CS 490-002 (Sp23)

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.

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

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 immediately before 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 write a well-defined program in a language you've never seen before.

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
- 5%: Optional Reading Responses
- 15%: Individual Projects
- 45%: Group Project
- 20%: Final Exam

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 three times during the semester:

- after the first optional reading is graded (~ halfway through the class)
- shortly before the final exam
- when I compute final grades.

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 optional readings. I expect you to read mandatory readings before coming to class that day, and participation quizzes (see Participation & Professionalism, below) will cover the mandatory readings only. During the semester, you **must** complete **at least two** optional readings and do an associated task. Most optional 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.

Some optional readings have a specific task associated with them; if you choose one of those, complete the task and then submit the result on Canvas. If there is no task associated with an optional reading, the task is to write a one-page reaction to the paper that explains what you have learned and submit that to Canvas. There is no required format for your reaction, but you might consider including any or all of the following:

- a one or two paragraph summary of the key points of the paper
- a list of the paper's strengths: what did the paper do well, scientifically?
- a list of the paper's weaknesses: what did the paper do poorly? Did it makes claims you didn't believe?
- a description of how this paper might be useful to a software engineer working in industry
- a description of how you might apply a lesson from the paper to your own work, now or in the future

When you submit an optional reading task, be sure to include in your submission the title of the optional reading.

One optional reading is due ~50% of the way through class (in spring semesters, around spring break). The other is due at the end of the semester. For the first optional reading, you aren't restricted to readings that are associated with lectures earlier in the semester: you are always welcome to read ahead.

Participation & Professionalism

Your participation & professionalism grade is composed of two scores.

First, your *Professionalism* score is based on the instructor's 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).

Second, your *Participation* score is based on reading quizzes (about topics from the mandatory readings only - quizzes will never cover optional 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 will always be easy if you did the reading). For full participation, you need to get at least a score of 60% 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 40% 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.

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 acknowledgements 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

Final Exam

The final exam will be held during the last regular class meeting of the semester. It will cover a range of topics discussed in lecture and/or in the mandatory readings, from any time during the semester. 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) and we will arrange an alternative.

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 another student's copy of an assignment, *as long as you cite your source*. "Another student's copy of an assignment" also includes students not currently enrolled in the course - e.g., students who took 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.

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 - 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.

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.

Consquences 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

You may use up to two late days on Individual Projects 1 and 2 (in total: either 2 on one of the two or one on each) without penalty. Assignments turned in after your second late day will not be accepted.

Your group may use up to two late days on the Group Projects (in total). Assignments turned in after your group’s second late day will not be accepted.

You may not use late days on Individual Project 0, the Group Project Presentation, or the Group Project Final Submission.

Acknowledgements

This course is heavily inspired by 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.