


AI-Driven Development — 30-Day

Challenge

Task 2 — Official Submission

 Class Slot: Friday — 6:00 PM to 9:00 PM

 Instructor: Sir Hamzah Syed

Uzma kanwal

Part A — Theory (Short Questions)

1. Nine Pillars Understanding

Q1 .Why is using AI Development Agents (like Gemini CLI) for repetitive setup tasks better for your growth as a system architect?

Ans. Using AI development agents (like Gemini CLI) for repetitive setup tasks helps your growth as a system architect because it frees you from manual, low-value work and lets you focus on high-level thinking. Automating setups saves mental energy, reduces mistakes, and speeds up project initialization. This helps you practice designing better architectures instead of wasting time on boilerplate. It also trains you to give clear instructions—an essential skill for architects—while keeping your workflow consistent and efficient. Overall, AI agents help you think smarter, work faster, and grow into a more strategic, big-picture engineer.

Q2 .Explain how the Nine Pillars of AIDD help a developer grow into an M-Shaped Developer.

Ans. .The Nine Pillars of AIDD help you grow into an M-Shaped Developer because each pillar builds a different skill you need to be strong in many areas, not just one.

Some pillars make you better at thinking and planning (like prompting, task breakdown, and design).

Some make you better at building and improving code (like code generation, refactoring, and testing).

Some help you work faster and smarter (like automation and AI agents).

Some help you communicate and learn better (like documentation and continuous learning).

When you grow in all these areas together, you gain multiple strengths instead of only one deep skill.

That combination is what makes you an M-Shaped Developer—strong in many areas, not limited to a single specialty.

2. Vibe Coding vs Specification-Driven Development

Q3. Why does Vibe Coding usually create problems after one week?

ans. Vibe Coding breaks after a week because it has no plan. You write whatever feels right in the moment, so the code isn't organized. After a few days, it becomes hard to understand, hard to change, and full of small mistakes.

Q4. How would Specification-Driven Development prevent those problems?

Ans. Specification-Driven Development prevents those problems because you decide what to build before you start coding. You write a simple plan that explains the feature, the rules, and how it should work. So instead of guessing later, you always have a guide to follow. This keeps your code clean, organized, and easy to understand—even after a week.

3. Architecture Thinking

Q5. How does architecture-first thinking change the role of a developer in AIDD?

Ans. Architecture-first thinking changes a developer's role in AIDD by shifting focus from just writing code to designing smart systems.

Developers plan the structure first—how components interact, where AI fits, and how data flows. Coding becomes about implementing the plan, not guessing or patching later. Developers start thinking like architects, making decisions that keep the system scalable, maintainable, and efficient. AI tools are used strategically, not randomly, to speed up work without creating messy code. Instead of just coding, the developer becomes a system thinker and problem solver, designing robust systems with AI support.

Q6. Explain why developers must think in layers and systems instead of raw code.

Ans. Developers must think in layers and systems because software is more than just code. Layers keep code organized, make it easier to maintain, and help parts work well together. Thinking in systems lets you build software that scales, stays reliable, and is easy to update, instead of messy code that breaks quickly.

Part B — Practical Task (Screenshot Required)

Task:

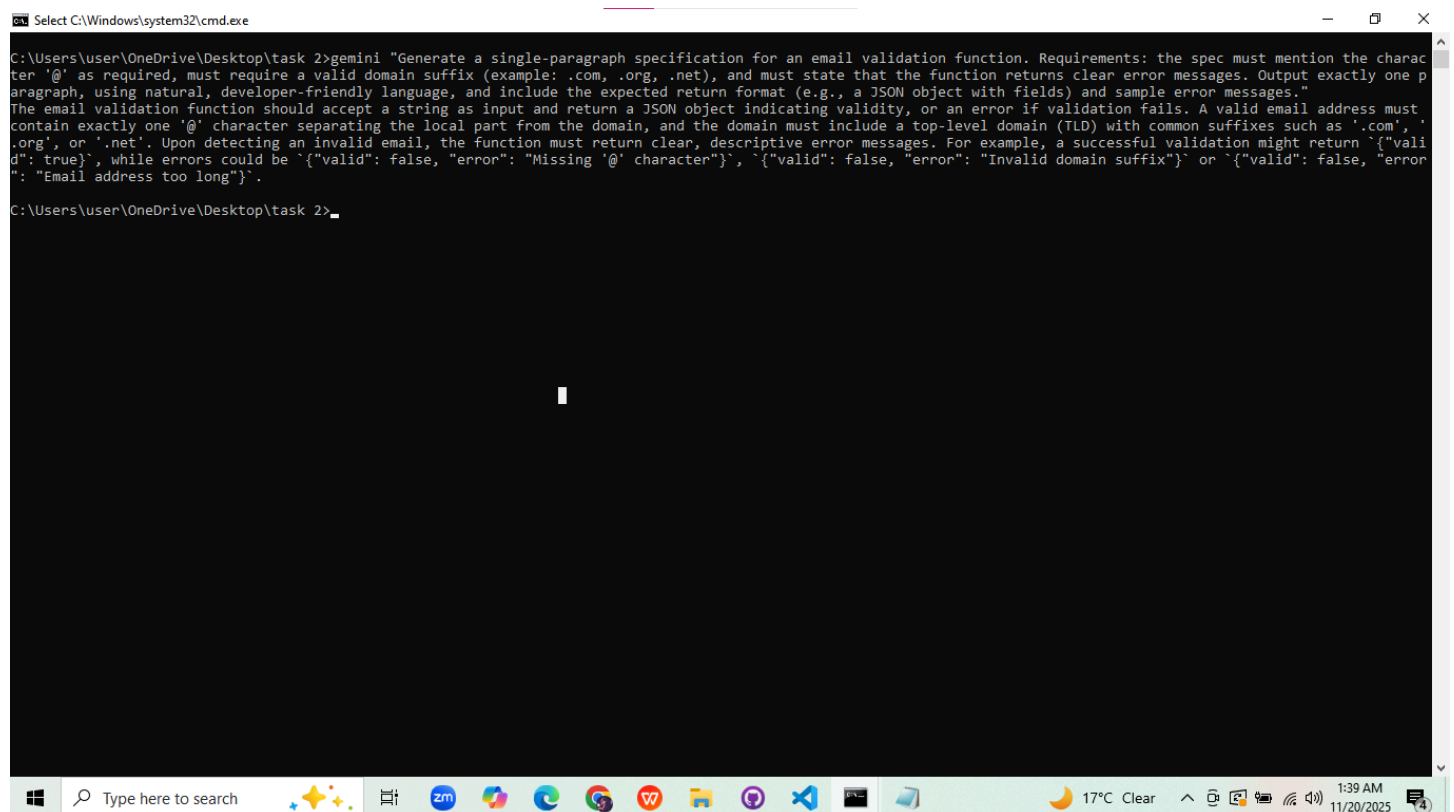
Using any AI CLI tool, generate a 1-paragraph specification for an email validation function.

Requirements:

Must contain “@”

Must contain a valid domain (e.g., .com, .org)

Should return clear error messages



```
Select C:\Windows\system32\cmd.exe
C:\Users\User\OneDrive\Desktop\task 2>gemini "Generate a single-paragraph specification for an email validation function. Requirements: the spec must mention the character '@' as required, must require a valid domain suffix (example: .com, .org, .net), and must state that the function returns clear error messages. Output exactly one paragraph, using natural, developer-friendly language, and include the expected return format (e.g., a JSON object with fields) and sample error messages."
The email validation function should accept a string as input and return a JSON object indicating validity, or an error if validation fails. A valid email address must contain exactly one '@' character separating the local part from the domain, and the domain must include a top-level domain (TLD) with common suffixes such as '.com', '.org', or '.net'. Upon detecting an invalid email, the function must return clear, descriptive error messages. For example, a successful validation might return '{"valid": true}', while errors could be '{"valid": false, "error": "Missing '@' character"}', '{"valid": false, "error": "Invalid domain suffix"}' or '{"valid": false, "error": "Email address too long"}'.
```

Part C — Multiple Choice Questions

1. : B. Clear requirements before coding begins
- 2.B. Thinking in systems and clear instructions
- 3.B. Architecture becomes hard to extend
4. B. Handle repetitive tasks so dev focuses on design & problem-solving
- 5.C. Deep skills in multiple related domains