COMP423 - Lecture 1

Please seat at tables with at least 3 people, ideally 4!
Only use Zones ABCD.

Kris Jordan / August 21, 2024

Class Start Checklist

1. Sit at a table with 3 or 4 people
2. Place backpack in bag under your seat
3. Write names of your group in corresponding corners of a whiteboard and place whiteboard in back brackets of table with names facing front of room
4. Have laptops put away to start class (when/if we need them, we'll get them out). iPads/notebooks are fine!

What are the CS Experience Labs?

- Community Co-Lab for Coworking (Solo, Pairs, and Groups)
- Productivity Rooms for Office Hours and Groups
  - 3x Pairing Rooms - Two Seats & Monitor
  - 2x Small Team Rooms - Five Seats
  - 2x Large Team Rooms - Seven Seats
- Collaborate with student orgs & CS Careers
- Upcoming: Workshops, Community Nights
- csxl.unc.edu > Coworking!

Virtues of a Great Teammate

Think, Team, Share

- 1m - Think for one minute individually: what are the virtues of a great teammate?

- 4m - On a whiteboard, as a table, share and record VIRTUES of great teammates
  - Don't filter, prune, or judge! This is brainstorming!
  - Give everyone an opportunity to contribute.

- 3m - On a separate whiteboard, identify the top 5 most important virtues to your table.

Classwork Submission

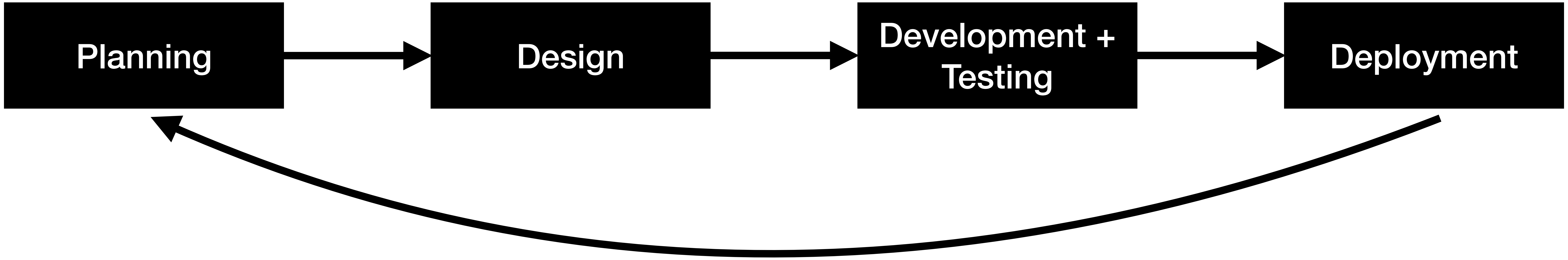
- Select ONE table member to be the submitter on Gradescope
- They should take a selfie/group photo of your table holding up Top 5 Virtues
- Submit the assignment on Gradescope
- After submitting, add everyone else to the assignment on Gradescope
- Everyone else: confirm you can see the submission on Gradescope

Tools and Engineering

“We become what we behold. We shape our tools and then our tools shape us.” -Marshall McLuhan

- Learning to wield (and *shape)* tools as a Software Engineer is paramount

- A **Toolchain** is the set of tools used in the sequence of designing, writing, building, testing, and deploying code throughout the **Software Development Lifecycle (SDLC)**



Tools for Software Engineering

- Operating System
- Command-line Interface
    - Controls processes, file system
    - Operates many tools
- Source Code Control (e.g. git)
- Integrated Development Environment (IDE) - e.g. VSCode
    - Program text editor with syntax highlighting
    - Language Server Protocol (LSP) system for checking types, code nav, etc.
    - Debugger
- Programming Language Platforms
    - Compiler / Transpiler (e.g. javac, tsc)
    - Language Runtime System (e.g. java virtual matchine / node's v8)
    - Test Systems and Frameworks
- Project Management Software

Group Exercise

- On your machines, open up a Terminal (or PowerShell / cmd.exe on Windows)
- Write down each team mate's versions (or "Error") when running:
  - git --version
  - python3 --version
  - node --version

Key Team Engineering Challenge: Environment Consistency

Combatting "But it Works on My Machine"

- ~ Pre-2000 - On-boarding involved installing environment directly to engineer machines (ideally scripted and automated, but often manual steps)
  - Real challenges in keeping everyone's machines on the same page
- ~ 2000 - 2015 - Virtual Machines
  - Run complete operating systems virtually. Heavier weight, slower. Mutable state of VMs still leads to challenges in developer consistency over time.
- ~ 2015 - 2020 - Emergence of Docker Containers in Development Flows
  - Containers and Images offer *immutability* and lightweight/fast operations
- 2020 - Dev(elopment)Containers in Microsoft VSCode and Cloud IDEs (e.g. CodeSpaces)
  - Entire IDE running in container for highly consistent environment.
  - Can run entire IDE on-line and in-browser (e.g. GitHub CodeSpaces)

Development Containers are becoming a Standard

https://containers.dev/

- Pre-built containers with great tooling for most popular languages:
  - https://github.com/devcontainers/images/tree/main/src
  - Including: C++, Java, JavaScript, PHP, Python, Ruby, Rust, Go, TypeScript
- Organizations / Engineers can fully customize custom built containers with the tooling they need
- Using this infrastructure, there is a *very* high degree of confidence in consistency across developer machines!

First DevContainer from Scratch

Requires Docker to be working (OK if it isn't! Work with partner.)

1. Be sure **Docker Desktop** is running
2. VSCode > Open Extensions > **Confirm DevContainers by Microsoft is installed**
3. File > Open Folder > New Folder (name it **ts-container**) > Open
4. **New Directory** named '**.devcontainer**'.
    1. Inside directory: New File named 'devcontainer.json'
    2. Save the contents found in the text to the right of this slide!
5. Open Command Palette and run "**Dev Containers: Reopen in Container**" (This will take a few minutes.)
6. Open a new Terminal and try running `**node --version**` and `**git --version**` and compare with your table.

```
{
   "name": "ts-devcontainer-demo",
   "image": "mcr.microsoft.com/vscode/devcontainers/typescript-node"
}
```

Starting a JavaScript REPL from the CLI

- From terminal, run a JavaScript REPL with: **`node`**
  - **`console.log("Hello, world.")`**
  - **`typeof(3)`**
  - **`let f = (x) => x * 2;`**
  - **`f(590)`**
  - **`.exit`**
- The JavaScript Runtime is **node.js** (accessed via command **`node`**)
- The node.js runtime repackages Google's V8 JavaScript Engine for development and server use. In a web browser, V8 is built into the client such as Chrome.

Managing Project Dependencies

- Typically bundled with node.js, the Node Package Manager is primarily a tool for establishing, managing, and updating **3rd party dependencies a.k.a. packages**
- Most (modern) language platforms have a preferred package manager! Streamlines the process of installing extra libraries your project needs.
- Examples of commonly used JavaScript packages:
  - Front-end / Back-end Frameworks
  - Testing Frameworks
  - Linting Tools (Automatic code tidying)
  - 3.1 million (!) open source packages hosted on npmjs.com
- The **npm** CLI Program also has support for *starting projects*, *running project tasks (start program, run tests, build for production, etc.), security checks, and more:* **`npm help`**

Starting a JavaScript Project with npm

1. To start a new project, in the DevContainer terminal: **`npm init`**

    - There are a few questions you are asked, read each but just press [Enter] until you are brought back to the command-line prompt starting with `node`

2. A file produced is called **package.json** (JSON: JavaScript Object Notation)

    - The file encodes an "anonymous object" that specifies project properties

3. In package.json, we'll use a more modern module system by adding the following line after the "version" line (don't forget : **"type": "module",**

4. In the "scripts" object, add a line (don't forget the ,) **"start": "node index.js",**

5. Try creating a file named **index.js** with contents **console.log("Hello, world!");**

6. Save and then in the terminal run: **npm start**

Adding a 3rd Party Package with npm

1. Let's add the popular library **chalk** to our project! This library makes it simple to use color coding in terminal program output!

2. Run the command: **npm install --save chalk**

    1. The **--save** long argument *saves* the dependency in package.json (go look!)

    2. Notice in dependencies, you see: **"chalk": "^5.3.0"**

    3. This also leads to a file named **package-lock.json** being created with more concrete details about the *exact* package(s) installed (and their dependencies!).

    4. Both of these files tend to be committed to project repositories.

3. Now you can use the chalk library, update your **index.js**

    ```
    import chalk from 'chalk';

    console.log(chalk.yellowBright("Hello, world"));
    ```

4. Do you remember how to run your program?