Good Vibes Agentic Coding Tips

Data Driven WI Conference 8/13/2025

Author: Derek Riley

- Rule/prompt evaluation mechanism
  - **T**ask-Specific Instructions
  - **A**rguments
  - **R**eusable Process Steps
  - **G**uided Examples and References
  - **E**xplicit Output Requirements
  - **T**emplate-Based Naming
  - **E**rror Handling and Edge Cases
  - **D**ocumentation
- Refactoring
  - Try adjusting the persona
    - *"As a senior software engineer who is an expert in maintainable java code…"*
  - Be specific with your goals
    - *Bad: "Refactor this"*
    - *Good: "Refactor the following function to improve its readability and maintainability (reduce repetition, use clearer variable names)."*
  - Refactor in steps- too much code can create distractions
- Debugging
  - Provide your intuition if you have it on the error
    - *"I suspect the root cause is due to the mishandling of XYZ edge case"*
  - Error messages can be copied/pasted, just ask the LLM that is the output you get and see how it responds
  - When something isn't working, provide the input and what you are observing
  - Make sure the LLM has the context of what the code is supposed to be doing
  - LLMs can easily add debugging output
  - Performance/benchmarking output or progress is also an easy ask
  - Some errors don't have error messages
    - Strategy: Ask the LLM to walk through the approach line-by-line
    - Example: *"Walk through this function line by line and track the value of total at each step. It's not accumulating correctly – where does the logic go wrong?"*
  - Sometimes focusing the LLM on a block of code can lead it to find a bug
  - Consider changing LLM Roles
    - Ask the LLM to be a "code reviewer" to identify mistakes or bad practices
- System Testing
  - *"Act like a user of this application trying to break it. Create different inputs that test error checking. "*
  - Best practice:
    - Have your system prompt instruct the Agent to write tests before writing code
    - Then only generate code that passes those tests
- Which LLM should I use?
  - Artificialanalysis.ai

Anti-Patterns

- Vague prompting
    - Not enough context
    - Symptom: hallucination
    - Cure: context + clarity
- Overloaded prompt
    - Too much complexity, detail, scope
    - Symptom: incomplete, incorrect implementation
    - Cure: reduce scope/context, break down tasks
- Missing a call to action
    - Providing context without a clear call to action at the end
    - Symptom: Not doing what you expect
    - Cure: Make a clear, direct ask at the end of your prompt
- Buffet Coding, 1 of everything
    - Dumping all files into the context window, letting the LLM find the needle in the haystack
    - Symptom: Long generation times, lots of tokens consumed
    - Cure: Pre-select code chunks for inclusion in the context window
- Lacking directory structure
    - Letting the Agent create the directory with minimal or no guidance
    - Symptom: Files with strange names in strange locations
    - Cure: Create a folder, naming template, and any other structural guidance using standard naming conventions before prompting
- Vague completion expectations
    - Insufficient definition of “done”
    - Symptom: Incomplete/incorrect code
    - Cure: Make success criteria/tests very clear
- Ignoring explanations
    - Accepting edits without scanning provided justification
    - Symptom: Not reading output
    - Cure: Pay attention!
- Positional references
    - Referring to code as “above” or “previous”
    - Symptom: Incorrect/incomplete implementation
    - Cure: Use specific file/method/variable/etc names
- Over specificity
    - Too much detail, assuming AI doesn’t understand common concepts
    - Symptom: Overly detailed prompts with definitions of common concepts
    - Cure: Build familiarity with prompting to know what level of detail is necessary
- Fixing generated code
    - Generated code is buggy, close but not quite
    - Symptom: Your generated code is close but often requires manual tweaks
    - **Cure: Fix the process and context, not the code**