MoDev

VIBE CODING

UPCOMING TALK

JULY 31

8:00AM EST

FOR PRO DEVELOPERS

TAMMY MCCLELLAN, SOLUTIONS ARCHITECT

WHAT IS VIBE CODING?

WHAT IT'S LIKE TO VIBE CODE

TOP 5 TIPS FOR CONTEXT ENGINEERING

Tip #1 - Sharpen the Axe" with a Solid PRD

"As Lincoln said, if I had six hours to chop down a tree, I'd spend the first four sharpening the axe. In context engineering, your PRD is that axe."

🔍 Why It Matters

- Copilot is only as good as your context: A well-crafted PRD gives Copilot the structured, domain-specific context it needs to generate relevant, accurate code.
- Reduces ambiguity: Clear requirements reduce the risk of hallucinations or misaligned suggestions.
- Improves prompt engineering: You can extract reusable prompts directly from the PRD (e.g., function descriptions, edge cases, constraints).

🧠 What to Include in a PRD for Copilot Context

- Problem Statement: What are we solving and why?
- User Stories or Scenarios: Helps Copilot understand the “who” and “how.”
- Functional Requirements: Specific features, inputs/outputs, and behaviors.
- Non-Functional Requirements: Performance, security, scalability.
- Edge Cases: These are gold for Copilot – they help it avoid generic assumptions.
- Glossary or Domain Terms: Especially useful for industry-specific language.

Tip #2 - "Big ideas need small steps – especially when you're coding with Copilot."

A detailed PRD gives you the *what* and *why* – breaking features into small tasks gives you the *how*.

- Each task becomes a **micro-context** that Copilot can reason about more effectively.
- Think of the PRD as the blueprint, and your tasks as the bricks – Copilot lays them better when they're well-shaped.

🧠 Why It Matters

- Copilot performs best when the scope of the prompt is narrow and specific.
- Smaller tasks reduce the chance of hallucination or irrelevant suggestions.
- It's easier to test, verify, and iterate on smaller code units (see Tips #2 and #3).

🛠️ How to Apply It

- Break down features into:
  - Single-responsibility functions
  - Isolated UI components
  - Discrete API endpoints
- Use comments to describe each task:

// Create a function that validates email format using regex

- Use GitHub Issues or ADO Work Items to track and describe each task – these can even be copy-pasted into Copilot Chat as context.

Tip #3 – Trust but Verify

"Copilot is confident – but confidence isn't correctness."

- Why It Matters
- GitHub Copilot is trained on vast amounts of code, but it doesn't *understand* your business logic, security constraints, or edge cases.
- It can generate syntactically correct but semantically flawed code – especially when context is thin or ambiguous.
- Trusting Copilot blindly can lead to bugs, vulnerabilities, or subtle logic errors.

🛠️How to Apply It

- Always review generated code like you would a junior developer's pull request.
- Run tests – unit, integration, and edge case scenarios – to validate behavior.
- Use linters and static analysis tools to catch issues Copilot might miss.
- Ask Copilot to explain its code in Copilot Chat:

"Explain what this function does and whether it handles null inputs."

Tip #4 – Always Test Everything

"If Copilot writes the code, you write the tests — or better yet, ask Copilot to write them too."

🧠 Why It Matters

- Copilot can generate code that *looks* right but behaves incorrectly under certain conditions.
- Without tests, you're flying blind — especially when Copilot introduces logic you didn't explicitly ask for.
- Testing is how you *close the loop* on context engineering: you gave it context, now verify the output against that context.

🛠️ How to Apply It

- Write tests for every Copilot-generated function, especially those involving:
  - Business logic
  - User input
  - External APIs
  - Security-sensitive operations

  "Add edge case tests for null, undefined, and empty inputs."
- Test the tests: Review them for coverage and correctness – Copilot can hallucinate here too.

Tip #5 - Use Conversation Flow

"Don't just prompt – *converse*. Copilot Chat works best when you treat it like a teammate, not a vending machine."

🧠 Why It Matters

- Copilot Chat is designed for **iterative, contextual dialogue** – not one-shot prompts.
- The more you build on previous turns, the more Copilot understands your intent, constraints, and style.
- This mirrors how real-world pair programming works: through back-and-forth clarification, refinement, and exploration.

🛠️ How to Apply It

Start with a clear ask, then follow up naturally:

“Write a function to parse CSV data.”

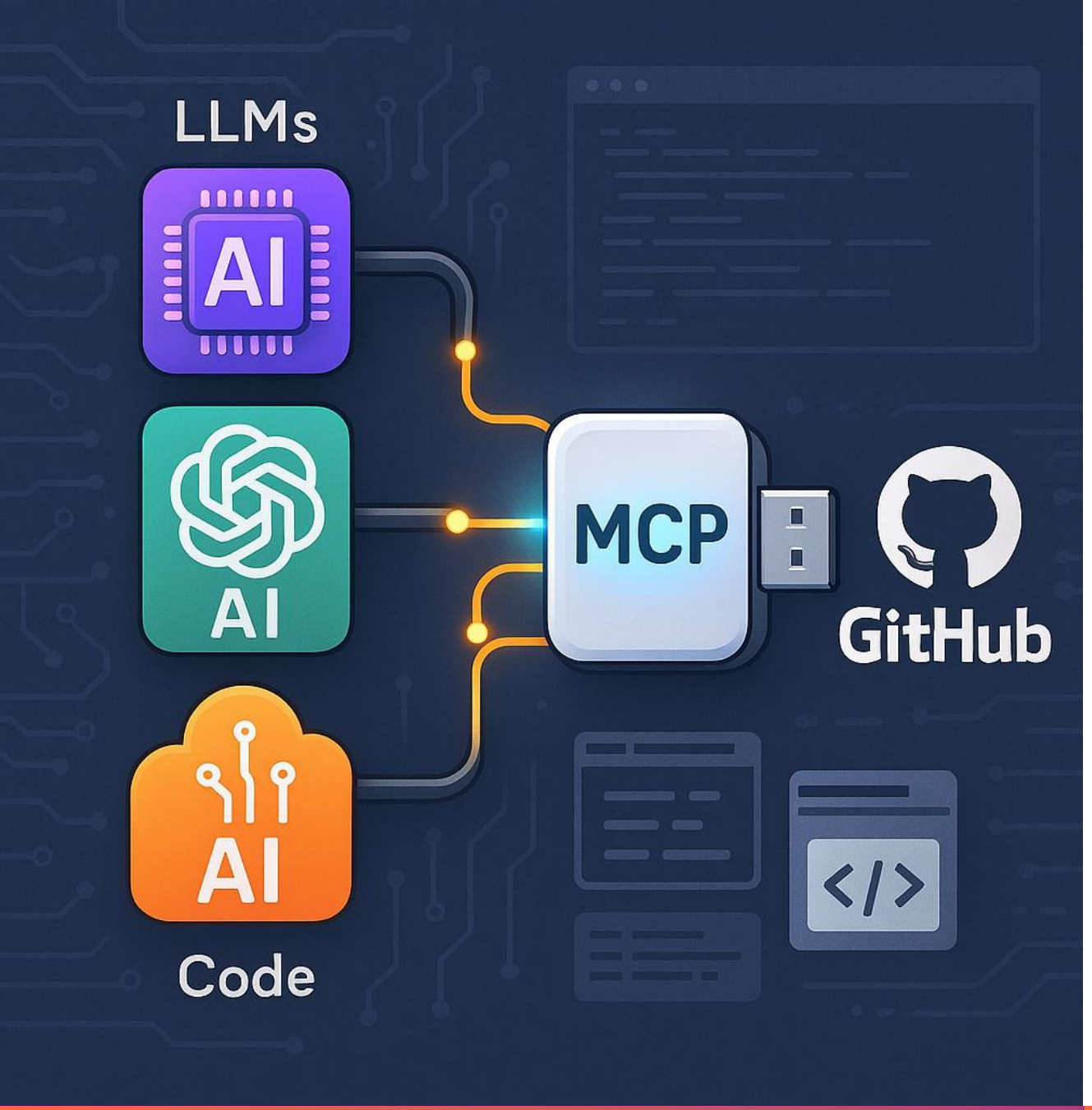
“Now make it handle quoted fields.”

“Add error handling for malformed rows.”

Conversation flow is how you layer context over time.

It’s especially powerful when combined with:

- Prompt files (Tip #2)
- Small tasks (Tip #4)
- PRD references (Tip #1)



WHAT IS MCP?

1. LLMs can't action. They Predict
2. LLMs + Agents – Now we have predictability + actions
3. MCP (Universal Language)
4. https://code.visualstudio.com/mcp
5. Created by Anthropic
6. Clients/Servers
7. Agents can use MCPs too

What is the Model Context Protocol (MCP)?

MCP is an open protocol that enables seamless integration between **AI apps & agents** and your **tools & data sources**. It solves the integrations problem much like the standards before it.