Big Ideas in Software Engineering (So Far)

	Big Idea / Theory	Practice / Tools
1.	Programming v. software engineering	Considering trade-offs; making principled engineering choices
2.	Version control & branch management	git, WSL, working on the command line
3.	Everything as code, virtualization / containerization	Docker
4.	Package & dependency management	apt, npm, poetry
5.	Testing	Small, medium, and large tests, Beyonce rule, etc.
6.	Static analysis	Linters, formatters, import sorters, etc.
7.	Continuous Integration	GitHub Actions, etc.
8.	Organizing work across a team	Pull requests, logical units of work.

Instructions

Please fill out this cheat sheet by addressing the topic associated with your group. Specifically, you will:

1. Come up with 2 exam questions (and answers) for your topic:
    - 1 question should be about theory and the other should be about tools / practical decision-making.
    - Feel free to draw from the course materials, ChatGPT, the Internet, etc.
2. Write down your questions and the answers in this Google Doc / cheat sheet under the appropriate topic.
3. Be prepared to share your questions / answers with the rest of the group in 15 minutes.
    - This is low pressure – it's just to get you thinking about what the exam might look like, and also to see what you remember / don't remember from the first half of the course.

Cheat Sheet

1. Programming v. Software Engineering

- Your questions / answers here (1 theory question and 1 practical question).

[Skip for now]

2. Version Control & Branch Management

- *Your questions / answers here (1 theory question and 1 practical question).*

What is a branch in version control and why do we use it?

Answer: A branch is a separate line of development that allows us to work on new features or fixes of a project without actually affecting the main code. Once we finish working and testing, the branch can be merged back into the main project.

What git command would you use to create and switch to a new branch called group3?

Answer: git checkout -b group3

- Does it matter what branch you're currently on when you make a new branch? Why or why not?

3. Everything as Code, Virtualization / Containerization

- Your questions / answers here (1 theory question and 1 practical question).

Practical question -

What are some key differences between virtualization and containerization, and in what scenarios would one be better than the other?

Answer:

- Virtualization - Abstracts the physical hardware to create multiple virtual machines. Each vm has its own operating system that runs on top of a hypervisor.
- Containerization - Uses the OS' kernel features to run multiple isolated environments on the same OS that share the same kernel.

- In resource limited scenarios, containerization is much less resource intensive since multiple full OS don't have to be created. Containerization starts up faster and is much more portable. VM's are great for legacy applications that cannot be ran on newer OS.

Theory question-

What are 2 concrete examples of how infrastructure can be everything as code?

Configuration as Code - System configurations are written in code using tools such as Ansible

Policy as Code - Security and compliance rule are expressed in code using tools like Open Policy Agent or HashiCorp Sentinel.

4. Package & Dependency Management

- Your questions / answers here (1 theory question and 1 practical question).
- How do package managers work and why are they useful? (Ex. apt, npm, brew, poetry)
    - Package managers work because they have a list of trusted programs that you can download directly from the terminal. They are useful because you do not have to download programs through websites that may not be trustworthy. It is also convenient to download things on the terminal. It also automatically manages your dependencies.
- For your machine, how would you install a program called curl from the terminal using your package manager?
    - Linux(Debian-based)/WSL: sudo apt-get install curl
    - MacOS: brew install curl
-

5. Testing : Your questions / answers here (1 theory question and 1 practical question).

Theory Question:

1. What is the Beyoncé Rule in software testing, and how does it relate to test size categories (small, medium, large)?

- "If you like it then you should've put a test on it". Test one small piece to isolate functionality.

  Small tests: small tests are fast, isolated unit tests; testing a single function.

  Medium tests: medium tests involve multiple components or services to test how few pieces work together.

  Large tests: large tests are full integration or end-to-end tests that simulate real behavior and environments

Practical Question:

1. Which command runs all Mocha tests in a Node.js project?

a) python -m pytest

b) npm start

c) npx mocha (WINNER! WINNER! CHICKEN DINNER)

d) mocha test.js run

2. How do you run your test suite with Poetry once pytest is installed?

a) pytest

b) poetry run pytest (WINNER! WINNER!)

c) python pytest.py

d) poetry test

6. Static Analysis

- What are some of the benefits of doing static analysis?
    - It increases consistency, can make your code more secure (by checking for depreciated dependencies), keeps your code modern/up-to-date, increase code readability, and it can save time by ensuring only effective code is submitted/pushed. And you can catch mistakes earlier
- What are the tradeoffs associated with static vs dynamic analysis?
    Static analysis does not execute the code, can be faster and lower risk than dynamic. Dynamic may catch issues at runtime not found by static analysis.
- What does effective static analysis look like? (Scalability, usability)
    Scalability: address scaling to produce results in a timely manner without slowing down the software development process.
    Usability: consider cost benefit tradeoffs for static analysis tool users
- What kinds of things does static analysis check for?
    - That your code follows style conventions and that you don't have any depreciated dependencies. It will validate type hints, and may check for possible errors or inefficiencies (linters do this, auto formatters don't do this).
- What makes static analysis *ineffective*?
    - When the feedback from the static analysis isn't helpful (causing frustration when trying to fix), or when there are false positives/false negatives. When linters and formatters are not consistent across a workflow or team it can cause a cycle of issues and frustration.
    - Definition of a false negative - Something slips by that should've been flagged.
    - Definition of a false positive - Something is flagged that actually has no issue.

- What tool can we use to format python code?
    black

Does java use or need any static analysis tools? Java uses compiler, so linter is less important but still need to consider formatting yourself

7. Continuous Integration

- Your questions / answers here (1 theory question and 1 practical question).

What are the advantages / disadvantages to using continuous integration (CI) ?
+Weed out bugs early
+Allows for quick feedback
+Greatly increases production speed
-Possibly resource-intensive
-Requires great discipline from the team
-Only as good as your tests

What might a presubmitter check for in a codebase designed for a calculator?

Code syntax

Code grammar using something like flake8

A few tests to check if the outputs are correct / code logic

8. Organizing Work Across A Team

- **Q:** In GitHub, what is the difference between creating a merge request, rebase and merge, and squash and merge?
- **A:** creating a merge commit pushes every commit in the working branch and applies it to the base branch via a merge commit.

  Rebase and merge rebases all of the commits in the working branch before being added to the main branch.

  Squash and merge squishes all of the commits in the working branch into a single commit before committing to the base branch.
- **Q:** Why should you NOT work on the main branch when utilizing version control?
- **A:** It is poor practice to work on the main branch because it can disrupt workflow for the entire project and introduce unseen errors and merge conflicts to the project that will directly affect the others in your team. It can force a whole project to backtrack to the last known functional version of main, which is a pain in the neck to deal with.