University California, Berkeley
Professional Masters in Molecular Science and Software Engineering

Online Degree
Chem 274B, 3 Units

Introduction to Software Engineering

Fall 2022
Syllabus

Tony Drummond,
LADrummond@berkeley.edu

Section I. Course Description.
This course will advance students' understanding of fundamental knowledge and techniques for developing complex software. Students will gain an in-depth view of computer system architecture as well as abstraction techniques as means to manage program complexity and software productivity.

Students will collaboratively develop a software engineering package, thus gaining experience in all aspects of the software development process. This course serves as a prerequisite to later MSSE courses: Data Science, Machine Learning Algorithms, Software Engineering for Scientific Computing, Numerical Algorithms Applied to Computational Quantum Chemistry, Applications of Parallel Computers and the Capstone Project.

Contribution of this course to the broader curricular objectives:
Required course for all MSSE students.

Section II. Prerequisites.
MSSE's Introduction to Programming Languages Bootcamp. Alternatively, for students opting out of the MSSE Introduction to Programming Languages Bootcamp, they will need to demonstrate knowledge of programming languages, software development tools and UNIX-based O/S commands.

Section III. Course Learning Objectives and Outcomes.
- This course will provide MSSE students with the required foundational knowledge of computer systems, algorithms and software engineering techniques to prepare them for more advanced computational sciences courses in the MSSE program and for students to start building sustainable and scalable computational science solutions.

- Students will also learn best software engineering and collaborative software development practices.

- Students will start to build their MSSE software portfolio, which is a graduation requirement that students will submit during their Capstone Project course.

Section IV. Reading List and Resources.

- [SMP2016] *Absolute C++* (6th Edition), Walter Savitch and Kenrick Mock, Pearson, 2016
- [CLRS2022] *Introduction to Algorithms* (fourth edition) by Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein: https://mitpress.mit.edu/books/introduction-algorithms-third-edition
- [WMW2020] *Software Engineering at Google*: Lessons Learned From Programming Over Time. Titus Winters, Tom Manshreck and Hyrum Wright.
- [PH2018] *Computer Organization and Design RISC (V Edition)*, David A. Patterson and John L. Hennessey, 2018. ISBN 0128122757.
- [IEEEFPS] *IEEE Floating Point Simulator*. https://www.h-schmidt.net/FloatConverter/IEEE754.html

Section V. Grading.

In this course, students' final grades will be computed by periodic assessments during the semesters. There are four individual homework assignments, totalling 40% of the student's final grade, and a final project assignment, totalling 40% of the student's final grade. Individual assignments will reflect the student's individual software development work in the class, these assignments will build up from problem-sets introduced in the asynchronous lectures and review of materials during synchronous discussion and lab sessions. For the final project, Students will work in cross-functional software development teams. The final project will also build up from software development best practices covered in the course and students experiences working on teams during the lab sessions.
Software engineering labs will be held every other week, and student participation and lab completions will account for 10% of the final grade. Students are also expected to participate during the synchronous discussion sessions answering questions and solving problems. Student participation in the synchronous discussions will account for 10% of the final grade.

In summary, the final grade breakdown is as follows;

- 40% individual student homework assignments,
- 10% synchronous sessions discussions and participation,
- 10% lab submission,
- 40% Final project assignment (cross-functional teams).

Section VI. Homework and Final Assignments *(see **Section VII** for more details)*

Assignment	Brief Description	Due Date	% Final Grade
Assignment 1	Analysis of algorithms, their computations and complexity.	Week 5	10%
Assignment 2	C++ implementation of algorithms and data structures.	Week 7	10%
Assignment 3	Scalability and performance of C++ code implementations	Week 11	10%
Assignment 4	Greedy algorithms implementations in C++	Week 13	10%
Final Assignment	Cross-functional team software development. Develop a framework for cellular automata modeling and simulations	Week 17	40%

Late Assignment Policy:

If a student needs to turn in a homework assignment after the deadline, they need to get approval from faculty and her or his grade will automatically lose 15% total grade for that particular homework assignment. ***Therefore, late assignment submissions will require justification and approval from the instructor.***

Faculty and GSIs will provide progress reports and feedback to students on the weekly basis. Students that miss a synchronous discussion or lab session will be asked to watch the video and summarize the discussion and provide answers to the questions and problems posed during the sessions. Faculty or GSIs will review students' make-up work. Late submissions for assignments will not be accepted.

Section VII. Course Structure and Schedule:

This course is designed as a 15 weeks of asynchronous online lectures, synchronous online discussions and laboratory practices, 5 student-led software engineering projects; 4 individual student projects and 1 cross-functional team project. Final projects are due on Week 17 after RRR week.

Every week, there are three hours of Faculty-led, asynchronous, web-based instruction; two hours weekly of online synchronous discussion. In addition, there are two hours of online synchronous lab discussions every other week.

Graduate Student Instructions will go over assignments and corresponding solutions after they are submitted. Outside class work should comprise about four hours a week for a total of nine hours per week.

The instructor will hold 1 hour/week for Office Hours. The GSIs will be available 10 hours/week. The course schedule and Office Hours will be posted on the course website.

Cross-Functional Teams.

Students will be organized in cross-functional teams. A cross-functional team is a group of students working on the same project and each team member has a different software development role in the team. Students will be asked to write individual reports on the team work and her or his individual contributions to the cross-functional team.

This course has been designed in full coordination with MSSE's Chem 274-A to maximize the course learning outcomes, improve the student's learning experience and transferability of the materials covered in both Chem 274A and Chem 274B to the other more advanced MSSE courses.

Chem 274-B is organized in seven modules as described in the following week-by-week schedule.

Module 1. Course Introduction.

This module covers Chem 274B Introduction and expectations, an introduction to computer systems; basic units (processing, storage, I/O), memory layout and hierarchies, working on a supercomputer, basic shell commands. Introduction to single- and multi-thread computing and best practices of software engineering.

Learning Objectives:

- Students will learn to identify key hardware components in a computer systems that impact the performance of software implementations of algorithms and computational applications,
- Students will learn basic Unix/Linux shell commands essential for Chem 274A, Chem 274B and other MSSE courses,
- Students deepen their understanding of high-level computer languages and data abstractions and machine language and data representations,
- Students learn key concepts pertaining to single-thread and multi-thread computing,
- Students learn from best software engineering practices.

- Week 1. This week starts with an Introduction to Chem 274 and the course expectations. Students are introduced to the learning objectives. This will also provide an introduction to Software Engineering and Computer systems, covering; Basic units (processing, storage, I/O), memory layout, multicore, GPUS and Supercomputers. Students will be introduced to NERSC and will apply for an account. Reading materials from [PH2018].

- Week 2. Covers an introduction to Floating Point Arithmetic, Binary codes, Kernel space, user space, Intro to secure shell (ssh) protocol, logging to the NERSC system and available resources. Students will learn the differences between processors, processes and threads, and will be introduced to multi-computer systems, and the client and server communication and computing model. Reading materials from [PH2018] and [IEEEFPS]

- Week 3. Will focus on an introduction to Software Engineering, Software engineering Scalability, Software engineering Productivity, Team Building, Onboarding and Code Review. Software for Equity, Software Longevity, Code Readability, Portability, Documentation and Licensing. Reading materials from: [WMW2020].

Module 2. Introduction to Algorithms and Performance Analysis.

This module covers algorithms and performance analysis. Students learn how to analyze and test algorithms, what are the properties of correct algorithms and to analyze the asymptotic behavior of algorithms. Students are also introduced to creating relevant tests for algorithms.

Learning Objectives:

- Students learn the critical elements of a well-written algorithm,
- Students learn how to analyze and run algorithms, and study their asymptotic behavior,
- Students deepen their understanding of complexity analysis, worst and expected case scenario,
- Students learn how to use plotting tools to compare the performance of different algorithms.

- Week 4. This week starts with Introduction to Algorithms, students learn how to analyze the behavior of algorithms through small data examples, they are introduced to Complexity Analysis (spatial and temporal), and identify the worst case complexity and expected case complexity. Reading materials from: [CLRS2022] .

- Week 5. Continues with the exploring of the asymptotic behavior of algorithms, introduces the amortized performance analysis, and students learn how to create tests and report performance using plots. Reading materials from: [CLRS2022]
  - Homework Assignment #1 is due on Friday of Week 5. Students analyze the complexity and performance of algorithms. They review a few typical errors working with scientific programming (e.g., round-off errors, NaN, etc)

Module 3. Introduction to Data Structures.

Introduction to basic data structures and their functionality. Students learn how to evaluate performance tradeoffs of using different data structures. Students write code using C++ classes to implement the data structures and meaningful tests.

Learning Objectives:

- Students learn basic data structures and their representations inside the computer's memory,
- Students learn hierarchical data structures and their properties,
- Students learn to apply object-oriented programming skills learned in Chem 274A to implement different data structures in C++,
- Students learn algorithms to parse hierarchical data structures.

- Week 6. Introduction to basic Data Structures; Vectors, Linked Lists, Queues, Priority-Queues, and Stacks. Reading materials from: [SMP2016] and [CLRS2022] .

- Week 7. Continues the explorations of Data Structures and Algorithms. Including an introduction to Binary Trees, AVLs, Hash Tables and Tree traversing Algorithms.
  - Homework Assignment #2 is due on Friday of Week 7. Student work on C++ implementations of algorithms using data structures (e.g. trees, priority queues and hash tables)

Module 4. Complexity and Performance Analysis.

This module furthers the students' understanding of complexity analysis through implementation of well-known sorting algorithms. Solving recurrence relations and comparing theoretical vs empirical performance analysis. They continue to also further their skills writing readable and well-documented software.

Learning Objectives:

- Students learn different sorting algorithms and implement them in C++,
- Students learn how to write meaningful tests cases for different algorithms and data structure implementations,
- Students learn how to evaluate different computational applications and algorithmic implementations.

- Week 8. Introduction to Sorting Algorithms; Insertion Sort, Selection Sort, Merge Sort and 3-way Merge Sort, Quick Sort and Heap Sort. Reading materials from: [SMP2016] and [CLRS2022] .

- Week 9. Continues the explorations of different data structures in the context of sorting algorithms. They learn how to measure the Performance of algorithmic implementations, write testing modules and measure the computational performance using well-defined metrics for time and space. Reading materials from: [CLRS2022] .

- Week 10. Introduction to Software Performance Optimization. Review the concepts of expected performance and randomized performance. Perform loop optimization, divide and conquer, recursion and meomization. Reading materials from: [CLRS2022] .

Module 5. Optimization and Graph Algorithms.

This module provides an introduction to algorithmic optimization techniques and their implementations. It also introduces basic graph data structures, their implementations and graph traversing algorithms.

Learning Objectives:

- Students learn about greedy algorithms and their implementation,
- Students learn about the Steepest Descent algorithm and its implementation,
- Students learn graph representations and their implementations using data structures,
- Students learn basic graph traversing and searching algorithms,
- Students learn about forming cross-functional teams.

- Week 11. Introduction to Greedy Algorithms and Steepest Descent algorithm. Introduction to the final project and cross-functional teams. Reading materials from: [CLRS2022].
  - Homework Assignment #3 is due on Friday of Week 11. Students work on C++ implementations of sorting algorithms and work on performance analysis and critique of the algorithms.

- Week 12. Introduction to Graphs: graph representations and graph traversing and searching algorithms: Depth First Search and Breadth First Search. Continue to work on the Final Project. Functional Team Work. Reading materials from: [CLRS2022].

Module 6. Introduction to Parallel Computing.

This module provides a brief introduction to code parallelization using OpenMP and MPI.

Learning Objectives:

- Students learn basic parallel computer concepts; data parallelism and instruction parallelism.
- Students learn how to parallelize code using OpenMP and run it on a supercomputer,
- Students learn how to parallelize code using MPI and run it on a supercomputer.
- Students learn how to install and test widely used software libraries like MPI on their own computers.

- Week 13. Parallel Programming I: Using OpenMP and cross-functional team work on final projects.
  - Homework Assignment #4 is due on Friday of Week 13. Students work on the C++ implementation of graph algorithms and Greedy algorithms. They practice best software engineering practices (e.g., code organization, distribution, build, and documentation)

- Week 14. Parallel Programming II: Using the Message Passing Interface (MPI), installing MPI in your own computers, programming a parallel application using MPI and running on multiple platforms. Cross-functional team work on the final project (Final Assignment).

Module 7. Debugging.

This module provides an introduction to debugging algorithms and C++ implementations. .

Learning Objectives:

- Students use the debugging tools they learned in Chem 274A (week 11) to debug and verify the behavior of different algorithms and applications.
- Students design input, output and processing components for their final project library, working in their cross-functional software development teams.

- Week 15. Debugging and code verification techniques. Final Project Cross Functional Team Work.

Final Project Assignment is due on Week 17 (during exam week). Students work in cross-functional teams in the development of framework to support Cellular Automata simulations. This assignment is due Wednesday of Week 17.

Section VIII. Course Completion Requirements.

Every student is required to view all of the online lectures (asynchronous lectures), complete online quizzes that foreshadow problem solving activities during the synchronous discussions. They also need to complete and submit all 4 homework assignments (40% of final grade), Final Project Assignment (40% of final grade) and attend all synchronous sessions.

Participation in activities and discussions during synchronous sessions account for 10% of final grade. Students that miss a synchronous session are required to watch the video and submit make-up work for the session to receive credit. Participation in activities and discussions during lab sessions account for 10% of final grade. Students that miss a lab session are required to watch the video and complete the exercises in the lab to receive credit.

Students can miss a maximum of 2 synchronous sessions, this includes a combination of synchronous Tuesday or Wednesday lab sessions. Thus, a student can have at most two make-up work during the semester. Faculty may consider special circumstances.

Students need a laptop or desktop computer with working audio and video devices. The computer needs to be able to UNIX-based commands, and access the NERSC supercomputer facility.

Section IX. Course Policy.

Attendance Policy:
The asynchronous lectures and synchronous check-in sessions are key to assess the students' progress with the materials covered in this course. Students are expected to participate in all synchronous sessions during the semester, as well as, watch all the asynchronous lectures online and complete the required background readings. Students are expected to attend the sessions on time.

Honor Code & Academic Integrity:
The community at UC Berkeley has adopted the following Honor Code: "As a member of the UC Berkeley community, I act with honesty, integrity, and respect for others." We should all strive to live up to this ideal and support others to do so as well. You should keep in mind that as a member of the campus community, you are expected to demonstrate integrity in all of your academic endeavors and will be evaluated on your own merits. So be proud of your academic accomplishments and help to protect and promote academic integrity at Berkeley. The consequences of cheating and academic dishonesty – including a formal discipline file, possible loss of future internship, scholarship, or employment opportunities, and denial of admission to graduate school – are simply not worth it. Anyone caught cheating in this course will receive a failing grade in the course and will also be reported to the University Center for Student Conduct.

Collaboration and Independence:
Unless otherwise instructed, homework assignments are to be completed independently and materials submitted as homework should be the result of one's own independent work.

Plagiarism & Ethics:
To copy text or ideas from another source without appropriate reference is plagiarism and will result in a failing grade for your assignment and usually further disciplinary action. For additional information on plagiarism and how to avoid it, consult these two resources for examples:

- **Academic Misconduct: Cheating, Plagiarism, and Other Forms**
- **Cite Sources**

Students with Disabilities:
If you need accommodations for any physical, psychological, or learning disability, please contact Heather Makiharju, Student Services Advisor at hmakiharju@berkeley.edu, or the **Disabled Students' Program.** An "individual with a disability" means any person who has a physical or mental impairment, which substantially limits one or more major life activities, who has a record of such an impairment, or who is regarded as having such an impairment.

Accommodation of Religious Creed:
It is the official policy of the University of California, Berkeley to permit any student to undergo a test or examination, without penalty, at a time when that activity would not violate the student's religious creed, unless administering the examination at an alternative time would impose an

undue hardship that could not reasonably have been avoided. Requests to accommodate a student's religious creed by scheduling tests or examinations at alternative times shall be submitted directly to the faculty member responsible for administering the examination. Reasonable common sense, judgment, and the pursuit of mutual goodwill should result in the positive resolution of scheduling conflicts. The regular campus appeals process applies if a mutually satisfactory arrangement cannot be achieved.

Disclaimer:

This syllabus is subject to change at any time by the instructor. Every effort will be made to give students ample notification of any changes.