Capstone Assessment Rubric: How to Read It

Capstone Assessment Rubric — How to Read It

This document explains **how your capstone will be assessed** and, more importantly, **how to use the rubric to guide your work** *before* the final submission.

Please read this before you optimize for anything.

Capstone Assessment Rubric (https://canvas.ubc.ca/courses/185076/pages/capstone-assessment-rubric)

The Big Idea

This capstone is not graded on whether your system is "successful," polished, or impressive.

It is graded on **what you learned by building it**.

In particular, we care about:

- What you *committed to* early
- Where the system *pushed back*
- How your thinking *changed* as a result
- What tradeoffs you chose to live with

If nothing pushed back, you did not go far enough.

Why the Rubric Is Intentionally Conflicted

Real systems design is a series of tradeoffs. You cannot maximize everything at once.

The rubric reflects this reality. Each dimension pulls in a different direction. Optimizing one often means sacrificing another.

That is not a bug. That is the point.

High-quality projects do **not** try to satisfy every dimension perfectly. Instead, they:

- Make tradeoffs explicit

- Show evidence of cost or loss
- Explain *why* those tradeoffs were chosen

How the Capstone Is Structured

You have **three opportunities** to demonstrate your learning:

1. **Code artifact** (your Git repository)
2. **Design reflection** (written, after the code)
3. **Learning showcase** (video, after the reflection)

Your **final capstone score is the maximum of the three**, not the average.

This means:

- You are allowed to struggle early
- You are allowed to change your mind
- You are allowed to learn late

You should think of this as a *trajectory*, not a single performance.

How to Use TA Feedback

Throughout the term, TAs will periodically review student repositories and leave feedback as **GitHub issues**.

Important clarifications:

- TA feedback is **formative**, not a grade
- TAs are helping you notice what your system is already telling you
- You are not expected to “fix everything” immediately

If a TA asks a question or points out a concern, that is a signal that there is *something worth learning there*.

The questions TAs raise often point toward the qualities the rubric assesses. If a TA asks about a decision, that's a signal there's something worth articulating in your final reflection.

The Five Rubric Dimensions (Explained)

Final grading uses five holistic dimensions. These are not checklists.

1. Commitment vs Flexibility

Did you make concrete commitments early enough for them to be tested — and were you willing to revise them when reality disagreed?

High-quality work shows:

- Early assumptions that were specific enough to be wrong
- Evidence of revision, abandonment, or justified stability

Staying vague avoids learning. Refusing to revise avoids learning.

---

2. Depth vs Breadth

Did you go deep enough somewhere for real constraints to appear?

High-quality work shows:

- At least one subsystem explored in depth
- Conscious decisions about what *not* to explore

Trying to do everything usually means learning nothing deeply.

Depth must be visible—in your code, your reflection, or your video. If you went deep but didn't articulate what you found, we cannot assess it.

---

3. Control vs Realism

Did you allow the system to behave in ways that were inconvenient or surprising, while still being understandable?

High-quality work shows:

- Some simplification or modeling
- Some encounter with messy, real behavior
- Awareness of what is *not* controlled

"Messy, real behavior" includes both production-scale surprises (network issues, cost spikes, cloud quirks) and development-time friction (race conditions, API limitations, debugging dead ends). Either counts if you learned from it.

Perfectly clean systems often lie. Completely chaotic systems teach nothing.

---

4. Optimization vs Understanding

Did you delay optimization until you understood the system — and then explore at least one optimization or cost concern?

High-quality work shows:

- Awareness of performance, cost, or reliability tradeoffs
- At least one optimization attempt or cost model
- Reflection on what that attempt revealed

Premature optimization hides learning. Avoiding optimization entirely does too.

5. Narrative Coherence vs Epistemic Honesty

Did you explain your work clearly *without* smoothing away uncertainty or contradiction?

High-quality work shows:

- A coherent explanation across artifacts
- Explicit acknowledgment of mistakes, uncertainty, or unresolved questions
- Clear separation between belief, evidence, and speculation

A perfect story with no scars is suspicious.

One Required Habit (Applies Everywhere)

At least once in your project, you must be able to say:

> “We deliberately sacrificed **X** to preserve **Y**, and here is what that cost us.”

If no choice made anything harder or impossible later, you did not push far enough.

The tradeoff must be *substantive*—connected to the nature of your system, not merely to project logistics. "We sacrificed test coverage for features" is ordinary. These are substantive:

- "We sacrificed consistency to preserve latency under partition, which meant users occasionally saw stale data."
- "We chose a single-leader design to simplify reasoning about state, which meant failover required manual intervention."

Name a cost that reveals something about distributed systems, not just about time management.

What a Strong Project Often Looks Like

- It is not the biggest or flashiest system
- It has visible scars in the repo history
- It contradicts its own early assumptions
- It leaves some questions unanswered — on purpose

Different students will peak in different artifacts. That is expected.

Final Advice

If you are optimizing for the rubric, optimize for **contact with reality**.

Build something small enough to understand, but real enough to push back.

That tension is where the learning lives.