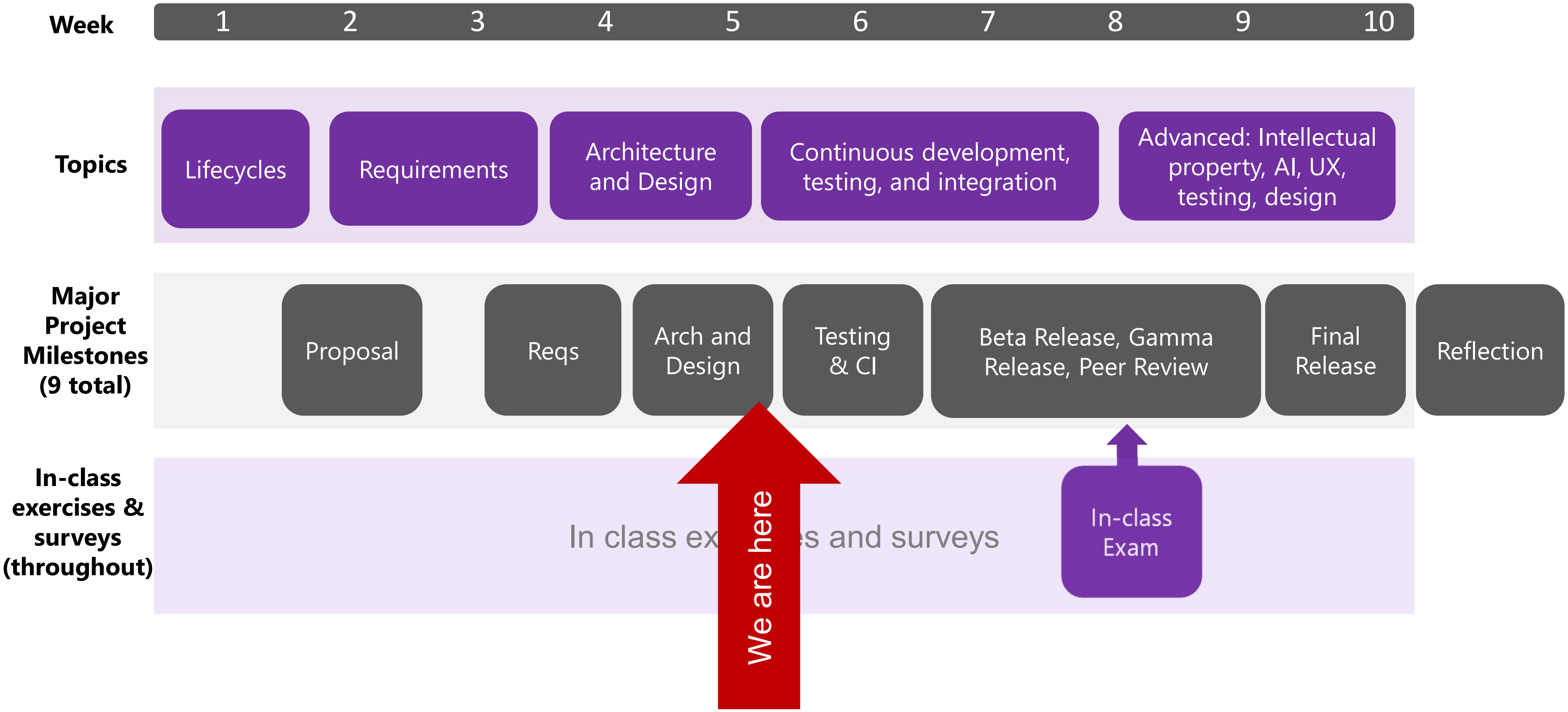
Build systems, continuous integration and delivery

CSE 403 Software Engineering

Winter 2026

Course overview: schedule



Week: 1, 2, 3, 4, 5, 6, 7, 8, 9, 10

Topics: Lifecycles | Requirements | Architecture and Design | Continuous development, testing, and integration | Advanced: Intellectual property, AI, UX, testing, design

Major Project Milestones (9 total): Proposal | Reqs | Arch and Design | Testing & CI | Beta Release, Gamma Release, Peer Review | Final Release | Reflection

In-class exercises & surveys (throughout): In class ex s and surveys | In-class Exam

We are here

Project tips

- Creating your project schedule
  - Include major class deliverables and dates
  - Include major integration and test points
  - [Milestone deliverables | Target date | Major tasks to make it happen]

- Week plan in your project status report (or scrum board)
  - Break down the tasks enough to assign who is delivering what this week
  - Improves clarity, understanding, and accountability

- User requirements
  - Consider all personas using your system, e.g., student, instructor, librarian
  - Formal use cases are conversations; remember to include system response and have a use case for each major feature / each persona

Today’s outline

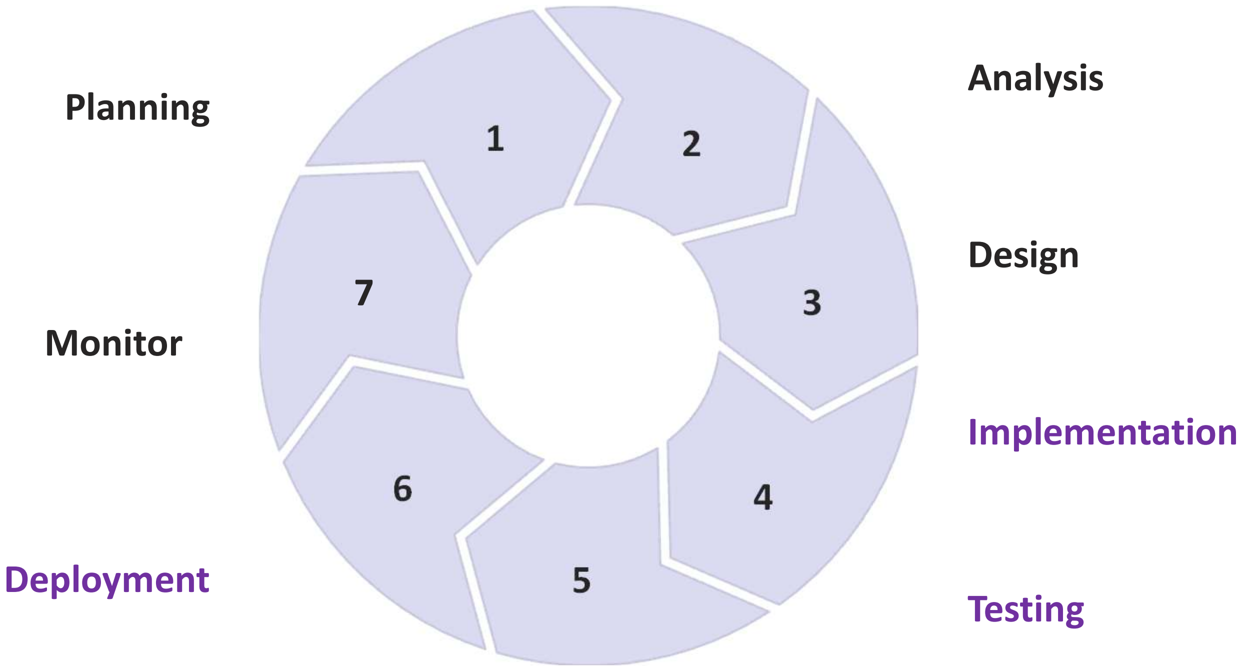
1. Build systems, as a component of …
2. Continuous integration and delivery/deployment systems

- What are these
- How do they relate
- Best practices
- Ideas to explore for your projects

See Appendix for topological sort and Calendar for devops readings

Software development lifecycle

Build/CI/CD fits primarily in Implementation, Testing, and Deployment stages



What does a developer do?

The code is written … now what?

- Get the source code
- Install dependencies
- Run static analysis
- Compile the code
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!
- Operate, monitor, repeat

What does a developer do?

The code is written … now what?

- Get the source code
- Install dependencies
- Run static analysis
- Compile the code
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!
- Operate, monitor, repeat

Which of these tasks should be handled manually?

What does a developer do?

The code is written … now what?

- Get the source code
- Install dependencies
- Run static analysis
- Compile the code
- Generate documentation
- Run tests
- Create artifacts for customers
- Ship!
- Operate, monitor, repeat

Which of these tasks should be handled manually?

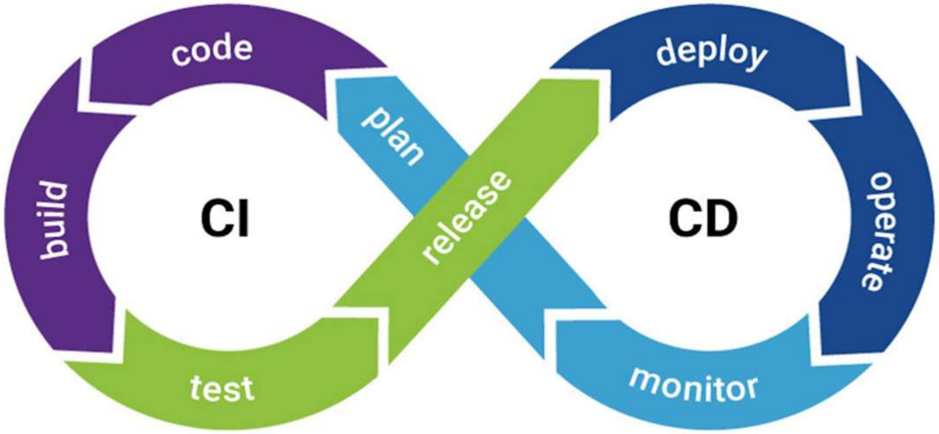
NONE!

Instead, orchestrate with a tool

Build system: a tool for automating compilation and related tasks

- Is a component of a **continuous integration/delivery/deployment system**

✓ Get the source code
✓ Install dependencies
✓ Run static analysis
✓ Compile the code
✓ Generate documentation
✓ Run tests
✓ Create artifacts for customers
✓ Ship!
✓ Operate, Monitor, Repeat



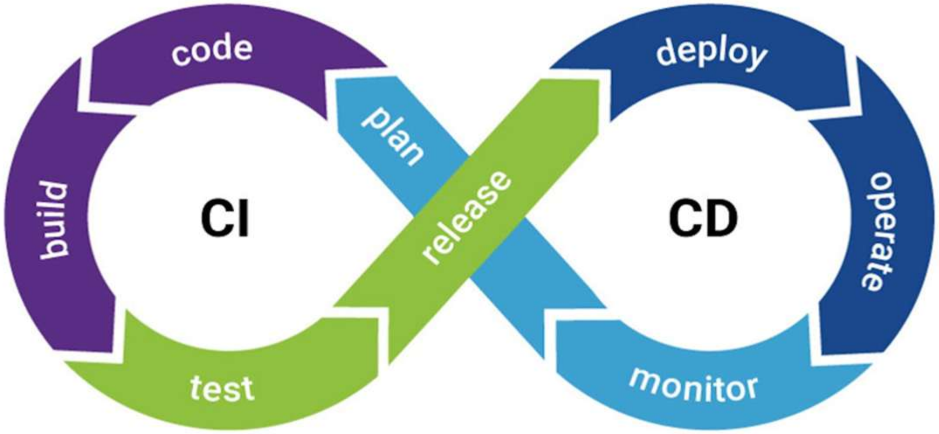
Instead, orchestrate with a tool

Build system: a tool for automating compilation and related **tasks**

- Is a component of a **continuous integration/delivery/deployment system**

These are all tasks handled by CI/CD systems

- ✓ Get the source code
- ✓ Install dependencies
- ✓ Run static analysis
- ✓ Compile the code
- ✓ Generate documentation
- ✓ Run tests
- ✓ Create artifacts for customers
- ✓ Ship!
- ✓ Operate, Monitor, Repeat



A build system has three main roles

1. Defines **tasks**

   Generally associated with getting source code and external resources, such as libraries, into an executable form

2. Defines **dependencies** among tasks (a graph)

3. **Executes** the tasks

Even build system tasks are code

- Should be tested
- Should be code-reviewed
- Should be checked into version control

A good build system is valuable to us

1. **Dependency management**
    1. Identifies dependencies between files (including externals)
    2. Runs the compiles in the right order to pick up the right dependencies
    3. Only runs the compiles needed due to dependency changes

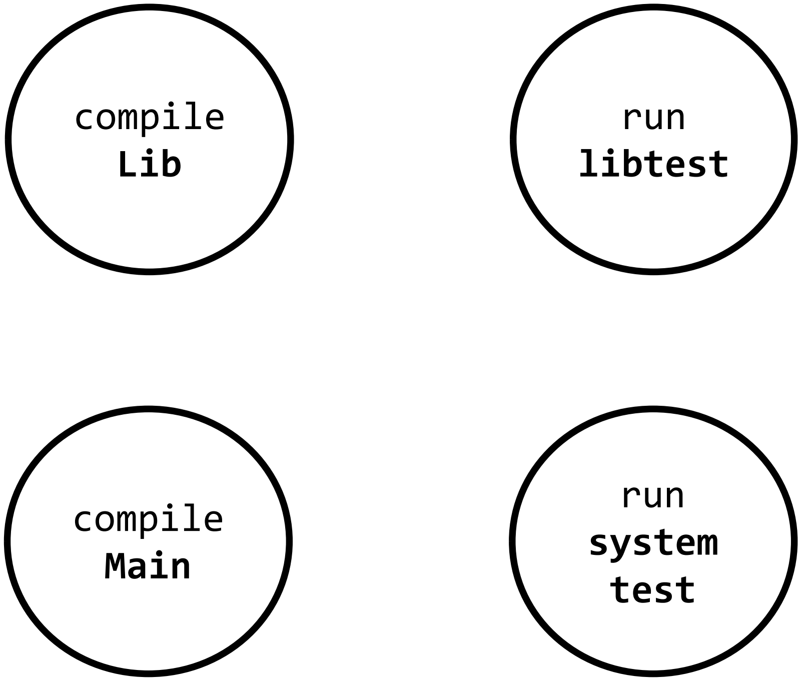
2. **Efficiency and reliability**
    1. Automates the build process so that new and old team members, even working in different dev environments, can move quickly from development to shipping code
    2. Eliminates the chance of missing steps due to tribal knowledge and/or simply errors

Here is a simple example code illustrating dependency management

```
% ls src/
    Lib.java
    LibTest.java
    Main.java
    SystemTest.java
```

Build systems: identify dependencies between tasks

```
% ls src/
    Lib.java
    LibTest.java
    Main.java
    SystemTest.java
```



compile **Lib**

run **libtest**

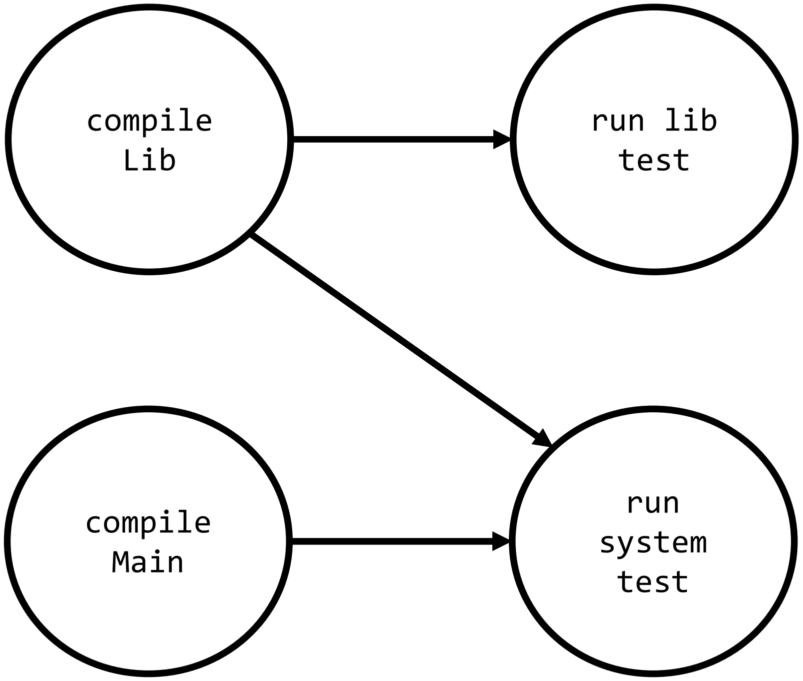
compile **Main**

run **system test**

What are the dependencies between these tasks?

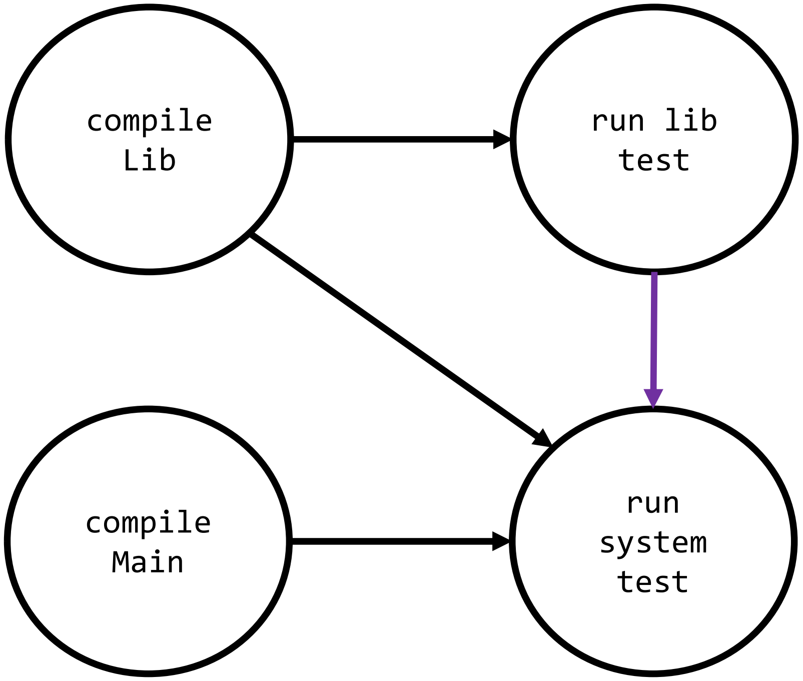
And why do I care?

Build systems: identify dependencies between tasks



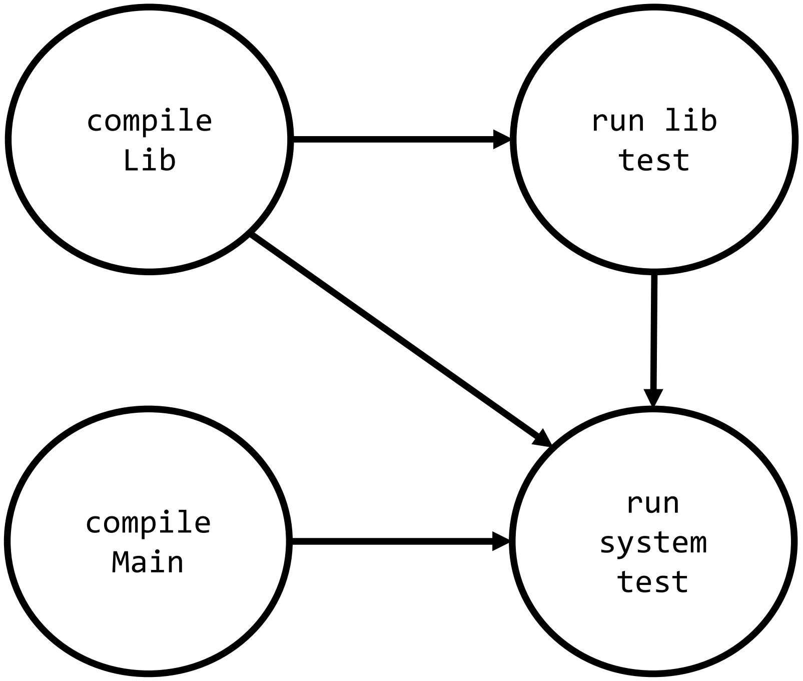
Arrow X to Y
if
Y depends on X

Build systems: identify dependencies between tasks



Build systems: identify dependencies between tasks

In what order should we run these tasks?



Tip: look for tasks with no dependencies and run those first

Build systems can determine task order

Large projects have thousands of tasks

- Dependencies between tasks form a directed acyclic graph
- Build tools use a topological sort to create an order to compiles
  - Order nodes such that all dependencies are satisfied
  - Implemented by computing indegree (number of incoming edges) for each node
  - No dependencies go first and open door to the others

External code (libraries) also can be complex

- Build systems can manage these dependencies as well!

A build system has three main roles

1. Defines **tasks** (and external resources, such as libraries)
2. Defines **dependencies** among tasks (a graph)
3. **Executes** the tasks

Consider a task for automated testing before the compile step, such as **static analysis**

Static analysis

Analyze source code for potential vulnerabilities
Run before the compile step

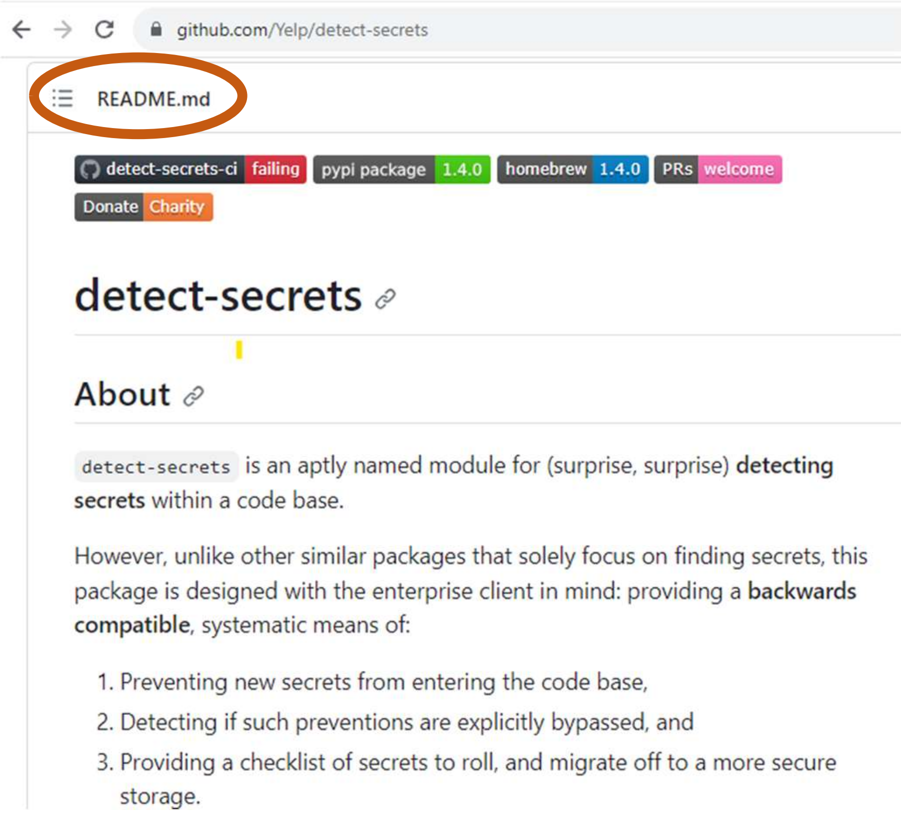
Examples:

- Credential scan
- Date scan
- Personal data scan
- Sensitive data scan

What might be others?

Is this worthwhile?

Build systems: opportunity for static analysis



detect-secrets

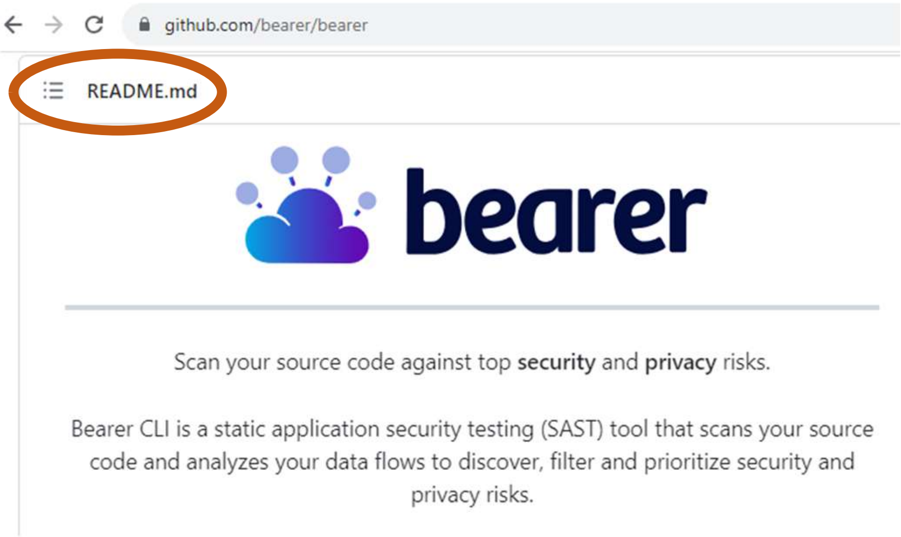
About

`detect-secrets` is an aptly named module for (surprise, surprise) **detecting secrets** within a code base.

However, unlike other similar packages that solely focus on finding secrets, this package is designed with the enterprise client in mind: providing a **backwards compatible**, systematic means of:

1. Preventing new secrets from entering the code base,
2. Detecting if such preventions are explicitly bypassed, and
3. Providing a checklist of secrets to roll, and migrate off to a more secure storage.

Could these types of static analysis tools be run earlier than build?



Scan your source code against top **security** and **privacy** risks.

Bearer CLI is a static application security testing (SAST) tool that scans your source code and analyzes your data flows to discover, filter and prioritize security and privacy risks.

Milestone 04: Research, evaluate and choose a build system for your project

Many other options!

Over to you to research

JAVA+		
	gradle	Open-source successor to **ant** and **maven**
	bazel	Open-source version of Google's internal build tool (blaze)
PYTHON		
	hatch	Implements standards from the Python standard (uses TOML files, has PIP integration)
	poetry	Packaging and dependence manager
	tox	Automate and standardize testing
JAVASCRIPT		
	npm	Standard package/task manager for Node, "Largest software registry in the world."
	webpack	Module bundler for modern JavaScript applications
	gulp	Tries to improve dependency and packing

Today's outline

- Build systems, as a component of …
- **Continuous integration and delivery/deployment systems**

  - What are these and
  - How do they relate
  - Best practices
  - Ideas to explore for your projects

Continuous integration

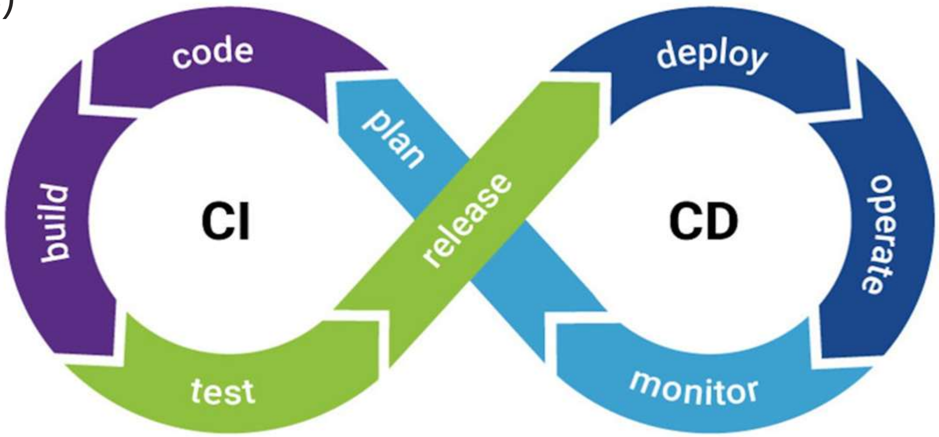
Purpose is to merge developer code changes into a shared repository multiple times a day, with automated builds and tests

Includes:

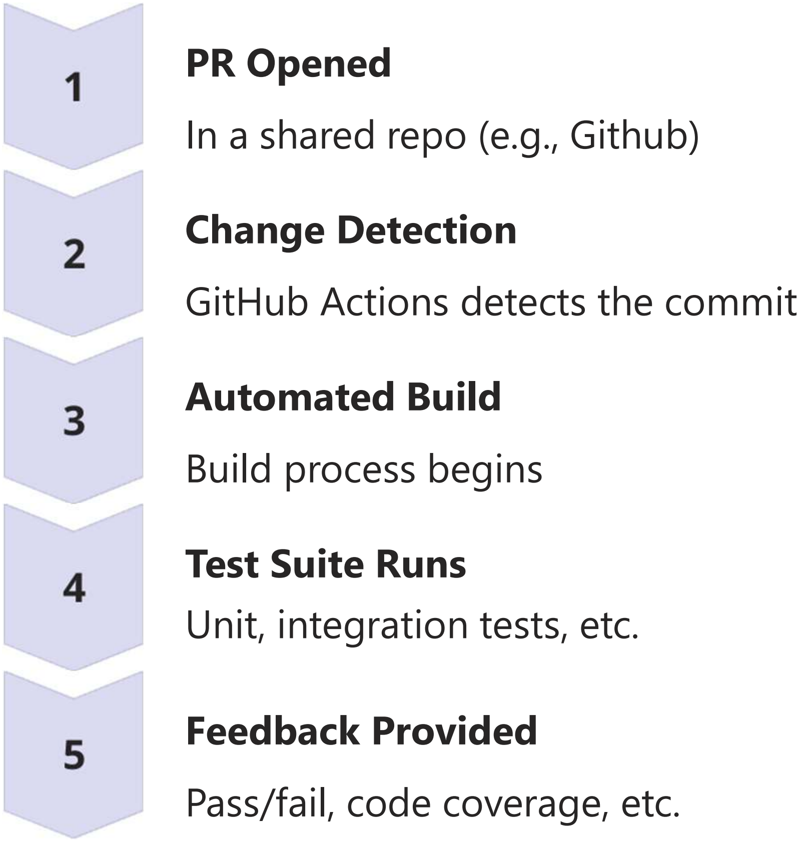
- Frequent commits (small, incremental changes)
- Automated builds triggered on every commit
- Automated tests for rapid feedback

Pros:

- Early bug detection
- Reduced integration headaches
- Improved team collaboration



Continuous integration workflow example



1. **PR Opened**

   In a shared repo (e.g., Github)

2. **Change Detection**

   GitHub Actions detects the commit

3. **Automated Build**

   Build process begins

4. **Test Suite Runs**

   Unit, integration tests, etc.

5. **Feedback Provided**

   Pass/fail, code coverage, etc.

Continuous integration basics

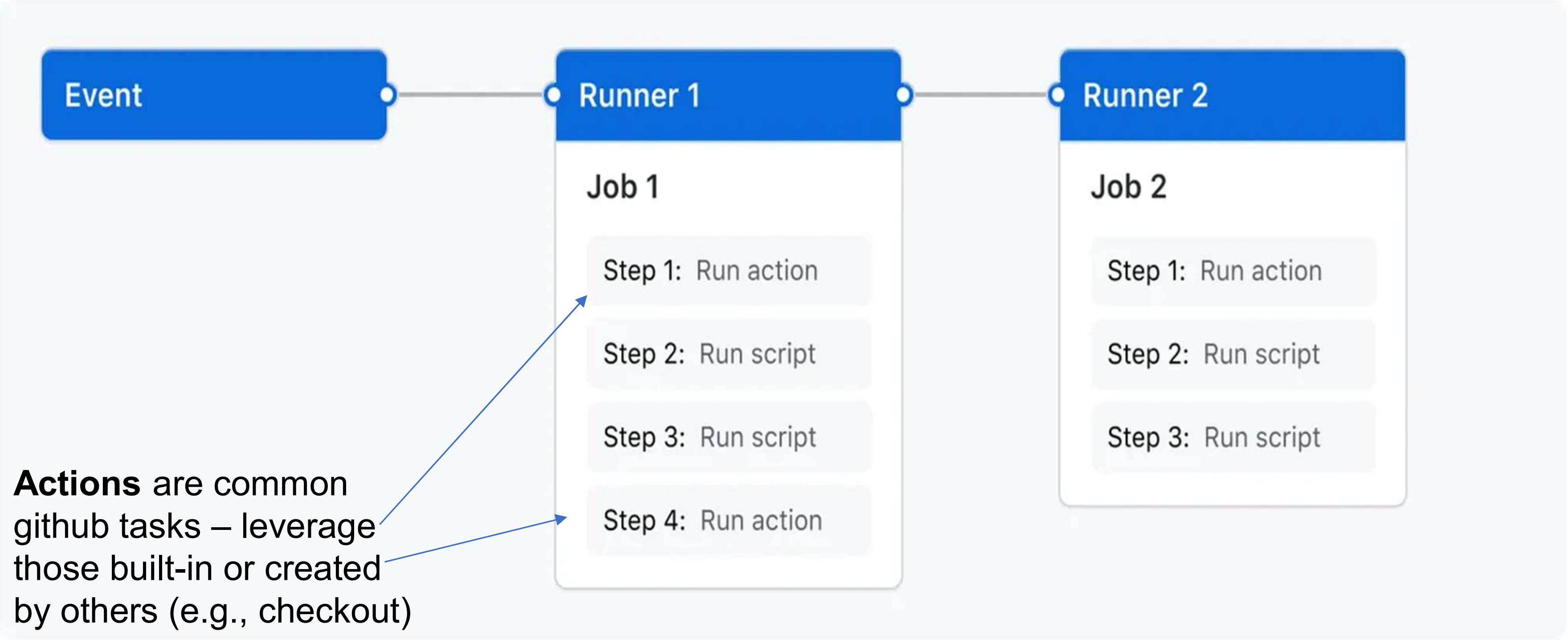
- A CI **workflow** is **triggered** when an **event** occurs in your [shared] repo
  - Example events
    - Push
    - Pull request
    - Issue creation

- A **workflow** contains **jobs** that run in a defined order
  - A job is like a shell-script and can have multiple steps
  - Jobs run in their own vm/container called a **runner**
  - Example jobs
    - Run static analysis
    - Compile, test
    - Deploy to test, deploy to prod

Using GitHub CI terminology but concepts span other CI systems

CI basics (w/ GitHub CI)

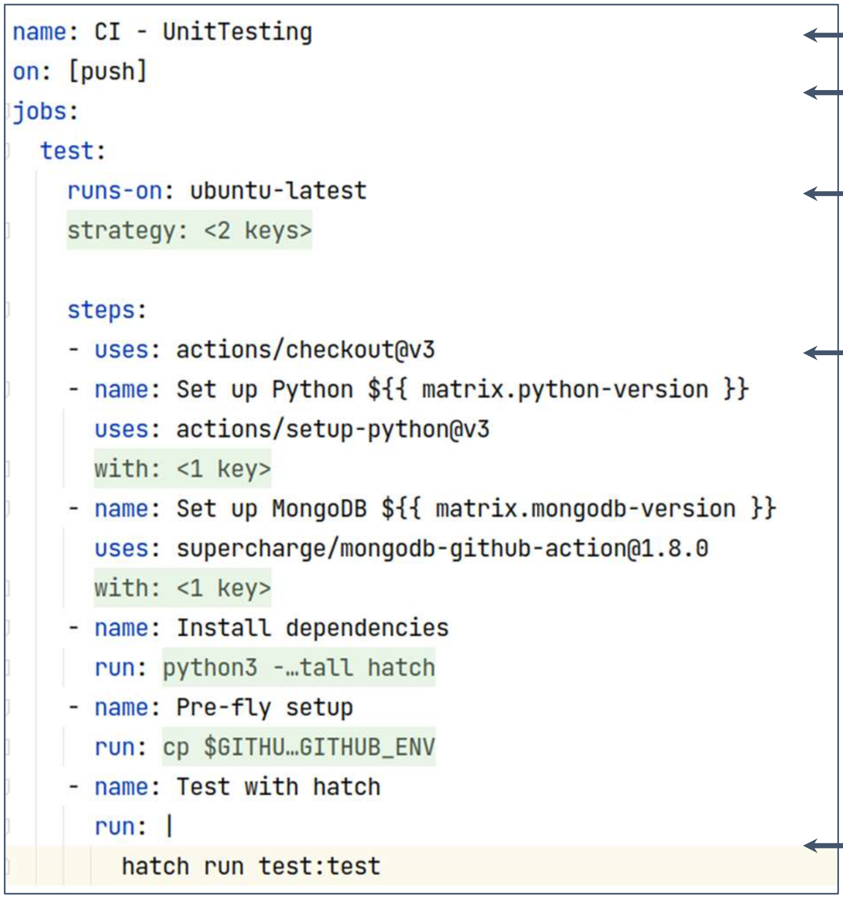
What SW architecture does this appear to be using?



Actions are common github tasks – leverage those built-in or created by others (e.g., checkout)

Example: CI with Github actions

Unit tests are triggered on every push of new code



```
name: CI - UnitTesting
on: [push]
jobs:
  test:
    runs-on: ubuntu-latest
    strategy: <2 keys>

    steps:
    - uses: actions/checkout@v3
    - name: Set up Python ${{ matrix.python-version }}
      uses: actions/setup-python@v3
      with: <1 key>
    - name: Set up MongoDB ${{ matrix.mongodb-version }}
      uses: supercharge/mongodb-github-action@1.8.0
      with: <1 key>
    - name: Install dependencies
      run: python3 -…tall hatch
    - name: Pre-fly setup
      run: cp $GITHU…GITHUB_ENV
    - name: Test with hatch
      run: |
        hatch run test:test
```

Workflow name

Trigger

Linux OS environment

Code reuse with established “actions”

One command to run test suite

Let’s look at some live CI workflows

hannahpotter/manual-code-review-examples

See: .github/workflows

Real 403 project

See: it runs lint and code coverage report too

CI vs CD: What’s the difference?

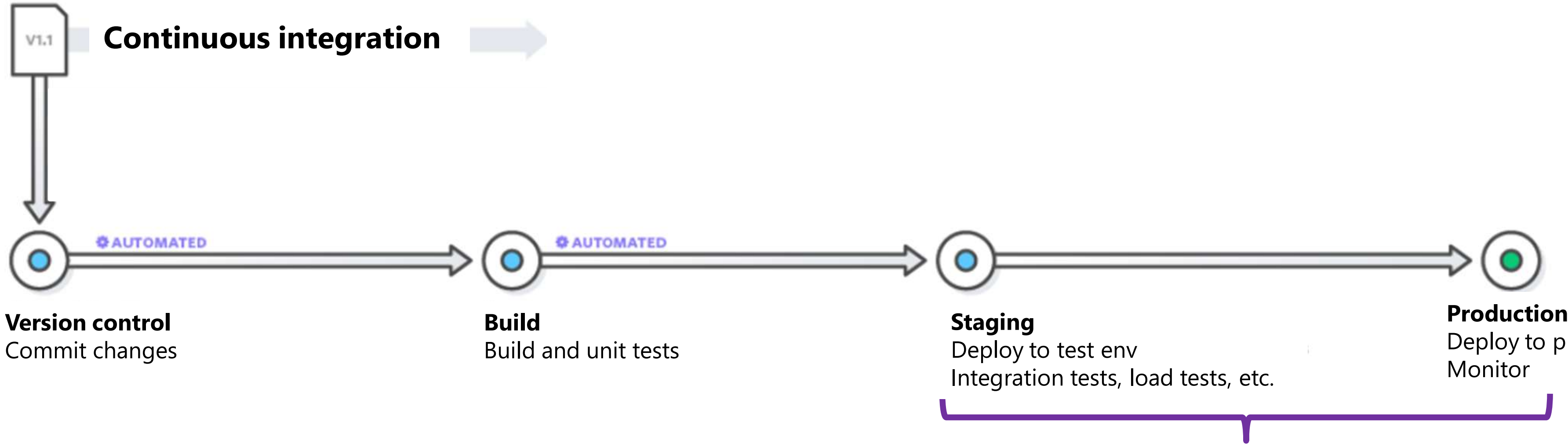
Continuous Integration (CI)

- Devs regularly integrate code into a shared repository
- System builds/tests automatically with each update
- Complements local developer workflows (e.g., may run diff tests)
- **Goal:** to find/address bugs quicker, improve quality, reduce time to get to working code

Continuous Deployment/Delivery (CD)

- Builds on top of CI
- Automatically pushes changes to [staging environment and then] production
- **Goal:** always have a deployment-ready build that has passed through a standardized testing process

CD vs CD: What’s the difference?



Continuous integration

Version control
Commit changes

Build
Build and unit tests

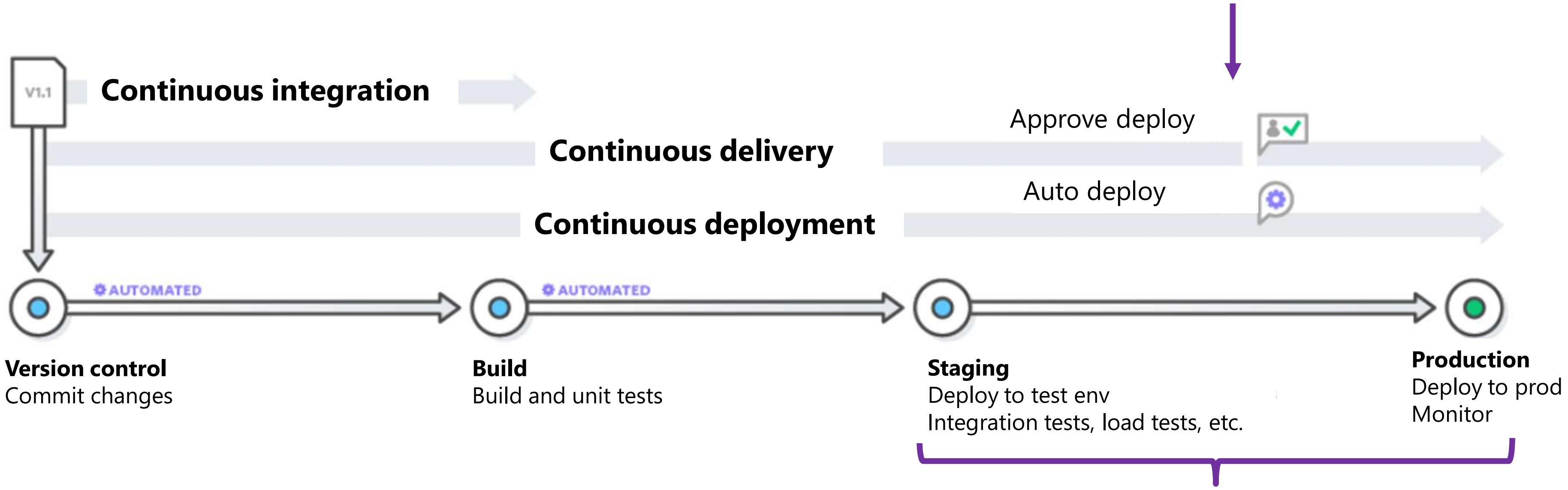
Staging
Deploy to test env
Integration tests, load tests, etc.

Production
Deploy to prod
Monitor

Staging before Production is very typical of industry practices

CD vs CD: What’s the difference?

Why would you not always automatically deploy to prod?



V1.1

Continuous integration

Continuous delivery

Approve deploy

Continuous deployment

Auto deploy

AUTOMATED

AUTOMATED

Version control
Commit changes

Build
Build and unit tests

Staging
Deploy to test env
Integration tests, load tests, etc.

Production
Deploy to prod
Monitor

Staging before Production is very typical of industry practices

Amazon example

CD vs CD: What’s the difference?

Continuous Delivery

- Codebase is always in a **deployable state**
- May require **manual approval** to push to production
- Common for mobile apps due to app store review process

Continuous Deployment

- Fully **automated release process** to production
- No manual steps once tests pass
- Common for web sites & backend systems

Milestone 04: Research, evaluate and choose a CI system for your project

Hosted Services

- GitHub Actions
- GitLab CI/CD
- CircleCI
- Travis CI
- Buildkite

Self-Managed Tools

- Jenkins
- TeamCity
- Bamboo

Supporting Technologies

- Docker for containerization
- Kubernetes for container orchestration
- Infrastructure as Code (Terraform, Ansible)

Consider these CI/CD scenarios…

No automated CI/CD system

- Manual build, integration, and releases
- Limited or no automated testing
- Long feedback loops
- Business impact?

No automated CI/CD system

- Manual build, integration, and releases
  - Large, infrequent code merges lead to conflicts discovered late
  - Error-prone and time-consuming deployment steps
- Limited or no automated testing
  - Bugs often caught in production
  - High risk of downtime
- Long feedback loops
  - Delayed discovery of issues
  - Slow response to user needs or market changes
- High cost business impact

Poorly implemented CI/CD system

- Incomplete or rarely used pipelines
- Minimal test coverage
- Unreliable pipelines
- Business impact?

Poorly implemented CI/CD system

- Incomplete or rarely used pipelines
  - Build/test stages not automatically triggered, skipped or inconsistent
- Minimal test coverage
  - Automated tests exist but don't cover critical functionality
  - Production bugs still leak through
  - False sense of security when pipelines pass without catching issues
- Unreliable pipelines
  - Frequent pipeline failures without clear resolution
  - Teams lose trust and revert to manual processes
- High cost business impact

Robust CI/CD system

1 **Fully automated build & test pipeline**
Every commit triggers a build and thorough suite of tests
Faster feedback; issues discovered and fixed early

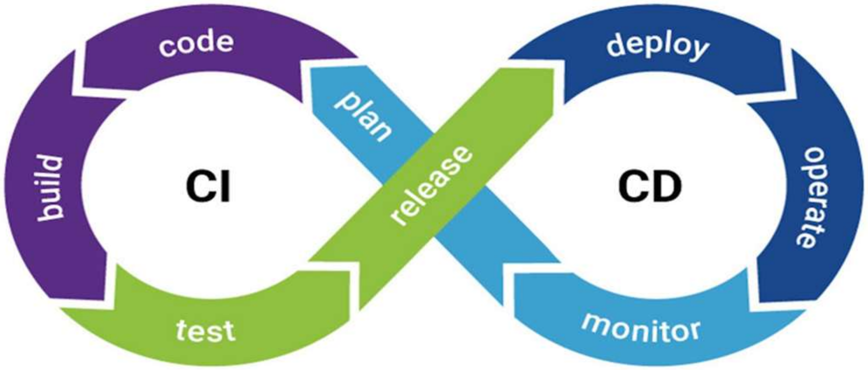
2 **Frequent, small releases**
Easier to deploy, roll back if necessary, and reduce release risk; Users see new features and fixes quickly

3 **High confidence in deployment**
Well-defined gating stages ensure only stable code is promoted; Post-deployment monitoring and automatic rollback if critical failures occur

4 **Positive business impact**
Faster time-to-market, improved quality & reliability, enhanced developer productivity, strong DevOps culture

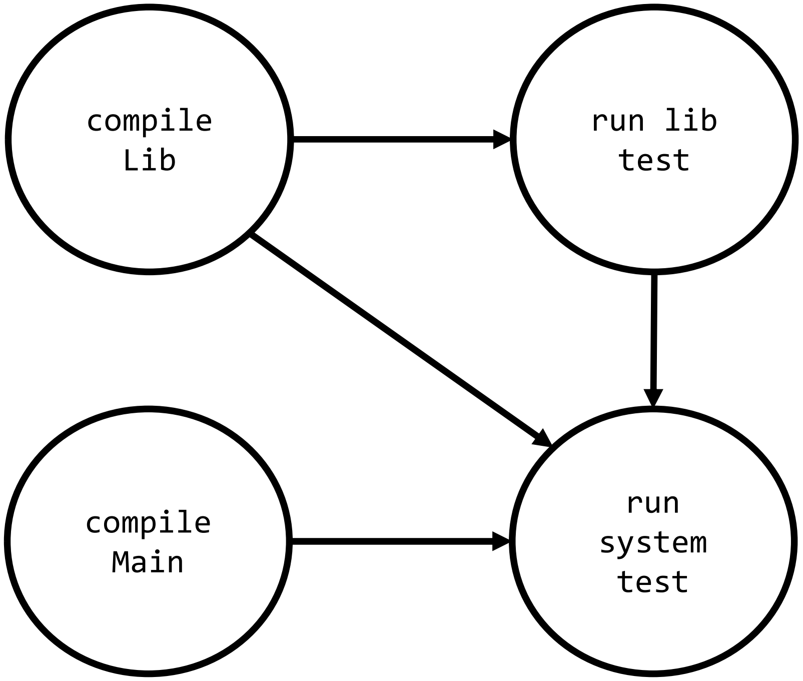
Summary

- Automate, automate, automate everything!
- Always use a build tool (one-step build)
- Use a CI tool to build and test your code on every commit
- Don’t depend on anything that’s not in the build file
- Don’t break the build!



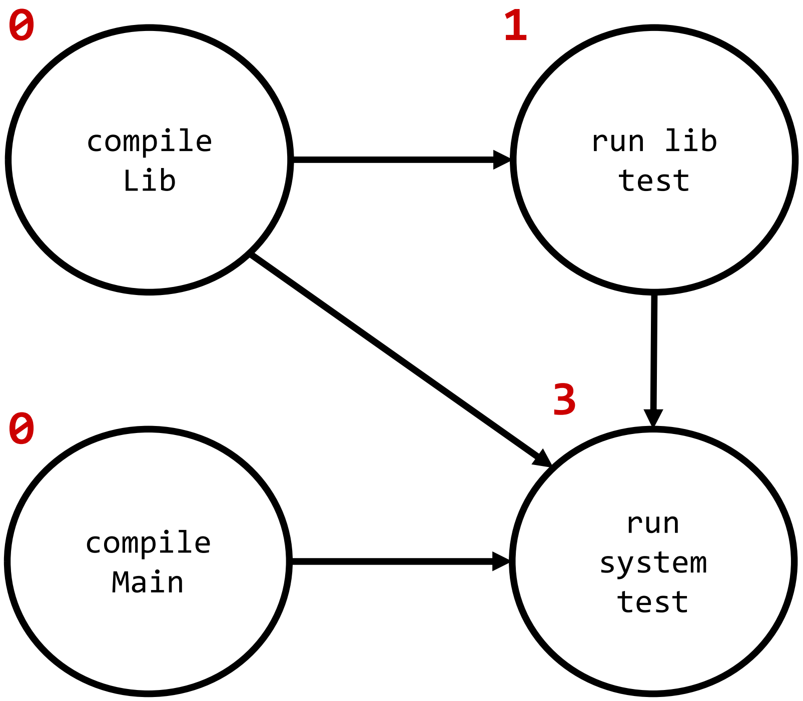
Appendix - Topological sort example

Build systems: topological sort

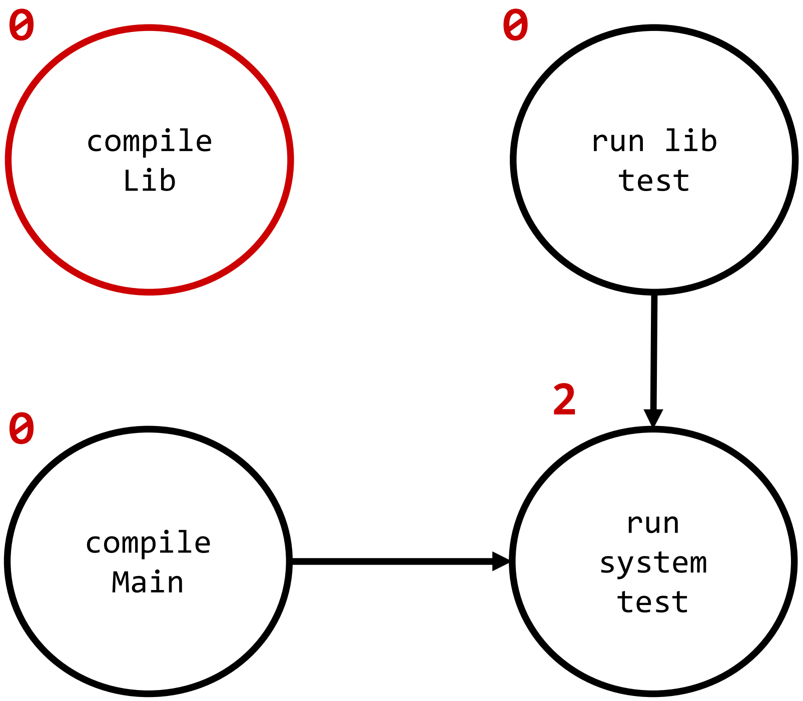


What’s the indegree of each node?

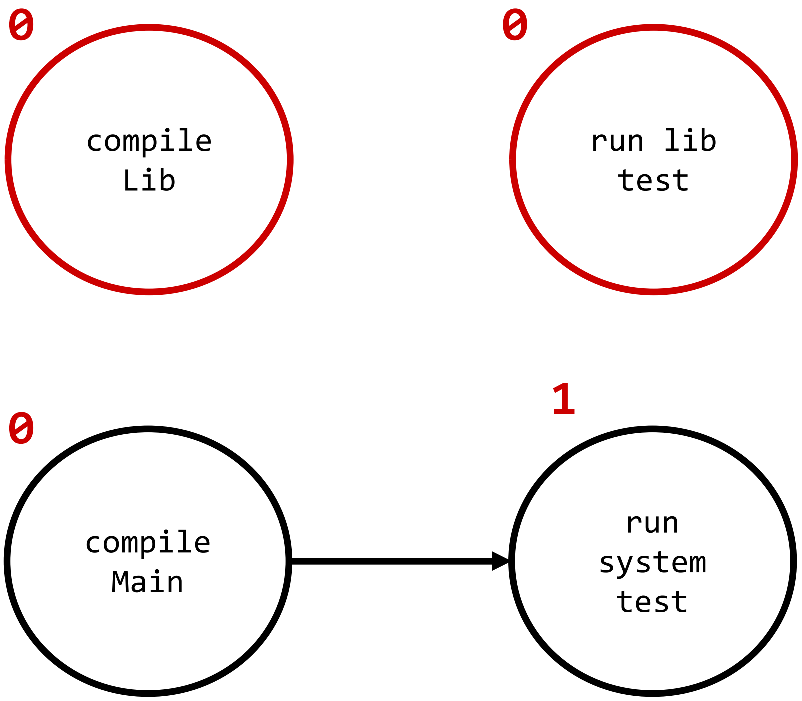
Build systems: topological sort



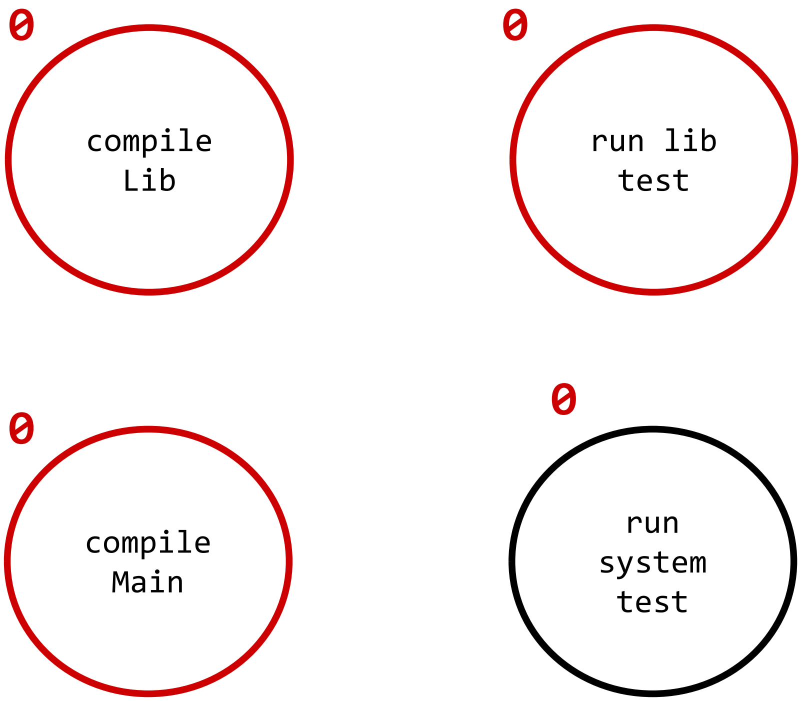
Build systems: topological sort



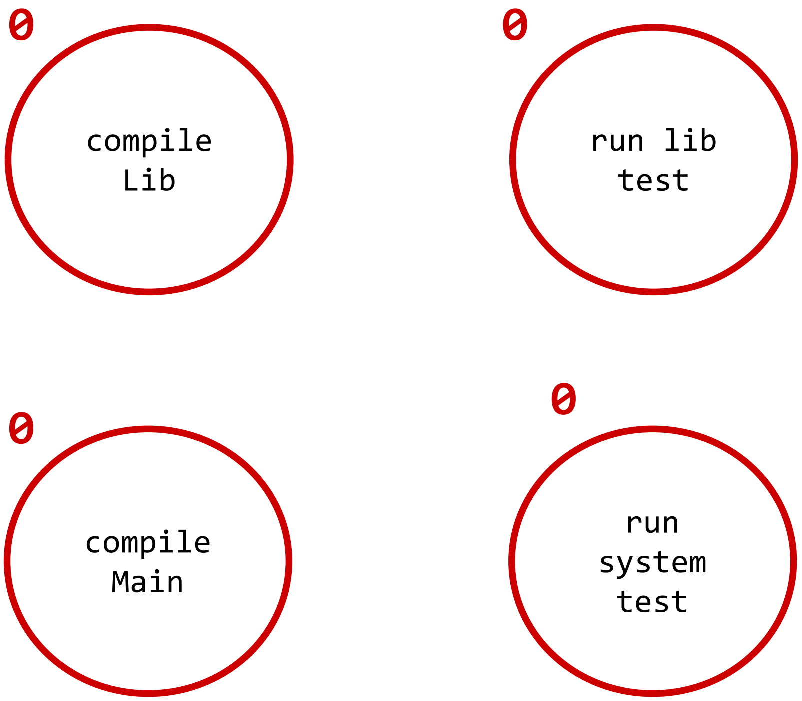
Build systems: topological sort



Build systems: topological sort



Build systems: topological sort



Build systems: topological sort

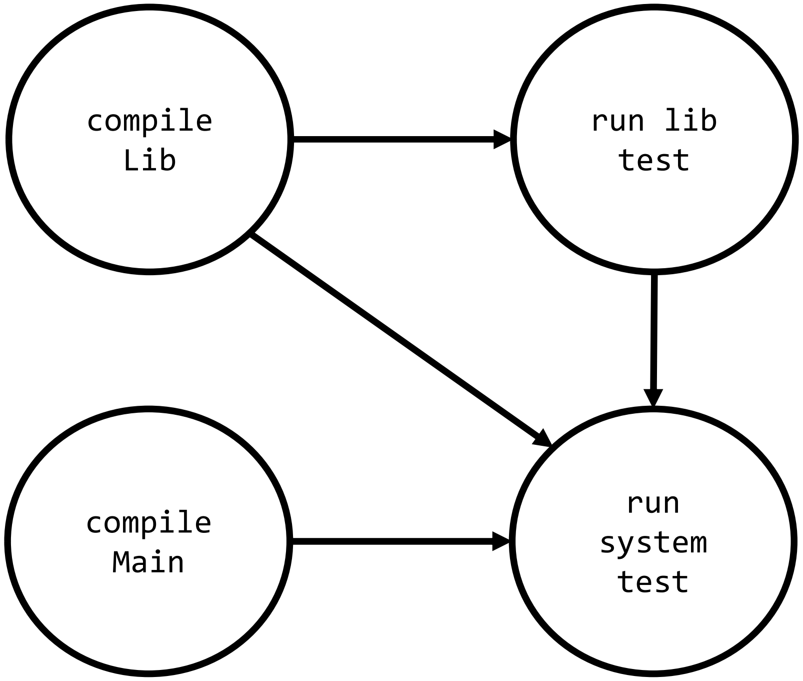
Valid sorts:

1. compile Lib, run lib test, compile Main, run system test

2. compile Main, compile Lib, run lib test, run system test

3. compile Lib, compile Main, run lib test, run system test

Which is preferable?



Let’s try writing our own simple CI workflow

Follow along at:

https://github.com/alv880/UW-CSE403-Alv-Projects

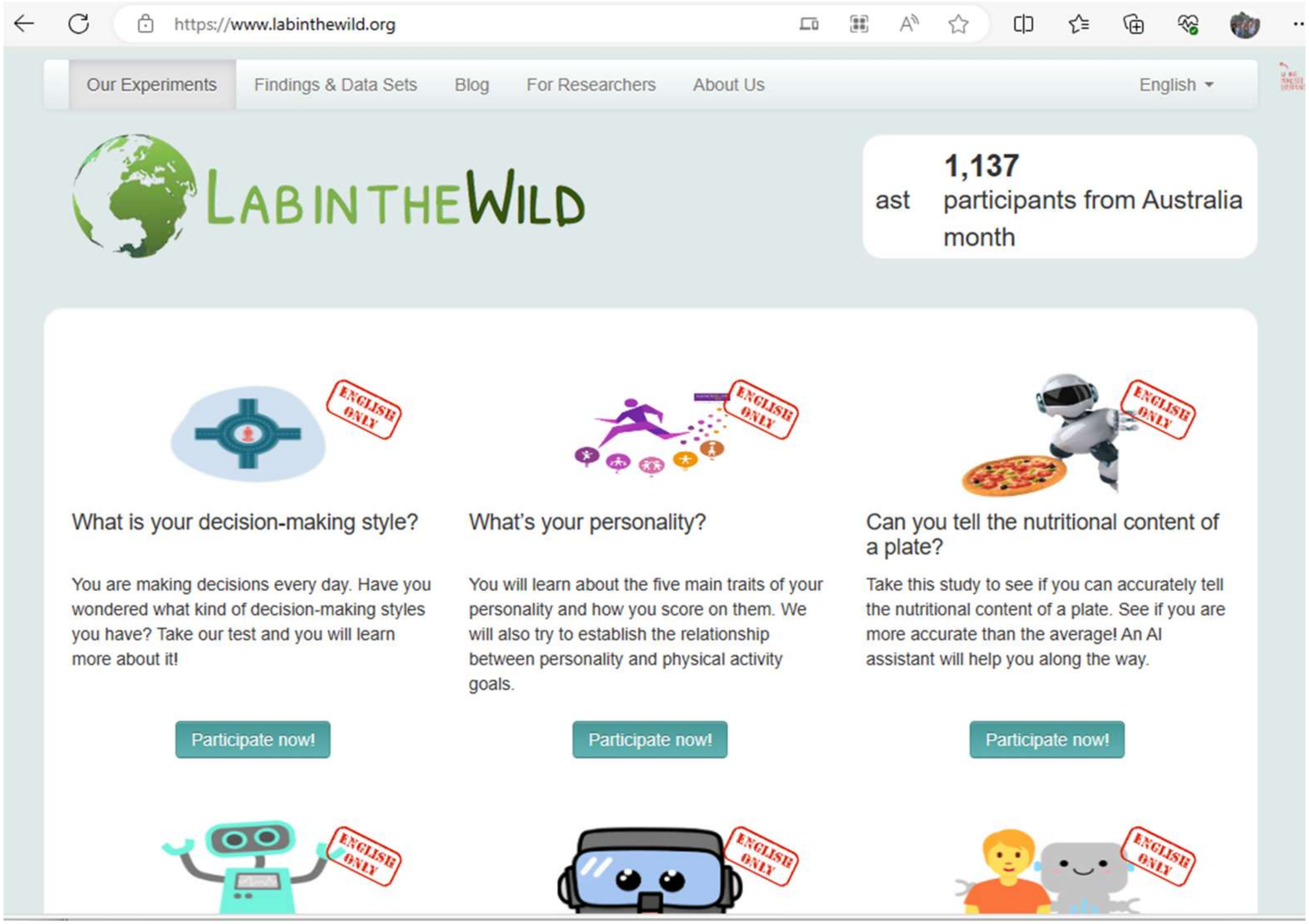
Github Actions resource:

https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions

Example: CI at work in CSE

Lab In The Wild is a research project drawing survey input from diverse community

– Nigini Oliveira UW researcher provided this example



ast **1,137** participants from Australia month

What is your decision-making style?

You are making decisions every day. Have you wondered what kind of decision-making styles you have? Take our test and you will learn more about it!

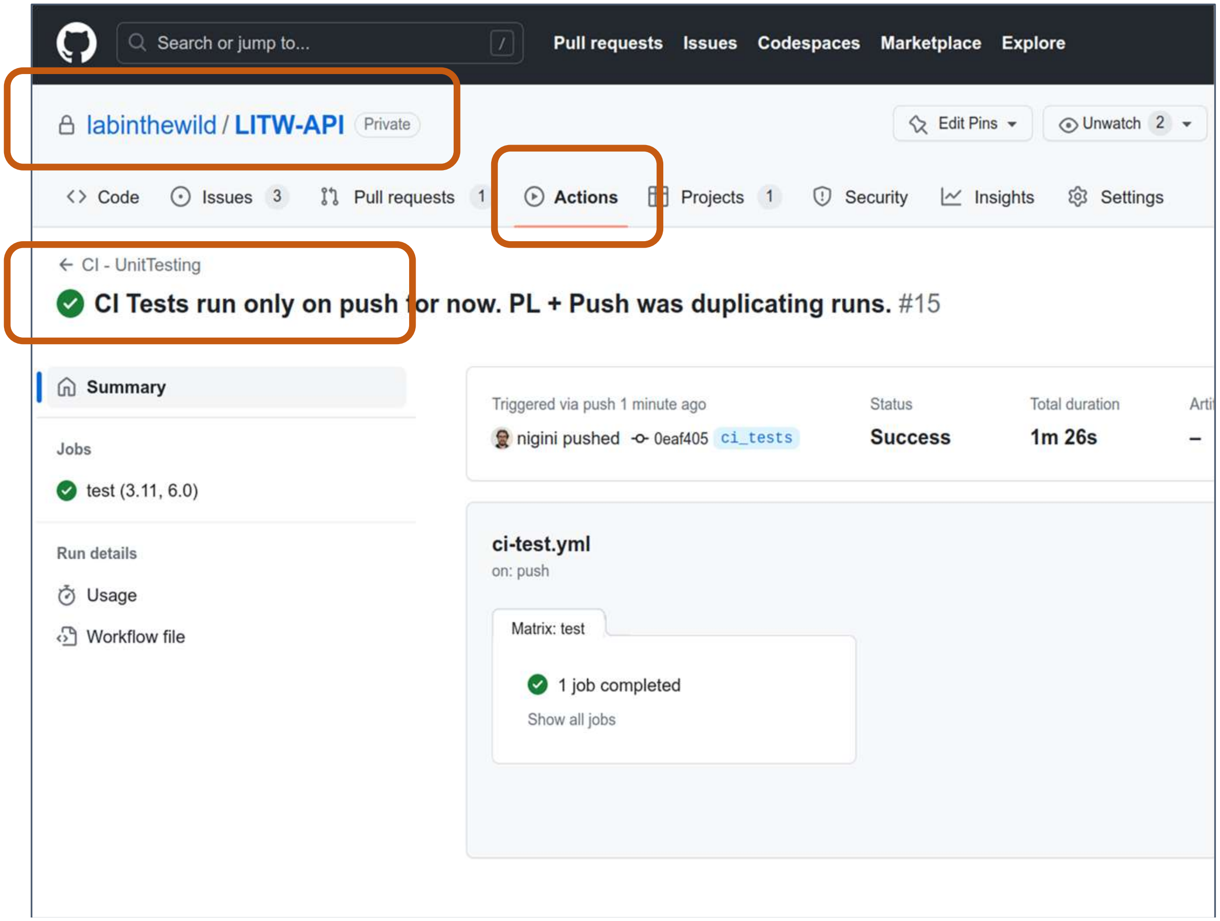
What's your personality?

You will learn about the five main traits of your personality and how you score on them. We will also try to establish the relationship between personality and physical activity goals.

Can you tell the nutritional content of a plate?

Take this study to see if you can accurately tell the nutritional content of a plate. See if you are more accurate than the average! An AI assistant will help you along the way.

Example: CI with Github actions



labinthewild / LITW-API Private

CI - UnitTesting

CI Tests run only on push for now. PL + Push was duplicating runs. #15

Jobs

test (3.11, 6.0)

Run details

Triggered via push 1 minute ago	Status	Total duration	Arti
nigini pushed 0eaf405 ci_tests	Success	1m 26s	–

ci-test.yml
on: push

Matrix: test

1 job completed

Let’s try writing our own simple workflow

Follow along at:

https://github.com/alv880/UW-CSE403-Au23-Projects

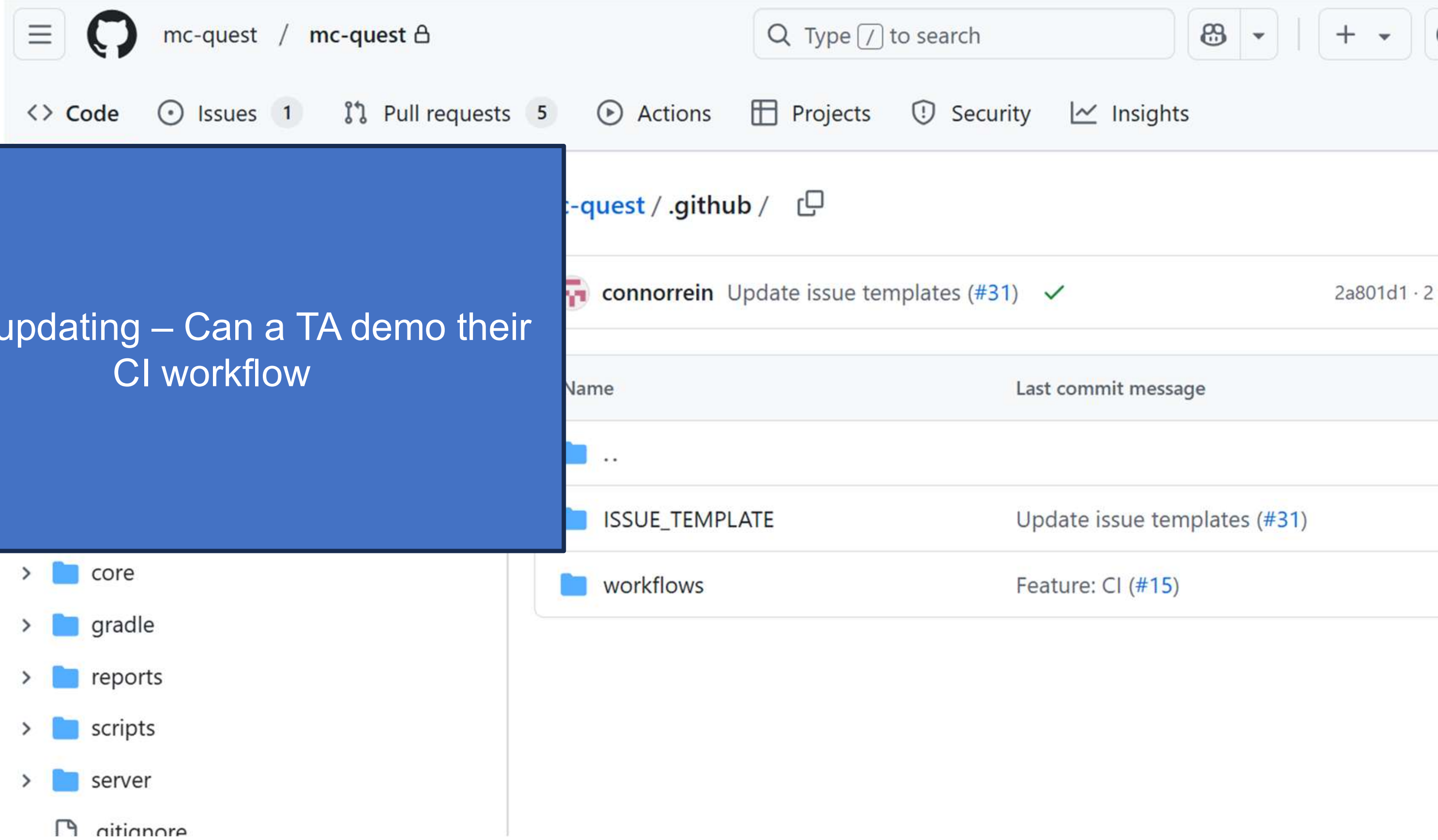
Real 403 project example at:

https://github.com/amgupta2/IntelliCue/blob/main/.github/workflows/ci.yml

Nice light starter tutorial – Automation Step by Step:
https://www.youtube.com/watch?app=desktop&v=ylEy4eLdhFs

Let’s look at a CI workflow from a CSE 403 project

Connor’s team’s repo



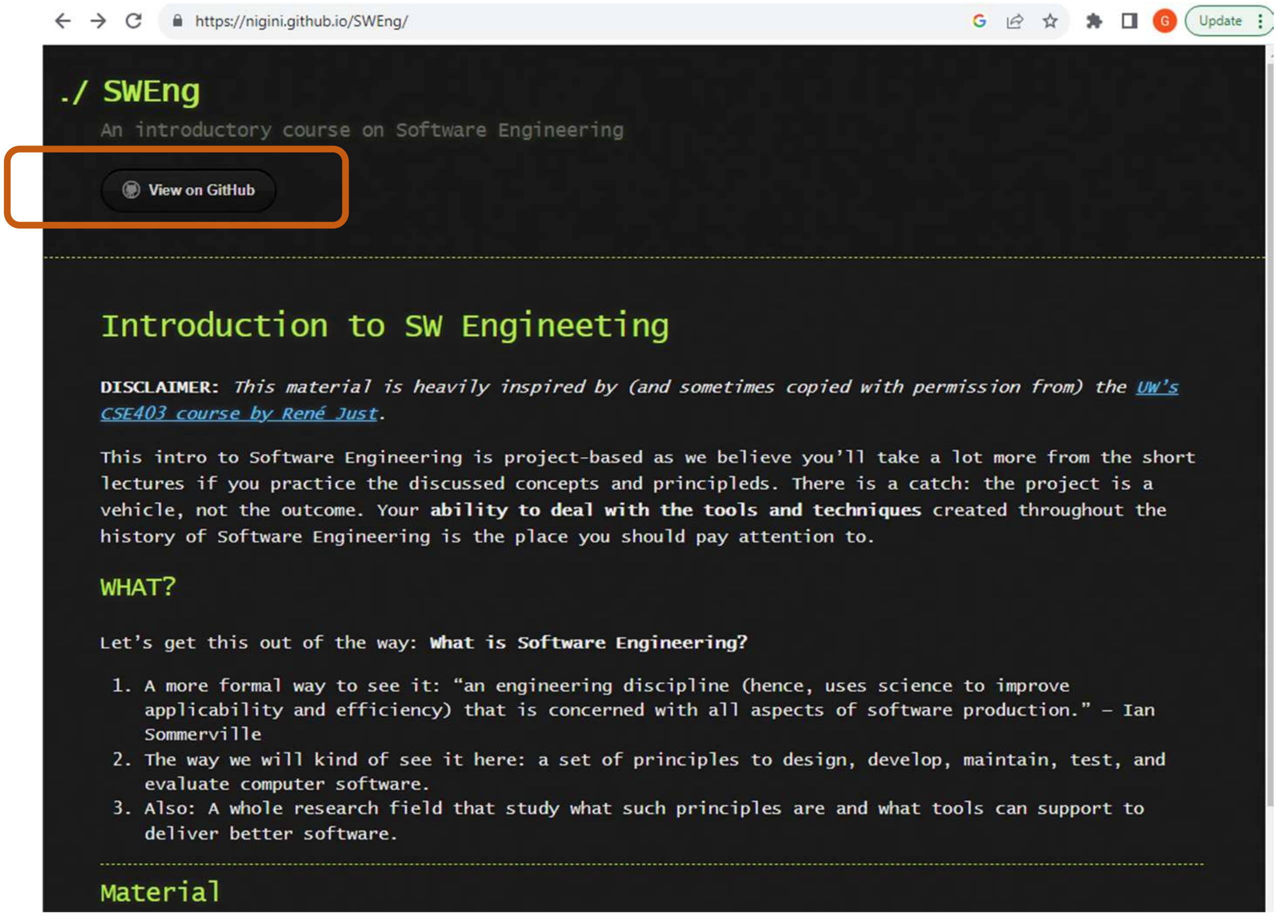
connorrein Update issue templates (#31) ✓ 2a801d1 · 2

Name	Last commit message
..	
ISSUE_TEMPLATE	Update issue templates (#31)
workflows	Feature: CI (#15)

Need updating – Can a TA demo their CI workflow

Example: continuous deployment with GitHub Pages (https://pages.github.com/)

Content updates trigger publishing the website update



./ SWEng

An introductory course on Software Engineering

Introduction to SW Engineeting

DISCLAIMER: *This material is heavily inspired by (and sometimes copied with permission from) the UW's CSE403 course by René Just.*

This intro to Software Engineering is project-based as we believe you'll take a lot more from the short lectures if you practice the discussed concepts and principleds. There is a catch: the project is a vehicle, not the outcome. Your **ability to deal with the tools and techniques** created throughout the history of Software Engineering is the place you should pay attention to.

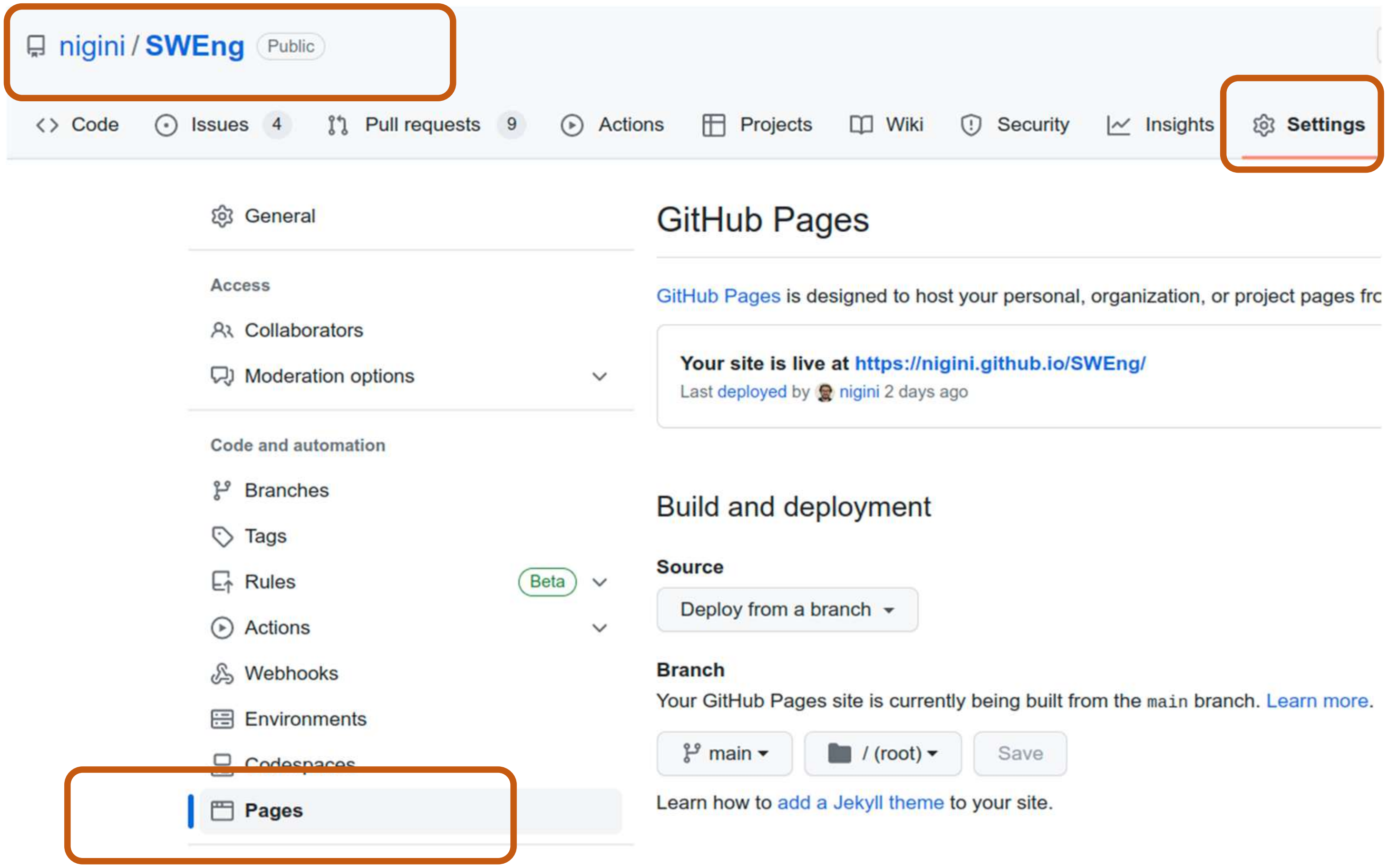
WHAT?

Let's get this out of the way: **What is Software Engineering?**

1. A more formal way to see it: "an engineering discipline (hence, uses science to improve applicability and efficiency) that is concerned with all aspects of software production." – Ian Sommerville
2. The way we will kind of see it here: a set of principles to design, develop, maintain, test, and evaluate computer software.
3. Also: A whole research field that study what such principles are and what tools can support to deliver better software.

Material

Example: continuous deployment config



GitHub Pages

GitHub Pages is designed to host your personal, organization, or project pages frc

Your site is live at https://nigini.github.io/SWEng/

Last deployed by nigini 2 days ago

Build and deployment

Source

Deploy from a branch

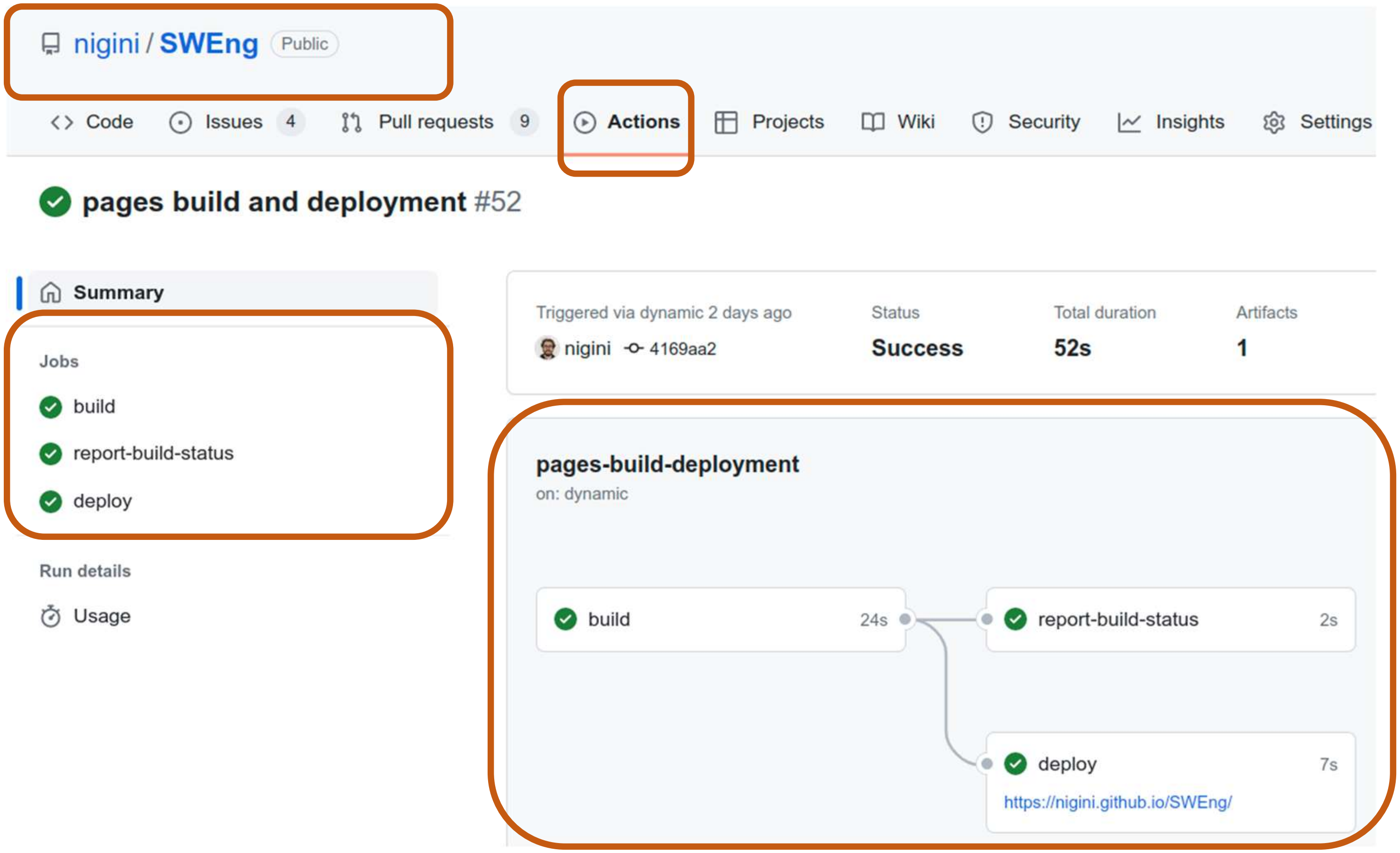
Branch

Your GitHub Pages site is currently being built from the `main` branch. Learn more.

main / (root) Save

Learn how to add a Jekyll theme to your site.

Example: continuous deployment config



nigini / **SWEng** Public

✓ **pages build and deployment** #52

Jobs

- ✓ build
- ✓ report-build-status
- ✓ deploy

Run details

Triggered via dynamic 2 days ago	Status	Total duration	Artifacts
nigini -o- 4169aa2	Success	52s	1

pages-build-deployment
on: dynamic

- ✓ build 24s
- ✓ report-build-status 2s
- ✓ deploy 7s https://nigini.github.io/SWEng/