AIDD-30- DAY – CHALLENGE (DAY 2)

Part A

Question #1

a) **Why is using AI Development Agents (like Gemini CLI) for repetitive setup tasks better for your growth as a system architect?**

Answer:

Using AI Development Agents like the Gemini CLI is better for your growth as a system architect because it removes repetitive setup work and frees your mind for high-level system design. Instead of spending time creating folders, installing packages, or writing boilerplate code, you focus on architecture, tool flow, context design, routing, and decision-making.

AI agents also reduce human errors, create clean and consistent project structures, and match modern industry workflows. They make you fast, productive, and more capable of building complex multi-agent systems. By shifting your role from “manual coder” to “system orchestrator,” AI agents help you grow into a stronger and more efficient system architect.

b) **Explain how the Nine Pillars of AIDD help a developer grow into an M-Shaped Developer.**

Answer

The Nine Pillars of AIDD help a developer become an M-Shaped Developer by giving them depth in many different areas of AI-native development. Each pillar adds a new specialization — like agent building, tool use, context engineering, streaming, safety, model selection, multi-agent systems, develops automation, and productivity workflows. Together, these pillars create multiple vertical strengths plus broad horizontal understanding across the whole AI development lifecycle. This combination of multiple deep skills + wide knowledge transforms an ordinary developer into a modern, versatile, industry-ready M-Shaped Developer.

Question # 2

a) **Why does Vibe Coding usually create problems after one week?**

Answer:

Meaning: Vibe coding = coding without planning.

You just write whatever comes to your mind in the moment.

Vibe coding breaks after a week because it has no planning, no structure, and no consistency — so the project becomes confusing and full of errors.

It’s asking why coding without planning or structure becomes messy and starts breaking after some days.

Vibe coding creates problems after one week because the project has no planning or structure. At first it works, but soon the code becomes messy, confusing, and difficult to remember. When you try to add new features, everything starts to break because nothing was organized from the start. That’s why vibe coding only works for a short time and fails in longer projects.

b) How would Specification-Driven Development prevent those problems?

Answer:

Specification-Driven Development (SDD) prevents vibe-coding problems because it makes you plan everything before writing code. You create a clear specification that explains the project structure, files, tools, flows, and rules. This gives your code a proper roadmap, so nothing becomes messy or random.

With SDD:

- every feature fits perfectly because it follows the plan
- code stays clean, organized, and easy to extend
- fewer bugs appear because everything is structured from the start

In short: SDD gives direction, consistency, and clarity — stopping the problems that vibe coding creates.

Specification-Driven Development prevents problems because you plan first and code later.

You make a clear guide of what to build, how files will look, and how everything should work.

So the project stays organized, clean, and easy to update, instead of becoming messy like vibe coding.

Brief Explanation: planning stops problems before they happen.

Question #3

a) **How does architecture-first thinking change the role of a developer in AIDD?**

Answer:

Architecture-first thinking means you design the system before writing any code.

In AIDD (AI-Driven Development), this changes the developer's role from a coder to a planner and system designer.

With architecture-first thinking:

- you focus on flows, tools, agents, and structure, not just typing code
- AI handles the small coding tasks
- you make the big decisions about how everything will work
- your job becomes orchestrating, not manually writing every line

In short: the developer becomes a system architect who designs, guides, and controls the AI — not just someone who writes code.

b) **Explain why developers must think in layers and systems instead of raw code.**

Answer:

Developers must think in layers and systems because real projects are too big to understand through raw code only.

Layers help you break the project into clear parts (like tools, agents, routing, context, UI), so everything stays organized and easy to manage.

When you think in systems:

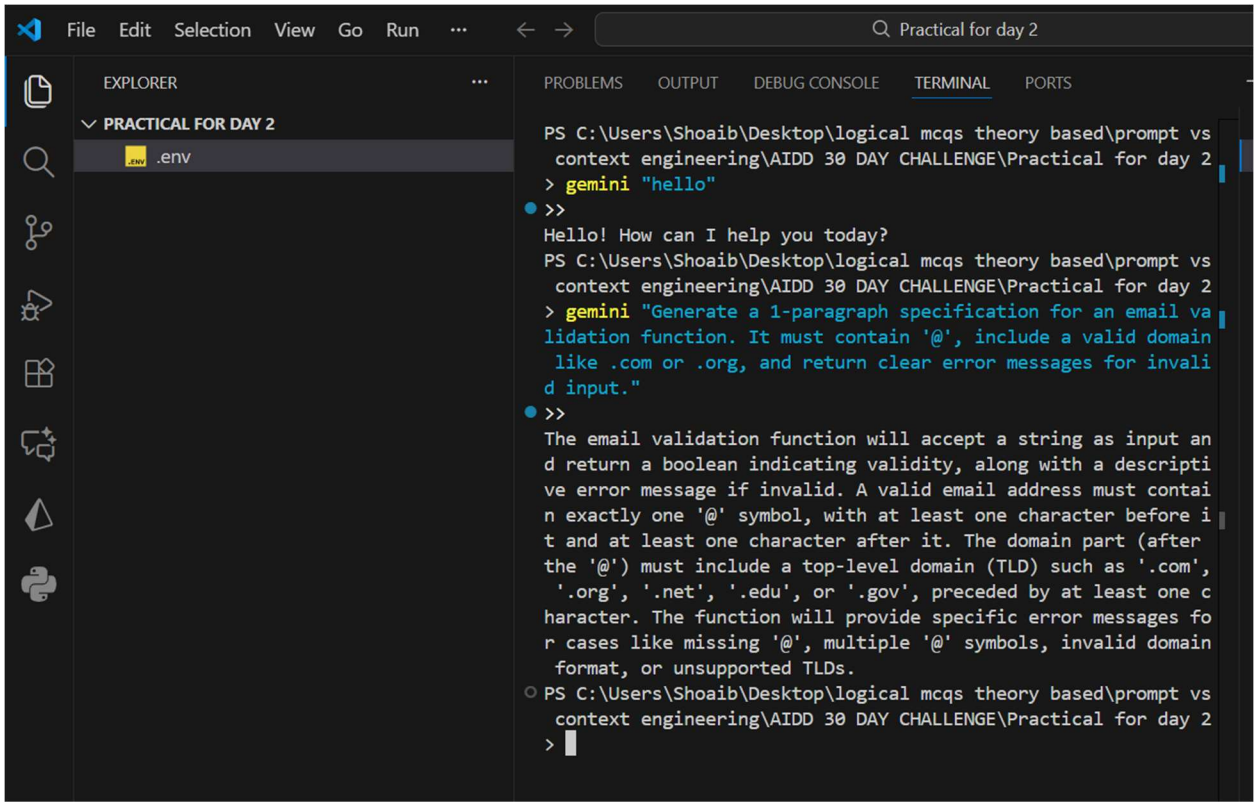
- you see how all parts connect
- you avoid confusion and messy code
- you can add new features without breaking old ones
- your project becomes easier to fix, test, and grow

In short:

Thinking in layers makes your project clear and structured.

Thinking in raw code makes it messy and hard to maintain.

PART B



EXPLANATION OF PART B:

1) **CLI Prompt (exact text you used)**

gemini "Generate a 1-paragraph specification for an email validation function. It must contain '@', include a valid domain like .com or .org, and return clear error messages for invalid input."

2) **CLI Output (Generated Specification)**

Specification:

The email validation function will accept a string as input and return a boolean indicating validity, along with a descriptive error message if invalid. A valid email address must contain exactly one '@' symbol, with at least one character before it and at least one character after it. The domain part (after the '@') must include a top-level domain (TLD) such as '.com', '.org', '.net', '.edu', or '.gov', preceded by at least one character. The function will provide specific error messages for cases like missing '@', multiple '@' symbols, invalid domain format, or unsupported TLDs.

PART C
MCQs

1. What is the main purpose of Spec-Driven Development?

A. Make coding faster

B. Clear requirements before coding begins ✓

C. Remove developers

D. Avoid documentation

Explanation: SDD ensures you plan everything first so coding becomes clean and error-free.

2. What is the biggest mindset shift in AI-Driven Development?

A. Writing more code manually

B. Thinking in systems and clear instructions ✓

C. Memorizing more syntax

D. Working without any tools

Explanation: AIDD focuses on designing flows and instructions, not typing raw code.

3. Biggest failure of Vibe Coding?

A. AI stops responding

B. Architecture becomes hard to extend ✓

C. Code runs slow

D. Fewer comments written

Explanation: Without planning, the project becomes messy and breaks when you add new features.

4. Main advantage of using AI CLI agents (like Gemini CLI)?

A. They replace the developer completely

B. Handle repetitive tasks so dev focuses on design & problem-solving ✓

C. Make coding faster but less reliable

D. Make coding optional

Explanation: CLI agents free your time from setup work so you can think like an architect.

5. What defines an M-Shaped Developer?

A. Knows little about everything

B. Deep in only one field

C. Deep skills in multiple related domains ✓

D. Works without AI tools

Explanation: An M-Shaped developer has many deep skill pillars + broad understanding.

Reflection:

This task highlights how rapidly the developer role is evolving in the AI-Native era. Concepts like the Nine Pillars, SDD, and Development Agents are not just tools—they represent a complete shift in how software is planned, built, and maintained. By practicing architecture-first thinking and learning to collaborate with agents, developers can grow from code writers into system-level thinkers, eventually becoming M-Shaped professionals who combine multiple deep skills with intelligent tooling.