Instruct

Author: Yuri Tricys

Date: December 19th, 2025

PIDP:3270

# CAPSTONE PROJECT

OPEN SOURCE SOFTWARE DEVELOPMENT:

## BUILDING EMACS FROM SOURCE: REFLECTION

## The Vision Behind the Lesson

Designing and recording this Emacs lesson — "Building Emacs 30.2 from Source (Linux, WSL, and macOS)" — was as much about forming my identity as an instructor as it was about teaching a technical skill.

My motivation came from three places. First, I've used Emacs to leverage productivity in writing, coding, data analysis, and research for the last twelve years. I know how transformative it can be for creatives and knowledge workers bogged down in fragmented tools and clumsy workflows. Second, my Emacs related posts, on readingworldmagazine.com, are my highest performing posts, with over 150,000 views since 2019; and, among those, the post named How to Compile Emacs 29 From Source on Windows in 2022 is the highest performing post. Third, I wanted to develop a way to communicate with those who want to learn what I have to teach. Media enhanced learning offers an opportunity to reach beyond the physical constraints of the classroom, and that is sometimes necessary when working in challenging niche areas like programming.

As an educator-in-training, my aspiration is to make time-intensive activities many people find boring, like mastering Emacs, navigable and even enjoyable. The lesson plan reflected that ambition: clear objectives, scaffolded videos, a mix of conceptual explanation and live terminal work. The resulting eight-video series is far from perfect, but it represents a genuine attempt to blend serious technical rigor with accessible narrative-style teaching.

## The Learning Curve: Mastering Content And Technology

In high-knowledge, niche areas the learning curve is known to be steep; content mastery is a big part of the package. In the modern digital media area, media production also presents a steep learning curve.

On the content side, I was drawing on more than a decade of accumulated practice. That journey has included JavaScript, PHP, shell scripting, R programming, data science, data bases, servers, email, Docker, Ollama, external AI API tooling, and HTML and other web development related technologies, plus a lot of system administration and design work. The lesson draws directly on that background. However, despite the long runway, I still had to deepen and reorganize my knowledge to teach it. For example, I needed to narrow down more exhaustively on Ubuntu related expertise — since I recently switched formerly from Windows to Ubuntu — and sharpen my explanations around the compilation side of software engineering, as represented by the GNU Autotools suite.

On the technology-of-teaching side, however, the learning curve was for me even steeper. I didn't set out intending to become a part-time sound engineer and video editor, but that is effectively what happened. This series was produced while I was:

- Learning DaVinci Resolve and OBS.
- Experimenting with microphones, screen-capture setups, and audio chains.
- Building presentation skill that matches modern media expectations.

In retrospect, I made some suboptimal technical choices. For instance, I processed audio during recording through a chain of filters. This limited what I could fix later in DaVinci Resolve, which has superior post-processing tools. I now understand "clean capture, process in post", a philosophy that enables better sound and more flexibility.

In terms of learning progress, this dual learning curve came with both breakthroughs and frustrations. Some of the breakthroughs included:

- The first time I watched back a segment and thought, "This actually sounds like a teacher, not just a person talking."
- Successfully debugging and demonstrating a full build in Docker on Ubuntu 25.04.
- Seeing the Mayer multimedia principles from PIDP 3240 "click" in practice – using visuals plus audio, reducing extraneous details, and aligning narration with on-screen actions.

The frustrations were just as real:

- Wrestling with audio artifacts I couldn't fully fix.
- Feeling that each improvement in clarity or pacing exposed new flaws.
- Realizing how modest edits, like cuts, zooms, and overlays, which I had to exclude, are time-consuming when you're still learning DaVinci Resolve.

Nonetheless, the accumulated technical effort is now baked into my evolving identity as an instructor who can teach "through" media, not just with "media".

## Teaching Through the Screen: A New Form of Presence

One of the defining but limiting characteristics of this lesson is that it was designed for asynchronous online consumption. It's limiting because the course objectives delineate an in-person synchronous teaching presentation, but that option wasn't available to me. As a result, there is no real-time back-and-forth, no visible classroom, no immediate nonverbal cues.

It's defining because the publishable digital format expands access to target markets, those students who want to learn Emacs, and the activities it enhances. Access to publicly available publishing platforms is the one of the defining characteristic of the modern technological age. Aligning my teaching style with what's happening now in the social media space is both practical and sensible.

However, teaching for online consumption requires a different approach. Students are more content-focused when learning on-line. They click on the lesson because they want the knowledge and skill, and they are trading off time for that voluntarily. They want to get through the material to solve their problem and advance. There is also endless content they can consume – including content created by the best creators, with the best funding, from around the world. In theory, with the current social media model, every student could learn from a handful of teachers. To stay relevant, my material has to adapt.

That shifts my role. If their motivation is solid and the content is genuinely useful, then it's delivering quality that matters. If the material is abstruse or boring, entertainment matters more.

For that reason, I should consciously adopt techniques to simulate presence. This is still a work in progress for me. For example, I can improve on:

- Direct address: speaking to "you" rather than "you all," and phrasing instructions conversationally.
- Rhetorical questions: "Have you ever wondered what actually happens when you run `./configure`?" even though no one can answer in the moment.
- Narrative framing: using stories, like the robot-on-Mars metaphor, to create a shared mental story.

While I made some effort to "speak into" a social context, I've yet to fully capitalize on all the affordances of asynchronous teaching.

Teaching through the screen highlights both the strengths and limitations of media-enhanced instruction. It gives me a more controlled, content-centric environment, where social noise is reduced and the learner's motivation is foregrounded. At the same time, it challenges me to find new ways to anticipate confusion, model a reflective mindset, and build in opportunities for learners to check their own understanding without me in the room.

## Honest Assessment: Where the Lesson Excelled

### Presentation and Teacherly Voice

For someone relatively new to public speaking, it takes a surprising amount of effort to learn to *sound* like a teacher – speaking at a consistent pace, minimizing verbal tics, structuring explanations into digestible chunks, and modulating tone. Even though I see a lot of room for improvement, I made significant strides in a relatively short period of time. The bridge-in segments, in particular, worked well: they combined motivating language, music, and a clear framing of why Emacs and source builds matter, which I think will help gain learner's attention early on.

### Depth and Integrity of Knowledge

The foundation for this lesson is years of consistent practice, living inside Emacs and its surrounding ecosystem – writing code, taking notes, doing data analysis, and building reproducible setups. That background allowed me to: curate a realistic set of dependencies, instead of just copying a random recipe. It gave me sight to anticipate real issues, like aligning the `libgccjit` version with the `gcc` version, and explain some of the trade-offs between a Flatpak/Snap install or a source build.

I think I did pretty good in that area. The decades of knowledge and practice that made those explanations possible, aren't visible, but they matter.

### Use of Multimedia Principles

Mayer's multimedia principles, introduced in PIDP 3240, were actually helpful in designing the material.

For example, in terms of **Multimedia & coherence,** I was able to use slides and screen captures effectively alongside narration – though I'm still getting to avoiding excessive on-screen text. The code example format is workable, and adding syntax highlighting instead of bold text was a successful endeavor. In fact, I think the entire theme and color pallet works well.

Note: I didn't borrow it from David Wilson, at System Crafters. The blue background is necessary because of the technology. Based on my lighting equipment, it's more effective to use a blue screen for the chroma filter than a green screen.

In terms of **Contiguity & signaling,** I aligned spoken explanations with the region of the screen where the learner should focus (e.g., the terminal output during `./configure`). I tried to signal important ideas – like the relationship between features and dependencies – by returning explicitly to the metaphor used.

These design elements contributed to a more coherent and engaging experience, even within the limitations imposed when learning new technical, production related software suites.

## Honest Assessment: Where the Lesson Fell Short

When I hold the actual delivery up against the lesson plan and rubric, several gaps are clear.

### Gaps Relative the Lesson Plan

1. **Assessing Prior Knowledge and Experience**

   In the plan, the pre-assessment section is robust: I outline prerequisites, and encourage knowledge check-ins: "Have you used a terminal?", "Can you install with `apt` or `brew`?" I also suggest opening a build log before starting. In the recordings, I mentioned prerequisites and readiness but didn't consistently *invite* learners to pause and explicitly self-assess. It often came across as material presented *at* them rather than a reflection *with* them.

2. **Providing Practice and Feedback**

   The participatory activities described in the plan are quite strong: learners are supposed to list Emacs use-cases, draw a `configure → make → make install → run` pipeline, and document dependency installation results. In practice, I didn't mention any of that material. I was focused fully on achieving a vocal tonality that's easy to listen to, and then mastering the media technologies.

3. **Closing the Lesson and Post-Assessment**

   The lesson plan ends with thoughtful reflection prompts, like: expectations vs reality, describing the build pipeline in their own words, listing errors and debugging steps, checking `system-configuration-features`. I did offer a summary and a basic bridge-out, but I didn't fully leverage these prompts. I could have devoted a more explicit conclusion, saying something like "Pause the video now and answer these questions in your log;" and, "Here's an example of what your answers might look like."

**Technical and Production Issues**

I think my biggest technical weakness was audio. By heavily processing the mic signal in real time (EQ, gain, noise reduction, compression, expansion, noise gating), I effectively "baked in" artifacts I couldn't remove later in post-production with DaVinci Resolve. The post-production tools are powerful, and I didn't get to a level of expertise where I could use them.

I also have a long way to go in terms of adding *entertainment value* through micro-editing. Channels like Alex Ziskind's demonstrate how frequent cuts, zooms, subtle sound effects, and visual changes can maintain attention without feeling gimmicky. It's true, they're using production studio calibre equipment – you can tell from the audio mix – but today, some of that quality can be achieved with a better software subscription and a little hard work.

While my videos do approach a David Wilson style, compared to Alex Ziskind and team, they're static. There are fewer camera moves, fewer pattern breaks, and longer uncut screencasts. Given modern attention spans and the competition for learners' time, this is a real weakness, even if the underlying information is solid.

Part of this was simply time and skill. I had to switch from OpenShot to DaVinci Resolve, and while I learned a lot from instructors like Casey Faris and Daniel Batal – both of whom model a student-centered, dynamic style – I didn't have the technological experience to match professional levels of polish.

These gaps are just part of the process. It takes awhile to master delivering on-screen demos with studio production quality audio and presentation. A more seasoned instructor/editor combination might have implemented more effective engagement patterns and embedded richer formative checks.

## Looking Ahead: A Road Map to Excellence

This project significantly shaped my teaching philosophy. I'm more convinced than ever that:

- Deep technical content *can* be made accessible with careful structuring, good metaphors, and thoughtful scaffolding.
- Online, asynchronous teaching is not “second best” – it is a distinct mode with its own strengths, and it rewards deliberate design.
- Online, media skill (editing, audio design, pacing) is now intrinsic to effective teaching, not an optional add-on.

Looking ahead, I see three overlapping time horizons.

### 1. Short-Term Goals

#### Improve Editing And Pacing

Use DaVinci Resolve more intentionally to introduce captions, frequent but purposeful zooms, callouts on important commands, and subtle sound design. Aim for shorter, more focused segments with fewer “dead” moments.

#### Improve Editing And Pacing

Be more concise and move more quickly. In entertainment and online education, the first few seconds of each segment are critical; if something doesn’t work quickly, show an alternate path instead of doing a deep dive.

### 2. Medium-Term Goals

#### More Sophisticated Bridge-ins And Narrative Arcs

Introduce short, clever opening scenarios or “micro-stories” at the start of each video that frame the material in human, relatable terms, without sacrificing quality of the information. The aim is to match the kind of “plotlines” or hooks used by creators like Alex Ziskind while keeping a strong teacherly presence like, for example, Casey Faris or Daniel Batal might do, since we’re on the topic of Davinci Resolve.

#### Richer Learner Interactions

Build more structured prompts, like explicit pauses or guided note-taking. For example, use phrases like: “Write down your exact `./configure` line now”, and provide optional self-check quizzes or worksheets that pair with the videos.

### 3. Long-Term Goals

#### Digital Teaching Portfolio

Build a coherent body of open educational resources – lesson plans, videos, scripts, code repositories – that can sell. Not only can such resources bring in revenue, but they can connect us, to both students and clients, who work with similar tools in similar ways. The YouTube model favors financed production settings, but there is still accessibility there, through exceptional visual design, audio quality, and useful instructional patterns.

#### Thoughtful Monetization And Access Strategy

Explore models where core conceptual content remains free or open, while extended, project-based series, coaching-oriented material, or contracts are paid. AI shifts the technology frontier outward, which makes every field more complex. Learners who want to operate in those fields will need accessible material.

**Hybrid Models of Interaction**

Eventually, I'd like to combine asynchronous video with live in-office group activities, community forums, or team-based activities where both flexibility and interaction are available.

Ultimately, this experience moved me toward a teaching philosophy grounded in: high respect for the learner's time and attention, and a willingness to invest in production craft as a part of being an educator.

## Conclusion: Embracing the Process

Creating and delivering this Emacs build lesson shows that excellence in teaching is a process, not a destination. The series demonstrates real strengths: a solid conceptual foundation, deep lived experience with the tools, clear objectives, and genuine attempts to apply multimedia learning principles. Learners who follow it carefully can reach a meaningful outcome: a working, custom-built Emacs and a much clearer mental model of how open-source software can be assembled on their machines.

At the same time, the gaps are instructive. I need to:

- Make learner-focused prompts and self-assessment more explicit.
- Integrate practice and feedback more deliberately, not just suggest practice in passing.
- Tighten summaries and bridge-outs that speak directly to what the learner just did and how it might have felt.
- Raise the production value – especially audio and editing – to match the quality of the content.

Moving forward, my commitment is to keep refining along those lines: more concise explanations, more intentional pacing, smarter use of editing and visual emphasis, and closer alignment between the formal lesson plan and the lived experience of the learner on the other side of the screen. The central challenge is to keep the "flow" of the lesson moving while weaving in interactive touches that encourage the learner to pause, reflect, and act – without losing them in the process.

In that sense, this project achieved what a good capstone project should achieve. It nudged me to see myself not just as a subject-matter expert, who happens to record videos, but as an emerging educator whose medium, methods, and mindset all have to evolve together.

## References

[Photograph of a rocket] [Photograph]. (n.d.). Internet Archive. Retrieved December 10, 2025, from https://archive.org/

AI Models Used In This Report: Various AI models were used to research, collect, and verify data, format arguments, and grammatically structure content. Models used include: Qwen3-coder:30b-a3b-q4_K_M, Gemma3:27b-it-q4_K_M, Llama3.1:70b-instruct-q3_K_M, GPT 5 Nano, GPT 5.1