TASK 02

30 DAYS CHALLENGE

FRIDAY BATCH (6:00 PM to 9:00 PM)

Submitted by:

Darakhshan Imran

Roll No. 00430657

Part A — Theory (Short Questions)

1. Nine Pillars Understanding

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

   AI-Driven Development (AIDD) is a specification-first methodology built on nine foundational pillars that work together as an integrated system. Now the role of developer transformed from user of AI tools to the orchestrator of an AI augmented development system.

   Using AI development agents for repetitive setup tasks is better for growth as a system architect because these agents remove the time spent on mechanical work and allow deeper focus on understanding systems, layers, and architecture. When AI handles tasks like project setup, configuration, boilerplate code, and environment preparation, more attention can be given to how different parts of the system connect and how decisions impact the overall design. Through the nine pillars of AIDD—like Specification-Driven Development, AI CLI Agents, and Universal Cloud Deployment—developers get to see entire workflows executed from a higher perspective instead of getting stuck in low-level setup steps. This helps build architectural thinking early, because the focus shifts from "typing code" to "designing systems," analyzing flows, writing clear specs, and understanding how tools integrate through standards like MCP and reusable vertical skills. The developer grows not by repeating setups but by observing, refining, and guiding the structure of the software as a whole.

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

   An M-shaped developer has little deep knowledge in multiple domains and the Nine Pillars of AIDD help a simple developer to transform in to an M-shaped developer. Because by adopting AIDD the developer focuses on the architect / system design and simultaneously able to understand how each pillar supports other domain.

   An M-shaped developer can be a backend expert who can integrate frontend with backend successfully and also deploy the final product on cloud with proper management of CI/CD and monitoring and also integrate language models. Hence an M-Shaped developer has deep knowledge in multi domains which includes frontend, backend, DevOps and AI/ML expert simultaneously.

   All the nine pillars of AIDD remove some barriers to transform a I-shaped or T-shaped developer into an M-shaped developer. The summary of each pillar how it helps the developer to be an M-shaped developer is discussed below:

- AI CLI & Coding Agents help to get expert-level help instantly—even for things developer never learned.
  Example: AI can write complex Kubernetes configs for you.

- Markdown as a Programming Language: Developer can write what he wants in simple English/Markdown, and AI converts it into working code.

- MCP Standard (Model Context Protocol) is a universal way to connect tools so developer don't need to learn every tool's API.

- AI-First IDEs enable a developer to use smart editors that deeply understand the project and give intelligent, context-aware suggestions.

- Linux Universal Dev Environment because one Linux-based environment that works everywhere from the laptop to the cloud.

- Test-Driven Development (AI Enhanced) through the help of this pillar the developer tests and type hints to stay correct, even when working in unfamiliar technologies.

- Specification-Driven Development enables developer to describe clearly about the task and AI generates consistent, high-quality code based on written specs.

- Composable Vertical Skills enable developer to install ready-made expert "skill modules" instead of learning entire domains yourself.

- Universal Cloud Deployment deploy complex infrastructure with one click no DevOps specialist required.

3. Vibe Coding Vs Specification-Driven Development?

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

Vibe coding is an instinctive development style which is based on intuition rather than conscious reasoning and logical analysis. Therefore, vibe coding starts to cause problems after the first week of development.

Usually, vibe coding starts without any written specifications and proper validation testing. Hence, initially the code works, the result is achieved, and the product is shipped by the developer confidently. But after that, the real twist happens when the maintainability and scalability of the code are compromised.

Even without written specs, collaborators have difficulty understanding other developers' work and ideas. Though vibe coding is good when the stakes are lower and the developer does it for learning and exploration purposes in solo projects, it is not suitable for production-level projects. Because due to the lack of proper specs and test validation, the code could crash, which questions the credibility of the developer or the company.

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

SDD inverts the order of workflow of vibe coding. In Specification Driven Development the developer begins with writing specification that make very clear the goal of building of a project. The developer knows what is he building and what the purpose and how the final project looks like.

After defining proper specs the developer writes tests that was the last step in vibe coding and results in failure of features when project scale-up.

Then the last step of implementation comes. That is performed by the AI agents as a co-partner that refactor and cleanup the implementation while testing each required and defined feature at every step this approach improves the overall structure of the project with clarity and high performance.

As SDD starts with proper spec document therefore it avoids human friction and ambiguity among the collaborators regarding the project, everything is written and clear to all co-partners even to the developer as well.

2. Architecture Thinking

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

In AIDD era the role of a developer is totally transformed in to a system architect from a conventional coder.

Now the developer first thinks about design system and write specification and validating the quality. AI agents work as co-partners that generate code, test and implement with continuous refinements until the results achieved.

A system designer is an orchestrator who use the AIDD pillars that increases the efficiency of a developer exponentially.

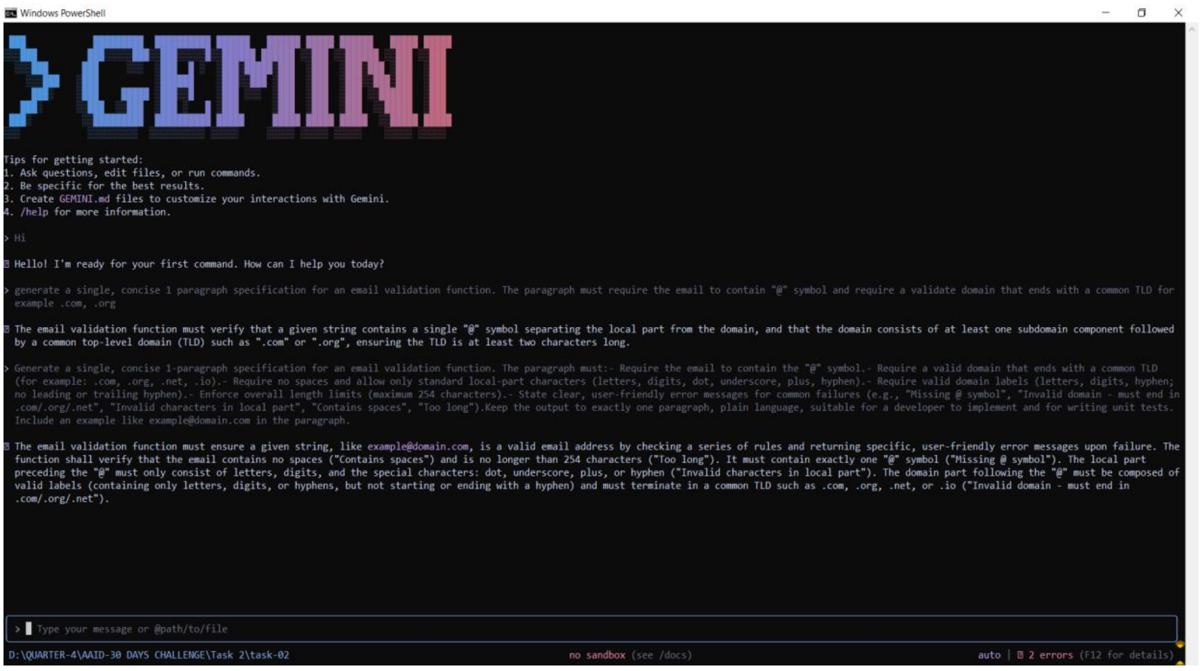
AI-Driven Development (AIDD) is a specification-first methodology that transforms developers into specification engineers and system architects.

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

Adapting AI Driven Development shifts the focus of a developer from writing raw code towards the designing the whole system that based on pillars of AIDD and increase the speed of delivering the task.

In AI-Driven Development, thinking in layers and systems becomes essential because the nine pillars shift the developer's role from writing raw code to designing how the entire product works. With AI CLI agents, Markdown-based specifications, and AI-first IDEs generating most of the low-level code, the focus naturally moves toward defining clear architecture instead of manually typing implementation details. The MCP standard, Linux universal environments, and universal cloud deployment also standardize how different tools and platforms connect, which encourages viewing a project as a set of interacting layers rather than isolated files. Test-driven and specification-driven development add structure by requiring developers to think about behaviors, boundaries, and responsibilities before code is written. Composable vertical skills further reinforce system-level thinking by letting developers plug in expert modules instead of reinventing deep technical layers. Altogether, these pillars push developers to design strong, scalable systems where each layer has a clear purpose while AI handles the code inside those layers making system thinking far more important than raw coding.

Part B — Practical Task (Screenshot Required)



```
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.

> Hi

Hello! I'm ready for your first command. How can I help you today?

> generate a single, concise 1 paragraph specification for an email validation function. The paragraph must require the email to contain "@" symbol and require a validate domain that ends with a common TLD for
  example .com, .org

The email validation function must verify that a given string contains a single "@" symbol separating the local part from the domain, and that the domain consists of at least one subdomain component followed
  by a common top-level domain (TLD) such as ".com" or ".org", ensuring the TLD is at least two characters long.

> Generate a single, concise 1-paragraph specification for an email validation function. The paragraph must:- Require the email to contain the "@" symbol.- Require a valid domain that ends with a common TLD
  (for example: .com, .org, .net, .io).- Require no spaces and allow only standard local-part characters (letters, digits, dot, underscore, plus, hyphen).- Require valid domain labels (letters, digits, hyphen;
  no leading or trailing hyphen).- Enforce overall length limits (maximum 254 characters).- State clear, user-friendly error messages for common failures (e.g., "Missing @ symbol", "Invalid domain - must end in
  .com/.org/.net", "Invalid characters in local part", "Contains spaces", "Too long").Keep the output to exactly one paragraph, plain language, suitable for a developer to implement and for writing unit tests.
  Include an example like example@domain.com in the paragraph.

The email validation function must ensure a given string, like example@domain.com, is a valid email address by checking a series of rules and returning specific, user-friendly error messages upon failure. The
  function shall verify that the email contains no spaces ("Contains spaces") and is no longer than 254 characters ("Too long"). It must contain exactly one "@" symbol ("Missing @ symbol"). The local part
  preceding the "@" must only consist of letters, digits, and the special characters: dot, underscore, plus, or hyphen ("Invalid characters in local part"). The domain part following the "@" must be composed of
  valid labels (containing only letters, digits, or hyphens, but not starting or ending with a hyphen) and must terminate in a common TLD such as .com, .org, .net, or .io ("Invalid domain - must end in
  .com/.org/.net").

> Type your message or @path/to/file
D:\QUARTER-4\AAID-30 DAYS CHALLENGE\Task 2\task-02          no sandbox (see /docs)          auto | 2 errors (F12 for details)
```

Part C — Multiple Choice Questions

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