

AI-Driven Development — 30-Day Challenge Day#2

Part A — Theory Questions

Question 1: Nine Pillars Understanding

Q1a: Why is using AI Development Agents (like Gemini CLI) for repetitive setup tasks better for your growth as a system architect?

Answer: Jab hum AI Development Agents ko repetitive tasks dete hain jaise project setup, boilerplate code, configurations wagera, toh hamara time bachta hai. Isse hum apna focus high-level thinking pe laga sakte hain jaise system architecture design karna, scalability sochna, aur different components ko efficiently connect karna.

Agar hum din bhar folders bana rahe hain aur packages install kar rahe hain, toh hamara dimag architect level ki thinking nahi kar payega. AI agents yeh boring kaam sambhal lete hain aur hum real problem solving pe focus kar sakte hain. Isse hum gradually architect mindset develop karte hain instead of sirf coder banne ke.

Q1b: Explain how the Nine Pillars of AIDD help a developer grow into an M-Shaped Developer.

Answer: Nine Pillars ek complete ecosystem create karte hain jisme developer ko multiple domains explore karne ka mauka milta hai. Pehle ek developer sirf ek cheez seekh pata tha kyunki time kam hota tha. Lekin ab AI agents repetitive kaam handle karte hain toh developer ko time milta hai different areas explore karne ka.

For example, agar main backend developer hoon, toh AI CLI tools mujhe DevOps, cloud deployment, testing automation jaise areas mein bhi kaam karne ki capability dete hain. Specifications likhna sikhta hoon, architecture

design karna seekhta hoon, agents ko orchestrate karna seekhta hoon. Isse naturally 2-4 domains mein deep expertise develop hoti hai aur main M-Shaped developer ban jata hoon instead of sirf ek skill pe atka rehna.

Question 2: Vibe Coding vs Specification-Driven Development

Q2a: Why does Vibe Coding usually create problems after one week?

Answer: Vibe Coding mein hum bina planning ke seedha code likhna shuru kar dete hain. Shuru mein sab theek lagta hai kyunki project chhota hota hai aur sab dimag mein hai. Lekin ek hafta baad jab project badh jata hai toh problems shuru hoti hain:

- Code ka structure unclear ho jata hai
- Kahan kya function hai yaad nahi rehta
- New features add karna mushkil ho jata hai kyunki architecture planned nahi tha
- Bugs dhundna difficult ho jata hai
- Agar kisi aur ko code dikhao toh wo samajh nahi pata

Basically, bina foundation ke ghar banaya hai toh wo gir jayega. Code messy aur unmaintainable ban jata hai.

Q2b: How would Specification-Driven Development prevent those problems?

Answer: Specification-Driven Development mein hum pehle clear specifications likhte hain ke kya banana hai, kaise behave karega, kaunse cases handle karne hain. Isse kai fayde hote hain:

- Pehle hi clear hai ke system kya karega toh code structure sochke banate hain
- Koi confusion nahi hota ke feature kaise kaam karega
- Agar baad mein koi bug aaye ya change karna ho toh spec dekh ke samajh jaate hain.

- Team mein kaam karna easy ho jata hai kyunki sabko specification se clear hai kya banana hai..
- Architecture planned hota hai toh extend karna ya modify karna simple rehta hai.

Simple words mein: Pehle blueprint bana lo, phir construction karo. Seedha construction mat shuru karo.

Question 3: Architecture Thinking

Q3a: How does architecture-first thinking change the role of a developer in AIDD?

Answer: Architecture-first thinking se developer ka role code writer se problem solver aur system designer ban jata hai. Pehle developer sirf code likhta tha jo task diya gaya. Ab developer pehle sochta hai:

- System ko kaise design karun ke future mein scale ho sake
- Different components kaise interact karenge
- Kaunsi technology stack best rahegi
- Security, performance, maintainability kaise ensure karun.

AI agents repetitive coding handle karte hain toh developer ko time milta hai architecture pe focus karne ka. Developer ab blueprint banata hai aur agents usse implement karte hain. Ye role zyada strategic aur high-level ho jata hai, jisse developer apni value zyada badha sakta hai.

Q3b: Explain why developers must think in layers and systems instead of raw code.

Answer: Layers aur systems mein sochna isliye zaruri hai kyunki modern applications complex hote hain. Agar hum sirf raw code mein sochenge toh:

- Sab kuch tangled ho jayega (spaghetti code)
- Ek part change karne se poora system break ho sakta hai

- Different teams kaam nahi kar paayengi efficiently
- Testing aur debugging bohot mushkil ho jayegi

Jab hum layers mein sochte hain (jaise frontend layer, backend layer, database layer, API layer) toh:

- Har layer apna kaam independently karti hai
- Ek layer change karo toh dusri layer pe impact kam hota hai
- Code reusable aur maintainable banta hai
- Different developers different layers pe parallel kaam kar sakte hain

Systems thinking se hum dekhte hain ke overall application kaise kaam kar raha hai, na ke sirf chhote code pieces. Ye approach large-scale applications ke liye bahut important hai.

Part B — Practical Task

```
PS C:\Users\arab> gemini hello
(node:5880) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:9932) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
Hello! How can I help you today?
PS C:\Users\arab> gemini "Create a detailed 1-paragraph specification for an email validation function that must check if email contains @ symbol, validate domain extensions like .com or .org, and return clear error messages"

(node:15028) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
(node:5244) [DEP0040] DeprecationWarning: The 'punycode' module is deprecated. Please use a userland alternative instead.
(Use 'node --trace-deprecation ...' to show where the warning was created)
An email validation function, 'validate_email', will be developed to accept a single string, the email address, as its input. This function must perform a series of checks to ascertain the validity of the email format. First, it will confirm the presence of exactly one "@" symbol, which is essential for separating the local part from the domain. Following this, it will analyze the domain to ensure it includes a valid top-level domain (TLD) extension, such as ".com", ".org", or ".net", by checking against a comprehensive list of recognized TLDs. The function is required to return specific, user-friendly error messages that clearly state the reason for failure—for instance, "Error: Missing '@' symbol," "Error: Invalid or unsupported domain extension," or "Error: The email address cannot be empty"—if any validation rule is violated. Should the email successfully pass all checks, the function will return a boolean value of "True" to indicate that the format is valid.
PS C:\Users\arab>
```

Part C — Multiple Choice Questions

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

Answer: B – Clear requirements before coding begins.

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

Answer: B – Thinking in systems and clear instructions.

3. Biggest failure of Vibe Coding?

Answer: B – Architecture becomes hard to extend.

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

Answer: B – Handle repetitive tasks so dev focuses on design & problem-solving.

5. What defines an M-Shaped Developer?

Answer: C – Deep skills in multiple related domains.

Reflection

This assignment has helped me understand how AI-Driven Development is transforming the role of developers. The Nine Pillars framework shows that modern developers need to think beyond just writing code – they must focus on architecture, specifications, and system design while letting AI agents handle repetitive tasks.

By practicing Specification-Driven Development and using tools like Gemini CLI, I can see how developers are evolving from code writers to system architects. The concept of M-Shaped developers makes sense in this context, as AI tools enable us to develop deep expertise in multiple complementary domains rather than being limited to a single skill.

