# CAS CS411: Software Engineering

**2024.FALL.SYLLABUS**

Dr. Peter B. Golbus

* Tues / Wed 10:30–12 CDS 906

TAs (Rooms TBD)

* Paula Lopez Burgos
  + Fri 10–1
* Aiden Fockens
  + Tues 4:30–6
  + Thurs 9:30–12
* Daniel Wang
  + Mon 1:30–3
* Jason Wang
  + Sun 3:30–5
  + Sat 1:30–3
* Kelvin Lin
  + Thurs 3:30–5
  + Sat 9:30–11
* Michelle Sun
  + Sun 11:30–1
  + Mon 3:30–5

# **About the course**

**What is Software Engineering and what does a Software Engineer do?** Software Engineering is the discipline of designing, building, and maintaining complex systems, built out of code, that solve real-world business problems. A Software Engineer architects and coordinates the development of entire software solutions, ensuring they are reliable, scalable, and maintainable.

While Software Engineers write code more than say, Structural Engineers mix concrete, Software Engineers are not programmers. They are Engineers, many, but not all, of whom spend varying amounts of their time writing code.

**How do code modules, programs, and software relate to each other?** In this course, we will explore how code modules, when combined, form programs. These programs, when working together, become the software that drives solutions to large-scale business challenges. Programs are built from smaller, self-contained modules, each with a specific role, and software integrates these programs into a cohesive system.

**What will you learn in this class?** We will focus on writing large-scale programs composed of "highly cohesive and loosely coupled" code modules. The success and reliability of these programs depend on the development processes used. While programming will be part of the course, our primary focus will be on how to deliver and deploy bespoke, scalable solutions to complex problems.

Entering this course, you have experience writing code, but you may not yet have had the opportunity to write actual programs. This course will guide you through that process. We will be working in Python and focusing on backend microservices, but only because that is my area of expertise. We will be discussing general principles that will still apply to your platform of choice. I will ensure that I flag anything that is specifically “Pythonic” and microservice backend-oriented to ensure that what we learn is as universal as possible.

We will begin by discussing the tools of software engineering[[1]](#footnote-1) before covering topics you should be aware of about how modern software works. We will end by talking about how software engineering projects are managed in practice.

We will also

* Become comfortable working from the command line, which is a necessary skill for all software engineers.
* Get our hands dirty managing modern distributed cluster-based environments. This will give you some understanding of what is going on Behind The Scenes**[[2]](#footnote-2)** that make the environments you rely on Just Work.**™**

# Textbook (singular)

We will be using What Every Engineer Should Know about Software Engineering, Laplante and Kassab. We will be using the **2nd Edition**. It’s very important that you have the **2023 version**.

I have also *recommended* a book: High Output Management, Andy Grove. The book is just that: recommended.

If you only read one “airport book” about management in your life, it should be this one. 5 stars!—me

# Delivery Tools

We use Ed Discussion as a repository for the slide sets for each class, copies of homework assignments and sample code. You should be enrolled already, so that when you log on to the site youll' see the course listed. If you are not enrolled, you can do so here: <https://edstem.org/us/join/HcQZTj>. We will *not* be using Blackboard.

Ed Discussion is also our tool of choice for communication. This includes all course related announcements as well as discussions, including group / project team discussions. Please do not send course staff individual email: class-related but nonpublic questions should go to private posts on Ed Discussion rather than to email. Email will get easily lost in overcrowded inboxes.

We’ll use a combination of GitHub, Ed Discussion and Gradescope for submitting work. Don’t worry if you don’t know what GitHub is—we’ll go over it. You can join the gradescope using the following code: **4J6XXR**.

# Grades:

|  |  |
| --- | --- |
| Readings | 5% |
| ~~Tophat~~ In Class Quizzes | 5% |
| HW | 25% |
| Tests | 2x 10% each |
| Final | 15% |
| Project | 20% |
| Peer Review | 10% |

# We will not assign letter grades to individual assignments and exams. When assigning final letter grades, we will look at the total points you earned and decide on the cut-offs for A, B, C, etc. Those cutoffs will likely be lower than the usual US high school cut-offs (90, 80, 70, etc.), because this class is harder than a typical high school class. To help you determine where you stand in the course, we will post averages of every assignment. If you are at the average, you are roughly around a B.

# Readings

The textbook claims to be “Socratic,” which appears to mean that the section headings are stated as questions. The reading assignments will have associated gradescope assignments where you will **answer** them ***in*** ***a few words to a few sentences****.* We will be spot-checking some of these to ensure that you have done the reading. **Do not spend much time on this**.

The questions will either cover a single section, e.g.

1.2.6: How do software engineers spend their time on the job?

* + Software engineers only spend about 10% of their time programming. They spend the rest of their time documenting, designing, discussing, etc. their projects.

Or a range of sections, e.g.

2.2.12—2.2.17

* This section describes the quality of software reliability, how it can be defined and measured, and the surprising way it changes over time

Scores will be rounded up as follows: if you score at least an 80%, your score will be rounded up to 100. If you score less than 80%, your score will be normalized out of 80. So if you scored 80%, you would receive full credit. If you scored 79%, you would receive 79/80 = 98.75%.

# ~~Tophat~~ In-class quizzes

In this course we will be doing the kind of brief, in-class multiple choice quizzes you are probably familiar with from other courses. The distinction is that Ed Discussion has functionality we can use for this so you will not need to purchase any additional licenses.

All quizzes will be graded 80% participation and 20% correctness. The final quiz score will be rounded up from 80 as are the readings.

# Tests / Final

The tests will be the same short answer format as the readings. A test needs to balance between going deep on a few topics and covering a wide range of topics superficially. The goal is to be broadly familiar with a wide range of topics so that you recognize them when they come up and know what to google, and therefore in this class we focus on the latter exclusively.

# Homeworks / Labs

There is only one way to learn to program, and that is by writing lots of code.[[3]](#footnote-3)

There will be approximately 6 homeworks related to designing and deploying a modern (toy) web-app. Some homeworks will be design-orientated. You will do these alone and have approximately 1.5 weeks to complete them. Others will be about “getting your hands dirty” writing code and getting it to work. These will be done in (randomly assigned) pairs. You will have approximately 2.5 weeks to complete these.

Homeworks will generally be released on Tuesdays and due at 11:59PM on the appropriate Friday. Homeworks will be accepted as late until the following A1 section, Tuesdays at 12:30, for a 15% penalty, i.e. 0.85 \* your score. This is so we can go over solutions.

Labs will be an optional time to receive tech support. It is also a time when you and your partner are guaranteed to both be available.

# Project

The project will be to essentially replicate the functionality of the homeworks in an application of your choice. You are encouraged to re-use the architecture / tech stack used in the homeworks, but you are not required to do so. The required components are:

* A decoupled frontend / backend
* (At least) one external API call
* A database-integration containing (at least) user accounts with salted password hashes
* Unit & integration tests
* Docstrings, logging and exception-handling
* Containerization

# Peer Review

We are not here to write code or to learn to write code. However, mastering the principles of Software Engineering will involve writing code, and will make you a better coder.

All coding homeworks in this class will be done in pairs, ideally using pair programming (see below). Don’t worry if you are not the strongest coder on your team, just contribute to the best of your ability. If you *are* the strongest coder on your team, please try to let your partners contribute.

**Pair programming:** Pair programming is the practice of having two engineers working together on a single computer, one as the “driver” who is actively using the mouse and keyboard, and the other as the “navigator.” You will quickly find out temperamentally which of you is which. This practice is very standard; it is widespread in industry. It is especially effective in pedagogical situations. You will be surprised to find that often neither of you will completely understand it, but that you will still be able to explain it to each other.

I’ve done this. I know it sounds painful, but it *works*. Really. If there are companies who pay two engineers to use a single computer, they must be seeing a greater than 2x gain in efficiency.

**Peer Reviews:** One of the objectives of this class is to mimic the experience of working in a professional setting, where collaboration with colleagues you didn't choose is essential.

Your performance as a team will be graded based on your “professionalism.” In this section, you'll evaluate your partner on a scale of one to five across several areas and provide a brief statement about your collaboration.

In a real job, your team's performance is visible across the company, while your individual contributions are seen by your team, your boss, and your “skip-boss.” This visibility is formalized through annual or bi-annual peer reviews. To reflect this, we have both a team grade and a peer review grade.

To advance in your career, you must be an effective team member contributing to collective success. Success can come in many forms, and you don't need to be the best coder to make meaningful contributions. However, contributing in some way is essential.

With that in mind, you'll assess your partner's communication and follow-through. For example, if your partner was actively engaged, but you ended up correcting their work because they needed guidance, that's a positive outcome. However, if you had to fix their work because they started and then didn’t follow through, that is not.

Long story short, to succeed in your career, people have to like you.

# ChatGPT

**DO NOT LET AI WRITE CODE THAT YOU PUBLISH ON THE INTERNET UNLESS YOU FULLY UNDERSTAND INTERNET SECURITY.**

If you use ChatGPT, you must also provide the prompts you used to generate the code. There is a line between

* “ChatGPT, implement these ten different things and talk to me about how to stitch them together”

and

* “ChatGPT, do my final project.”

It remains to be seen how thin that line is.

If you do use ChatGPT, we are more than happy to look at any code but we will not look at your prompts (other than to provide guidance about whether your prompts are fine-grained enough to get credit).

I use ChatGPT all the time, but I also know how to tell when it’s wrong and what to do about it. Use at your own risk. Also consider that **if all you learn how** **to do is use** **LLMs**, **you will** **not be able to pass a job interview**.

**DO NOT LET AI WRITE CODE THAT YOU PUBLISH ON THE INTERNET UNLESS YOU FULLY UNDERSTAND INTERNET SECURITY.**

# On code reuse

You are expected to use as many resources as you can when doing your assignments. (Before you ask a question, did you google it? Did you check source code we used in class and in the various repos that I’ve shared? Did you look on Ed Discussion?) It is entirely appropriate to “copy-pasta” blocks of code, so long as you cite your sources. This is done all the time, as you can tell because it has a cute nickname.

You may not turn in someone else’s program as your own work, even if it is properly cited. However, you can implement it yourself. If you find a solution you want to use that’s more than a few lines of code, you can always translate it into English, and then turn the English back into code without looking at the original source.

1. E.G. architectural and software design principles. Things like git and IDEs are *programming* tools. [↑](#footnote-ref-1)
2. Have you thanked your friendly neighborhood DevOps / Infra team today? [↑](#footnote-ref-2)
3. And fixing it. Mostly fixing it, actually. [↑](#footnote-ref-3)