1. Course Introduction and Scope

Lecturer: Paul Eggert

1.1 Software Construction vs Software Engineering

- CS 35L is about software construction.
- Prerequisite: CS 31 (basic algorithms, data structures, C++).
- CS31 covered basic, sequential code on single computer cores, which is now only one small part of modern software development.

1.2 Rapid Evolution in Software Development

- Previously, knowledge “half-life” in CS was ~15 years.
- Now decreasing due to rapid advances like machine learning.
- Students will constantly have to learn new software construction techniques throughout careers.

1.3 Generative AI and Software Construction

- Instructor is not a machine learning expert but emphasizes its growing influence.
- Tools like ChatGPT, Copilot, Cloud Code can help but cannot be fully relied upon.
- AI assistants’ current ~75% accuracy is not good enough for production code.
- Students must still know how to code and verify results manually.

---

2. Course Structure

2.1 Projects and Assignments

- Group Project (5 people per team) - Practical software construction experience.
- Individual Assignments (6 total) - Cover various technologies and programming skills.

2.2 Technologies Used

- Recommended: Node.js, React (JavaScript-based stack).
- Additional topics/tools: Bash, Emacs, Make, Git, Python.

2.3 Tools and Learning Objectives

- Learn to rapidly adopt and use new software technologies.
- Build competence in pragmatic programming rather than theoretical deep-dives.

---

3. Core Topics

3.1 File Systems

- Covered via the POSIX model.
- Linux is used as the base system.
- Topic includes:
  - File abstractions
  - Storage structure
  - Permissions and metadata

Table Example:

Concept	Definition
File	A named data container in a filesystem

Directory	A file that lists other files
POSIX	Portable Operating System Interface (standard)

3.2 Scripting

Definition:

Writing programs (scripts) used to automate tasks, often with interpreted or command-based languages.

Languages Covered:

- Bash (shell scripting)
- Emacs Lisp
- Python

Tools:

- Emacs: open-source text editor and development environment.
- Interactive shell operations.

Examples:

Tool	Use Case
Bash	Automating build tasks, scripting tools
Emacs Lisp	Writing scripts in Emacs
Python	General-purpose scripting

3.3 Build and Distribution

Concepts:

- Building: Converting source into executable programs.
- Distribution: Installing programs for use on other systems.

Tools:

- make
- npm (for JavaScript projects)

Challenges:

- Consistency across systems.
- Managing dependencies.

3.4 Version Control

Key Ideas:

- Track and manage changes to code.
- Support collaboration.

Tool Used:

- Git

Concepts:

Concept	Explanation
Commit	A set of changes saved to the repository

Branch	A separate line of development
Merge	Combining changes from different branches
Tag	A named commit usually used for release versions

3.5 Low-Level Debugging and Dynamic Linking

Focus:

- Slightly higher-level than CS33 topics.
- Emphasis on understanding compiled artifacts behavior.

Tools:

- GDB (GNU Debugger)
- Dynamic linkers

3.6 Client-Server Architecture

Model:

- Server holds shared state (e.g., database, schedules).
- Clients communicate with the server but not with each other.
- Communication is indirect and goes through a server.

Example Application:

UCLA Groundskeeping Scheduler

Component	Role
Server	Stores all task and personnel schedules
Clients	Users update/view schedules

---

4. Comparison to CS130 (Software Engineering)

4.1 Topics Not Included in CS35L

Not Covered Fully	Mentioned Briefly
Scheduling Projects	Integration
Thorough Testing	Configuration
Project Forensics	Data Management
CI/CD (e.g., Docker)	Security Basics
Large-Scale Deployment	Networked Systems

4.2 Brief Lectures Promised on:

- Prompt engineering
- Data management (e.g., schemas, ACID principles)
- Security basics
- Distributed architectures (1/8 of a lecture)
- Deployment considerations

---

5. Software Construction Project

5.1 Goals

- Create a usable app with real-world potential.
- Demonstrate quick uptake of unfamiliar technologies.

5.2 Constraints and Expectations

- No C++
- JavaScript is encouraged, not mandatory.
- Node.js + React stack suggested.
- Client-server model is required.

5.3 Example: Groundskeeping App

- Task: Manage and schedule groundskeeping teams.
- Must support dynamic assignments as people call in sick or trees fall.
- Includes client-server spec and data sharing.

---

6. Tools and Languages

6.1 JavaScript/Node/React

Tool	Role
Node.js	Backend server scripting
React	Front-end UI development

6.2 Bash and Shell Use

- Used for scripting and shell operations.
- Students must be familiar with shell commands.

6.3 Emacs

- Text editor and development ecosystem.
- Written partly by Eggert.
- Includes scripting using Emacs Lisp.

6.4 Python

- Used for nondemanding scripting tasks.
- High-level general-purpose language.

---

7. Practical Infrastructure

7.1 CSNet Environment

- Students will use school’s server network.

7.2 Login and Linux Usage

- SSH into CSNet servers.
- Understand command line interface (shell).
- Identify:
  - Shells: Command interpreters, e.g., Bash
  - Applications: Programs like Emacs/less

Example Commands:

Command	Action
ssh username@lnxsrv11.cs.ucla.edu	Log into CSNet
uname -r	Kernel version
cat /proc/cpuinfo	View CPU info
lsb_release -a	Show system info

---

8. Administrative and Logistics

8.1 Grading Weights

Component	Percentage
Assignments (6)	18%
Class Participation	1%
LA Feedback Surveys	0.5%
Course Evaluations	0.5%
Midterm Exam	18%
Final Exam	27%
Project (Group Work)	35%

8.2 Lateness Policy

Penalty increases geometrically:
P = 3^(n-1) if n is the number of days late

Days Late	Penalty (%)
1	1%
2	3%
3	9%
4	27%
5	81%

Hard deadline: Nothing accepted past Friday of Week 10.

8.3 Academic Integrity

- Submissions must be original.
- Avoid hard coding or copying answers.
- Do not submit content sourced from direct online solutions.

8.4 Using Generative AI

Usage allowed under constraints:
- Log every prompt and AI response.
- Submit logs and a reflection (“after-action report”).
- Log is required as part of your assignment submission.

8.5 Evaluation and Participation

- Use of Piazza encouraged for Q&A.
- Participation earns small bonus credit.
- Emphasis on asking questions and clarifying doubts.

8.6 Resources and Office Hours

- CSNet info, assignments, schedule located on official course page.
- Extra tutoring via UPE, office hours, peer discussions encouraged.

---

9. Primer on Emacs and Linux Basics

9.1 Shells and Processes

- Shell: Interface to execute programs (CLI).
- Bash: Default shell used.
- Processes: Running executable programs.

Process Tree:

- systemd → bash → less/emacs/subshell

9.2 Command Line Interface (CLI)

- Command line environment pointedly non-GUI.
- Efficient for repetitive tasks, scripting.

9.3 ASCII and Control Characters

- ASCII: 7-bit character set
- Control characters: Use bit manipulation
- Example:
  - Ctrl-A = 0x01 (clear upper bits)
  - Meta key: Used to set highest bit

9.4 Basic Emacs Controls

Command	Operation
Ctrl-X Ctrl-C	Exit Emacs
Ctrl-H	Help Menu
Ctrl-H K (Key)	Show function of a Keystroke
Meta-X	Run command by name

---

10. Summary

This lecture sets the groundwork for CS 35L, a course focused on modern software construction practices beyond simple algorithm design. It frames software development in practical terms, emphasizing real-world technologies, version control, scripting, debugging, system interaction, and manageable project design using client-server architecture. It also explores how generative AI tools are beginning to change the development workflow—though caution is advised when using them. Key infrastructure such as the Linux command line, Emacs, and common Unix tools will be explored in depth. The course structure balances individual exercises with a collaborative capstone project, providing both technical experience and exposure to core skills in pragmatic software engineering.