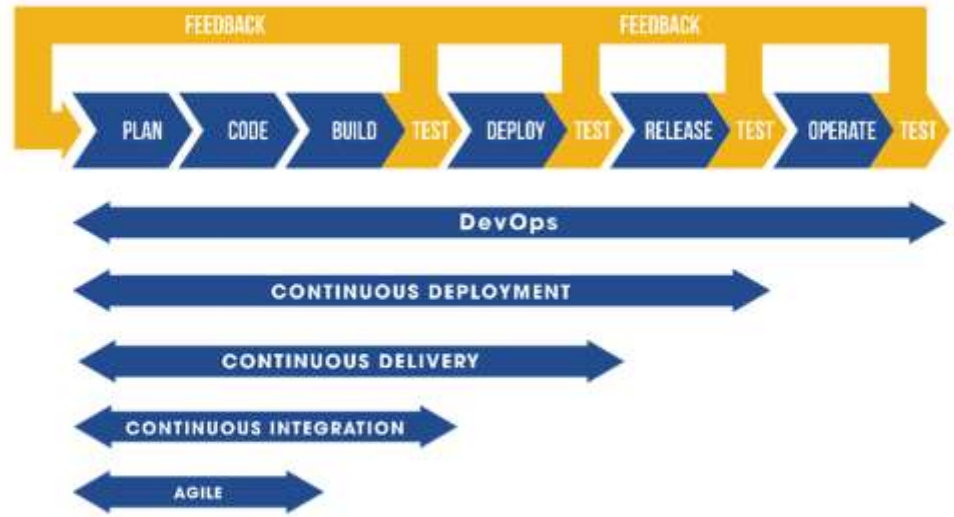
Figure 1.1 *The deployment pipeline*

- The aim of the deployment pipeline is threefold.
- **First**, it makes every part of the process of building, deploying, testing, and releasing software visible to everybody involved, aiding collaboration.
- **Second**, it improves feedback so that problems are identified, and so resolved, as early in the process as possible.
- **Finally**, it enables teams to deploy and release any version of their software to any environment at will through a fully automated process.
- Hence , now our goal actually ***is to find ways to deliver high-quality, valuable software in an efficient, fast, and reliable manner.***


Develop
Build
Test
Deploy
Release
CONTINUOUS
DELIVERY


CI/CD
Plan
Code
Build
Test
Release
Deploy
Operate
Continuous Integration
Continuous Delivery


Continuous Delivery
Dev
Application Test
Integration Test
Acceptance Test
Production
Automatic trigger
Manual trigger
Dev
Application Test
Integration Test
Acceptance Test
Production
Continuous Deployment


FEEDBACK
FEEDBACK
PLAN
CODE
BUILD
TEST
DEPLOY
TEST
RELEASE
TEST
OPERATE
TEST
DevOps
CONTINUOUS DEPLOYMENT
CONTINUOUS DELIVERY
CONTINUOUS INTEGRATION
AGILE

Principles of Software Delivery

1. Create a Repeatable, Reliable Process for Releasing Software:-

- Releasing software should be easy because you have tested every single part of the release process hundreds of times before.
- The repeatability and reliability derive from two principles: automate almost everything, and keep everything you need to build, deploy, test, and release your application in version control.

- Deploying software ultimately involves three things:

1. Provisioning and managing the environment in which your application will run (hardware configuration, software, infrastructure, and external services).
2. Installing the correct version of your application into it.
3. Configuring your application, including any data or state it requires.

2. Automate Almost Everything

- Automation is a prerequisite for the deployment pipeline, because it is only through automation that we can guarantee that people will get what they need at the push of a button.
- You don’t need to automate everything at once. You can, and should, automate gradually over time.
- There are some things it is impossible to automate.(eg:- Demonstrations of working software to representatives of your user community cannot be performed by computers.) However, the list of things that cannot be automated is much smaller than many people think.

3. Keep Everything in Version Control

- Everything you need to build, deploy, test, and release your application should be kept in some form of versioned storage.
- This includes requirement documents, test scripts, automated test cases, network configuration scripts, deployment scripts, database creation, upgrade, downgrade, and initialization scripts, application stack configuration scripts, libraries, toolchains, technical documentation, and so on.
- All of this stuff should be version-controlled, and the relevant version should be identifiable for any given build.

4. If It Hurts, Do It More Frequently, and Bring the Pain Forward

- Integration is often a very painful process. If this is true on your project, integrate every time somebody checks in, and do it from the start of the project.
- If testing is a painful process that occurs just before release, don’t do it at the end. Instead, do it continually from the beginning of the project.
- If releasing software is painful, aim to release it every time somebody checks in a change that passes all the automated tests.
- If you can’t release it to real users upon every change, release it to a production-like environment upon every check-in.
- If creating application documentation is painful, do it as you develop new features instead of leaving it to the end.

5. *Build Quality In*

- “Build quality in” “Bring the pain forward” --catch defects as early in the delivery process as possible and the next step is to fix them.
- Delivery teams must be disciplined about fixing defects as soon as they are found. (Eg:- A fire alarm is useless if everybody ignores it.)

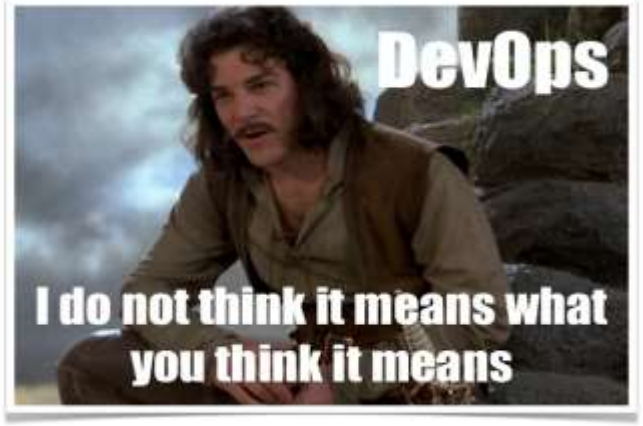
6. *Done Means Released*

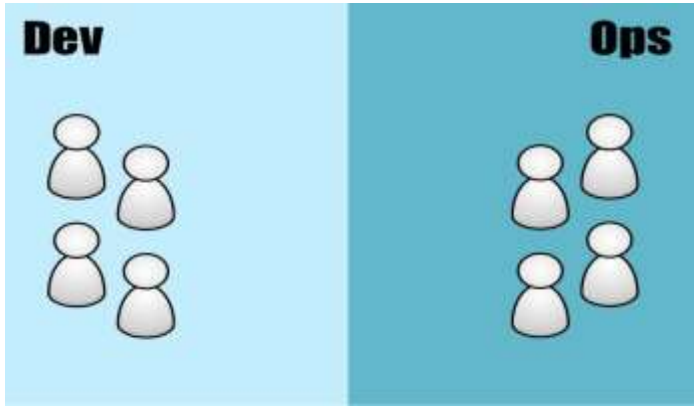
- A feature is only done when it is delivering value to users.
- There is no “80% done.” Things are either done, or they are not.

7. Everybody Is Responsible for the Delivery Process

- everybody within an organization is aligned with its goals, and people work together to help each to meet them. Ultimately the team succeeds or fails as a team, not as individuals.

- This is one of the central principles of the DevOps movement. The DevOps movement – encouraging greater collaboration between everyone involved in software delivery in order to release valuable software faster and more reliably.





8. Continuous Improvement

- It is worth emphasizing that the first release of an application is just the first stage in its life.
- All applications evolve, and more releases will follow. It is important that your delivery process also evolves with it.
- The whole team should regularly gather together and hold a retrospective on the delivery process.
- This means that the team should reflect on what has gone well and what has gone badly, and discuss ideas on how to improve things.
- Somebody should be nominated to own each idea and ensure that it is acted upon.Then, the next time that the team gathers, they should report back on what happened.
- This is known as the *Deming cycle*: plan, do, study, act.