

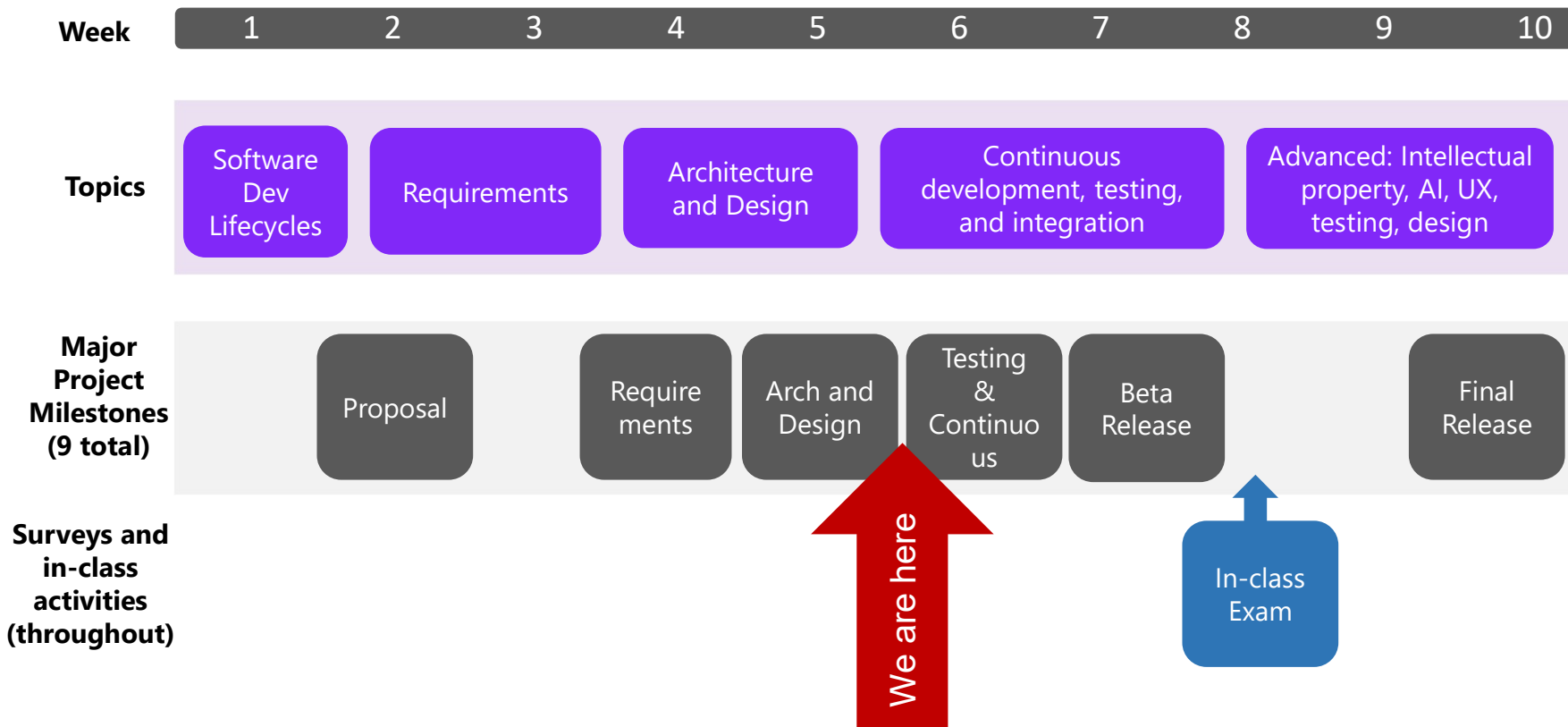
# Build systems & continuous integration and deployment

CSE 403 Software Engineering

Winter 2025

# Course overview: schedule

**Important:** See Calendar and Canvas for current details of topics and assignments



# Today's outline

- Build systems
- Continuous integration and deployment systems
  - What are these
  - How do they relate
  - Best practices
  - Ideas to explore for your projects

Have a question for [Zach Sperske](#) (industry speaker from Affirm, Inc., next Wed)?  
[Question link on Calendar and in Ed.](#)

# 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
- Perform 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?

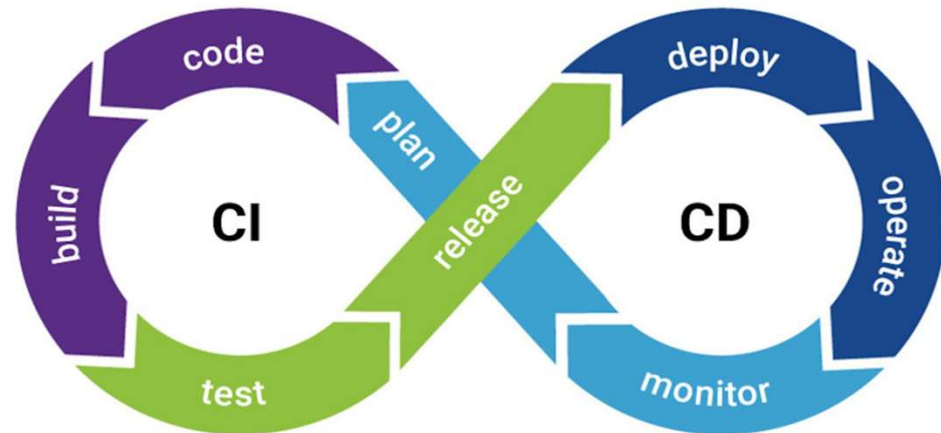
**NONE!**

# Instead, orchestrate with a tool

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

- Is a component of a **continuous integration/deployment system** as today we automate more than just the build step of producing shippable software

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



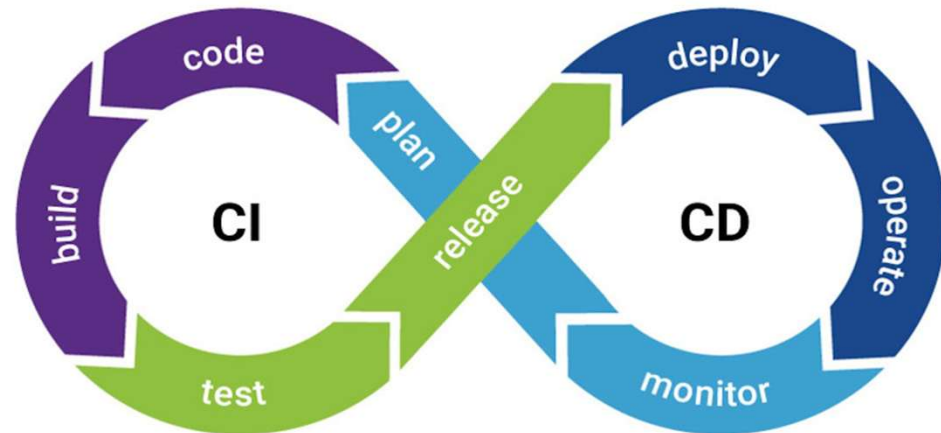
# Instead, orchestrate with a tool

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

- Is a component of a **continuous integration/deployment system** as today we automate more than just the build step of producing shippable software

## These are all tasks!

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



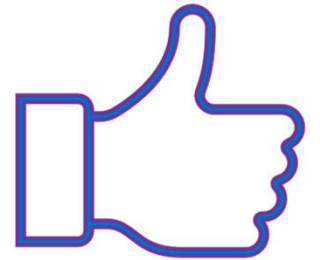


# Even build system **tasks** are **code**

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

# Adding to our software engineering best practices list

- Automate, automate, automate everything!
- Always use a build tool (one-step build) 😊
- Use a continuous integration 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!



# 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 simple errors

# A build system has three main roles

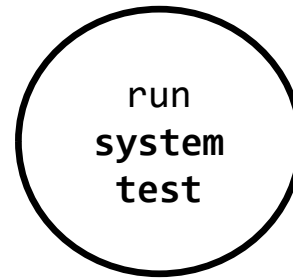
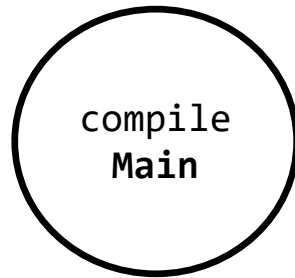
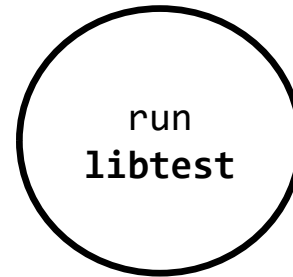
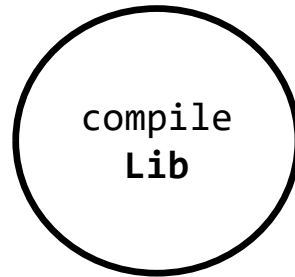
1. Defines **tasks** (and external resources, such as libraries)
2. Defines **dependencies** among tasks (a graph)
3. **Executes** the tasks

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

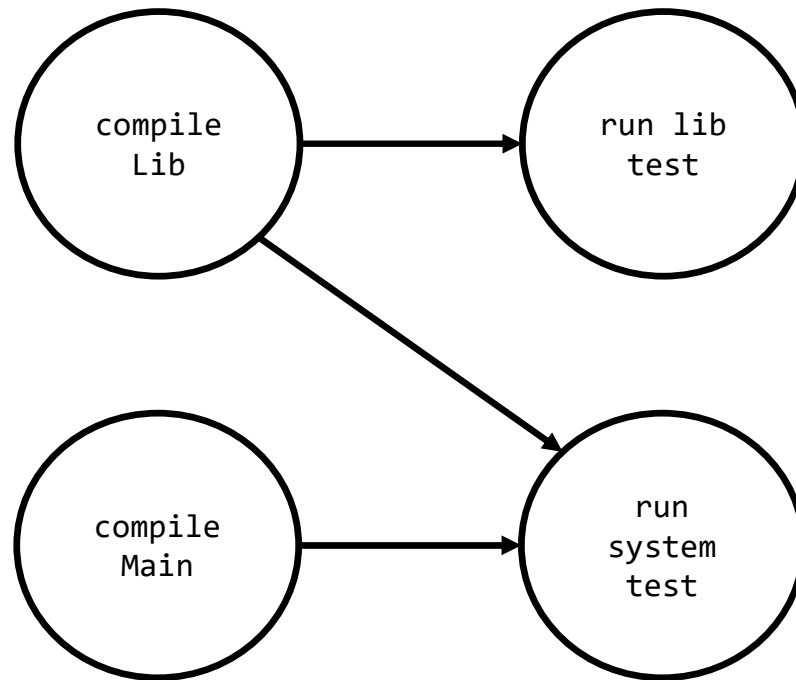
```
% ls src/  
Lib.java  
LibTest.java  
Main.java  
SystemTest.java
```



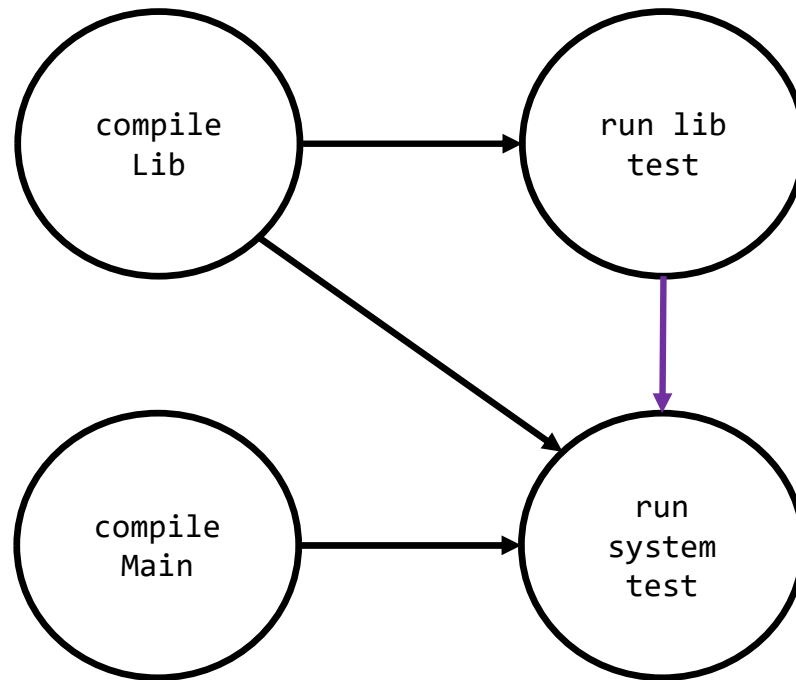
What are the dependencies between these tasks?

And why do I care?

# Build systems: identify dependencies between tasks



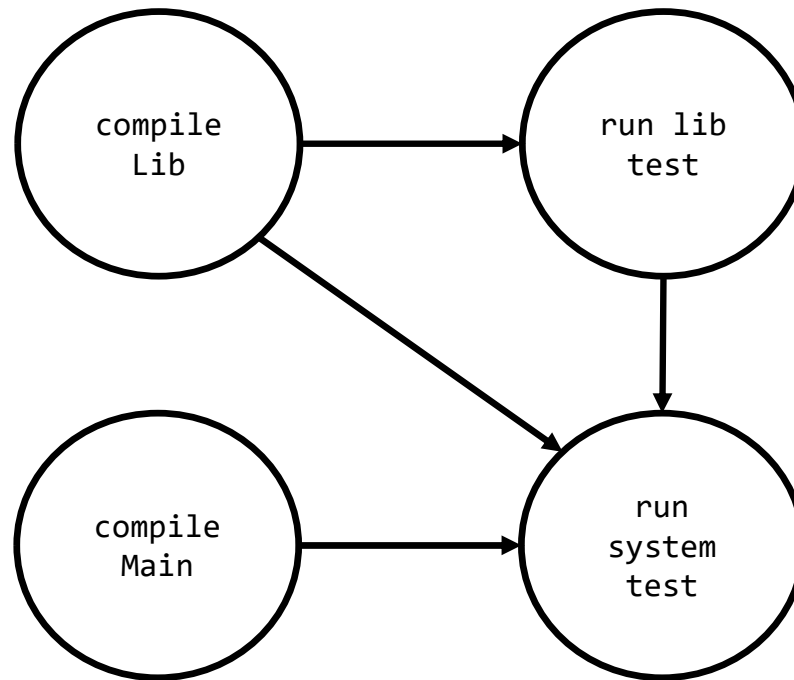
# Build systems: identify dependencies between tasks





# 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 compile
  - 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
  - See Appendix for example

## 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

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

github.com/Yelp/detect-secrets

☰ README.md

detect-secrets-ci failing pypi package 1.4.0 homebrew 1.4.0 PRs welcome Donate Charity

## 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?

github.com/bearer/bearer

☰ README.md



# bearer

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.

# Here's an example build system 'input'

Basic-Stats  
"ant"  
**build.xml**

(from last week's in-class  
exercise)

Simple-C  
"make"  
**Makefile**

## 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 <b>ant</b> and <b>maven</b>
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
- **Continuous integration and delivery/deployment systems**
  - What are these and
  - How do they relate
  - Best practices
  - Ideas to explore for your projects



# CI/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 (CD) [Continuous Delivery]

- 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



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



**Jenkins**



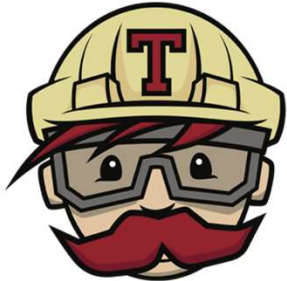
GitHub Actions



AWS  
CodePipeline



**Azure Pipelines**



Travis CI



**GitLab**



circleci



**Bitbucket Pipelines**

# Continuous integration basics

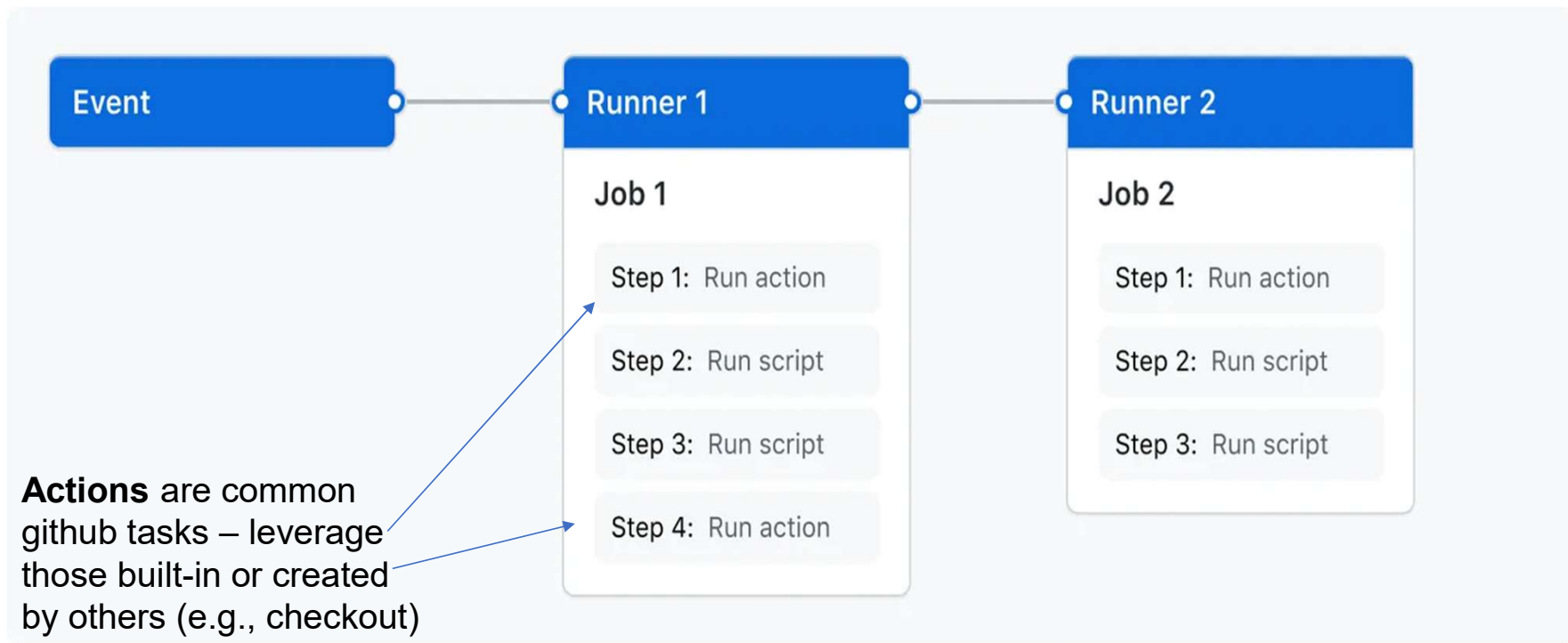
- A CI **workflow** is **triggered** when an **event** occurs in your [shared] repo
  - Example events
    - Push
    - Pull request
    - Issue creation
- A workflow contains **jobs** that run in a defined order
  - A job is like a shell-script and can have multiple steps
  - Jobs run in their own vm/container 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 is this using?



# 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 -m pip install hatch
      - name: Pre-fly setup
        run: cp $GITHUB...GITHUB_ENV
      - name: Test with hatch
        run: |
          hatch run test:test
```

Workflow name

Trigger

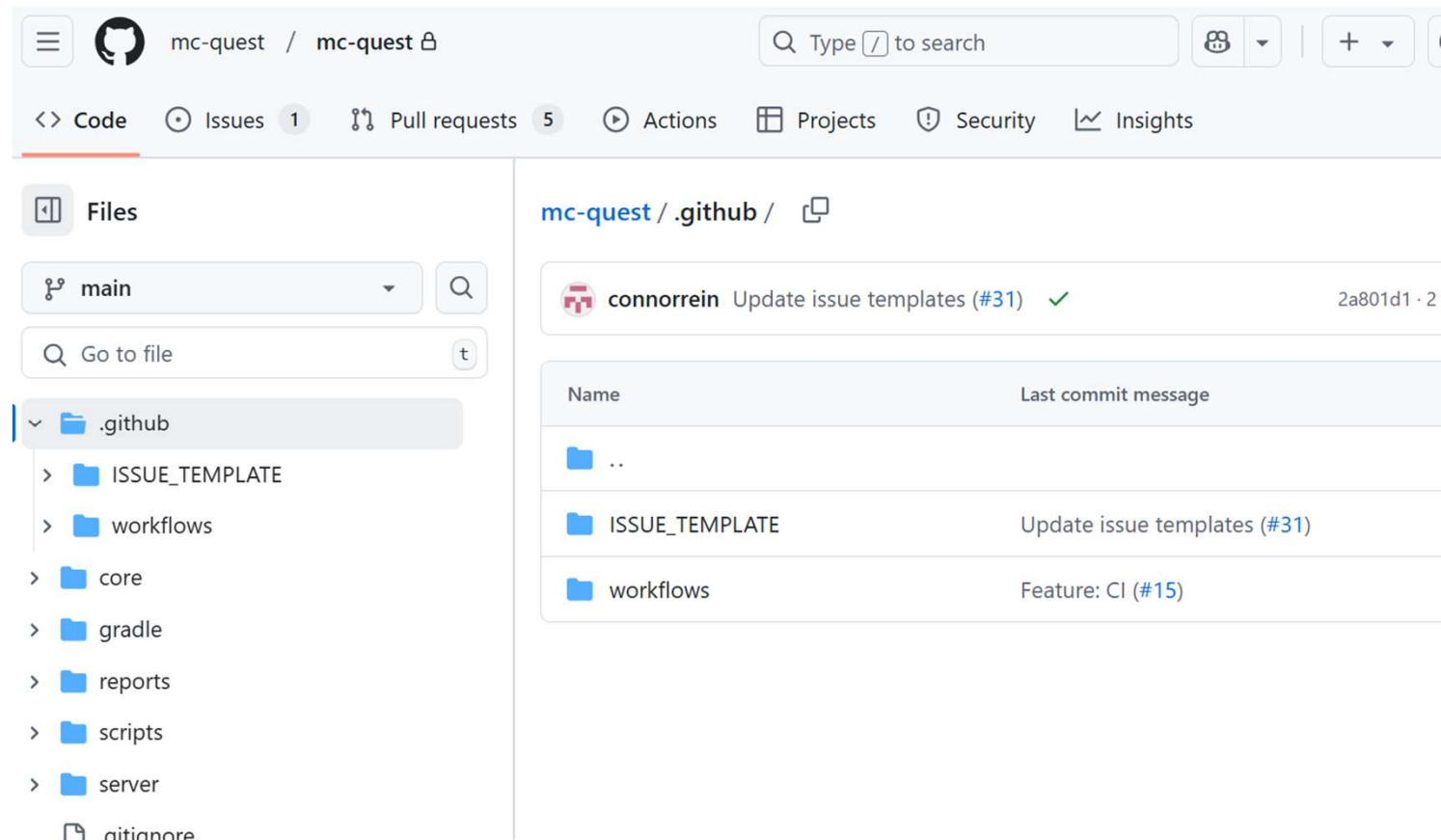
Linux OS environment

Code reuse with established "actions"

One command to run test suite

# Let's look at a CI workflow from the MC-Quest CSE 403 project

Connor's  
team's  
repo



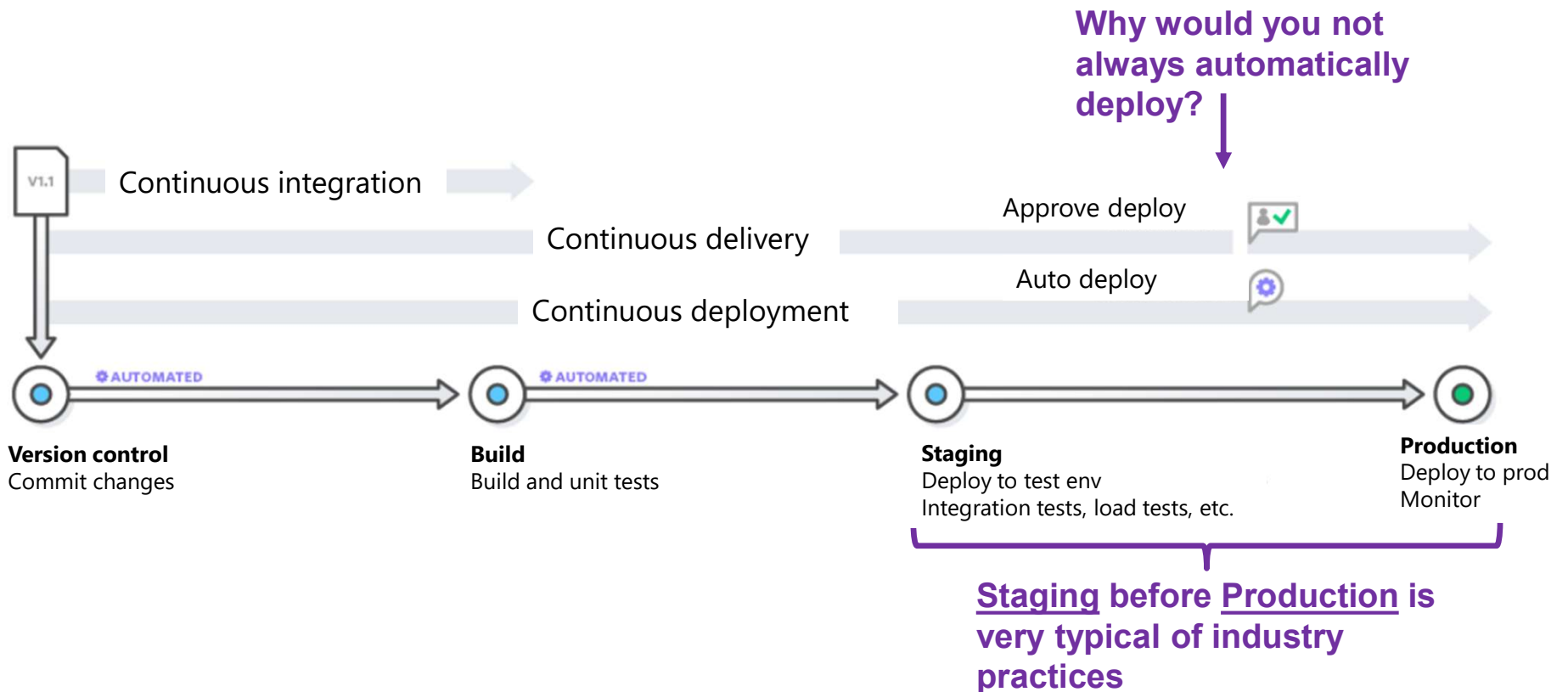
The screenshot shows the GitHub interface for the 'mc-quest' repository. The top navigation bar includes 'Code', 'Issues 1', 'Pull requests 5', 'Actions', 'Projects', 'Security', and 'Insights'. The left sidebar shows the file explorer with the following structure:

- Files
  - main
  - Go to file
  - .github
    - ISSUE\_TEMPLATE
    - workflows
  - core
  - gradle
  - reports
  - scripts
  - server
  - nitianore

The main content area displays the CI workflow section for the '.github' directory. It shows a commit by 'connorrein' titled 'Update issue templates (#31)' with a green checkmark and commit hash '2a801d1 · 2'. Below this is a table with the following data:

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

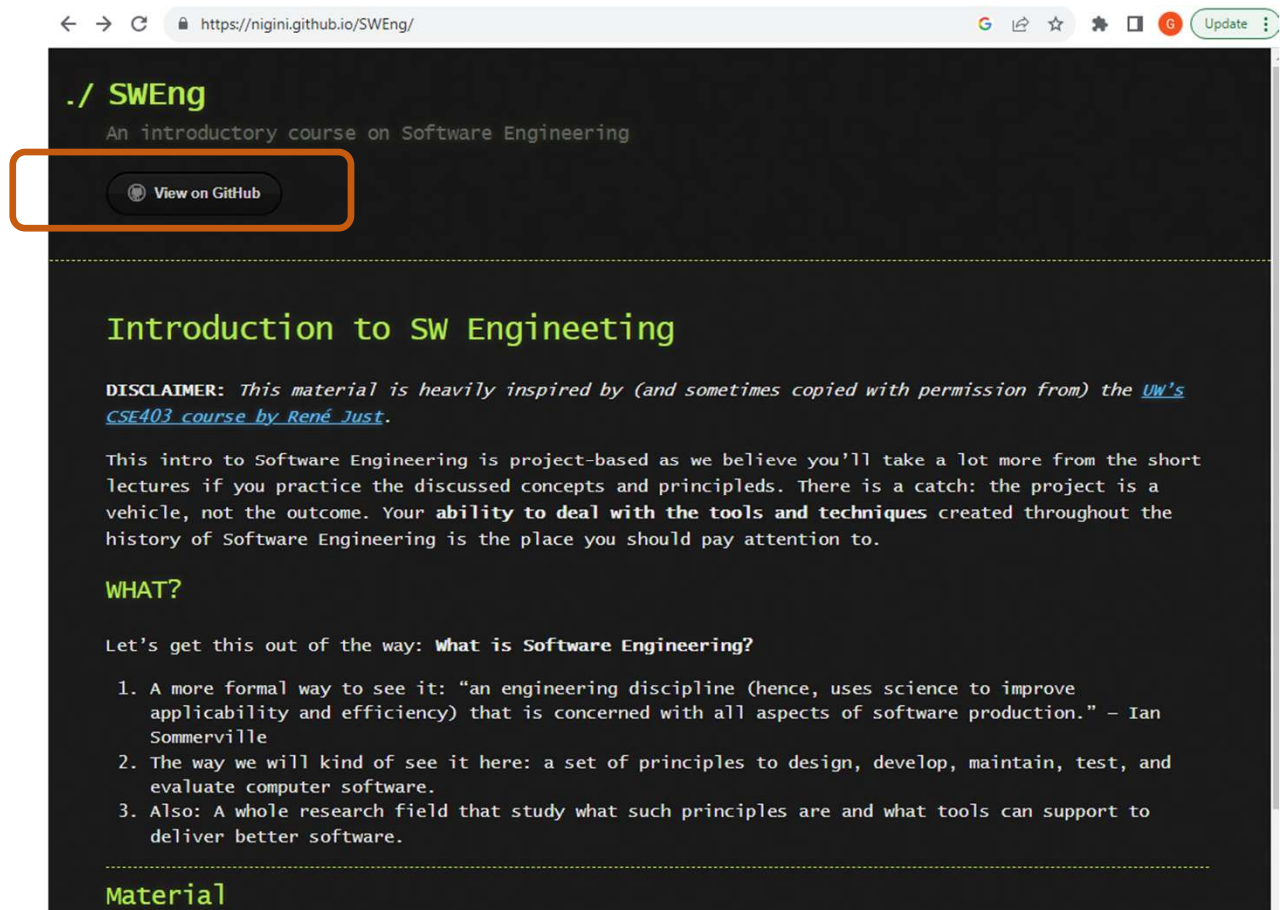
# Continuous delivery/deployment basics



Amazon example

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

Content updates trigger publishing the website update





# Example: continuous deployment config

The screenshot shows the GitHub repository settings for 'nigini / SWEng' (Public). The 'Settings' tab is selected in the top navigation bar. The left sidebar contains a list of settings categories: General, Access (Collaborators, Moderation options), Code and automation (Branches, Tags, Rules (Beta), Actions), Webhooks, Environments, Codespaces, and Pages. The 'Pages' category is highlighted. The main content area is titled 'GitHub Pages' and displays the following information:

- GitHub Pages** is designed to host your personal, organization, or project pages for free.
- 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.](#)
- Branch selection: `main` (dropdown), `/ (root)` (dropdown), `Save` (button)

Learn how to [add a Jekyll theme](#) to your site.

# Example: continuous deployment config

The screenshot shows a GitHub repository page for 'nigini / SWEng'. The 'Actions' tab is selected, displaying a workflow named 'pages build and deployment #52'. The workflow is triggered via dynamic and has a status of 'Success' with a total duration of 52s and 1 artifact. The workflow steps are: 'build' (24s), 'report-build-status' (2s), and 'deploy' (7s). The 'deploy' step includes a URL: <https://nigini.github.io/SWEng/>.

**Repository:** nigini / SWEng (Public)

**Navigation:** Code, Issues (4), Pull requests (9), **Actions**, Projects, Wiki, Security, Insights, Settings

**Workflow:** pages build and deployment #52

**Summary:**

- Jobs
  - build
  - report-build-status
  - deploy
- Run details
- Usage

Triggered via	Status	Total duration	Artifacts
dynamic 2 days ago	Success	52s	1

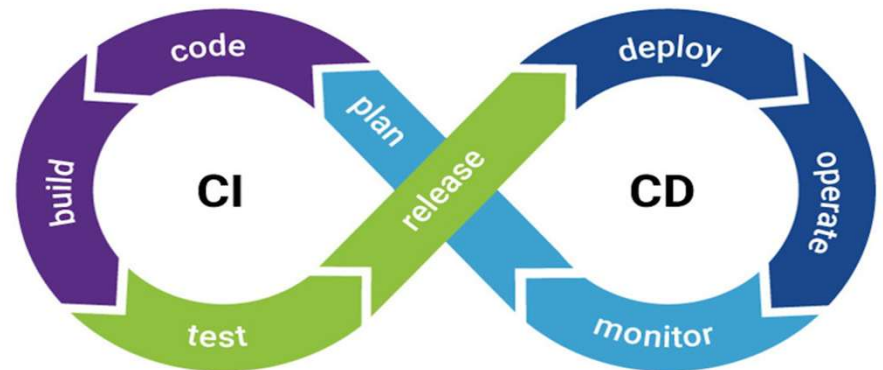
**pages-build-deployment**  
on: dynamic

```
graph LR; build[build 24s] --> report-build-status[report-build-status 2s]; build --> deploy[deploy 7s];
```

<https://nigini.github.io/SWEng/>

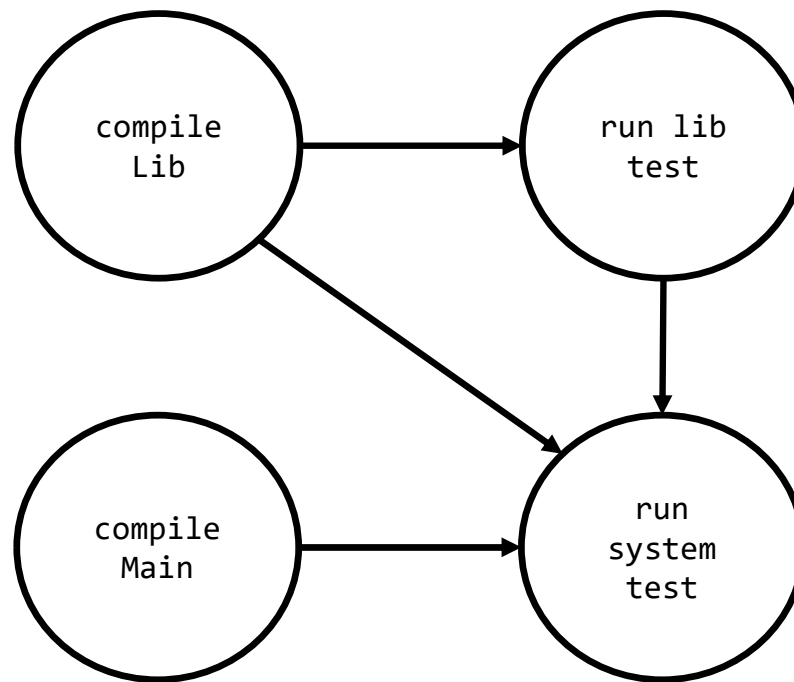
# Summary of best practices for build and continuous integration

- 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!



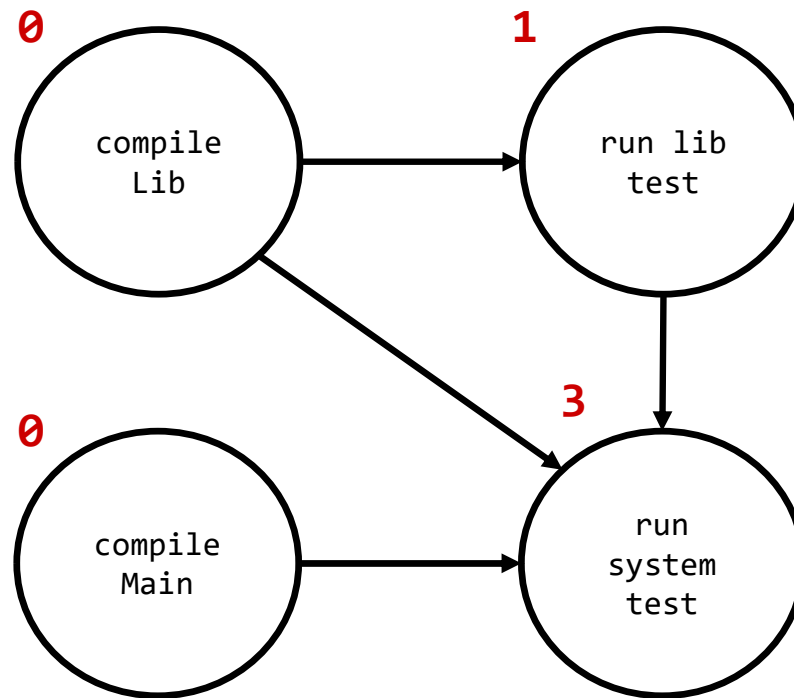
# Appendix - Topological sort example

## Build systems: topological sort

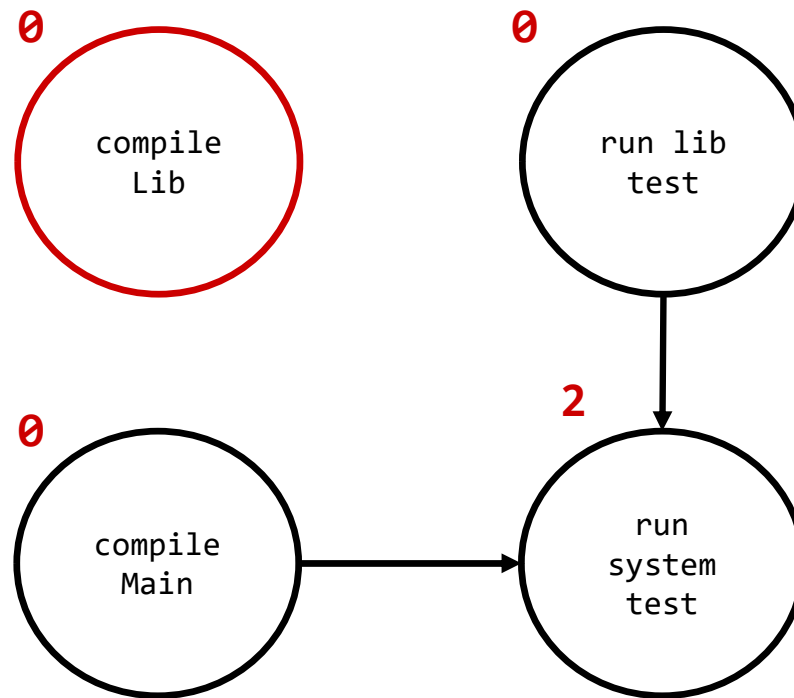


What's the indegree of each node?

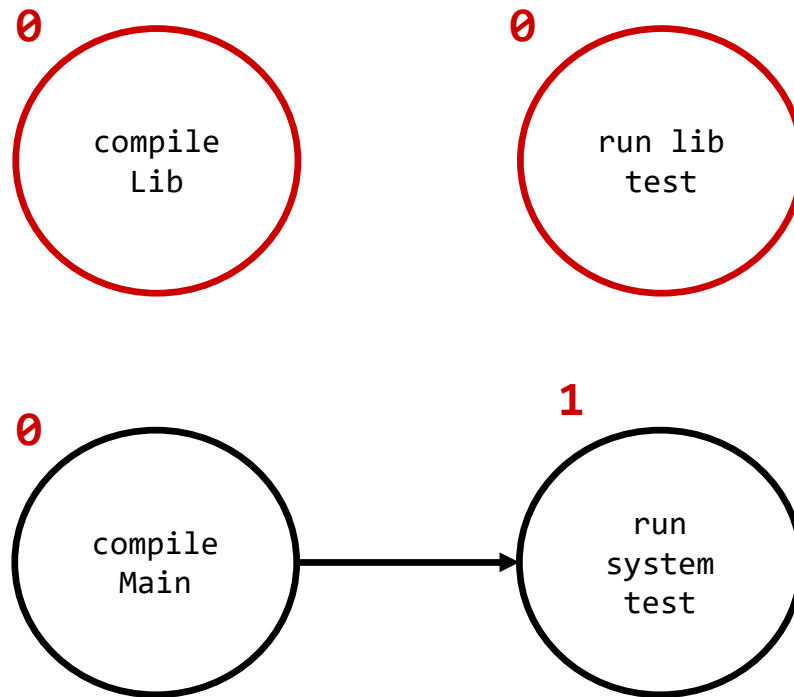
## Build systems: topological sort



## Build systems: topological sort

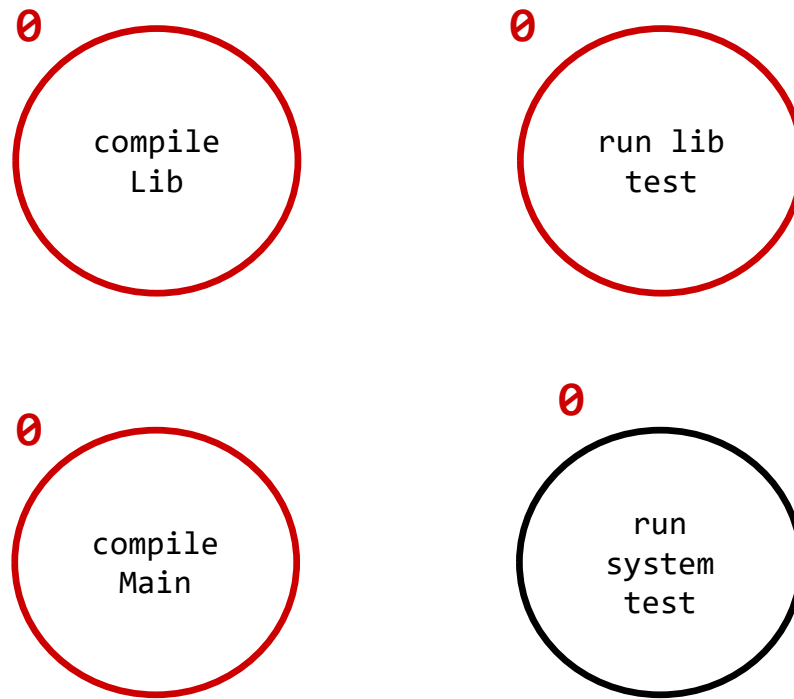


## Build systems: topological sort

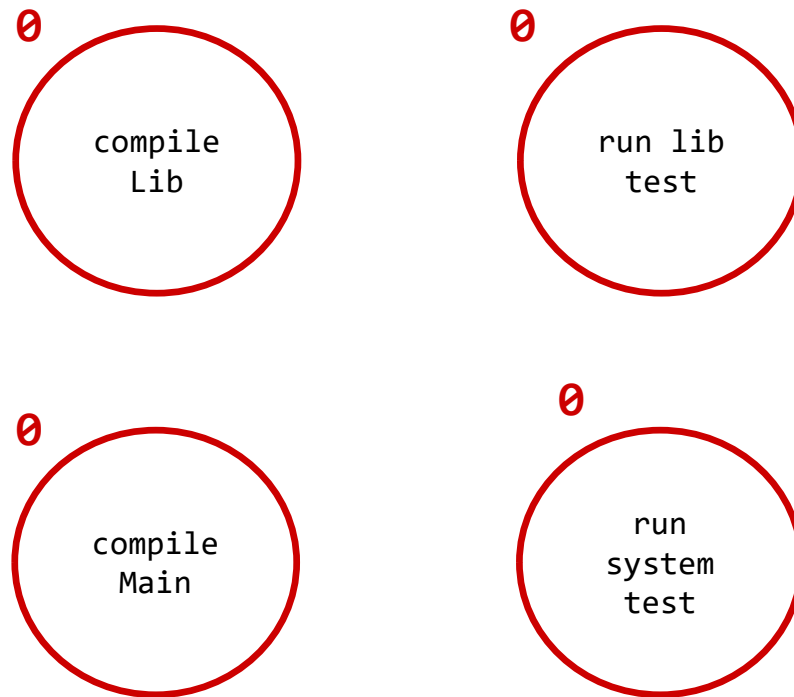




## Build systems: topological sort



## Build systems: topological sort



## Build systems: topological sort

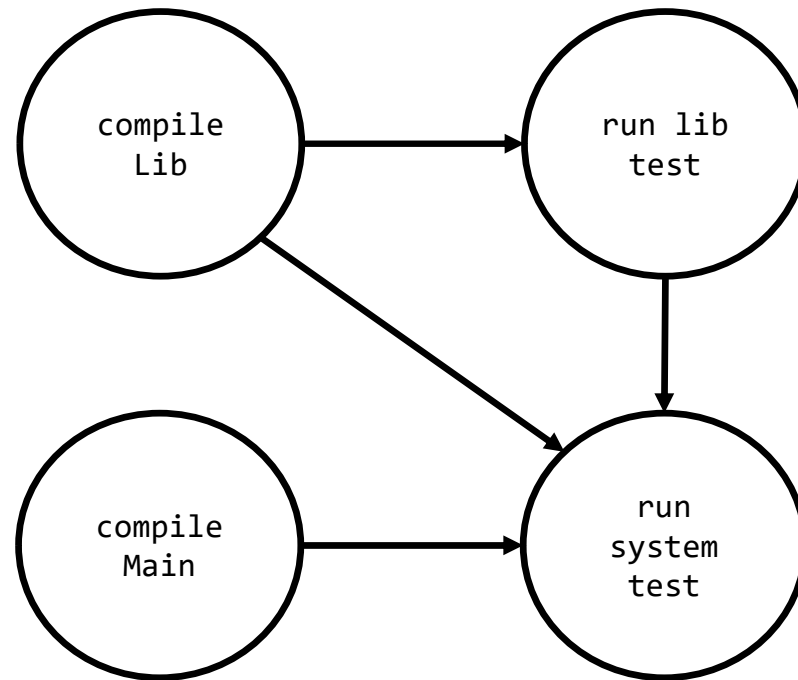
Valid sorts:

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:

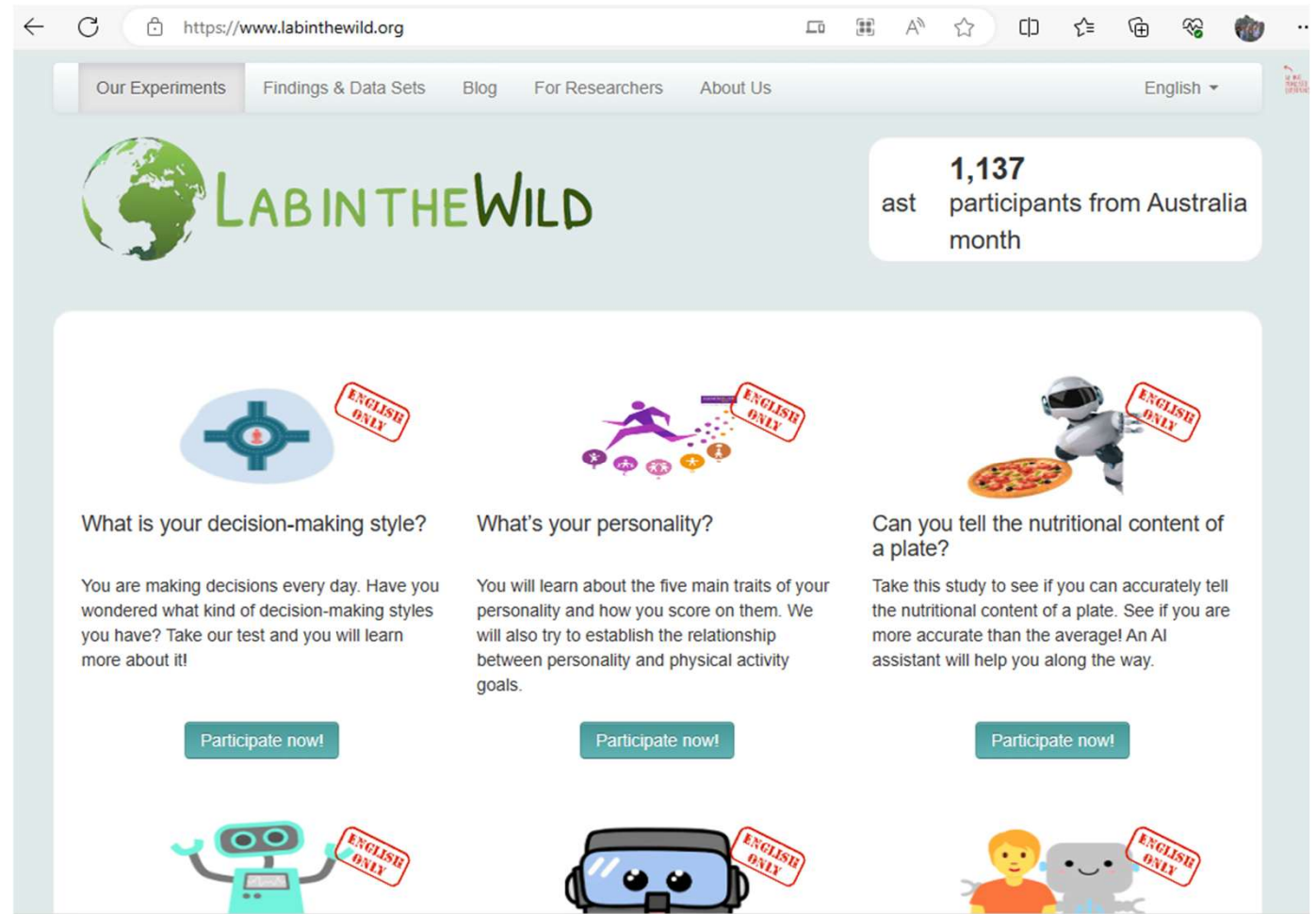
<https://docs.github.com/en/actions/learn-github-actions/understanding-github-actions>

# Example: CI at work in CSE

## [Lab In The Wild](https://www.labinthewild.org)

is a research project drawing survey input from diverse community

– Nigini Oliveira  
UW researcher  
provided this  
example



The screenshot shows the website <https://www.labinthewild.org>. The navigation bar includes "Our Experiments", "Findings & Data Sets", "Blog", "For Researchers", and "About Us". The language is set to "English".

The main header features the "LAB IN THE WILD" logo with a globe icon. A statistics box on the right indicates "1,137 participants from Australia month".

Three experiment cards are displayed:

- What is your decision-making style?**: Includes a crosshair icon and a description: "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?**: Includes an icon of a person running and a description: "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?**: Includes an icon of a robot with a pizza and a description: "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."

Each card has a "Participate now!" button and a red "ENGLISH ONLY" stamp. At the bottom, there are three more icons: a robot, a car, and a person with a robot, each with a red "ENGLISH ONLY" stamp.

# Example: CI with Github actions

The screenshot shows the GitHub Actions interface for a repository named 'labinthewild / LITW-API'. The 'Actions' tab is selected, showing a workflow run titled 'CI Tests run only on push for now. PL + Push was duplicating runs. #15'. The run is in a 'Success' state and was triggered by a push from user 'nigini' 1 minute ago. The workflow file 'ci-test.yml' is shown, with a matrix job 'test' that has completed successfully. The interface includes a search bar, navigation links for Pull requests, Issues, Codespaces, Marketplace, and Explore, and a sidebar with links to Code, Issues, Pull requests, Actions, Projects, Security, Insights, and Settings.

labinthewild / LITW-API Private

Code Issues 3 Pull requests 1 Actions Projects 1 Security Insights Settings

CI - UnitTesting

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

Summary

Jobs

- test (3.11, 6.0)

Run details

- Usage
- Workflow file

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

Show all jobs