Best Practices for Working with AI Agents: A Verification-Driven Approach

Working effectively with AI agents requires a fundamental shift in how we approach development. While AI can generate vast amounts of code instantly, the primary challenge is no longer authorship, but **verification**. Modern software engineering with AI is less about crafting the "perfect prompt" and more about maintaining a **disciplined, step-by-step process**.

Here is a comprehensive guide on how to optimally interact with AI agents, supported by real-world examples.

1. The Mindset Shift: Engineering Over Prompting

In the AI era, your core value shifts from typing speed to three essential competencies: **Problem Definition, Decomposition, and Verification**.

- **You are the Architect, AI is the Typist:** You are entirely responsible for the logic, security, and data flow.
- **Avoid the "One-Shot Trap":** A common beginner mistake is the "5-second high"—asking the AI to generate a complete application from a single sentence. This creates a massive **technical debt of understanding**. If you cannot verify the output, you do not own the code, making it a liability rather than an asset.

2. Precise vs. Imprecise: The Power of Constraints

Your prompts must be **highly precise when it comes to rules, constraints, and edge cases**. Ambiguity is the enemy of secure AI-generated code.

- **Example: The Server-Side Cart Calculator** If you simply ask an AI to "build a shopping cart," you risk getting vulnerable client-side logic where a user could manipulate prices. Instead, you must define a strict trust boundary where the server is the single source of truth. A precise prompt establishes rigid constraints:

- **Ignore Client Prices:** Explicitly state to never accept a price sent from the browser.
- **Validation Constraints:** Define mathematical rules, such as $Quantity \ge 1$ and $Tax/Discount \ge 0$.
- **Order of Operations:** Mandate that discounts must be applied *before* calculating tax.
- **Precision:** Require the system to round money to 2 decimal places (or use integers/cents to avoid floating-point errors).

3. Short vs. Long Prompts: The Iterative Workflow

Instead of writing one massive prompt, the most effective strategy is **iterative prompting**. Start with a structured, medium-length prompt to define the goal and constraints, then transition to short, highly focused commands to build and refine the output incrementally.

- **Example: Rapid UI Iteration via Short Prompts** During the development of a real-time events dashboard, a developer used extremely short prompts to polish the UI once the foundational context was established by the AI.
  - The developer prompted: *"make the bar chart smaller and horizontal. different colors for channels. randomly assigned."*.
  - Because the AI already understood the established architecture, this short prompt was enough for the agent to formulate a highly detailed implementation plan—creating a compact horizontal layout and using a deterministic hash-to-color function so that channels kept a stable pseudo-random color across page reloads.
  - The developer then followed up with rapid micro-prompts like *"increase font contrast of labels of bar chart"* and *"the colors of the background and the overlays do not match the dark"*. The AI executed these perfectly by tuning CSS overlay tokens and applying theme-aware colors.

4. The 7-Step Verification Loop: Trust but Verify

Never assume the agent's first output is flawless. You must **treat AI-generated code like code from a stranger—useful, but untrusted until proven by tests**. Fundamentals matter more than ever: security, data flow, and edge-case thinking are your primary tools.

To ensure quality and maintain control, adopt this repeatable **7-Step Iterative Loop**:

1. **Define the Goal:** State the objective in one clear sentence.
2. **Establish Rules:** List the non-negotiable technical constraints (what *must* be true).
3. **Provide Examples:** Define the exact expected Input $\rightarrow$ Output mappings.
4. **Identify Edge Cases:** List "weird" or bad situations the system needs to handle.
5. **Request a "Small Piece":** Ask for a specific function or logic gate, not the whole system.
6. **Demand Tests:** Require the AI to provide runnable assertions to prove its logic.
7. **Iterate:** Treat failing tests as data. Use them as a "flashlight" to refine your next prompt and fix ambiguities in your rules.

- **Example: Actively Verifying System Logic** Verification isn't just about automated tests; it's also about actively questioning the AI's architectural decisions. When the AI added data filters to a dashboard's charts, the developer didn't just accept the code. They verified the logic by asking: *"how do the filters work? do they filter visible data or data on the server?"*. Only after the AI confirmed that the filters were applied securely on the server side via SQL query parameters, did the developer instruct the AI to solidify this architecture: *"document the changes do you?"*, ensuring the verified logic was permanently recorded in the project's README.

Revision #1

Created 2026-03-13 11:13:11 UTC by Carsten

Updated 2026-03-13 11:23:35 UTC by Carsten