CS 356: Software Design

Cal Poly Humboldt
Department of Computer Science
Syllabus—Spring 2025

Labs	Tuesdays 3:00–4:20 room FH 202		
Course Credit	3 units		
Instructor	Ben Kovitz	**Office**	BSS 344
Email	blk14@humboldt.edu	**Office Phone**	(707) 826–3492
Office Hours	MW 3:00–4:00 and by appointment		
Texts	We will be reading excerpts from the following: *Design Patterns* by Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides (the "Gang of Four"). 1994. *UML Distilled,* 3rd ed., by Martin Fowler. 2003. *Effective C++,* 3rd ed., by Scott Meyers. 2005. *Effective Modern C++* by Scott Meyers. 2015. *Refactoring,* 2nd ed., by Martin Fowler. 2019. *Design Patterns in Modern C++20* by Dmitri Nesteruk. 2022. All of these books are available free on-line to Humboldt students at O'Reilly Learning: `https://go.oreilly.com/humboldt-state-university`. All but the last are also available fairly inexpensively in hardcopy as used books (recommended).		
Prerequisites	CS 201 (Requirements Engineering); concurrently: CS 325 (Database Design)		
Canvas Link	`https://canvas.humboldt.edu/courses/76623`		
Final Exam	Tuesday, May 13, 2025, 3:00–4:50 p.m. in room FH 202		

Course Description

In this course, you will learn how to design software, mainly by designing software—and implementing it. And you will learn partly by reading and discussing classic writings about software design.

This course will be all labs, no lectures. Here's how the course will work:

- The first few class sessions will be design exercises, mostly omitting implementation in code.

- After that, we will spend most of the semester developing a large piece of software—designing *and* implementing. All code will be written during class, collaboratively, by "mob programming" (see below).
- There will be assigned readings before every class session. Some will come from the texts listed above and some will come from articles that I find. Expect about 10–40 pages of reading a week.
- Each reading will have one student assigned to give a short presentation on it—three to five minutes at the start of the next class. We'll have a brief in-class discussion after each presentation.
- After each class session, you are required to write some notes summarizing what you learned that day—one or two paragraphs of complete sentences, up to a page if you like. I'll read these periodically to see how the class is going.
- Use of AI is allowed and encouraged. During the first half of the semester, we will limit AI to documentation look-up and debugging. In the second half of the course, we will run wild with AI, using it for everything we possibly can, seeing what works and what doesn't work.
- We might spend the last few sessions of the semester on some high-level design exercises of the sort that are common in job interviews.

Along the way, you will learn:

- How to design and implement software collaboratively
- How to sketch out a design with UML diagrams
- Design patterns
- How to improve a design by refactoring
- Basics of user-interface design
- How to use modern C++ effectively (beyond basic object-orientation)

This is a highly experimental course. Many things will go wrong. I have never done mob programming myself, and I've only recently been experimenting with AI-assisted programming. We will all be learning this at the same time. Expect many "course-corrections" throughout the semester.

You must show up! This is class is pretty much all in-class activity. Attendance and active participation during class are mandatory.

We will choose the exact project after we finish the design exercises. I have asked people around campus for software that they need written. We might choose one of theirs or we might choose a project of our own. The constraints are that C++ must be a reasonably good fit as an implementation language, the project must be complex enough that it will call for a variety of design patterns, and it must be feasible to implement within the semester.

Mob Programming

Mob programming is a way of writing software "where the whole team works together on the same thing, at the same time, in the same space, and at the same computer." (*Software Teaming,* 2nd ed., by Woody Zuill and Kevin Meadows, 2022, p. 2.)

During mob programming, one person, called "the driver", is at the keyboard. Everyone else should be able to see their screen. The team rotates through the driver role every 7 minutes (or whatever rotation time we set—we'll experiment).

Another role is "the navigator". The navigator tells the driver what to do next. The driver functions as an intelligent input device, translating higher-level ideas into keystrokes and code. A maxim: "For an idea to go from your head into the computer, it must go through someone else's hands." —Llewellyn Falco.

We will explore roles that the other team members can play. Generally you should be checking what's going on at the keyboard for errors or better ways to do it, discussing them with the navigator, and finding other ways to support the team.

We'll try to make most decisions by consensus, but occasionally there will be disagreements. When the time required to try two ideas is short, we'll try them both rather than argue. Sometimes, though, we will need to make a group decision that not everyone agrees with. At these times, the principle to follow is "Disagree and commit"—that is, commit to implementing the team's idea and making it work as well as you can, even if you disagree with it.

Near the start of the course, all of us will work as one mob so we all learn the basics. After we know how to do it, we'll likely break frequently into smaller mobs that work in parallel.

Your Time Outside of Class

Since all code will be written in class, you won't need to write code as homework nor will you need to schedule times with other students on team projects. You should still block out about 2–4 hours of uninterrupted time each week, though, for the readings, for preparing your presentations, and for making your after-class notes. During some weeks, we may assign some research work to help with the project, or you might simply want to explore or code up an idea on your own initiative. So, allow that in some weeks, you might need as much as 6 hours of time outside of class.

Topics

We will cover some large subset of the following topics. The exact topics will result from the readings and from the problems and techniques that arise organically in the course of designing and implementing the software for the semester project. At the end of the semester, we'll look over this list and see which topics we covered—and what we learned about design that was not on the list.

OO Analysis
nouns → classes
verbs → methods
UML class diagram
UML sequence diagram
UML state diagram
UML use-case diagram
UML activity diagram
design documents

Deciding What to Do Next
test-driven design (TDD)
YAGNI
Write the calling code first
Top-down or bottom-up? Start from what you know
Premature optimization is the root of some evil
stubs

Planning
agile & waterfall
the cost-of-change curve
risk-mitigation
build one feature at a time
learning happens during software development
pauses to learn something
spikes

Ancillary but Important
requirements / specifications
unit tests
git / version control

Good and Bad Code
refactoring
coding standard
clear names > comments
code smells
technical debt

General Concepts & Maxims
information-hiding
What is the simplest thing that could possibly work?
Choose an output for every possible input
the DRY principle (aka OOAO)
Design code to be testable
Favor immutability
Fail visibly
abstraction layers
leaky abstractions
look-up table
Every problem in computer science can be solved by adding a layer of indirection …except for the problem of too many levels of indirection
Make representations that have no invalid states
coupling and cohesion
design by contract
ripple

OO Maxims
aggregation > inheritance
S: single responsibility
O: open/closed principle
L: Liskov substitution principle
I: interface segregation
D: dependency inversion
Law of Demeter

C++
resource acquisition is initialization (RAII)
Curiously Recurring Template Pattern (CRTP)
rule of 3 / rule of 5
smart pointers

GUI Design
user model vs. implementation model
paper prototypes
Principle of Least Amazement
user thinks out loud

Design Patterns
Wrapper (Adapter)
Factory, factory method
Observer
Composite
Decorator
Model–View–Controller
Interpreter
State
Strategy
Iterator
Template methods, template classes
Visitor
value objects
Producer-Consumer Pattern

Architecture
3-tier client-server architecture
DSL (domain-specific language)
microservices

Real Life
You're paid for working code, not diagrams and design documentation
The users don't sign the checks
"The code you write today pays for your time to debug it tomorrow."

Grading

Each part of the course will be weighted in your final grade for the course as follows:

Participation	50%
Learning notes	5%
Presentations	25%
Final exam	20%

To get a passing grade, however (a grade of C– or better), you must score at least 50% in each of the above categories.

Letter-grade cut-offs are as follows, where s is your overall score for the course:

$93 \leq s$	A
$90 \leq s < 93$	A–
$87 \leq s < 90$	B+
$84 \leq s < 87$	B
$80 \leq s < 84$	B–
$77 \leq s < 80$	C+
$74 \leq s < 77$	C
$70 \leq s < 74$	C–
$65 \leq s < 70$	D+
$60 \leq s < 65$	D
$s < 60$	F

Additional Information

- Students are responsible for knowing the information about campus policies, procedures, and resources on the Syllabus Addendum website linked below. The site includes topics such as learning outcomes; registration policies; academic honesty policy; attendance and disruptive behavior policy; standards for student conduct; prevention and reporting of discrimination, harassment, and retaliation; animals on campus policy; emergency procedures; resources for students with disabilities; learning and advising resources; counseling and psychological services; financial aid; IT Help; and more. `https://academicprograms.humboldt.edu/content/syllabus-addendum`

- It is the student's responsibility to notify the instructor in advance of the need for accommodations and to provide documentation to the university (SDRC or Dean of Students).

- If you do any online communication in connection with this course, remember that university regulations regarding disruptive behavior extend to the online environment. Appropriate online behavior (i.e., netiquette) is expected.

This syllabus is subject to change. Changes will be announced on Canvas. Be sure to check your Cal Poly Humboldt email regularly so you're aware of course updates and announcements.