Lab 4: September 16, 2026

Project Planning in Practice + Getting an Interview in 2026

Agenda

- Mentor Meeting Expectations
- Planning in Practice
- Lab Planning Activities
- Job Search in 2026: Getting the Interview

Mentor Meeting Expectations

- Meet regularly with your mentor (weekly recommended; at least bi-weekly + sprint planning required).
- For the first meeting: introduce the team, share your project proposal, and agree on a recurring time.
- Use your GitHub Project board to drive the discussion: what is done, next, blocked, and risky.
- All team members should attend, participate, and be ready to share progress.
- Ask for technical feedback early—especially on scope, architecture, dependencies, and testing.

Planning in Practice: What Matters Now

You have a project direction. Now turn it into work the team can actually execute.

Define what “success this semester” means.

Break large ideas into small, testable tasks.

Put tasks on GitHub Projects and update the board every week.

Use instructor + mentor meetings to adjust scope early—not in December.

Fall 2026: Simple Roadmap

Use the course schedule, but plan around a few major checkpoints

NOW → OCT. 6

Scope + plan
First sprint
Demo -1

OCTOBER

Design + build
Presentation 1
Writing 2

NOVEMBER

Individual progress
Presentation 2
Writing 3 + alpha prep

DECEMBER

Presentation 3: Dec. 2
Demo 1: Dec. 8
Writing 4: Dec. 13

Course schedule: https://gw-cs-sd-26-27.github.io/ | Dates Dec. 2 / Dec. 8 / Dec. 13 updated for this lab

The Three Finals Are Closer Than They Look

DEC. 2

Presentation 3

Can you clearly explain what you built and why?

DEC. 8

Demo 1

Does the alpha project work end-to-end?

DEC. 13

Writing 4

Is the final design documented and defensible?

Treat them as one connected deadline: presentation + working alpha + documented design.

Plan Backwards from the Alpha Demo

1. **DEC. 8** What must work in the alpha demo?
2. **LATE NOV.** What must be integrated and testable?
3. **EARLY NOV.** What components must already exist?
4. **OCTOBER** What is the smallest end-to-end path?
5. **THIS WEEK** What can each person finish in 7 days?

Simple Planning Framework

GIST idea, simplified for your GitHub board

GOAL
Semester outcome

MILESTONE
Visible checkpoint

STEP
Chunk of work

TASK
Smallest assignable item

For day-to-day work, spend most of your time at the Task level.

Example: Safe Walking App

Reusing last year's example, but breaking it into smaller work

Goal

Demonstrate a safer walking-route prototype

Milestones

- Map input works
- Safety score works
- Route combines map + score
- End-to-end demo works

Example tasks

- Get API key + render map
- Choose a safety-data source
- Implement scoring function
- Write unit tests for scoring
- Connect route result to UI
- Test one complete user flow

What Makes a Good Task?

- Starts with an action verb
- Has one owner
- Has a target/due date
- Has clear acceptance criteria
- Notes dependencies or blockers

Too big

“Build the backend”

Better

“Add /route endpoint returning a route for two coordinates”

Scope: Must / Should / Could

MUST

Required for the Dec. 8 alpha demo

SHOULD

Improves quality if time allows

COULD

Nice-to-have; cut first if needed

If everything is “must-have,” nothing is prioritized.

Dependencies and Risk: Move Unknowns Earlier

What must happen before this task can start?

Do you depend on an API, dataset, hardware device, account, or external approval?

Does one teammate’s work block another teammate?

Which technical assumption is still unproven?

What is the fallback if the risky approach fails?

GitHub Projects = Your Source of Truth

The syllabus explicitly uses the board to track task progress

- Every active task should be visible on the board.
- Use clear status: To Do → In Progress → Review/Test → Done.
- Each card should have an owner, target date, and acceptance criteria.
- Update the board before instructor and mentor meetings.
- If the board is stale, your mentor cannot see where the team needs help.

Course syllabus: https://gw-cs-sd-26-27.github.io/syllabus/

Monthly Sprint Planning: Keep It Simple

Choose one milestone the team wants to reach in the sprint.

Pull the highest-priority tasks from the backlog.

Assign owners and identify dependencies.

Leave some buffer for bugs, integration, and surprises.

Revisit the plan every week with your instructor and mentor.

Planning is a hypothesis—you are allowed to revise it when you learn more.

Lab Activity 1 — Define Fall Success

10 minutes • Work as a team

- Write one sentence: “By Dec. 8, our alpha demo will allow a user to …”
- List the 3 most important things that must work.
- Move nice-to-have features into “Should” or “Could.”

By the end

1 fall goal +
3 must-have outcomes

Lab Activity 2 — Work Backwards from Dec. 8

10 minutes • Work as a team

- Create 3–5 milestones between now and the alpha demo.
- Include at least one integration milestone and one testing milestone.
- Give each milestone a target week—not just “sometime in November.”

By the end

A simple milestone timeline

Lab Activity 3 — Break One Milestone into Tasks

15 minutes • Work as a team

- Choose the next milestone.
- Create 8–12 tasks that can usually be finished in less than one week.
- Give every task an owner, target date, and acceptance criteria.
- Mark dependencies and unknowns.

By the end

A task list the team can start tomorrow

Lab Activity 4 — Update Your GitHub Project

15–20 minutes • Work as a team

- Create or update the cards on your GitHub Project board.
- Choose the tasks for the next two weeks.
- Mark blockers and risky tasks clearly.
- Share the board during the mentor/instructor discussion.

By the end

An updated board that is usable in the next meeting

Job Search in 2026

Getting the interview is the first problem to solve

2026 Job Market: Why Getting an Interview Is Harder

+1.6%

Projected new-grad hiring vs. Class of 2025 (NACE: nearly flat)

−7.5%

Entry-level postings year-over-year as of May 2026 (Indeed)

~−30%

Software development postings vs. Feb. 2020 baseline (June 2026, Indeed)

~70%

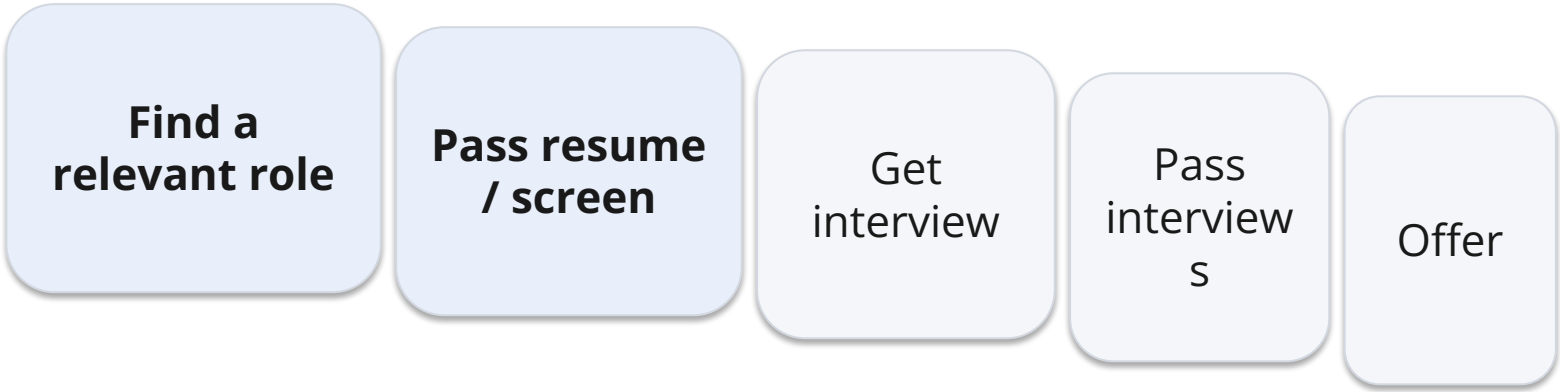
Employers using skills-based hiring for entry-level recruiting (NACE)

The market is not closed—it is more selective. Your first challenge is getting through screening.

Sources: NACE Job Outlook 2026; Indeed Hiring Lab, July 2026

Think Like a Funnel

Today’s focus is the top of the funnel



If you are not getting interviews, do not only practice LeetCode—improve targeting, resume evidence, visibility, and networking.

1. Search Broader, Not Randomly

Do not search only for “New Grad Software Engineer.”

Also consider: Software Engineer I, Developer, QA/Automation, Data/ML, Security, Cloud/DevOps, Systems, and Solutions roles.

Look beyond large tech companies; software hiring is spread across many industries.

Use location flexibility when you realistically can.

Read the requirements, but do not self-reject because you miss every “preferred” skill.

Context: LinkedIn U.S. Software Engineer Talent Landscape (Feb. 2026)

2. Resume: Make Screening Easy

For most new grads, keep it to one clear page.

Put the role-relevant technical skills where a recruiter or screening system can find them quickly.

Use project bullets that show action + technology + evidence—not a list of course names.

Tailor the top third of the resume to the job description.

List only skills and experiences you can explain in an interview.

2026 trend: skills-based hiring is used by about 70% of employers in NACE's survey

Use Senior Design as Proof of Work

Your project should show:

- The problem + user
- Your individual contribution
- Technical decisions + tradeoffs
- A working demo or tested feature
- Evidence: users, tests, latency, accuracy, scale, etc.

Resume bullet pattern

Built [feature] for [user/problem] using [technology], resulting in [measured evidence].

Do not invent a metric. Use real evidence from your project.

3. Make Your GitHub Easy to Evaluate

Pin your best 2–4 projects.

For each important repo, add a short README: problem, demo, stack, setup, and architecture.

Link a working demo or short video when possible.

Keep secrets and API keys out of the repository.

For team projects, be ready to explain exactly what you personally built.

4. Network Before You Need a Referral

Use mentors, alumni, career fairs, and short informational conversations to learn about roles and teams.

In a first conversation, ask for advice and feedback—not “Can you refer me?”

If you later find a specific role that fits, follow up with a concise message.

Your project mentor is first a project mentor: build a professional relationship before asking for career help.

GW Career Services networking guidance: https://careerservices.gwu.edu/networking-strategy

5. Apply Early + Track It

Set job alerts and check new postings regularly.

Apply soon when the role is a reasonable match; many entry-level positions close quickly.

Track: company, role, date, status, contact, and follow-up.

Mix large tech firms with smaller companies and non-tech employers that hire software talent.

If you submit many applications but get no screens, revise targeting and resume evidence—not only interview prep.

Where to Find Roles

GW Handshake — jobs and internships specifically available to GW students and alumni.

SimplifyJobs New-Grad Positions — frequently updated list of entry-level tech roles.

LinkedIn Jobs + employer career pages — use alerts and apply at the source when possible.

GW Career Services / career coach — resume review, search strategy, and interview support.

GW Handshake: https://careerservices.gwu.edu/handshake | SimplifyJobs: https://github.com/SimplifyJobs/New-Grad-Positions

Use AI Carefully in the Job Search

Good uses: compare your resume to a job description, brainstorm keywords, revise bullets, and practice recruiter questions.

Do not fabricate experience, metrics, or technical skills.

Avoid sending hundreds of nearly identical AI-written applications.

Know every line on your resume and every major decision in your projects.

Skills-based hiring means you may be asked to demonstrate what you can actually do.

NACE Job Outlook 2026; LinkedIn SWE Talent Landscape 2026

Once You Get the Interview

A common path: recruiter screen → coding/assessment → technical + behavioral rounds → final/team discussion.

The exact process varies by company and role.

Follow the company's instructions about AI tools during assessments—do not assume they are allowed.

Review your own resume and Senior Design project deeply; expect questions about your decisions and tradeoffs.

Technical Interview Prep: Keep It Focused

Refresh the data structures and algorithms most relevant to your target roles.

Practice explaining your approach before you code.

Write clean code and test edge cases.

Practice behavioral stories from teamwork, debugging, design choices, and project setbacks.

Use mock interviews after you begin getting screens so practice matches the real pipeline.

Helpful Resources

A short list—use these rather than collecting 50 job-search links

GW Handshake

Jobs + internships posted for GW students/alumni

NACE Job Outlook 2026

Current new-grad hiring + skills-based hiring context

GW Networking Strategy

How to run informational interviews and build relationships

Tech Interview Handbook

Free coding / behavioral / interview preparation

SimplifyJobs New-Grad Positions

Frequently updated list of entry-level tech roles

Course schedule + syllabus

Use course milestones to build portfolio evidence early

Market context used in this lab: NACE, Indeed Hiring Lab, LinkedIn Economic Graph (2026)