CS 490-001 (Au25)

Syllabus

Welcome to CS 490: Guided Design in Software Engineering! This is a course about building software that satisfies stronger properties than "it works": it's about building software that is correct, reliable, maintainable, testable, and usable. That is, this course will teach you about how to build software *well*.

The course is structured around giving you experience in both the *theory* (through readings and lectures) and the *practice* (through programming assignments and a large group project) of software engineering. However, this course has a strong emphasis on the practical: the assignments mirror real engineering activities, most readings are blog posts from engineers, etc. My overarching goal in this class is to expose you to how high-quality software engineering is done at the world's best software engineering firms.

Course Outcomes

Official course outcomes:

- Students will be able to explain the major theories and methods applicable to professional software engineering.
- Students will be able to design, implement and evaluate a computer-based system to meet desired needs.
- Students will be able to function effectively on a team to accomplish a goal.
- Students will be able to use current techniques, skills and tools necessary for computing practice.

My course design is based additionally on these unofficial outcomes:

- Students will be able to assess the quality of software engineering being done at some future workplace
- Students will be competent software engineers that I wouldn't be worried about hiring as an engineering manager

Prerequisites

Technically this course requires CS 280 and CS 288, which are both courses about *how* to program. I will assume in this course that you know how to program: that is, that if I tell you to go write some code, you'll be able to go do it. Since this course focuses on how to program **well** (i.e., how to engineer software!), you first need to know how to program at all. I'll also assume some familiarity with command line tools, debugging, and using a search engine: I expect that if I ask you to go write code in some language you've never seen before, you'll be able to find the necessary components online, find an online tutorial on the syntax, and figure out how to write that code. Put another way, I won't teach you how to write a program: this course already assumes that you can do that.

As an analogy to carpentry, classes like CS 113 teach you how to build the equivalent of a software cabinet. CS 280 and CS 288 teach you how to build something like a software shed: pretty big, but still small enough for one person to do on their own by combining the skills they learned building cabinets. CS 490 is about the equivalent of building a software high-rise: not only is it more than a single-person job, but also there's lots of other things you need to worry about that don't come up when you're building a shed.

The first homework assignment is due right at the drop date, and is intended to let you check that you have the basic skills that will be needed for this course: it requires you to make a trivial change to a big program in a language you've never seen before.

However, software engineering is a broad topic that requires a synthesis of knowledge, and students will benefit from almost all of the rest of the undergraduate curriculum. We will use concepts from most other courses in the curriculum, including but not limited to: CS theory (CS 341), operating systems (CS 332), algorithms (CS 435), data structures (CS 114), security (CS 351), and databases (CS 331). You don't *need* to have taken these courses before you take CS 490, but I'll bring up topics from them when they're relevant to the course, so you'll get more out of CS 490 if you have seen those classes first (or are taking them concurrently).

Topics

- What is Software Engineering?
- Software Engineering process
- Version control
- Code review

- Programming in teams
- Testing, including coverage, continuous integration, test-driven development, mutation testing, and fuzzing
- Requirements and specifications
- Programming languages
- Build systems
- Static analysis
- Debugging
- Architecture and design, including design patterns, microservice design, designing for security, and designing for scale
- Technical debt, refactoring, and maintenance
- DevOps, logging and post-mortems
- Open source software

Grading and Assignments

Your grade is composed of the following sub-scores (in no particular order):

- 15%: Participation & Professionalism
- 15%: Individual Assignments
- 35%: Group Project
- 35%: Exams (15% for the mid-term, 20% for the final)

This class will be curved: when grading, I prefer to use the whole range available rather than scores in a tight range. That is, if an assignment is worth 10 points, I will give grades at all the points between 0 and 10. I will project your raw scores onto the final distribution twice during the semester:

- after the mid-term exam
- shortly before the final exam

You will be notified of your current projected class grade via email at each of these points.

Readings and Reading Responses

Each lecture has two kinds of readings: mandatory and "Your Choice" readings. I expect you to read mandatory readings before coming to class that day, and reading quizzes (see Participation & Professionalism, below) will cover the mandatory readings only. During the semester, you **must** complete **at least two** "Your Choice" readings: one before the mid-term, and another before the final. Most "Your Choice" readings are research papers from the software engineering literature: the idea is that you will do a deeper dive on two topics that interest you. The "Your Choice" readings will be checked on the exams: see the "Your Choice" reading page.

Participation & Professionalism

Your participation & professionalism grade is composed of two scores.

First, your *Professionalism* score is based on the instructors (both the professor and TAs!) impression of how well you participated in class, with deductions for distracting other students and credit for asking and answering questions (either in person or on the course discussion board). Professionalism during the project (especially in interactions with your group's TA mentor) is also a major component.

Second, your *Participation* score is based on reading quizzes (about topics from the mandatory readings only - quizzes will never cover "Your Choice" readings) at the beginning of most lectures. You get half credit on these quizzes just for being there, and half credit for answering the reading questions correctly (the questions are supposed to be easy if you did the reading). For full participation, you need to get at least a score of 70% on all quizzes over the whole semester (this gives you space to e.g., miss a reading quiz because you were sick or have a family emergency - there are no excuses for missing reading quizzes). Put another way, you can miss up to 30% of the reading quiz points and still get full participation points.

These policies are designed to encourage you to come to class. A big part of the goal of this class is to help you develop an intuition for what good software engineering looks like, and without coming to class you won't get the full benefit of that intuition.

Remote Participation

Generally this class does not support remote participation: teaching is much more effective, in my experience, when everyone is physically present. However, I understand that sometimes you are sick, traveling, or otherwise unable to come to class. I will arrange for remote participation in any particular lecture as long as you request it at least one hour in advance (if you're sick or in some other emergency) or 24 hours in advance (if you're traveling or otherwise planning to be unable to come to class). Notify the instructor via email if you need to participate in a particular class remotely.

Asking Questions

There is a course Discord server which you can use to ask (and answer) questions about any of the course topics or for help with the homework. Participating on Discord is optional, but if you do participate in a productive manner (especially by answering other student's questions!), it will have a positive impact on your participation score.

Course Project (both Individual and Group)

The assignments and project for this class are designed to mirror the experiences of a software engineer joining a new development team: you will be "onboarded" to our codebase, make several individual contributions, and then form a team to propose, develop and implement a new feature. The codebase that we'll be developing on is a remote collaboration tool called Covey.Town. Covey.Town provides a virtual meeting space where different groups of people can have simultaneous video calls, allowing participants to drift between different conversations, just like in real life. Covey.Town is inspired by existing products like Gather.Town, Sococo, and Gatherly.IO — but it is an open source effort, and the features will be proposed and implemented by you! All implementation will take place in the TypeScript programming language, using React for the user interface.

At the end of the semester, the instructors and TAs will evaluate all of the student projects, and select the best (in terms of usability, code quality, test suite quality, and overall design) to merge into the open source Covey.Town codebase on GitHub repository. No additional course credit will be awarded to these teams, but these students will have the opportunity to receive public recognition for their project (in the form of a pull request merged into our repository and acknowledgments in the project's contributors list).

The project will provide hands-on experience to complement the skills taught in this class, requiring students to be able to:

- Work effectively in a small team
- Enumerate and prioritize development tasks
- Propose, design, implement and test a new feature in an existing non-toy software application
- Write code that their team members can read and review
- Review teammates' code
- Analyze a proposed software architecture
- Use relevant software tools, such as:
  - TypeScript
  - Visual Studio Code (or similar IDE)

- Git
- Mocha and Jest
- Twilio's Programmable Video API
- Postman

Exams

There are two exams in this course:

- a mid-term, which is held in class about halfway through the semester (worth 10% of your course grade)
- a final exam, which is held during the university-scheduled final exam slot (worth 15% of your course grade)

Both exams will cover a range of topics discussed in lecture and/or in the mandatory readings, from any time during the semester up to the point when the exam is held. The exam will be comprehensive, covering many of the topics we discuss; I may ask about anything we covered in class or that you were supposed to read. The exam will be conducted in person. Contact the course staff privately via email if you are not able to attend for any reason (e.g., you are sick or need special accommodations) and we will arrange an alternative. See the exams page for more information.

Collaboration Policy

Collaboration is generally encouraged in this course, as is consulting online resources. You are permitted to copy small amounts of code from any source except someone else's copy of an assignment, *as long as you cite your source*. "someone else's copy of an assignment" also includes students not currently enrolled in the course - e.g., students who took (or are taking) this class in previous semesters or took classes that used similar individual projects at other institutions. To make this more clear, here are some examples of acceptable and unacceptable collaboration on a programming assignment in this course:

Acceptable collaborations:

- Discuss problems/solutions/anything with any number of other students (as long as you don't look at each other's code).
- Copy a short (about 10 lines or fewer - use your judgment) snippet from stackoverflow.com or a similar source, as long as you include a comment with the source URL.

- Copy code written by one of your teammates during the group project for another part of the group project.
- Copy code from the output of a generative AI tool such as ChatGPT that you prompted yourself, if you include a link to a record of your interaction with the model (e.g., ChatGPT's "share" feature) as a code comment.

Unacceptable collaborations:

- Copy code directly from another student on an individual project.
- Copy code from another group on a group project.
- Copy a significant portion (more than about 10 lines of code or a single method - use your judgment) of your assignment from the internet, even if you cite your source.
- Copy a short snippet from the internet without citing your source.
- Copy code from the output of a generative AI tool (such as ChatGPT) without citing your source
- Copy code from the output of a generative AI tool prompted by someone other than you (or your teammates, for the group project)

These rules are intended to mimic what is acceptable in industry when working as a software engineer: using the resources available to you, such as your teammates and the wider internet, is always allowed. But, it would be illegal to copy code from a competing company working on a similar product.

Consequences of Violating the Collaboration Policy

(From the University)

"Academic Integrity is the cornerstone of higher education and is central to the ideals of this course and the university. Cheating is strictly prohibited and devalues the degree that you are working on. As a member of the NJIT community, it is your responsibility to protect your educational investment by knowing and following the academic code of integrity policy that is found at: http://www5.njit.edu/policies/sites/policies/files/academic-integrity-code.pdf.

Please note that it is my professional obligation and responsibility to report any academic misconduct to the Dean of Students Office. Any student found in violation of the code by cheating, plagiarizing or using any online software inappropriately will result in disciplinary action.

This may include a failing grade of F, and/or suspension or dismissal from the university. If you have any questions about the code of Academic Integrity, please contact the Dean of Students Office at dos@njit.edu"

Late Policy

All deadlines are final; no late work will be accepted for credit. We are always happy to give you feedback on late work if you contact us by email.

Acknowledgments

This course is heavily indebted to a number of other courses in software engineering at other universities, especially:

- Jon Bell's CS 4530 at Northeastern (special thanks to Jon and his colleagues for their permission to re-use the Covey.Town project materials.)
- Wes Weimer's EECS 481 at the University of Michigan
- Michael Ernst's CSE 403 at the University of Washington

As a student, if you're looking for more materials (or just a different perspective) on any of the topics we cover, you might start with those (excellent) courses.

CS 490-001 (Au25)

Calendar

Week 1

Sep 1:	**No Class (Labor Day)**	**Totally Optional, For Fun Readings**: Gross' The Grug Brained Developer and Kingsbury's Reversing the technical interview
Sep 3:	**Introduction**	**Mandatory reading**: the Individual Project 0 Specification and the syllabus (No reading quiz today, but these are fair game for any subsequent reading quiz.) **Your Choice reading**: Brooks' No Silver Bullet

Week 2

Sep 8:	**INDIVIDUAL PROJECT 0** DUE	
Sep 8:	**Code-level Design**	**Mandatory reading**: Spolsky's The Joel Test (note that this article is from 2000, so the examples are a little dated), Gransee's Opinions on Opinionated Formatters, the Prettier team's Option Philosophy, and Wikipedia's Law of Triviality.

Your Choice reading: Ajami et al.'s Syntax, predicates, idioms — what really affects code complexity?

Sep 10: **Reading Code**

Mandatory reading: Atwood's Learn to Read the Source, Luke, Coleman's How to quickly and effectively read other people's code, and the Individual Project 1 Specification

Your Choice reading: Endres et al.'s Relating Reading, Visualization, and Coding for New Programmers: A Neuroimaging Study

Week 3

Sep 15: **Testing (1)**

Mandatory reading: Shore's The Art of Agile Development: Test-Driven Development

Your Choice reading: Saff and Ernst's An Experimental Evaluation of Continuous Testing During Development

Sep 17: **Testing (2)**

Mandatory reading: Petrovic's Mutation Testing

Your Choice reading: Memon et al.'s Taming Google-Scale Continuous Testing

Week 4

Sep 22:	**INDIVIDUAL PROJECT 1** DUE	
Sep 22:	**Testing (3)**	**Mandatory reading**: SQLite's How SQLite is Tested and the Group Project Specification. **Your Choice reading**: Barr et al.'s The Oracle Problem in Software Testing: A Survey
Sep 24:	**Version Control**	**Mandatory reading**: Ernst's Version control concepts and best practices and Thompson's My favourite Git commit **Your Choice reading**: De Rosso et al.'s Purposes, concepts, misfits, and a redesign of git

Week 5

Sep 29:	**INDIVIDUAL PROJECT PROPOSAL** DUE	
Sep 29:	**Process**	**Mandatory reading**: The Agile Manifesto and its Twelve Principles (this should be a quick read, but I suggest you think about what it is advocating for at least a few minutes before moving onto the next article) and Santo's "Waterfall" doesn't mean what you think it means **Your Choice reading**: Anda et al.'s Variability and Reproducibility in Software Engineering: A Study of Four

		Companies that Developed the Same System
Oct 1:	**Working in Teams**	**Mandatory reading**: Fowler's Two Pizza Team and Arguelles' My favorite coding question to give candidates (and why) **Your Choice reading**: Behroozi et al.'s Hiring is Broken: What Do Developers Say About Technical Interviews?
Oct 3:	**Team Assignments** published no later than this date.	

Week 6

Oct 6:	**Requirements and Specifications (1)**	**Mandatory reading**: Spolsky's How to be a Program Manager and Ubl's Design Docs at Google. **Your Choice reading**: Ernst et al.'s The Daikon system for dynamic detection of likely invariants
Oct 8:	**Requirements and Specifications (2)**	**Mandatory reading**: Wayne's Using Formal Methods at Work. **Your Choice reading**: Lamport's Introduction to TLA

Week 7

Oct 13:	**PROJECT PLAN** DUE	

Oct 13:	**Code Review**	**Mandatory reading**: Google's How to do a code review (read all six linked sub-pages in the bulleted list) **Your Choice reading**: Bacchelli and Bird's Expectations, Outcomes, and Challenges Of Modern Code Review
Oct 15:	**Languages**	**Mandatory reading**: Howarth's Why Discord is Switching from Go to Rust and Schwab's Safety through Incompatibility **Your Choice reading**: Hoare's Hints on Programming Language Design (shorter than it looks!)

Week 8

Oct 20:	**Build Systems**	**Mandatory reading**: Atwood's The F5 Key Is Not a Build Process **Your Choice reading**: Mokhov et al.'s Build Systems à la Carte
Oct 22:	**Static Analysis**	**Mandatory reading**: Ayewah et al.'s Experiences Using Static Analysis to Find Bugs and Schwartz-Narbonne's How to integrate formal proofs into software development **Your Choice reading**: Chapter 2 ("Abstract Interpretation") of

		Ernst's Notes on Program Analysis

Week 9

Oct 27:	**REVISED PROJECT PLAN** DUE	
Oct 27:	**Debugging (1)**	**Mandatory reading**: Zeil's Debugging – Using Hypotheses to Track Down the Culprit and Taylor's Debugging **Your Choice reading**: Ko and Myers' Designing the WhyLine: A Debugging Interface for Asking Questions about Program Behavior
Oct 29:	**Debugging (2)**	**Mandatory reading**: Zeller's Automated Debugging: Are We Close? and Alpert's TODOs aren't for doing **Your Choice reading**: Cleve and Zeller's Locating Causes of Program Failures
Oct 31:	PROJECT **DEPLOYMENT DEMO** DUE	

Week 10

Nov 3:	**Mid-term Exam (in-class)**	see the exams page for old and practice exams (with keys)
Nov 3:	FIRST **PROJECT TEAM SURVEY** DUE	
Nov 5:	**Software Architecture**	**Mandatory reading**: Kästner's Thinking Like a Software

		Architect and Ross' How Architecture Diagrams Enable Better Conversations **Your Choice reading**: Garlan's Software Architecture
Nov 7:	PROJECT **WIZARD-OF-OZ DEMO** DUE	
Week 11		
Nov 10:	**Design Patterns**	**Mandatory reading**: Fowler's Writing Software Patterns (read up to, but not including, "Common Pattern Forms") and Lewis and Fowler's Microservices **Your Choice reading**: Kellogg et al.'s Verifying Object Construction
Nov 12:	**Tech debt, refactoring, and maintenance (1)**	**Mandatory reading**: Allman's Managing Technical Debt **Your Choice reading**: Kim et al.'s A Field Study of Refactoring Challenges and Benefits
Week 12		
Nov 17:	**Tech debt, refactoring, and maintenance (2)**	**Mandatory reading**: Spolsky's Things you should never do, part I and Majors' Friday Deploy Freezes Are Exactly Like Murdering Puppies

Your Choice reading: Scully et al.'s Machine Learning: The High-Interest Credit Card of Technical Debt

Nov 19: **DevOps (1)**

Mandatory reading: Sloss' "Introduction" and Baye's "Emergency Response" from Google's Site Reliability Engineering

Your Choice reading: Dean and Barroso's "The Tail at Scale"

Nov 21: PROJECT **PRELIMINARY DEMO** DUE

Week 13

Nov 24: **DevOps (2)**

Mandatory reading: Lunney and Lueder's "Postmortem Culture: Learning from Failure" from Google's Site Reliability Engineering and Luu's "Postmortem Lessons"

Your Choice reading: Xu et al.'s "Do Not Blame Users for Misconfiguration"

Nov 26: **No class** (Friday classes meet for Thanksgiving holiday)

Nov 26: SECOND **PROJECT TEAM SURVEY** DUE

Week 14

Dec 1: **Free and Open-source Software**

Mandatory reading: Stallman's Why Open Source Misses the Point of Free Software and

		Zaitsev's The Future of Open Source is Polarized **Your Choice reading**: Terrell et al.'s Gender differences and bias in open source: pull request acceptance of women versus men
Dec 3:	**Software Engineer Panel**	**Mandatory reading**: none, but you must submit a question by December 2 AoE **Your Choice reading**: none

Week 15

Dec 8:	**What is Software Engineering?**	**Mandatory reading**: Shaw's "What makes good research in software engineering?" **Your Choice reading**: read 10 abstracts in the latest ICSE proceedings and make a list of words you don't know. Then, look up at least 5 of those words and write a brief definition. Send me the list of words, the 5 definitions, and the titles of the papers whose abstracts you read over email. (The reading quiz question for this reading will cover a topic that everyone who chooses this reading has read.)
Dec 10:	ALL GROUP PROJECT **FINAL DELIVERABLES** DUE	

Dec 10:	TBD/Slack, for now
Dec 12:	Group project demos for the instructor must be done by this date AoE. Attendence (in-person) is required. At least one group member must bring a laptop with a working demo of your group project (running in a publicly-accessible, deployed covey.town instance). Your team can sign up for a timeslot here.
Dec 13:	THIRD **PROJECT TEAM SURVEY** DUE
Dec 13:	**INDIVIDUAL REFLECTION** DUE

Week 16

Dec X:	Final exam at TBD
Dec 18:	ALL GROUP PROJECT **FINAL DELIVERABLES (RE-SUBMISSION)** DUE

CS 490-001 (Au25)

Projects / Project Overview

Project Overview

The individual and team projects for this class are designed to mirror the experiences of a software engineer joining a new development team: you will be "onboarded" to our codebase, make several individual contributions, and then form a team to propose, develop and implement a new feature. The codebase that we are be developing on is a remote collaboration tool called Covey.Town. Covey.Town provides a virtual meeting space where different groups of people can have simultaneous video calls, allowing participants to drift between different conversations, just like in real life. Covey.Town is inspired by existing products like Gather.Town, Sococo, and Gatherly.IO — but it is an open source effort, and the features will be proposed and implemented by you! All implementation will take place in the TypeScript programming language, using React for the user interface.

Overview of Project Deliverables

Date	Deliverable	Description
9/29/25	Individual Project Proposals	Propose a feature for Covey.Town and specify preferences for teammates
10/3/25	Team Assignment	Teams will be assigned based on individual proposed features.
10/13/25	Preliminary Project Plan	As a team, propose and plan a new feature for Covey.Town that can be implemented within 7 weeks
10/27/25	Revised Project Plan	Refine the scope of your feature based on staff feedback, define detailed requirements and project acceptance criteria.
10/31/25	Deployment Demo	Show that you can deploy a lightly-modified copy of covey.town to a remotely-accessible machine.
11/3/25	First team survey	Let us know how you think the project is going.
11/7/25	Wizard-of-Oz Demo	Show what your project will look like, once it is complete, to your project mentor and the instructor. This demo

Date	Deliverable	Description
		doesn't require you to show any working code: it is purely about design.
11/21/25	Preliminary Demo	Demonstrate one user story to your project mentor and the instructor. In this demo, you need to actually be running your own code (unlike the Wizard-of-Oz demo).
11/26/25	Second team survey	Let us know how you think the project is going.
12/10/25	Project Implementation and Documentation	Deliver your new feature, including design documentation and tests.
12/12/25	Final Demo	You will demo your feature to the instructor by this date.
12/12/25	Third team survey	Let us know how you think the project is going.
12/17/25	Project Implementation and Documentation (Resubmission)	If your final demo does not meet your project goals, you may schedule another demo with your project mentor and the instructor no more than one week after the last day of class (12/17). Re-submit your code and documentation immediately before the demo.

All assignments are due on the specified date, AoE (i.e., before the beginning of the next day anywhere on Earth, which is at 7am EST the next day).

Summary of Project Grading

Your overall project grade (which will account for 45% of your final grade in this course) will be the weighted average of each of the deliverables.

- Planning Documents
  - 7.5% Preliminary Project Plan
  - 10% Revised Project Plan
- Activities During the Project
  - 5% Meetings with Mentor and Team Surveys
  - 10% Ongoing development progress, including code reviews
- Final Deliverables
  - Code
    - 20% Final implementation of your feature

        - 10% Final test suite of your feature
    - Report
        - 5% Feature Overview
        - 7.5% Technical Overview
        - 7.5% Process Overview
- Demos
    - 1% Deployment demo
    - 2.5% Wizard-of-Oz demo
    - 4% Preliminary demo
    - 10% Final demo

In cases where team members do not equally contribute to the project, we may assign different grades to different individuals, up to an extreme of deducting 50% of the team project grade for a student. We will evaluate each individual's contribution on the basis of a variety of factors, including progress reports at meetings, through inspecting version control history, through each students' self-reflection, and through each students' peer evaluation (during and/or) at the end of the project. We will make regular efforts to collect and distribute this feedback throughout the project — our ultimate goal is for all students to participate and receive full marks.

Team Formation

All projects will be completed in a team of 3-4 students (most teams will have 4). Part of the first deliverable for the project will be a team formation survey: you will be able to indicate your preferences for teammates. The instructors will assign students to the teams based on a number of factors including your responses to the survey.

Individual Project Proposal

You'll write a one-page proposal for a feature. You only need to explain the feature *from a user perspective* in this document. We'll create groups so that people whose individual proposals are similar are grouped together. Your feature should be something that can be implemented within the timeframe allotted (5-7 weeks), and will be implemented in a fork of the main Covey.Town codebase.

Team Meetings with Mentor

Each team will be assigned a member of the course staff as a mentor, who will also serve as your point of contact for project grading. During the first week after project teams are announced, you will have a "Kickoff Meeting" with your mentor, where you will meet your mentor and have the

opportunity to share any early ideas that you might want feedback on before submitting the preliminary proposal. Once project begins in full force, you will have regular standup meetings with your mentor (scheduled at your team's and your mentor's convenience, at least once every week) in order to help ensure that you are making progress on the project, and to help address problems that you encounter (be they technical or non-technical problems).

Preliminary Project Plan

All projects will involve frontend and backend development of a new feature for Covey.Town. Once teams have been formed, you and your team will decide what kind of new feature you would like to build. We suggest starting with one of your individual proposals, but you're welcome to come up with something new together, too, if you'd like. Talk to your mentor! Given that you will be up-to-speed on the Covey.Town codebase (and have been introduced to TypeScript, React, NodeJS, and testing frameworks), and that you will have a team of three or four, we expect that the feature that you propose will be more complex than the feature implemented in the individual projects.

The project plan will focus on two sections:

- User stories and conditions of satisfaction that describe the feature that you plan to implement.
- Work breakdown: Map your user stories to engineering tasks. Assign each task to a team member (or pair of team members), provide an estimate for how long each task will take, a rationale for that estimate, and schedule those stories into sprints.

Creating a GitHub Repository

Your team's development must take place within a private GitHub repository in our GitHub Classroom. To create your repository, each member of your team should follow these instructions:

1. Sign in to GitHub.com, and then use our invitation to create a repository with the covey.town codebase. Check to see if one of your groupmates has created a group already - if so, select it to join it. Otherwise, you should enter your group number and the current semester (e.g. "Group 7-Au25") as the team name.
2. Refresh the page, and it will show a link to your new repository. Click the link to navigate to your new repository. This is the repository you will use for the project.

This repository will be private, and visible only to your team and the course staff. After the semester ends, you are welcome to make it public - you have complete administrative control of the repository.

Revised Project Plan

Based on the feedback that you receive from the course staff, you will revise your preliminary project plan, creating a more detailed plan to implement your new feature.

The project plan will include:

- Revised user stories and conditions of satisfaction (based on feedback on the preliminary project plan)
- Revised work breakdown (based on feedback on the preliminary project plan)

Your team will self-organize, as agile teams do, and will use the work breakdown and schedule as the basis for your check-ins with your team's mentor.

Project Implementation and Documentation

You will be assigned a mentor for your project who will work closely with you for the entire project. You will coordinate with the mentor to setup weekly meetings and regular sprint demos. Peer evaluation will also be used. Your final team deliverable will be a "release" of your new feature on GitHub (with tests), and will be accompanied by a demo. *Optionally,* you may also open a pull request to merge your feature into our main repository (submitting a pull request, or the pull request being merged into our codebase is independent of the grade you receive, but provides a platform for more visiblity of your project).

Your final team deliverable will include:

- The implementation of your new feature
- Automated tests for your new feature
- A report

Accompanying the final team deliverable will be an *individual reflection*, which every student must submit on their own, which will include your reflections on:

- The evolution of your project concept: How does the project that you delivered compare to what you originally planned to deliver? What caused these deviations?
- The software engineering processes that you feel could have been improved in your project: were there any procesess that in hindsight, you wish that you followed, or wish that you followed better?
- Your team dynamic: Provide a frank (and ideally, blameless) postmortem of your and your teammates collaborative performance and participation. If you had to do this same project over with the same teammates, what would *you* have done differently (or not) to improve your team's overall performance?

CS 490-001 (Au25)

Exams

My exams are generally cumulative: anything we've covered in the course up to the point at which you take the exam is fair game. I may also include questions about assigned mandatory readings, homework assignments, or any other class content that you are supposed to have viewed. Notably, this does not generally include the "Your Choice" readings: I don't expect you to have read all of those, so questions about them will always offer you a choice of which reading to answer a question about. See the "Your Choice" readings page for more information about how the "Your Choice" readings will be assessed on the exams.

My exam design philosophy is to aim for a wide range of question difficulties: I try to include both some questions that I think every student should get right and some questions that I think are difficult enough that only those who have deeply understood multiple concepts that we covered in class will even be able to answer them in a reasonable way, and everything in between.

To help you prepare for this semester's exams, below you can find links to exams from previous semesters, all of which have solutions ("keys"). Some of these exams cover the whole course (anything labeled "final"), so you'll want to be careful when studying for the midterm—not everything on these exams will have been covered by then. In addition, keep in mind that the set of topics changes a bit semester-to-semester, so it's possible that these exams include some topics that we didn't cover, and that your exams this semester might include topics that these exams ignore. These exams are provided "as-is" to help you study, but please don't over-rely on them.

I strongly recommend that before looking at a "key" for one of these exams, you sit down and attempt the exam yourself, under something like exam conditions (quiet room, no interruptions, etc.): this will help you more to prepare for this semester's exam than just reading the solutions.

Sp23 Practice Final (key)

Sp23 Final (key)

Au23 Midterm (key)

Au23 Final (key)

Au24 Midterm (key)

Au24 Final (key)