Fundamentals of CI/CD

Continuous Integration

The practice of merging all developers' working copies to a shared mainline several times a day. It's the process of "Making". Everything related to the code fits here, and it all culminates in the ultimate goal of CI: a high quality, deployable artifact! Some common CI-related phases might include:

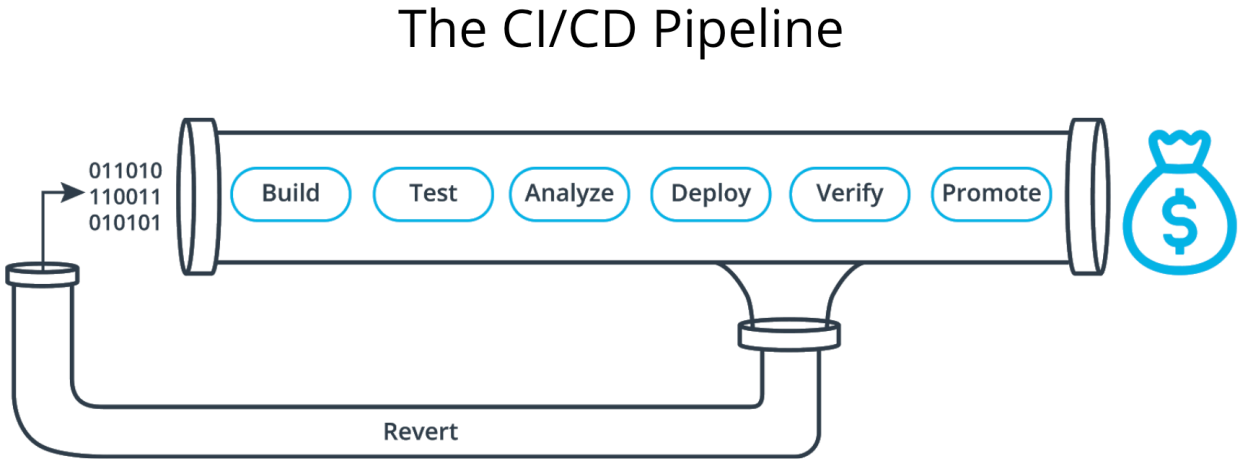
- Compile
- Unit Test
- Static Analysis
- Dependency vulnerability testing
- Store artifact

Continuous Deployment

A software engineering approach in which the value is delivered frequently through automated deployments. Everything related to deploying the artifact fits here. It's the process of "Moving" the artifact from the shelf to the spotlight. Some common CD-related phases might include:

- Creating infrastructure
- Provisioning servers
- Copying files
- Promoting to production
- Smoke testing (aka Verify)
- Rollbacks

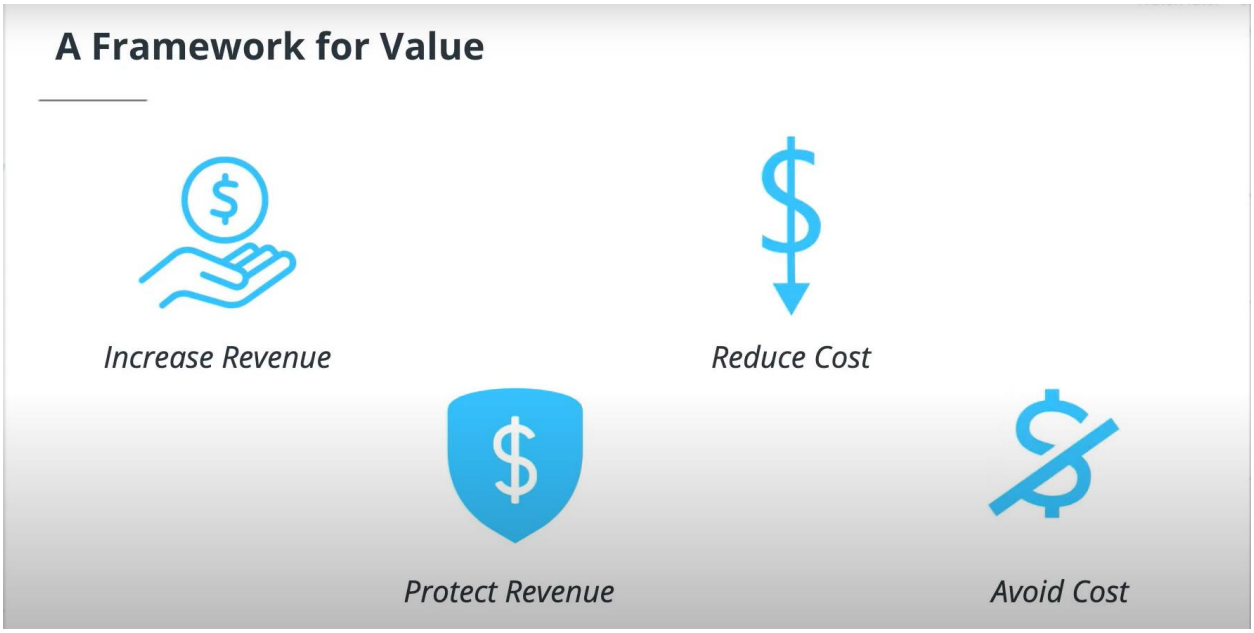
The Phases of CI/CD Pipeline

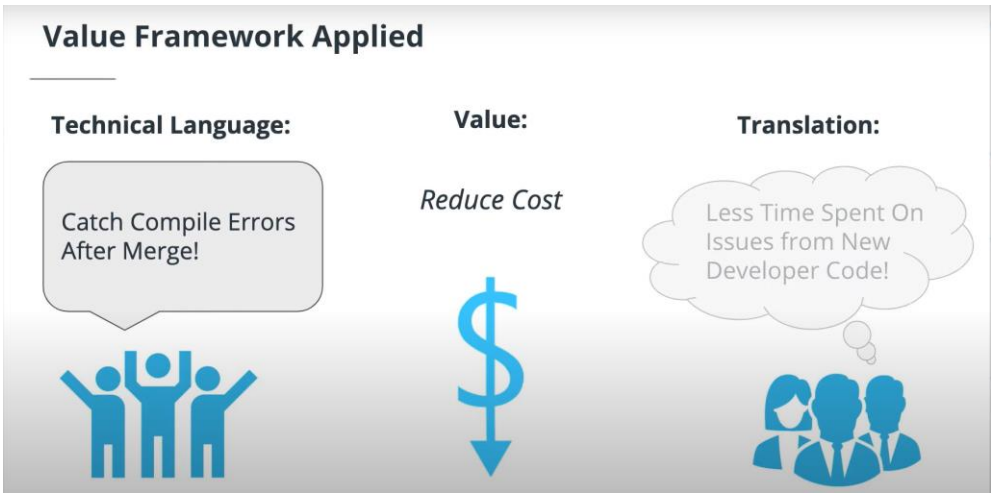


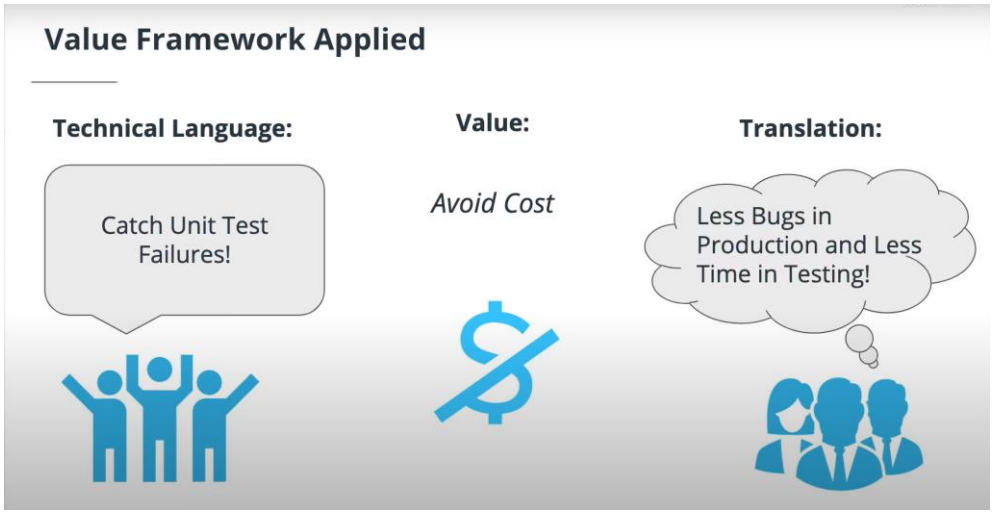
Key Terms

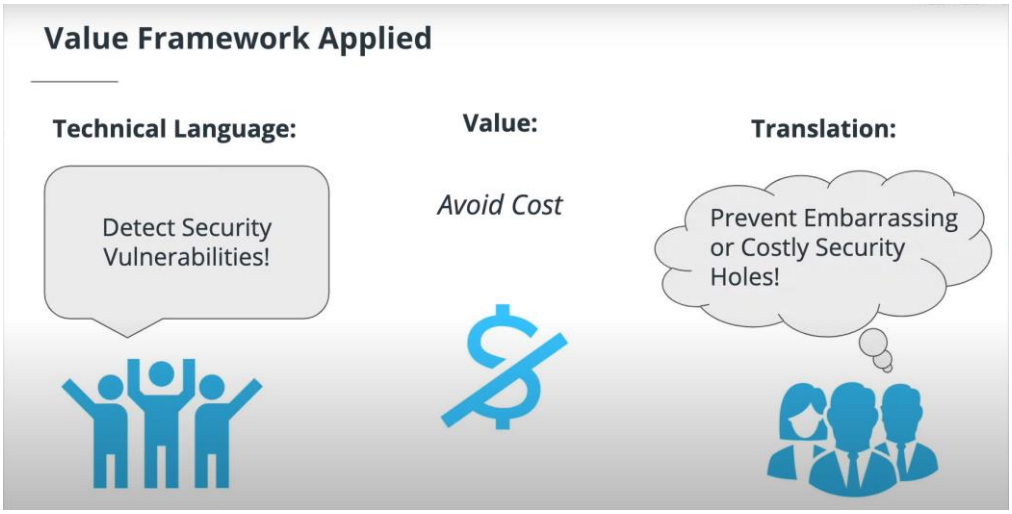
- **Pipeline**: A set of data processing elements connected in series, where the output of one element is the input of the next one.
- **Continuous Integration**: The practice of merging all developers' working copies to a shared mainline several times a day.
- **Continuous Delivery**: An engineering practice in which teams produce and release value in short cycles.
- **Continuous Deployment**: A software engineering approach in which the value is delivered frequently through automated deployments.
- **Infrastructure as Code**: The management of infrastructure using code.
- **Provisioning**: The process of setting up IT infrastructure.
- **Artifact**: A product of some process applied to the code repository.
- **DevOps**: A set of practices that works to automate and integrate the processes between software development and IT teams.
- **Testing**: A practice that seeks to ensure the quality of the software.

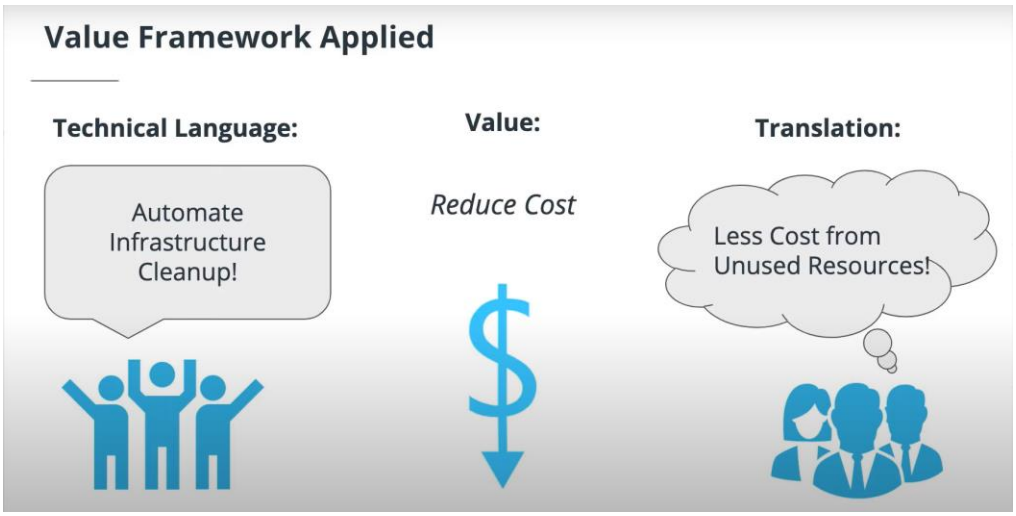
Benefits of CI/CD

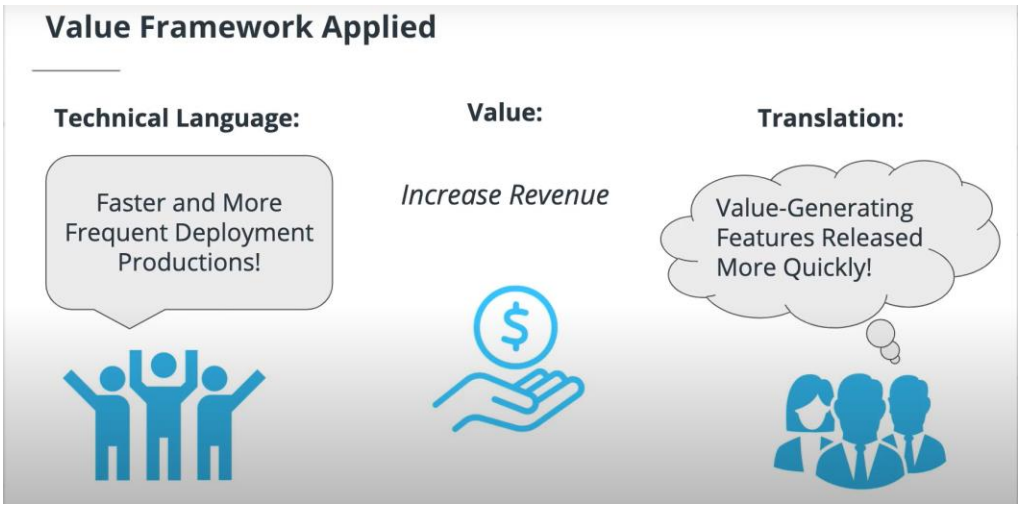



Value Framework Applied
Technical Language:
Catch Compile Errors After Merge!
Value:
Reduce Cost
Translation:
Less Time Spent On Issues from New Developer Code!


Value Framework Applied
Technical Language:
Catch Unit Test Failures!
Value:
Avoid Cost
Translation:
Less Bugs in Production and Less Time in Testing!


Value Framework Applied
Technical Language:
Detect Security Vulnerabilities!
Value:
Avoid Cost
Translation:
Prevent Embarrassing or Costly Security Holes!





Why Is Translation Important?

If the decision-makers in your company can't understand why something is important or adds value, they might not approve it. This usually isn't because they're ignorant or trying to be difficult, this is often because they don't understand how the technical changes translate into benefits for the company. Using language they understand will make everyone's lives better and easier!

Technical Language	Value	Translation
Catch Compile Errors After Merge	Reduce Cost	Less developer time on issues from new developer code

Technical Language	Value	Translation
Catch Unit Test Failures	Avoid Cost	Less bugs in production and less time in testing
Detect Security Vulnerabilities	Avoid Cost	Prevent embarrassing or costly security holes
Automate Infrastructure Creation	Avoid Cost	Less human error, Faster deployments
Automate Infrastructure Cleanup	Reduce Cost	Less infrastructure costs from unused resources
Faster and More Frequent Production Deployments	Increase Revenue	New value-generating features released more quickly
Deploy to Production Without Manual Checks	Increase Revenue	Less time to market
Automated Smoke Tests	Protect Revenue	Reduced downtime from a deploy-related crash or major bug
Automated Rollback Triggered by Job Failure	Protect Revenue	Quick undo to return production to working state