

COMP1531

 Software Engineering
8.2 - Deployment - Intro

In this lecture

Why?

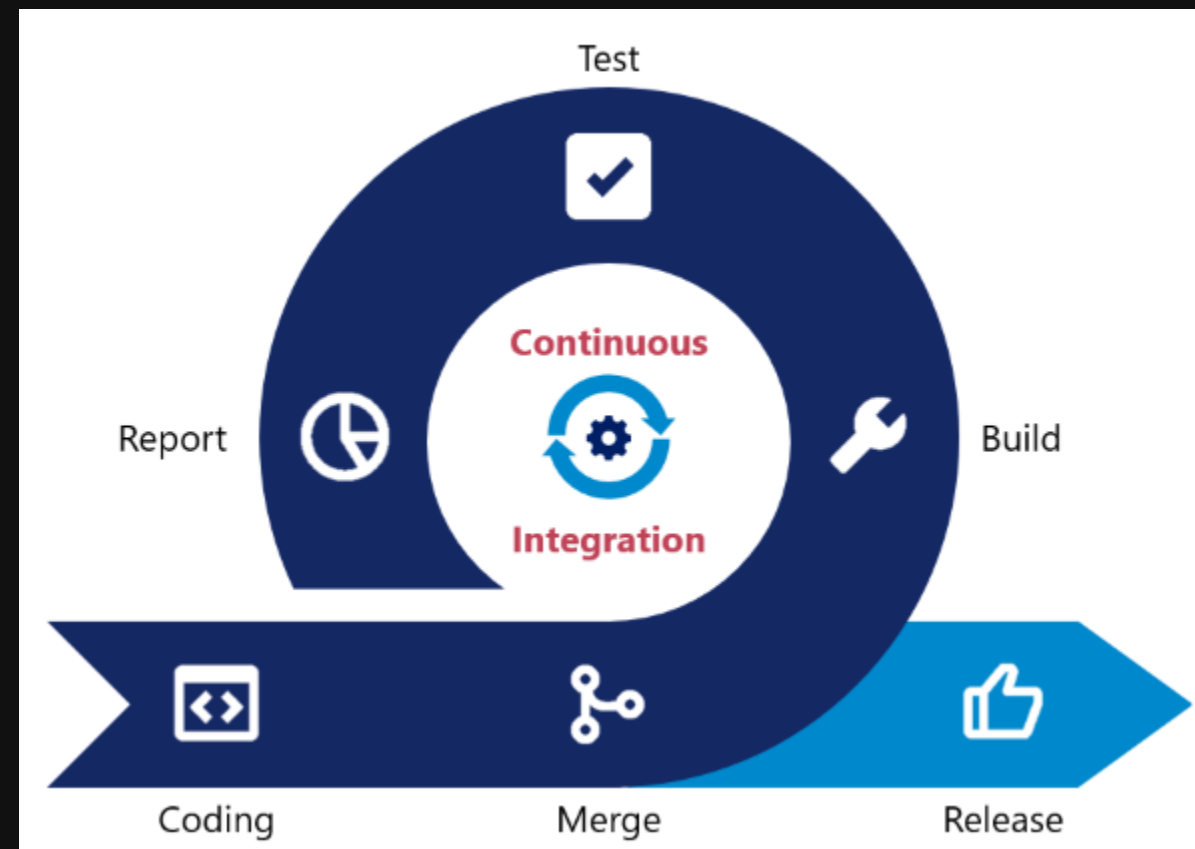
- The purpose of most software is for people to use it, and for that to happen we need processes to make it available for usage

What?

- Deployment history
- Continuous Delivery
- Continuous Deployment
- DevOps

Continuous Integration

Continuous integration: Practice of automating the integration of code changes from multiple contributors into a single software project.



Software Deployment

Deployment: Activities relating to making a **software system available for use.**



Diagrams sourced from atlassian, gitlab, microsoft

Simple example: CSE

Every CSE student has a **public_html** folder that is exposed to the internet.

Historical Deployment

Historically, **deployment** was a much less frequently occurring process.

Code would be worked on for days at a time without being tested, and deployed sometimes years at a time. This is largely due to software historically being a physical asset

Something changed

Two major changes have occurred over the last 10 years:

- Increased prevalence of web-based apps (no installs)
- Improvement to internet connectivity, speed, bandwidth

These changes (and more) have allowed for the pushing of updated software to **users** to be substantially more possible. Subsequently, users have come to expect more rapid updates.

**A movement from software as an asset, to software as a service,
has catalysed this transition**

Software as a service (Sass)



Service vs Asset

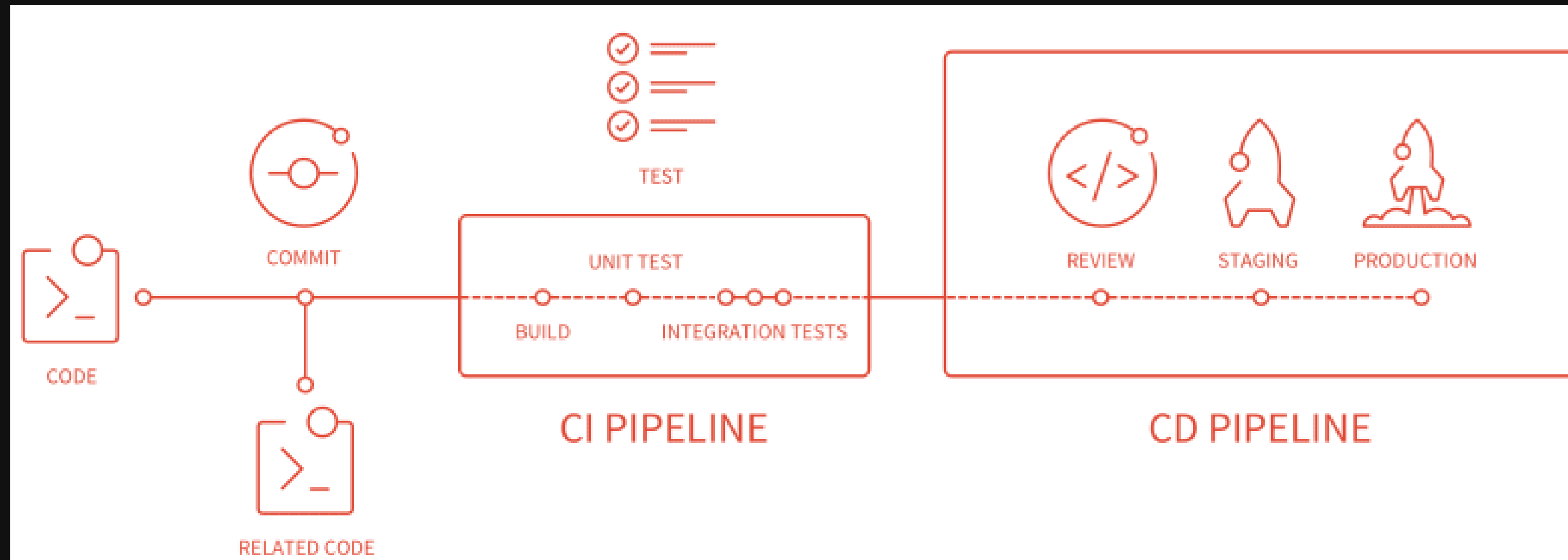
A simple case study can be found in [Microsoft's movement of Windows from shipping a product, to shipping a service.](#)

Cloud services

- Numerous cloud services offer the ability to "easily" deploy your web applications
 - Amazon Web Services
 - Google App Engine
 - Heroku

Modern Deployment

To achieve rapid deployment cycles, modern deployment isn't as simple as pushing code. Rather, a heavily **integrated** and **automated** approach is preferred.



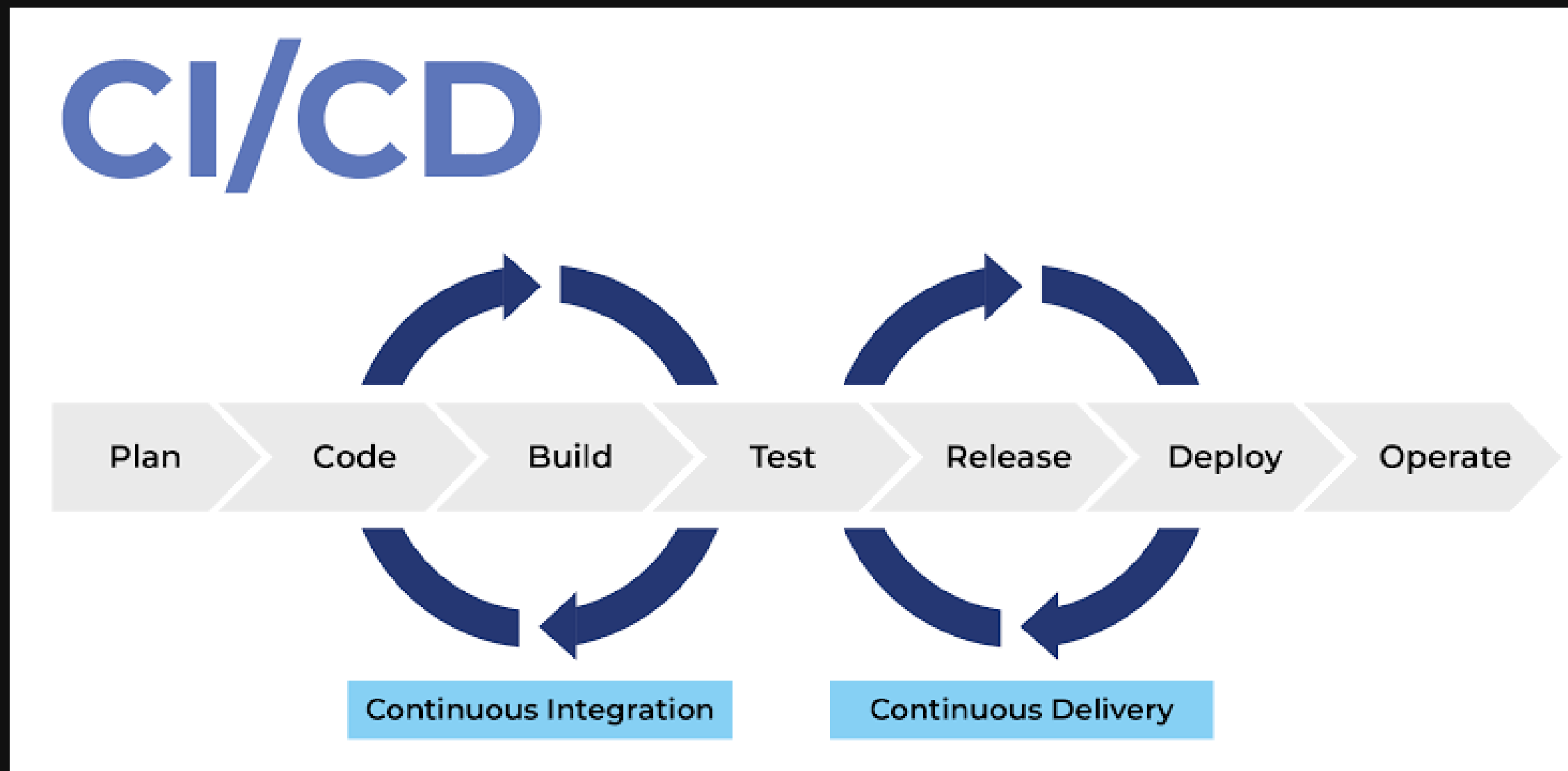
Continuous Delivery

Continuous delivery: Allows accepted code changes to be deployed to customers quickly and sustainably. This involves the **automation of the release process such that releases can be done in a "button push"**.

Continuous Delivery

- Many companies will have a daily or weekly "ship"
- Often there is some "sign off" process before things are finally shipped
- Since the process is highly controlled, less likely to make mistakes during testing

CI/CD relationship



CD: Readings

- <https://www.atlassian.com/continuous-delivery/principles>
- <https://about.gitlab.com/product/continuous-integration/>

Release Methods

Continuous delivery is often concerned with more than just going from "your computer" to a "production environment".

Often we have various stages of release
e.g. dev => test => prod.

As you move down the stages, things tend to be more stable.

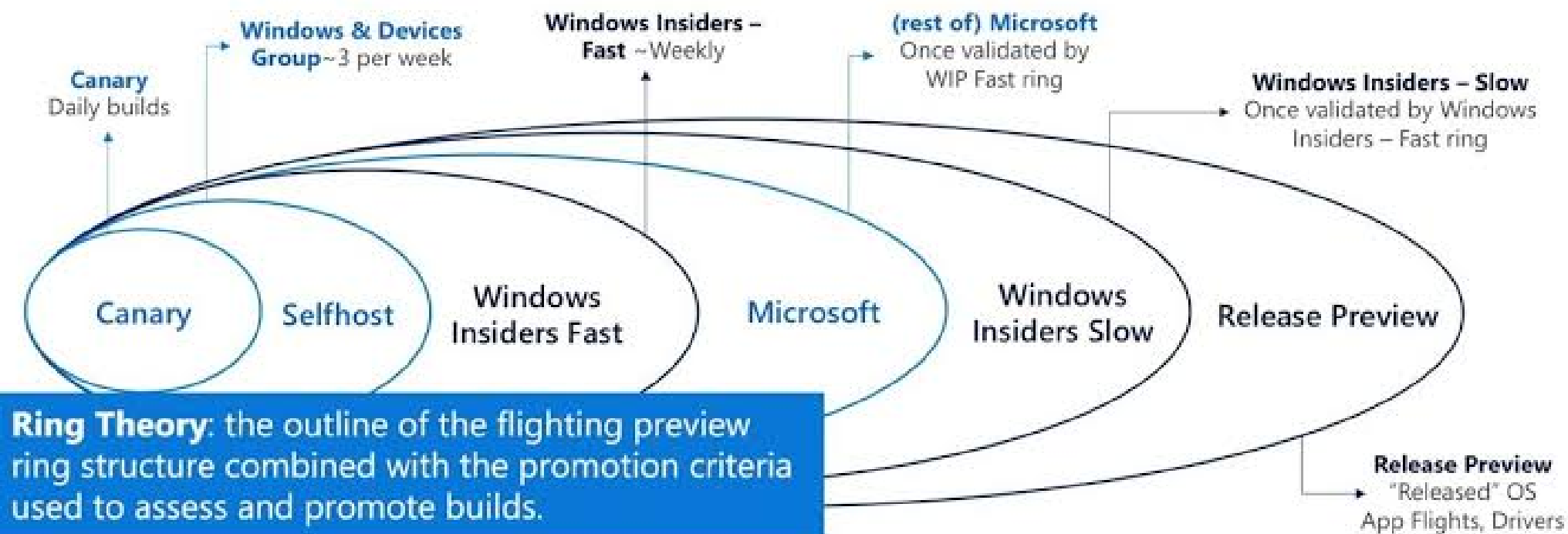
Different deployments

It is common to have 3 core tiers:

- **dev:**
 - released often, available to developers to see their changes in deployment
- **test/staging:**
 - As close to release as possible, ideally identical to prod
- **prod:**
 - Released to customers, ideally as quickly as possible

"Flighting"

WINDOWS INSIDER PROGRAM RING THEORY

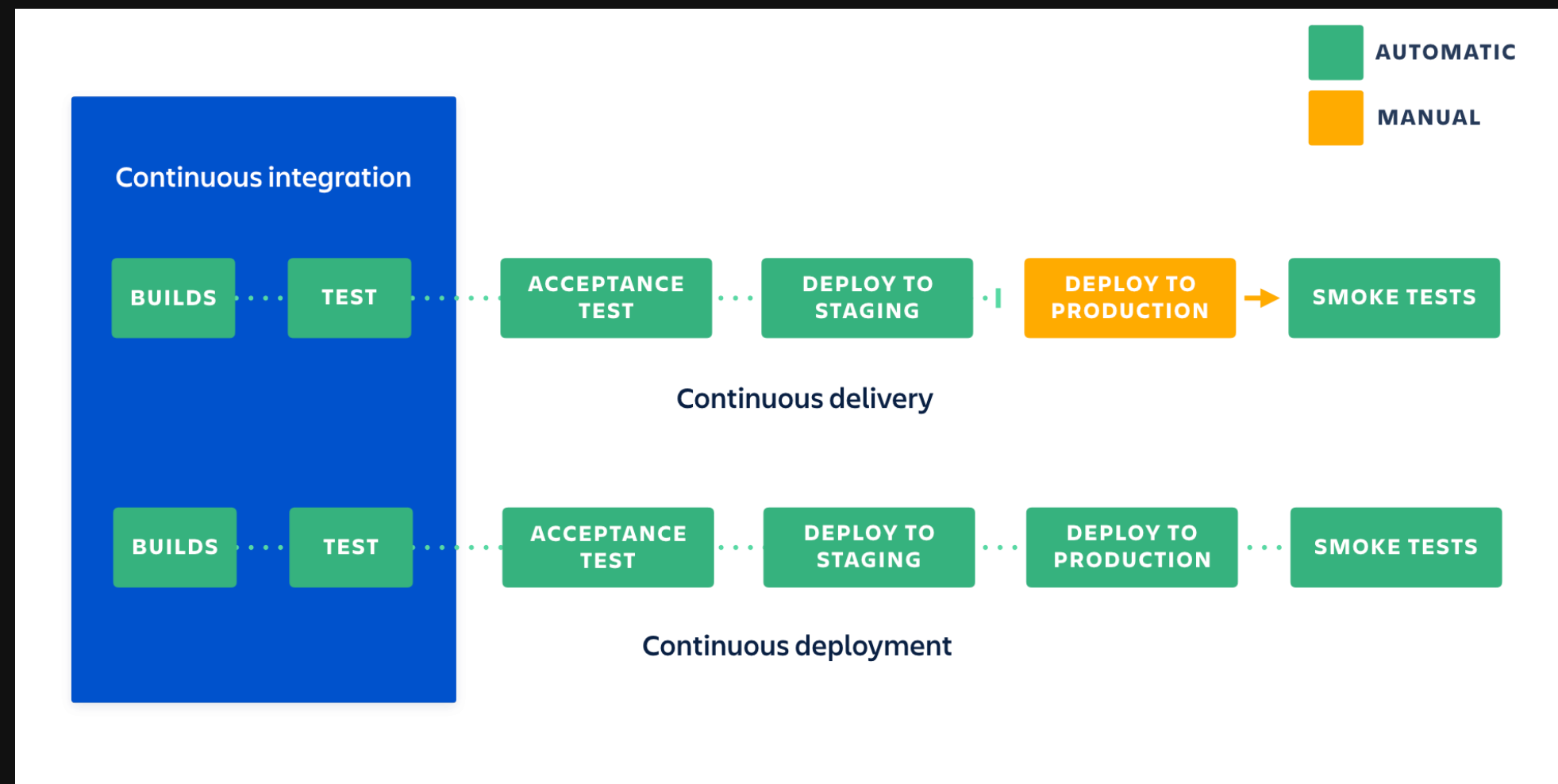


- Builds/content go to progressively larger audiences
- Organizations should setup their own rings with 1% of devices on Windows Insider Slow

■ MS Internal ■ Public

Continuous Deployment

Continuous Deployment is an extension of Continuous Delivery whereby changes attempt to flight toward production automatically, and the only thing stopping them is a failed test



CD: Further Reading

- <https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment>

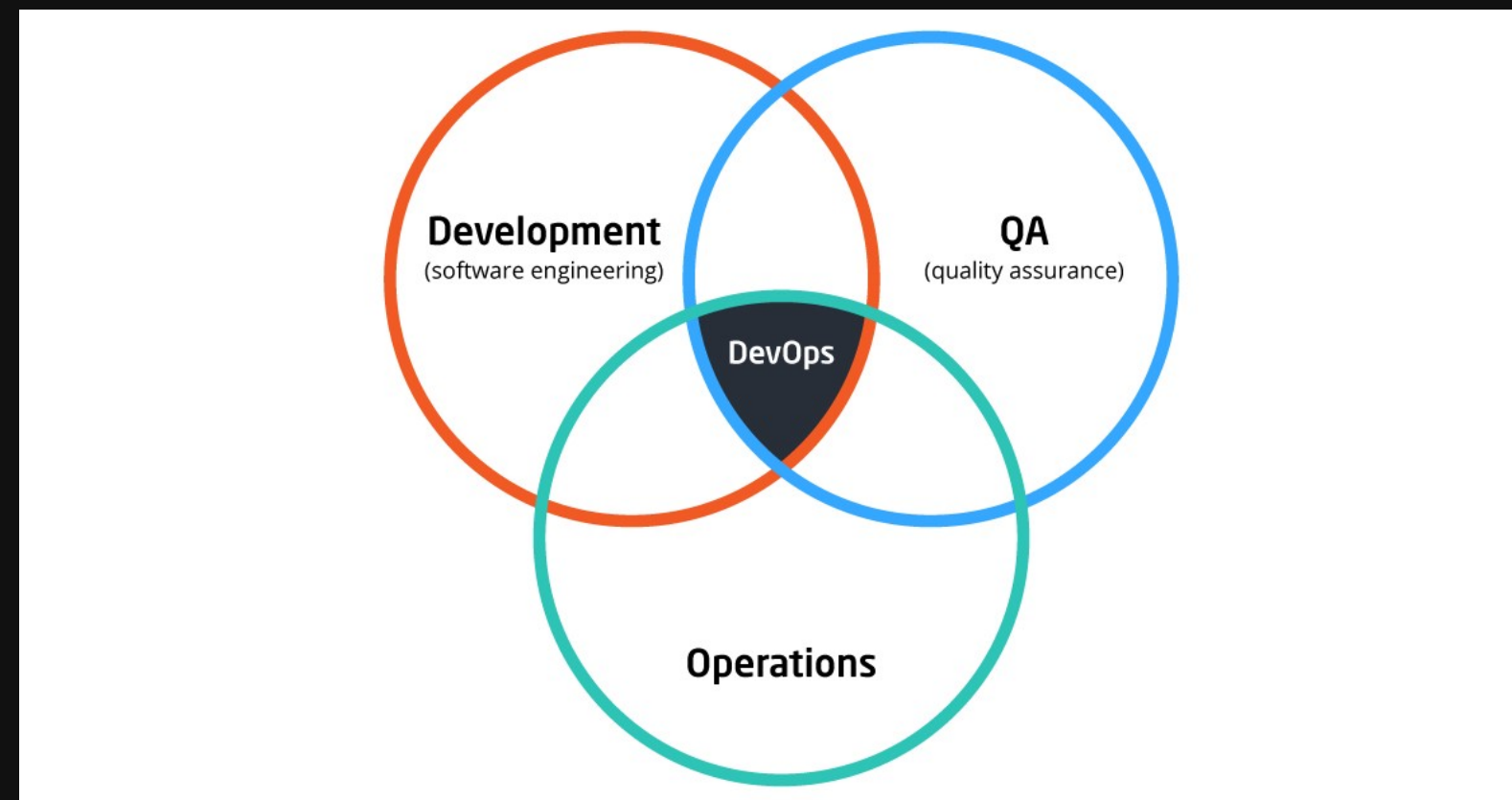
Deploying on your own: AlwaysData

For 21T3 COMP1531 has decided to use a free service known as "alwaysdata" to let students deploy their **backend** to the cloud.

Instructions of how to set this up are found in the project repository for iteration 3. We have pre-recorded a brief demo in lectures.

DevOps

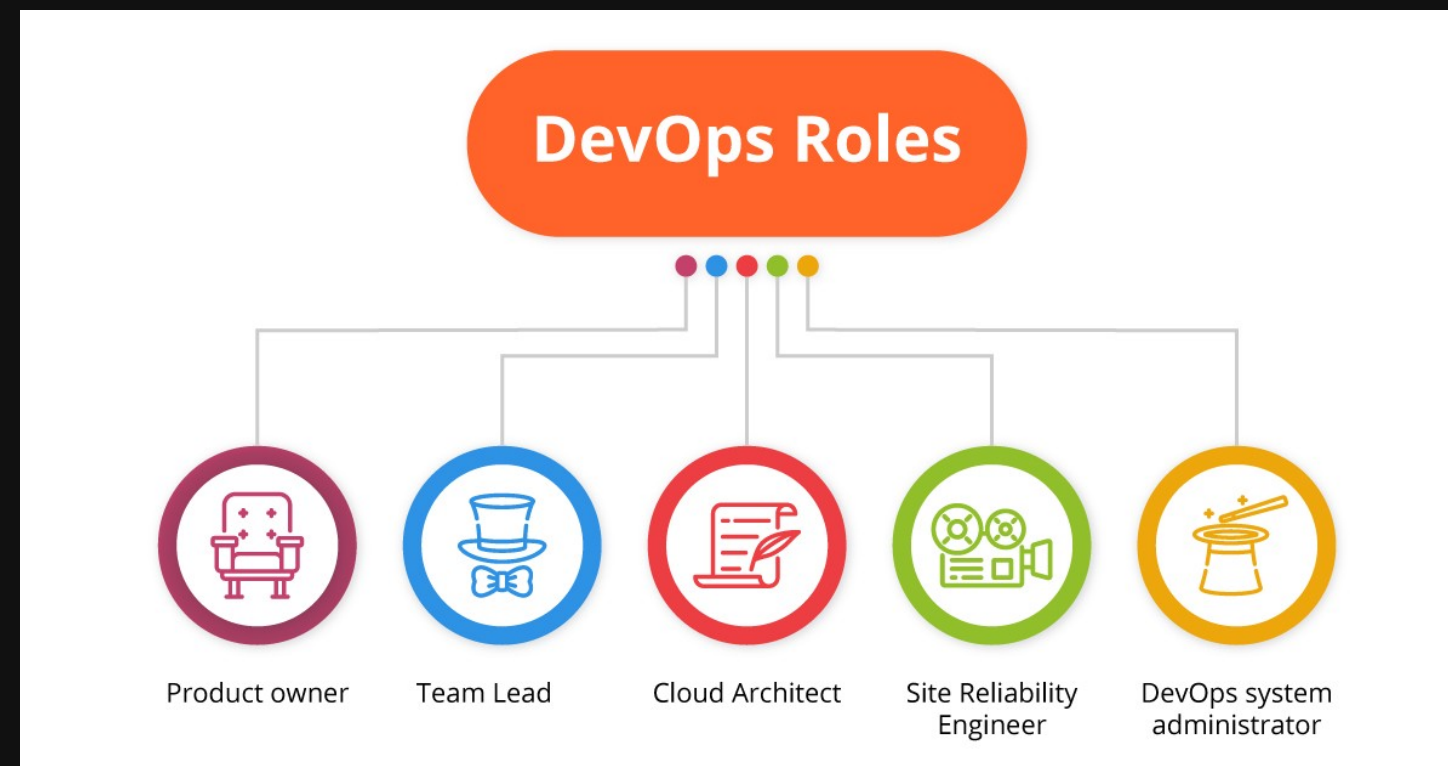
A decade ago, the notion of dev ops was quite simple. It was a role dedicated to gluing in the 3 key pillars of deploying quality assured software



DevOps is a set of practices intended to reduce the time between committing a change to a system and the change being placed into normal production, while ensuring high quality [Wikipedia. Yes, Wikipedia]

DevOps

As development teams become less silo'ed, modern DevOps is less a role, and more a series of roles or aspect of a role.



Source & Reading: <https://hackernoon.com/devops-team-roles-and-responsibilities-6571cfb56843>

Maintenance & Monitoring

Maintenance: After deployment, the use of analytics and monitoring tools to ensure that as the platform is used and remains in a healthy state.

Monitoring often has two purposes:

- Preserving user experience: Monitoring errors, warnings, and other issues that affect performance or uptime.
- Enhancing user experience: Using analytical tools to monitor users or understanding their interactions. Often leads to customer interviews and user stories

Maintenance

Maintenance: After deployment, the use of analytics and monitoring tools to ensure that as the platform is used and remains in a healthy state.

Health is defined by developers, but often consists of:

- Monitoring 4XX and 5XX errors
- Ensuring disk, memory, cpu, and network is not overloaded

Often these aren't actively monitored, but rather monitored with alerts and triggers

Feedback

