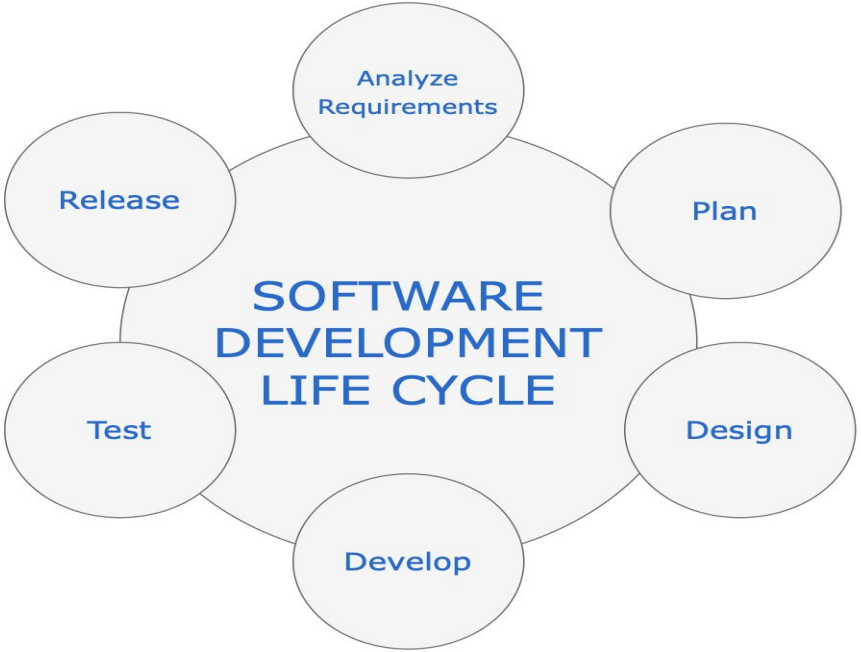
CI/CD

Continuous Integration / Continuous Delivery

Adopted from LinuxFoundationX LFS167x

For **TIC 4302 Information Security Practicum II Course**

SDLC - Software Development Lifecycle



Software development follows a flow, starting with identifying new features, planning, doing the actual development, committing the source code changes, running builds and tests (unit, integration, functional, acceptance, etc.), and deploying to production.

SDLC - Software Development Lifecycle

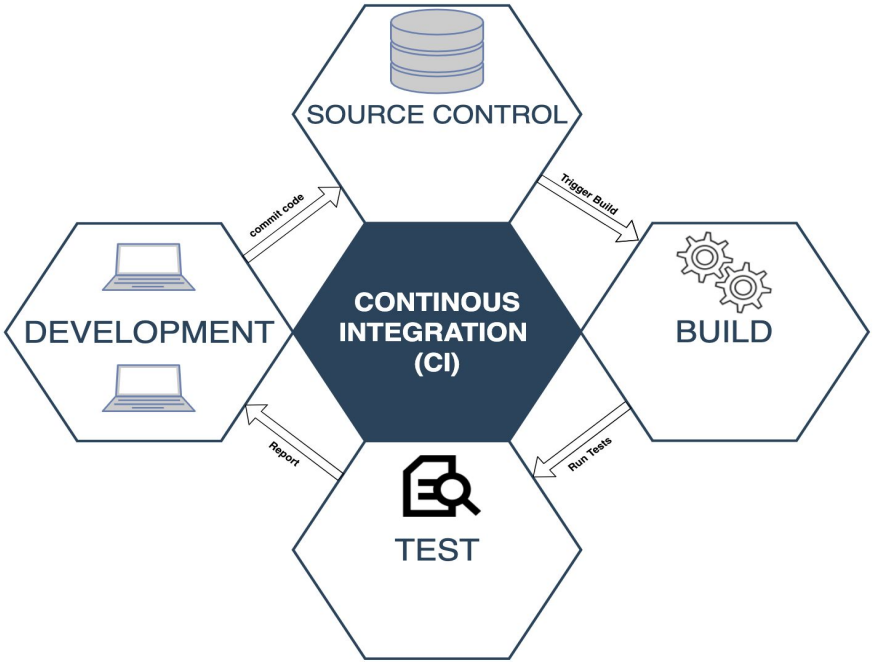
With a traditional **waterfall software delivery approach**, developers work independently for long time, but they have no clue about issues during the integration phase.

Agile software development is introduced to change the way software development by delivering incremental working software frequently rather than big bang waterfall releases

Delivering software frequently meant **producing stable code** for every incremental release but needs better approach for the integration.

Continuous Integration (CI) offered a ray of hope and started to gain in popularity.

Continuous Integration



According to Martin Fowler,

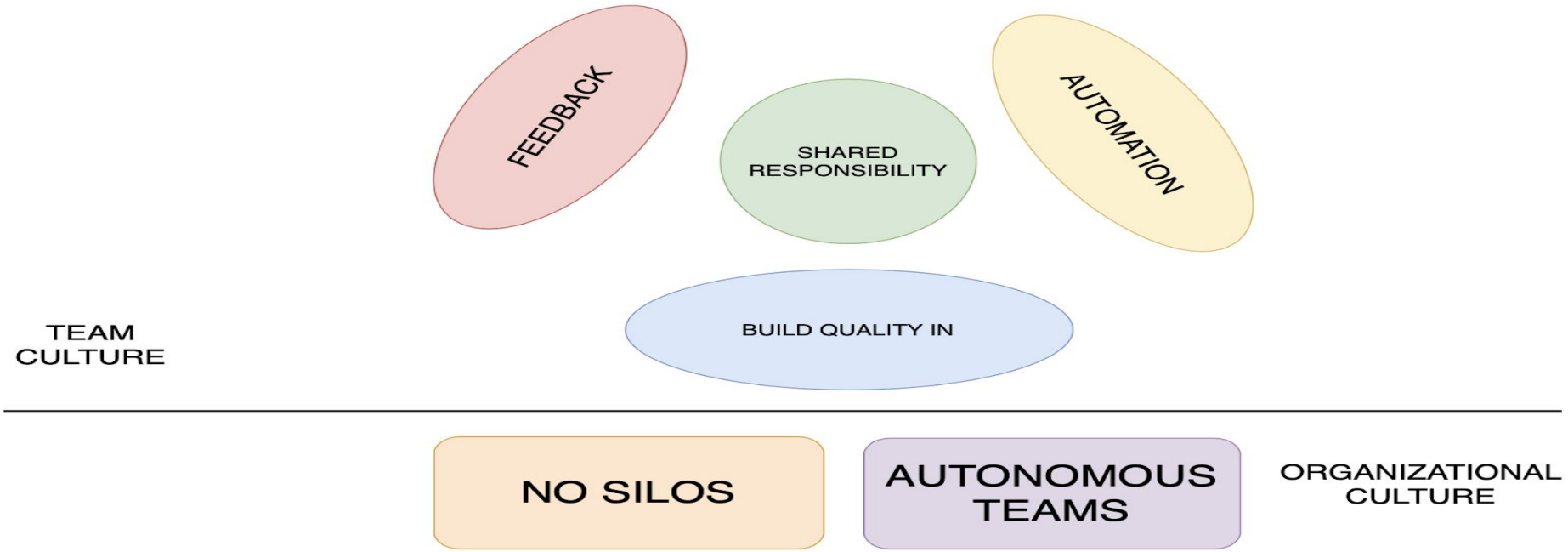
"Continuous Integration (CI) is a software development practice where members of a team integrate their work frequently; usually each person integrates at least daily - leading to multiple integrations per day. Each integration is verified by an automated build (including test) to detect integration errors as quickly as possible".

Continuous Integration

In order to implement Continuous Integration, you will need:

- **A version control system** to store all the source code checked in by the teams, and acts as the single source of truth.
- **Automated build and unit test** to ensure every commit to the version control system can be built and tested by an independent continuous integration server.
- **Feedback** for any developer's change that breaks the build so they can take the necessary action (example: email notifications).
- **Agreement on ways of working** to ensure everyone on the team follows the practice of checking-in incremental changes rather than waiting till they are fully developed.

DevOps


FEEDBACK
SHARED RESPONSIBILITY
AUTOMATION
BUILD QUALITY IN
TEAM CULTURE
NO SILOS
AUTONOMOUS TEAMS
ORGANIZATIONAL CULTURE

DevOps

- every single team a**re equally responsible** for the release of the new software, which means teams need to work in close collaboration
- The DevOps is originated from Agile software development that strongly emphasizes **collaboration amongst teams** in software delivery process.
- **Automation** of each of the stages and **constant feedback** cycles are also considered extremely important.
- **Continuous Delivery** provides a framework to achieve the goals of DevOps through automation, and continuous feedback loops.

Continuous Delivery

Continuous Delivery is a logical extension of Continuous Integration. While Continuous Integration lets you automate the software build and test process, Continuous Delivery automates the full application delivery process by taking any change in code (new features, bug fixes, etc.) all the way from development (code commit) to deployment (to environments such as staging and production). It ensures that you are able to release new changes to your customers quickly in a reliable and repeatable manner.

According to Martin Fowler, who coined the term Continuous Delivery,

"Continuous Delivery is a software development discipline where you build software in such a way that the software can be released to production at any time."

Continuous Delivery

You’re doing continuous delivery when:

- Your software is deployable throughout its lifecycle.
- Your team prioritizes keeping the software deployable over new features.
- Anybody can get fast, automated feedback any time somebody makes a change.
- You perform push-button deployment of any version to any environment on demand.

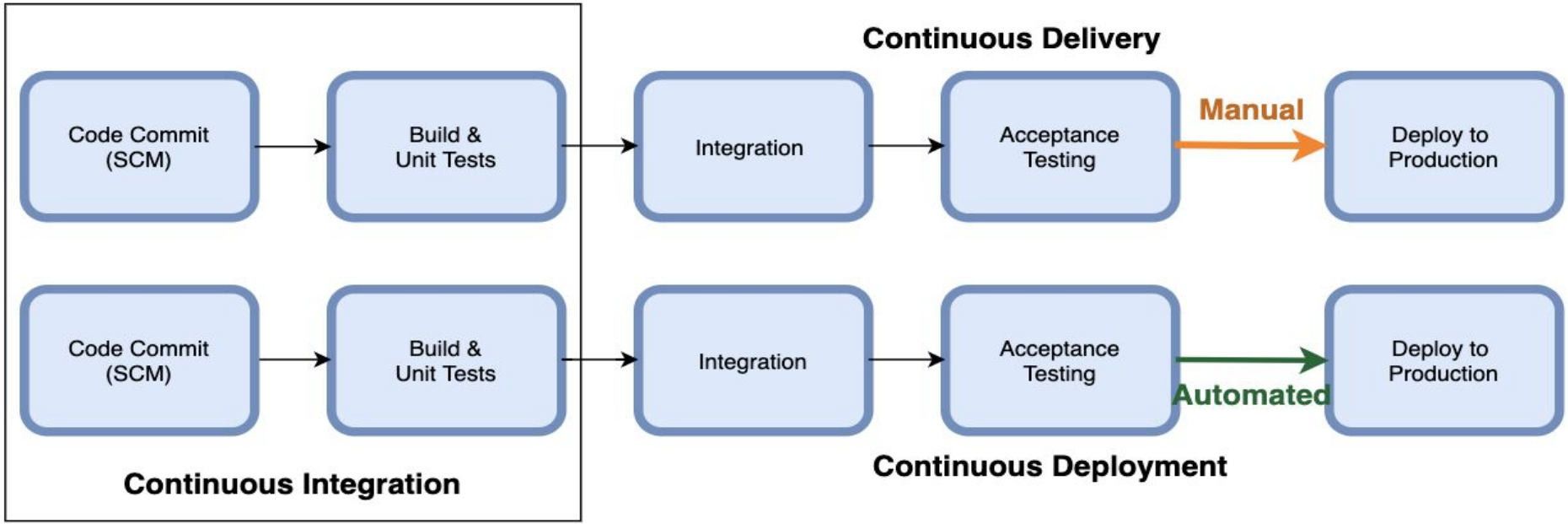
To achieve continuous delivery you need:

- A close, collaborative working between everyone involved (i.e., DevOps culture)
- Extensive automation of all possible parts using a deployment pipeline.

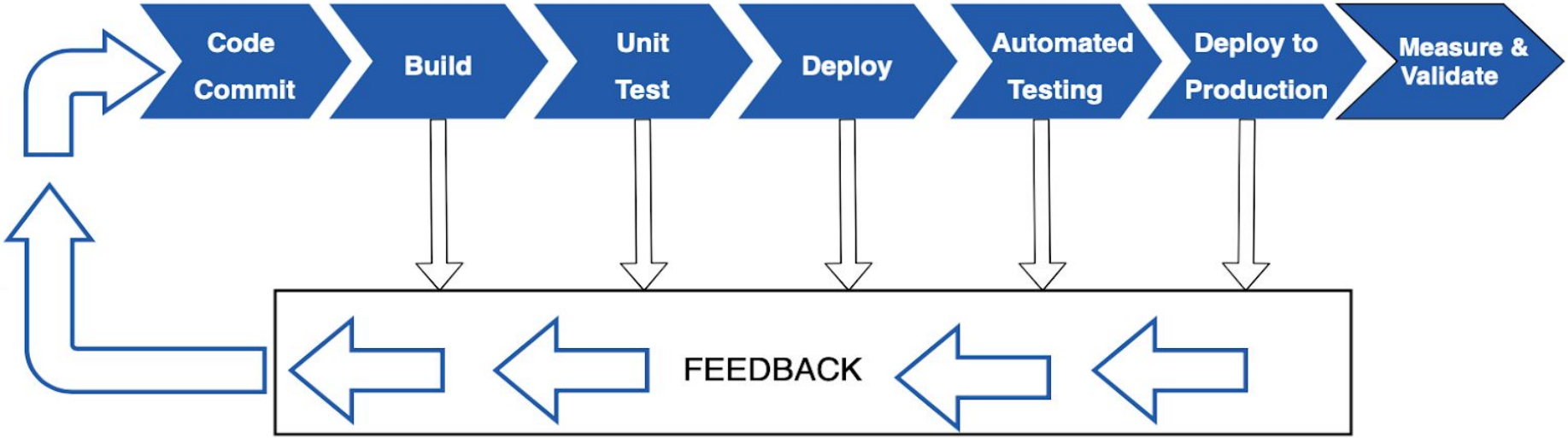
Continuous Deployment

- Continuous Delivery and Continuous Deployment are **often used synonymously** but in reality these concepts have a different meaning.
- Continuous Delivery gives you the **capability to deploy to production frequently** but you are **not automating the deployment** because of manual approval or business consideration.
- Continuous Deployment **automate way of deploying** your releases to production but it **needs continuous delivery** in order to be able to perform it.
- It is use for companies like Netflix, Amazon, Google, and Facebook automatically deploy to production **multiple times a day**.

Continuous Deployment



Deployment Pipelines



Deployment Pipelines

- Deployment pipelines (or Continuous Delivery pipelines) are the cornerstone of Continuous Delivery as they automate all the stages (build, test, release, etc.) of software delivery process.
- Some of the benefits are an automated pipeline allows all stakeholders to monitor the progress, eliminates the overhead of all the manual work, provides quick feedback, and more importantly builds confidence on the code quality.
- The deployment pipeline run starts with a developer committing source code change into a version control repository. The CI server detects the new commit, compiles the code, and runs unit tests. The next stage is deploying the artifacts (files generated from a build) to staging or a feature test environment where you run additional functional, regression, and acceptance tests. Once all the tests are successful, you are ready to deploy into production. In case of failure during any stage, the workflow stops and an immediate feedback is sent back to the developer.

Tools for Deployment Pipeline

In order to automate the various stages of your deployment pipeline, you will need multiple tools. For example:

- a version control system such as Git to store your source code
- a Continuous Integration (CI) tool such as Jenkins to run automated builds
- test frameworks such as xUnit, Selenium, etc., to run various test suites
- a binary repository such as Artifactory to store build artifacts
- configuration management tools such as Ansible
- a single dashboard to make the progress visible to everyone
- frequent feedback in the form of emails, or Slack notifications.

And that's not all. You will also need a single tool that can bring all these tools together to achieve CI/CD goals which is to automate software delivery.

CI/CD Tools	Feature: Pricing	Feature: Licensing	Feature: Hosting	Feature: Extensions/Plugins
Atlassian Bamboo	Paid	Closed	On-premises	**
Bitbucket Pipelines	Paid	Closed	Cloud/On-premises	**
AWS CodePipeline	Paid	Closed	Cloud	*
Azure Pipelines	Paid	Closed	Cloud	**
Circle CI	Paid	Closed	On-premises/Cloud	***
CloudBees Core	Paid	Closed	On-premises/Cloud	*****
GitHub Actions	Paid	Closed	On-premises/Cloud	***
GitLab CI/CD	Free/Paid	OSS/Closed	On-premises/Cloud	*
Google Cloud Build	Paid	Closed	Cloud	**
Jenkins	Free	OSS	On-premises/Cloud	*****
Travis CI	Paid	Closed	On-premises/Cloud	***

Thank You!