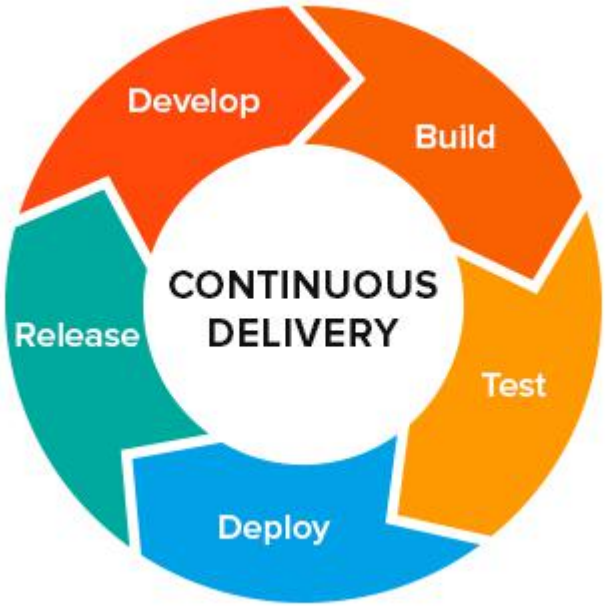
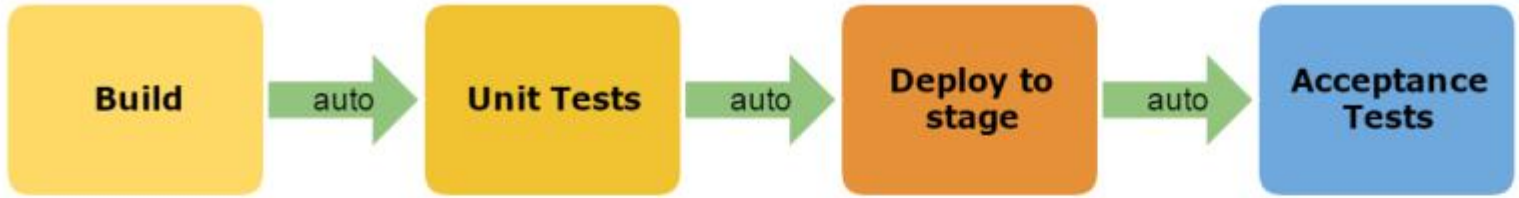
CONTINUOUS DELIVERY



- **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.
- 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** is a software development practice where code changes are automatically prepared for a release to production.
- Continuous delivery lets developers automate testing beyond just unit tests so they can verify application updates across multiple dimensions before deploying to customers.
- In practice, continuous delivery focuses on automated deployment pipeline. This may have one or more manual approval gates prior to reaching 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:-

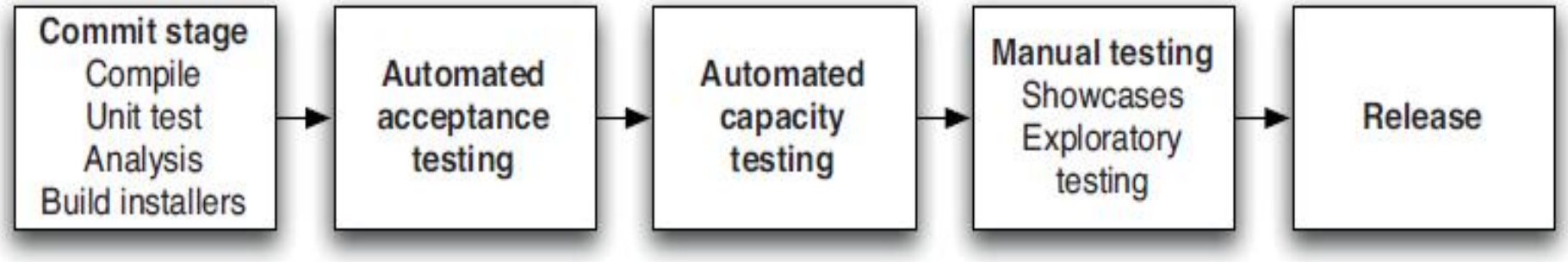


Figure 1.1 *The deployment pipeline*

The way the deployment pipeline works is as follows:-

- Every change that is made to an application's configuration, source code, environment, or data, triggers the creation of a new instance of the pipeline.
- One of the first steps in the pipeline is to create binaries and installers.
- The rest of the pipeline runs a series of tests on the binaries to prove that they can be released.
- Each test that the release candidate passes gives us more confidence that this particular combination of binary code, configuration information, environment, and data will work.
- If the release candidate passes all the tests, it can be released.

- 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.

- Our goal as software professionals is to deliver useful, working software to users as quickly as possible.
- Speed is essential because there is an opportunity cost associated with not delivering software. (You can only start to get a return on your investment once your software is released.)
- Find ways to reduce ***cycle time :--*** the time it takes from deciding to make a change, whether a bugfix or a feature, to having it available to users.

- Delivering fast is also important because it allows you to verify whether your features and bug fixes really are useful.
- An important part of usefulness is quality. Our software should be fit for its purpose.
- our goal should always be to deliver software of sufficient quality to bring value to its users.
- So while it is important to deliver our software as quickly as possible, it is essential to maintain an appropriate level of quality.

- Hence , now our goal actually ***is to find ways to deliver high-quality, valuable software in an efficient, fast, and reliable manner.***
- In order to achieve these goals—low cycle time and high quality—we need to make frequent, automated releases of our software.
- **Automated:-** If the build, deploy, test, and release process is not automated, it is not repeatable. Every time it is done, it will be different, because of changes in the software, the configuration of the system, the environments, and the release process. Since the steps are manual, they are error-prone, and there is no way to review exactly what was done.

- **Frequent.** If releases are frequent, the delta between releases will be small. This significantly reduces the risk associated with releasing and makes it much easier to roll back. Frequent releases also lead to faster feedback.
- Feedback is essential to frequent, automated releases. There are three criteria for feedback to be useful.

1. Any change needs to trigger the feedback process.
2. The feedback must be delivered as soon as possible.
3. The delivery team must receive feedback and then act on it.

1. Every Change Should Trigger the Feedback Process

- A working software application can be usefully decomposed into four components: executable code, configuration, host environment, and data.
- If any of them changes, it can lead to a change in the behavior of the application. Therefore we need to keep all four of these components under control and ensure that a change in any one of them is verified.

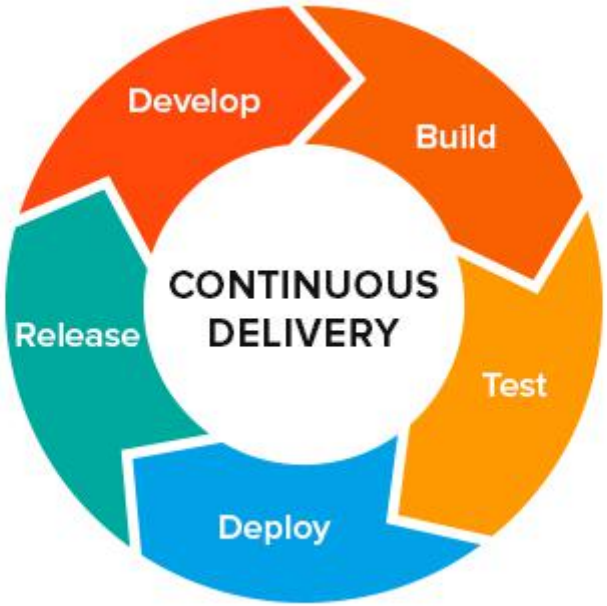
2. The Feedback Must Be Received as Soon as Possible

- The key to fast feedback is automation.
- If you have manual processes, you are dependent on people to get the job done. People take longer, they introduce errors, and they are not auditable. Moreover performing manual build, test, and deployment processes is boring and repetitive.

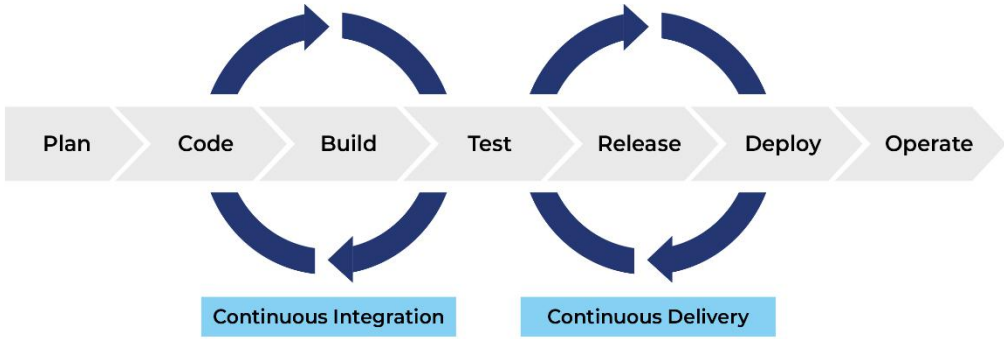
3. The Delivery Team Must Receive Feedback and Then Act on It

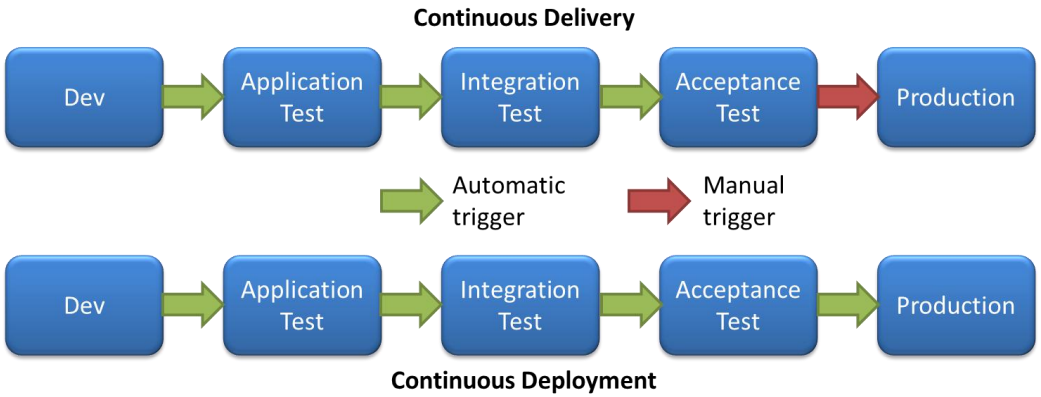
- It is essential that everybody (developers, testers, operations staff, database administrators, infrastructure specialists, and managers) involved in the process of delivering software is involved in the feedback process.
- If people in these roles do not work together on a day-to-day basis, it is essential that they meet frequently and work to improve the process of delivering software.
- A process based on continuous improvement is essential to the rapid delivery of quality software.
- Iterative processes help this kind of activity—at least once per iteration a retrospective meeting is held where everybody discusses how to improve the delivery process for the next iteration.

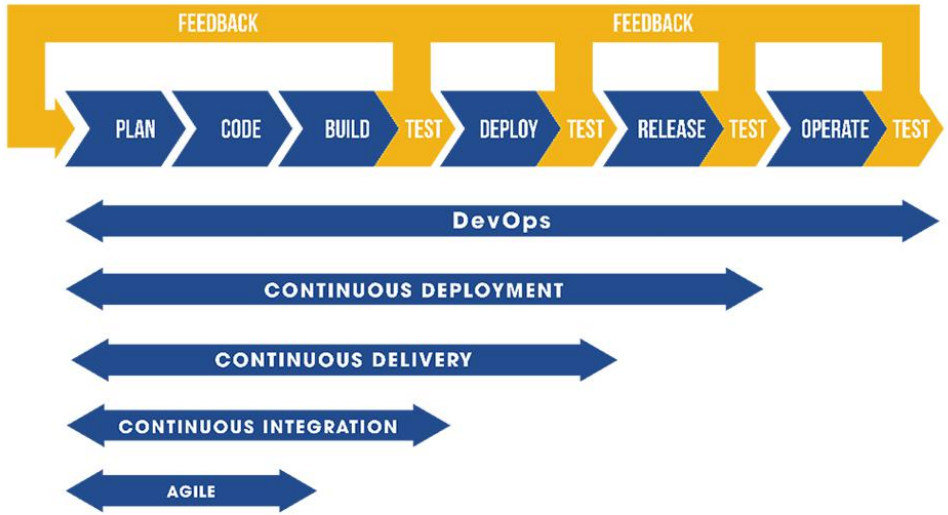
- Being able to react to feedback also means broadcasting information. Using big, visible dashboards (which need not be electronic) and other notification mechanisms is central to ensuring that feedback is fed-back and makes the final step into someone's head.
- Finally, feedback is no good unless it is acted upon. This requires discipline and planning.
- When something needs doing, it is the responsibility of the whole team to stop what they are doing and decide on a course of action. Only once this is done should the team carry on with their work.


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. It should be as simple as pressing a button.
- It 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

- Most development teams don’t automate their release process because it seems such a daunting task.
- It’s easier just to do things manually. Perhaps that is true the first time they perform a step in the process, but it is certainly not true by the time they perform that step for the tenth time.
- 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.
- That is, these *change sets* should have a single identifier, such as a build number or a version control change set number, that references every piece.

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.)

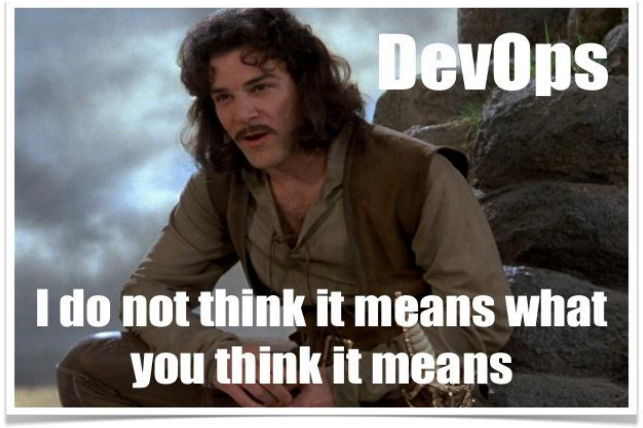
6. Done Means Released

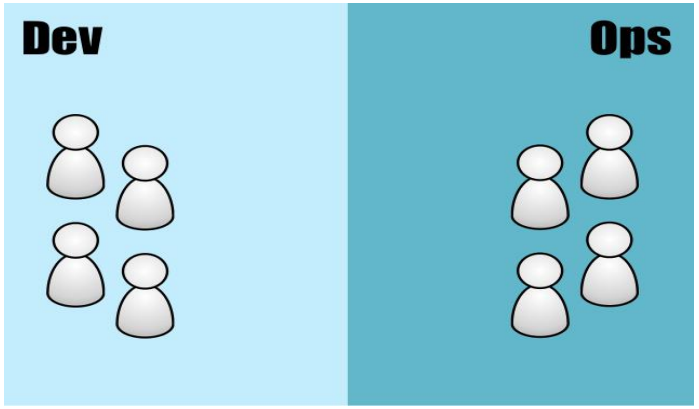
- a feature is only done when it is delivering value to users.
- For some agile delivery teams, “done” means released into production. This is the ideal situation for a software development project.
- There is no “80% done.” Things are either done, or they are not.

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.
- If you are working in a small organization or in a relatively independent department, you may have complete control over the resources that you need to release software. If so, fantastic. If not, realizing this principle may require hard work over a long period of time to break down the barriers between the silos that isolate people in different roles.

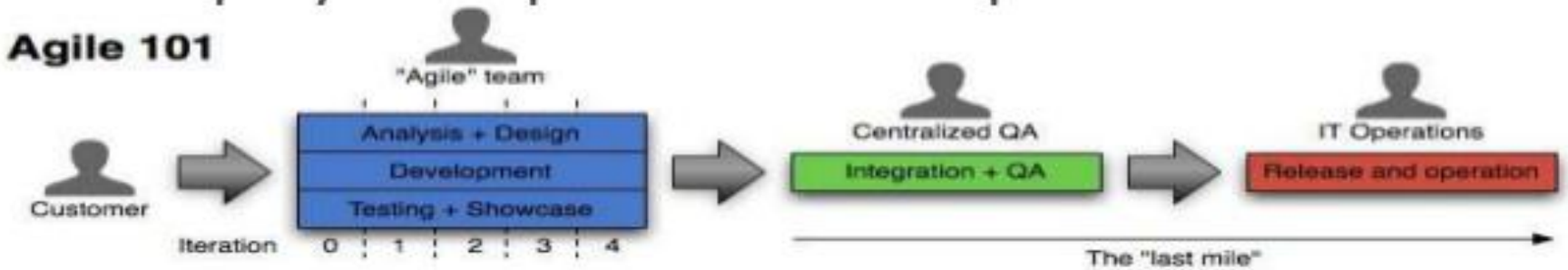
- This is one of the central principles of the DevOps movement. The DevOps movement – encouraging greater collaboration between everyone involved in software delivery in order to release valuable software faster and more reliably.





Continuous Delivery

- Taking each CI build and run it through deployment procedures on production or



Acceptance test driven development process
Tight collaboration between business and delivery teams
Cross-functional teams include QA and operations
Automated build, testing, db migration and deployment
Incremental development on mainline with continuous integration
Software always production ready
Releases tied to business needs, not operational constraints

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.