

AIDD 30-Day Challenge — Task 2 (Complete Submission)##

Asma Yaseen & Yusra Saleem Class Coordinator
Under the Supervision of Sir Hamzah Syed

##By: Darakshan Anjum##

##Date : 18-11-2025##

Part A — Theory (Short Questions)

Answer in your own wording

1. Nine Pillars Understanding

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

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

2. Vibe Coding vs Specification-Driven Development

Why does Vibe Coding usually create problems after one week?

How would Specification-Driven Development prevent those problems?

1. **AI CLI & Coding Agents** are command-line AI tools (like Claude Code, Gemini CLI, Copilot CLI) that work directly inside your terminal and interact with your actual codebase. These tools are revolutionary because, for the first time, a developer isn't working alone — the AI can read your entire project, run commands, generate code, fix bugs, and even build features just from natural language.

2. **Markdown as a Programming Language** means that in Specification-Driven Development, your Markdown specs act like real “source code.” You write your requirements in simple, structured English, and AI agents read those specs and turn them into actual implementations. This is revolutionary because it removes the old barrier where humans had to translate ideas into strict programming syntax.

3. **The MCP Standard** is a universal way for AI agents to connect to tools, databases, and services without custom integrations. Instead of writing new code every time you want an AI agent to talk to a system, MCP gives you one common interface that works everywhere. It's maintained by Anthropic and used by agents like Claude Code, and developers can build MCP servers for anything—from APIs to cloud services.

4. **AI-First IDEs** like Zed and Cursor are development environments built specifically for constant collaboration with AI. Unlike older editors where AI was just an add-on, these tools are designed so you can talk to the AI, refactor code, generate features, and navigate your project all

through natural language. Zed focuses on speed and multiplayer AI, while Cursor specializes in inline AI code editing.

5.** Linux Universal Dev Environment** makes the command line consistent everywhere—Windows (via WSL2), Mac, Linux, and cloud IDEs. This means AI agents can run the same shell scripts on any machine without worrying about platform differences, making setup and automation smooth and predictable. Tools like WSL2, macOS Terminal, Docker, and Codespaces all support this. It’s essential for AI CLI agents, SDD workflows, and cloud deployment, ensuring scripts and containers work everywhere.

6.** Test-Driven Development (TDD)** is about writing tests first, then coding to pass them. With AI-generated code, TDD is crucial because it automatically verifies correctness and documents expected behavior. Tools like pytest, Jest, and JUnit let AI write both tests and implementation. This approach builds confidence: the AI can iterate until all tests pass, making code reliable, safe, and self-documenting.

7.** Specification-Driven Development (SDD) with SpecKit Plus** makes Markdown specs the “source of truth.” Instead of humans translating requirements into code, AI agents read well-structured specs and implement them directly. SpecKit Plus provides templates, workflows, and tools to manage these specs, ensuring code stays aligned with requirements.

8.** Composable Vertical Skills** are reusable expertise modules for AI agents, like a “Django security best practices” skill. They let AI agents apply deep domain knowledge without needing every developer to be an expert. This democratizes specialist knowledge, integrates with AI agents, MCP tools, and SDD workflows, and runs reliably in standardized Linux environments.

9.** Universal Cloud Deployment** makes production-ready deployment accessible using Docker, Kubernetes, Dapr, Kafka, and Ray. AI agents can generate deployment configs, scale apps globally, and connect seamlessly to CI/CD pipelines.

ANS. Vibe Coding becomes a problem after a week because the code has no clear plan. You keep adding things on top of each other, and soon the structure gets messy, hard to debug, and even harder to extend. What felt fast at the start turns into confusion later.

Specification-Driven Development avoids this because you decide the rules first. You write down what the system should do, how it should behave, and how parts should connect. With a clear map, the code stays organized, predictable, and easier to grow no surprises later.

3. **Architecture Thinking**

How does architecture-first thinking change the role of a developer in AIDD?
Explain why developers must think in layers and systems instead of raw code.

ANS. Architecture-first thinking turns a developer into a system designer, not just a coder.

Instead of jumping straight into writing functions, you start by understanding how the whole system should behave. Its flows, boundaries, risks, and future needs. This makes your decisions smarter and your work more strategic.

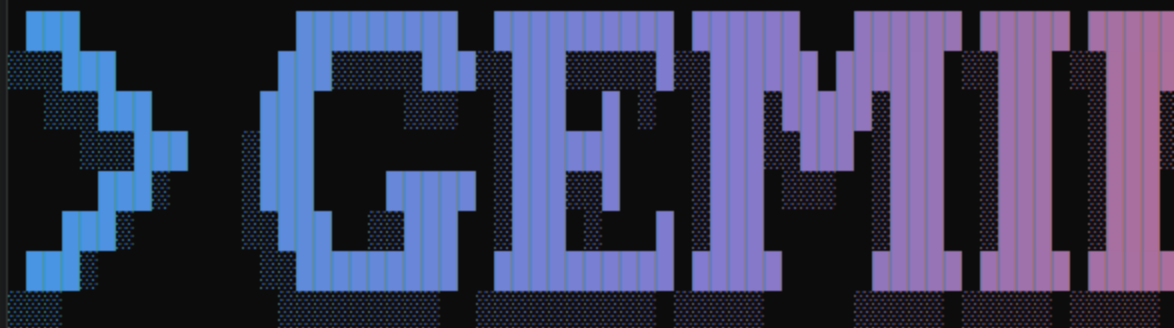
****Developers must think in layers and systems because real projects are bigger than individual lines of code.****

Layers give structure, separation, and clarity. Thinking in systems helps you understand how pieces talk to each other, how data flows, and how changes in one part affect everything else.

This mindset leads to scalable, stable, and long-lasting software.

Part B — Practical Task (Screenshot Required)

Windows PowerShell



Tips for getting started:

1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. Create GEMINI.md files to customize your interactions with Gemini.
4. /help for more information.

You are running Gemini CLI in your home directory. It is recommended to create a GEMINI.md file in your home directory to customize your interactions with Gemini.

> Type your message or @path/to/file

English (United States)

“Generate a one-paragraph technical specification for an email validation function. The specification should clearly state that the email must contain an '@' symbol, must include a valid top-level domain such as .com or .org, and must provide meaningful and user-friendly error messages when validation fails.”



```
Windows PowerShell

> GEMINI

Tips for getting started:
1. Ask questions, edit files, or run commands.
2. Be specific for the best results.
3. Create GEMINI.md files to customize your interactions with Gemini.
4. /help for more information.

> generate a one paragraph specification for an email validation function.the specification clearly state that email must contain an '@' symbol must include top-level domain such as .com or .org, and must provide meaningful and user ferirndly error messages when messages fails.

The email validation function validate_email(email_address: str) -> Tuple[bool, Optional[str]] will rigorously check if a given string constitutes a valid email address. It must ensure the presence of a single "@" symbol separating the local part from the domain, and verify that the domain includes a top-level domain (e.g., ".com", ".org", ".net") with at least two characters following the last dot. The function will return a tuple: (True, None) for a valid email, or (False, "Error Message") for an invalid one, where the error message is clear, concise, and user-friendly, indicating the specific reason for validation failure (e.g., "Missing '@' symbol", "Invalid domain format", "Missing top-level domain").

You are running Gemini CLI in your home directory. It is recommended to run in a project-specific directory.

> | Type your message or @path/to/file

no sandbox (see /docs) auto
```

Part C — Multiple Choice Questions

1. B.
2. B
3. B
4. B
5. C