University California, Berkeley

Master of Molecular Science and Software Engineering

Software Engineering Fundamentals for Molecular Sciences

CHEM 274B, 3 Units

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.

Course format: This course is designed as 13 weeks of: three hours weekly of Faculty-led, asynchronous, web-based instruction; two hours weekly of web-based synchronous discussion; and two hours of web-based, synchronous lab every other week. GSIs will go over homework assignments and practice exercises that are quantitative, prepare students for their homework assignments and post answer guides to homework assignments after they are submitted. Outside class work should comprise about four hours a week for a total of nine to eleven hours per week. Throughout the course, we will develop a series of assignments related to the development of a web-based platform where users can:

- Input molecular structures in different formats.
- Visualize the molecules in 2D and 3D.
- Calculate basic molecular properties.
- Predict more complex properties integrating pre-existing ML models.

- Store and retrieve molecular data.
- Access all functionality programmatically via an API.
- Monitor and receive user feedback for continued optimizations and integrations.

Project assignments, final projects, lecture exercises, and labs will enable development of the final product and highlight how each component can be extended to other applications in molecular sciences, with references to real products that use them.

Prerequisites: Prior exposure to basic programming methodology at the level taught through MSSE's Introduction to Programming Languages Bootcamp. Python and C++ programming language concepts covered in MSSE's "Programming Languages for Molecular Sciences: Python and C++," though not before they are covered in the referenced course.

Reading List and Resources:

Required and recommended readings will be posted weekly on the course website. Required readings (books, articles, etc.) will be free to access or available through UC Berkeley's library. Weekly readings will be designated as required or recommended, but reading all of them will provide the most benefit.

Learning objectives for this course: Upon successfully completing this course, students will be able to

- Build foundational knowledge of software engineering principles, data structures, algorithms, and optimization techniques.
- Develop skills to identify and integrate new technologies into a pre-existing codebase.
- Learn best software development practices in traditional and machine learning oriented projects at a collaborative level and with the end user in mind.
- Add practical projects to their MSSE software portfolio.

Grading: There will be 3 individual programming projects, asynchronous "lecture" exercises, and a group final project, in addition to participation in synchronous labs and discussion.

- Individual programming projects will reflect the student's individual software development work in the class and build up on the case studies introduced through asynchronous lectures. Each programming project will be due 4 weeks

after being assigned. The third individual programming project will be due at the same time as the group final project.
- Lecture exercises are interspersed between asynchronous video lecture material and are meant to simulate walking through solving a software development project as a collaborative pair of developers. Problems in the lecture exercise will be related to the development of a single, overarching software package that we will develop during the span of the course.
- Discussions and labs will extend covered concepts through short, practical scenarios. Attendance is required and discussion and lab assignments are scoped to be completed during the session.
- The final project ties together all material covered in the course and tasks students with replicating a software development product end-to-end, similar in scope to the overarching course project, with minimal aid from the instructor.

An online coding platform with an auto-grader will be used for the lecture exercises. Submissions will be reviewed by GSIs if necessary. Individual and final programming projects will be graded by GSIs or faculty.

The final grade breakdown is as follows:

- 36% programming projects (each project worth 12%),
- 25% final project,
- 15% lecture exercises,
- 15% programming labs,
- 9% synchronous session discussion and participation

Missed Synchronous Sessions: Missing synchronous sessions (discussion and lab) will result in no credit being awarded for the missed session unless the absence is pre-arranged with the instructor.

Late work: Late work will not be accepted for lecture exercises. For programming projects, late work will be accepted for a penalty of 10% per day until a week after the assignment due date. Programming projects turned in more than a week late will not be accepted without pre-approval from the instructor.

Accommodations for Disabilities UC Berkeley is committed to creating a learning environment that meets the needs of its diverse student body including students with disabilities. If you have a disability, or think you may have a disability, you can work with the Disabled Students' Program (DSP) to request an official accommodation. https://dsp.berkeley.edu/. If you already have an accommodation letter from DSP, please

check to make sure that the letter is submitted through the DSP system (there is no need to email a separate copy). If you would like to set up an individual meeting to discuss your accommodations, please contact the instructors.

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

Course requirements: Each student is required to view all of the online lectures, do all the online quizzes, submit all homework assignments, and attend discussion and lab sessions. A laptop, workstation, or access to a UNIX-style account is required, as is installation of an Emacs or VI editor.

Office hours: The instructors will be available 2 hours per week for one-on-one consultation during office hours. The GSIs will be available 4 hours a week. The course schedule and office hours will be posted on the course website. The instructors will also be available for synchronous open class discussion two hours per week.

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.

Course Schedule:

- **Module 1:** Weeks 1, 2, 3, 4, 5
- **Module 2:** Weeks 6, 7, 8, 9, 10
- **Module 3:** Weeks 11, 12, 13

Week #	Concepts	Product Status
1	Course introduction, expectations, and outline of course-long product. We will cover business requirement documentation and how to translate it to system design documentation, life cycle models, project planning and estimation, and diagramming (UML, user journey). By the end, we will have a mental map for the components that we will cover throughout the course and which components are out-of-scope for future learning. We will also have covered concepts to prepare for team-oriented software development (productivity, onboarding, code review, code readability, documentation, licensing, versioning)	Defined documentation on customer requirements, system design, and operational structure
2	Data structures for molecular representation. Strings, arrays, hash tables, stacks & queues, bloom filters.	Supports several molecular representation formats, conversion between multiple input/output formats, basic property calculations and similar molecule look ups
3 4 5	Algorithms for molecular analysis, such as substructure searching, molecular fingerprint, similarity calculations, SMILES/SMARTS processing, sorting applications, and compound library design. Analysis of algorithm complexity and performance. Advanced extensions of previously covered data structures to highlight object-oriented design pitfalls, antipatterns, and value of design patterns. Introduction to graph data structures for molecular representations and algorithms.	

6	RESTful API design, microservices architecture, relational vs NoSQL databases for molecular data, efficient querying of chemical databases	Web-accessible via UI along with API integrations
7	Principles of scientific data visualization, libraries for 2D and 3D molecular visualization, explanatory vs exploratory visualizations, how to lie (and detect liars) with visualizations	View molecule structure and plots of properties of uploaded molecules
8	Advanced graph algorithms (e.g., searching and traversing algorithms), greedy algorithms, optimization algorithms.	Improved performance of features and feature extensions
9	Basic parallel programming concepts in Python (Numba, Cython) and C++ (CUDA), processing large files and considerations to latency and scalability.	
10	Test driven development, unit testing framework, continuous integration, user-acceptance testing, integration testing, code debugging strategies.	Test set coverage of existing features
11	Basic machine learning concepts for molecular property prediction, integrating an opaque machine learning model, differences in traditional software project lifecycle compared to machine learning project lifecycle.	Integration of more advanced property prediction capabilities
12	Offline evaluation, online evaluation, experimentation principles and design, A/B testing, multi-armed bandits, statistical testing applied to software development.	Harness for continuous evaluation of property predictors & for evaluating new features or candidate models in production settings

13	Static deployment, dynamic deployment (device, server), deployment strategies, versioning, caching, model serving, model monitoring, model maintenance. Retrospective on course-long project and next steps. Discussion on future trends in software engineering and molecular sciences.	Set up tracking and protection against potential issues to the platform's reliability and scalability. Project retrospective to identify future directions.