How to approach the homeworks

Key takeaways

1. v1 hands you its constraints, so an agent may pass quickly; learning starts there.
2. Make the agent explain what it built; verify the explanation against the repo.
3. Each later beat changes a seam your v1 map should already name.

Common misunderstandings of agent-built homework

Which claim would you defend right now? Choose before the evidence.

“If the v1 tests pass, the first beat is finished.”

“The agent wrote the code, so it already understands the design for me.”

“The next beat is just another implementation prompt.”

Keep your choice. Look for the mechanism or counterexample that would make it fail.

Real projects never start from a full spec

The homeworks arrive in beats because requirements are discovered by building

In real development	In the homeworks
Nobody can write the final requirements on day one	Only the current beat’s SPEC is frozen and buildable
Needs surface once v1 meets users and real load	The next beat unlocks only after your v1 works
You get direction—“we will need X”—not a finished spec	v1 warns what to design for; the beat’s exact demands stay sealed

The next beat is a comprehension test: predict what will strain before the agent edits.

source: assignment-design.md — the change beat and its reveal policy

Course v1 supplies what open-ended projects make you find

The assignments isolate the skills that come after problem discovery

In an open-ended project	In a course v1
Find a credible starting point	A starter repository and frozen API are provided
Decide acceptable behavior	The SPEC pins observable semantics
Invent acceptance evidence	Public tests and a harness define the floor
Bound scope and tradeoffs	Non-goals and a measured baseline constrain the work

The course is not claiming requirements are known in real life; it is moving you to the next skill.

source: assignment-design.md and the active homework SPECs

An agent may mostly one-shot v1

Example: email-index gives the agent a precise contract and a running test harness

A passing v1 can arrive quickly

```
$ mindex update --db mail.db corpus/
indexed 12,418 messages

$ pytest -q
all public tests passed
```

- Let the agent read the SPEC, repository instructions, code, and tests, then implement v1.
- The errors that remain are conceptual — a wrong reading of the spec, an edge case no public test touches — and the code “looks right and may even pass basic tests.”
- Once the tests pass, stop: ask it to trace what it built before you request the next beat.

Passing is where learning starts: the failures that remain are the ones tests cannot see.

source: Osmani, Saboo & Kartakis, The New SDLC With Vibe Coding (Google, May 2026)

Have the agent teach you the system it built

Ask for a map tied to this repository — questions shown for email-index

Map	Question for email-index	Evidence you should expect
Request path	How does mindex update reach the database?	Files, functions, and SQL
State	Where do messages and postings live?	Schema and keys
Invariant	Why should update equal a fresh build?	Transaction boundary and test
Failure	What is the smallest violating execution?	Counterexample or experiment
Change seam	What does in-place mutation strain first?	Exact assumption in v1

If the explanation cannot cite the repository, it is not teaching you this lab.

Ask for explain-back, not a summary

Force the agent to connect the spec, code, evidence, and counterexamples

Ask the agent to explain the implementation it actually produced

```
Trace one search from the CLI to the postings table.
Name the invariant that keeps index == corpus, and cite the exact
statements that commit a message together with its postings.
Construct the smallest crash point that would violate it.

Which query shape should this schema lose, and why?
When Resilience adds in-place mutation, which assumption breaks first?
```

A useful explanation predicts a failure and survives a random challenge.

Connect every claim to evidence

Running example: why should email-index update match a fresh build?

STUDENT → AGENT

"After I append one message and run update, why should searches match a fresh build?

Cite the requirement, the code path in this repository, the observed test, and one result that would prove the claim false."

asks for evidence

WHAT A USEFUL ANSWER CONNECTS

Requirement	SPEC.md: update must match a fresh build.
Code path	The answer names the functions and SQL that make this true here.
Evidence	The rebuild-equivalence check passes against the independent oracle.
Falsifier	A unique-token message that goes missing or appears twice disproves the claim.

The student verifies each link against the repository.

source: hw/email-index/SPEC.md and the rebuild-equivalence checks

When to specify and when to let the agent explore

You do not need to pin every loop

Pin this	Let the agent choose
observable behavior	helper function layout
schema invariants	boilerplate SQL call sites
performance budget	micro-optimizations after measurement
failure cases	test-file organization
non-goals	internal names that are easy to review

Pin what grading observes and what breaks silently; delegate what a review can cheaply catch.

Key Takeaways

1. A passing v1 is evidence to interrogate, not the finished assignment.
2. Explain-back with citations—path, state, invariant, evidence, change seam—is how v1 becomes your model.
3. Build your v1 map before the next beat — each sealed beat changes a seam the map should already name.