CPSC 455

Development and Release Processes in Mature Organizations

By
Dorothy Ordogh

Development and release processes in *mature* organizations

Intro

How do mature organizations structure, manage, build, deploy, and host code?

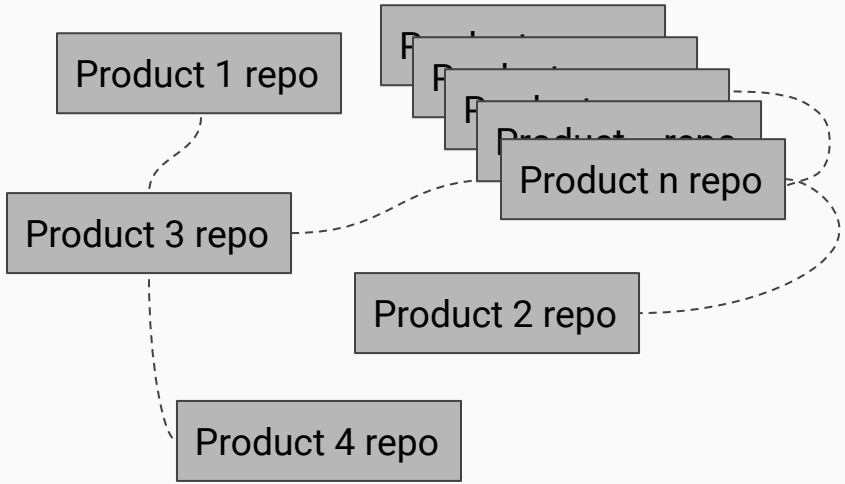
&&

Why is it important?

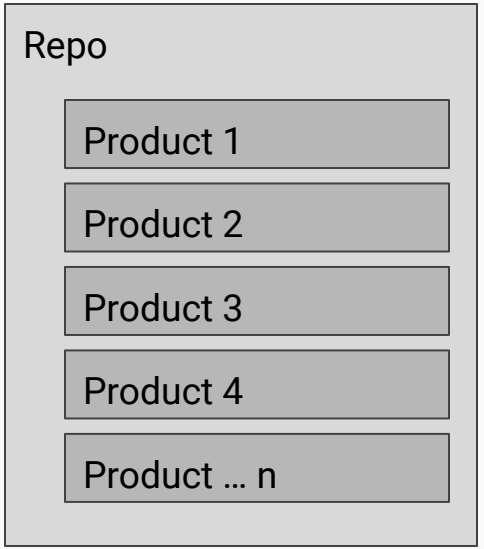
Structure

Code-base Structure

Many repos

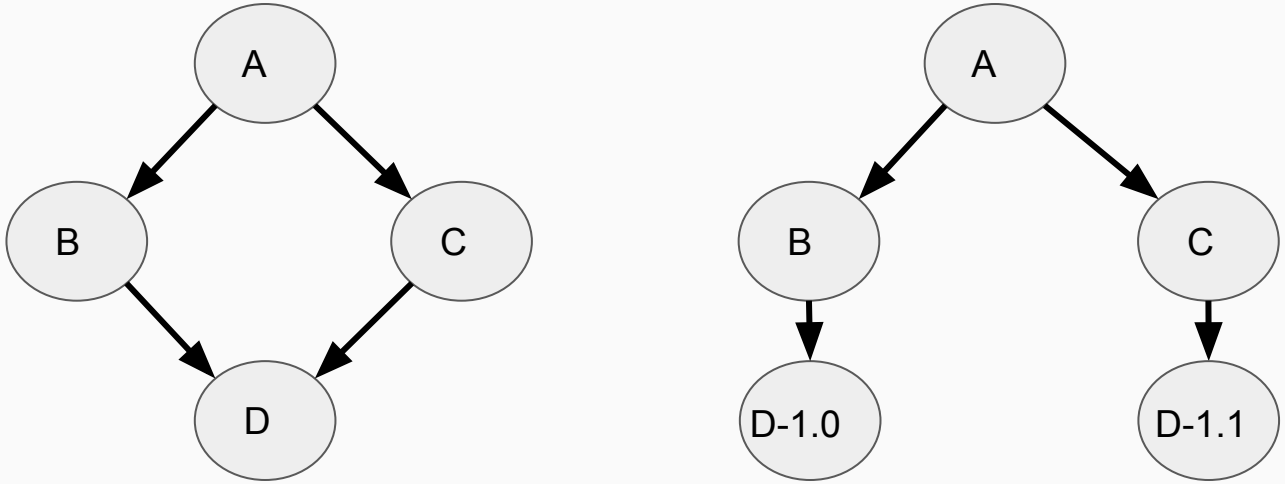


Monorepo



Code-base structure

Diamond dependencies



Code-base structure

Many repos

Pros:

+ smaller repos
+ easier to maintain
+ tools often work better with smaller repos

Cons:

- limited access to other repos
- dependency hell
- coordination overhead

Monorepo

Pros:

+ source dependencies
+ access to everything
+ onus on the person making the change not to break everyone

Cons:

- hard to maintain
- need specialized tools because of the size of the repo, often slow

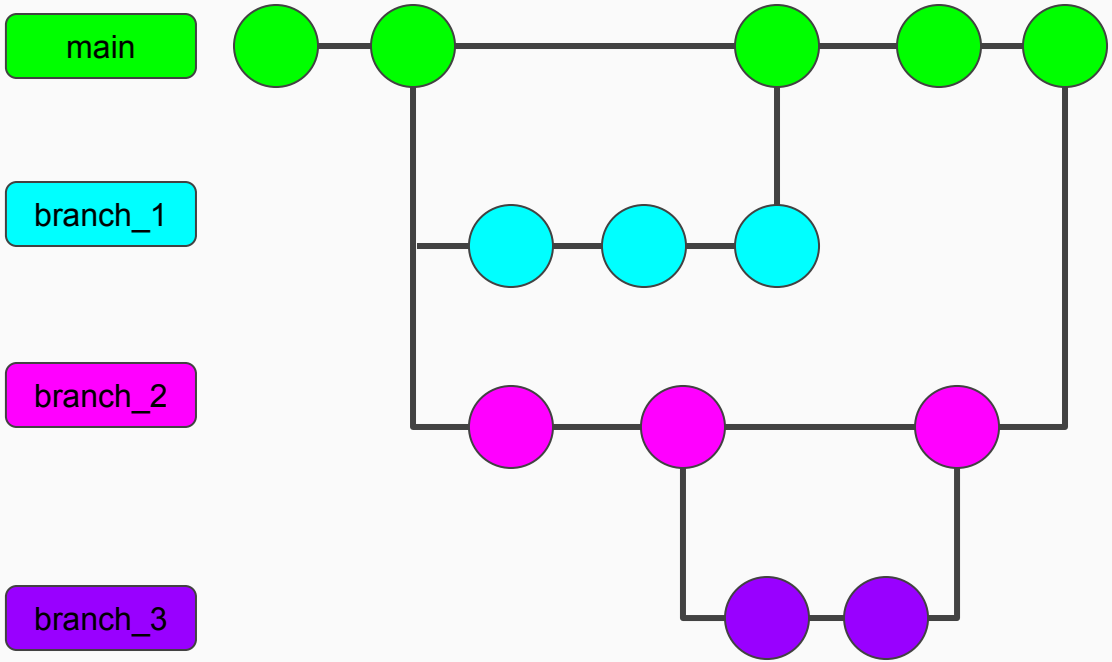
Manage

Version control systems

Manage

Which system is used depends on a number of factors. What do you think these factors are?

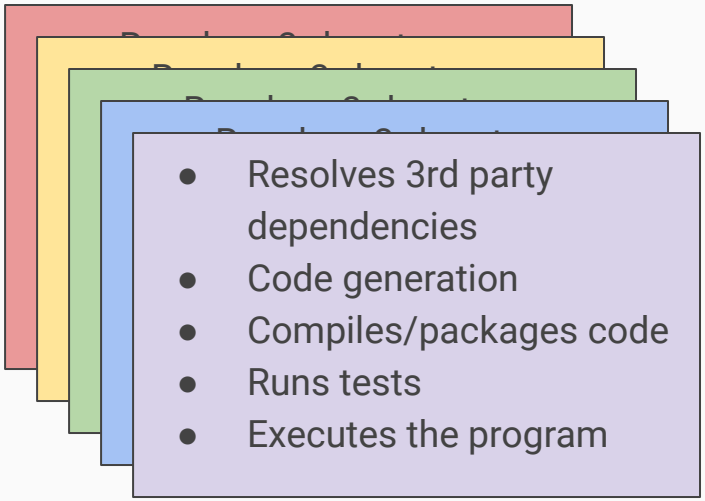
What is done before code is merged?

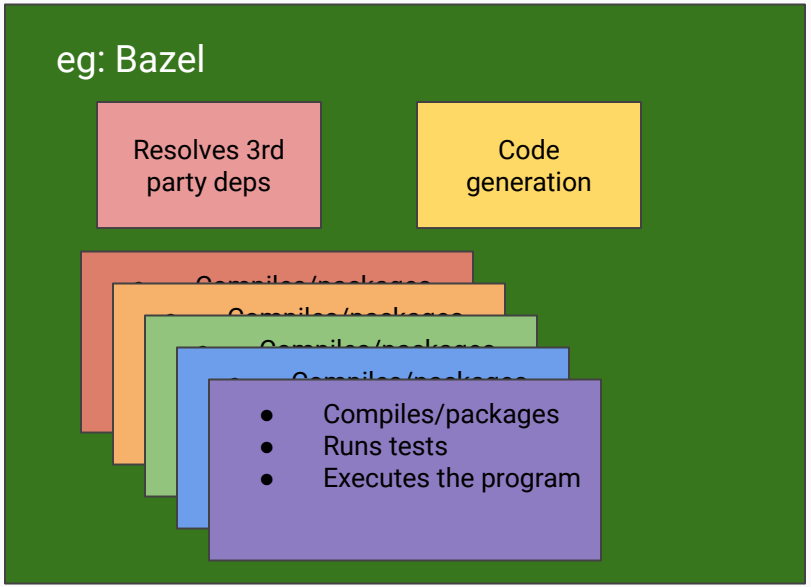


Build

Build

Language specific vs polyglot build tools





Deploy & Host

Deploy

Continuous Integration (CI)

Automatic testing of changes before integrating changes into the main branch.

- Building
- Running and passing integration and unit tests
- Optional: code coverage, quality, syntax, etc.

Goal: main branch is a “clean state”

Continuous Delivery (CD)

Automated process to deploy software with some human intervention.

- Happens after CI
- Deploys to testing and staging environments

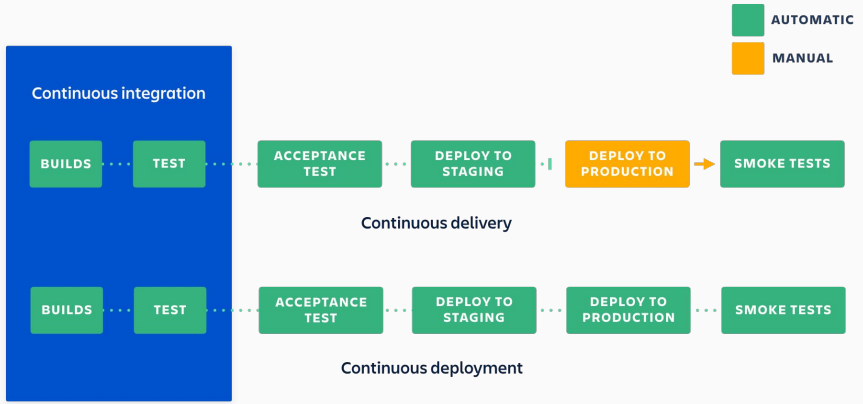
Goal: prove code is deployable

Deploy

Continuous Deployment

Totally automated deployment of changes.

- Must complete CI and CD first



Source: Atlassian

Goal: automatically deploys changes so they are consumable by users

Deploy

Examples of GitHub Actions

Small sample repo: https://github.com/dotordogh/ubc-cpsc-455/actions

Twitter Util: https://github.com/twitter/util/actions

Deploy

Shipping with zero downtime ensures your product is always available.

What are some strategies that you might have heard of?

Deploy

Shipping with zero downtime ensures your product is always available.

Blue/Green deploys

- Two sets of identical environments with the current version and the new version. Traffic is routed from the current to the new version
- Quick, and easy to rollback versions if new version has a bug, but very expensive (especially with very frequent deploys)

Rolling deploys

- One set of nodes and the new version is deployed gradually across the set
- Cost effective, more than two versions can be deployed at the same time, but the process can be lengthy and rollback can be difficult

Canary deploys

- One set of nodes and changes deployed to a small subset for testing. Many versions can be deployed at the same time

A/B deploys

- Very similar to canaries (used for testing) but only two versions are running at the same time

Host

Where software is deployed so that it is consumable.

Can you name some hosting options mature organizations would use?

Host

Cloud

- Third party (AWS, GCS, etc)
- Elasticity and auto scaling depending on traffic is usually included
- Cheaper for smaller systems because you don't need to invest in the capital of infrastructure (building, servers, wiring, electricity, etc)
- May have to share resources with other companies/services

Data center

- When cloud doesn't cut it anymore (disk sharing, not enough capacity, etc)
- Have more control over your own data centers and servers
- Need lots of capital for infrastructure
- Lots of infra needs to be built for this to be effective

Further reading

- Release engineering - Google SRE book
- Diamond dependency problem
- Visual guide to version control
- Illustrated guide to distributed VCS
- Difference between a compiled and interpreted language
- What is an executable?
- Difference between continuous integration, delivery, and deployment
- Deployment strategies with zero downtime
- Hosting