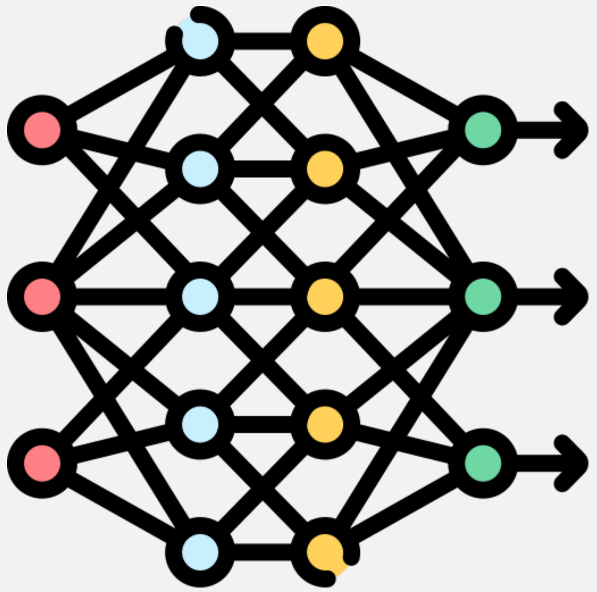
CEN 5016: Software Engineering

Spring 2026

University of Central Florida

Dr. Kevin Moran

Week 6 - Class I: A Software Engineer's Guide to LLMs Pt. II



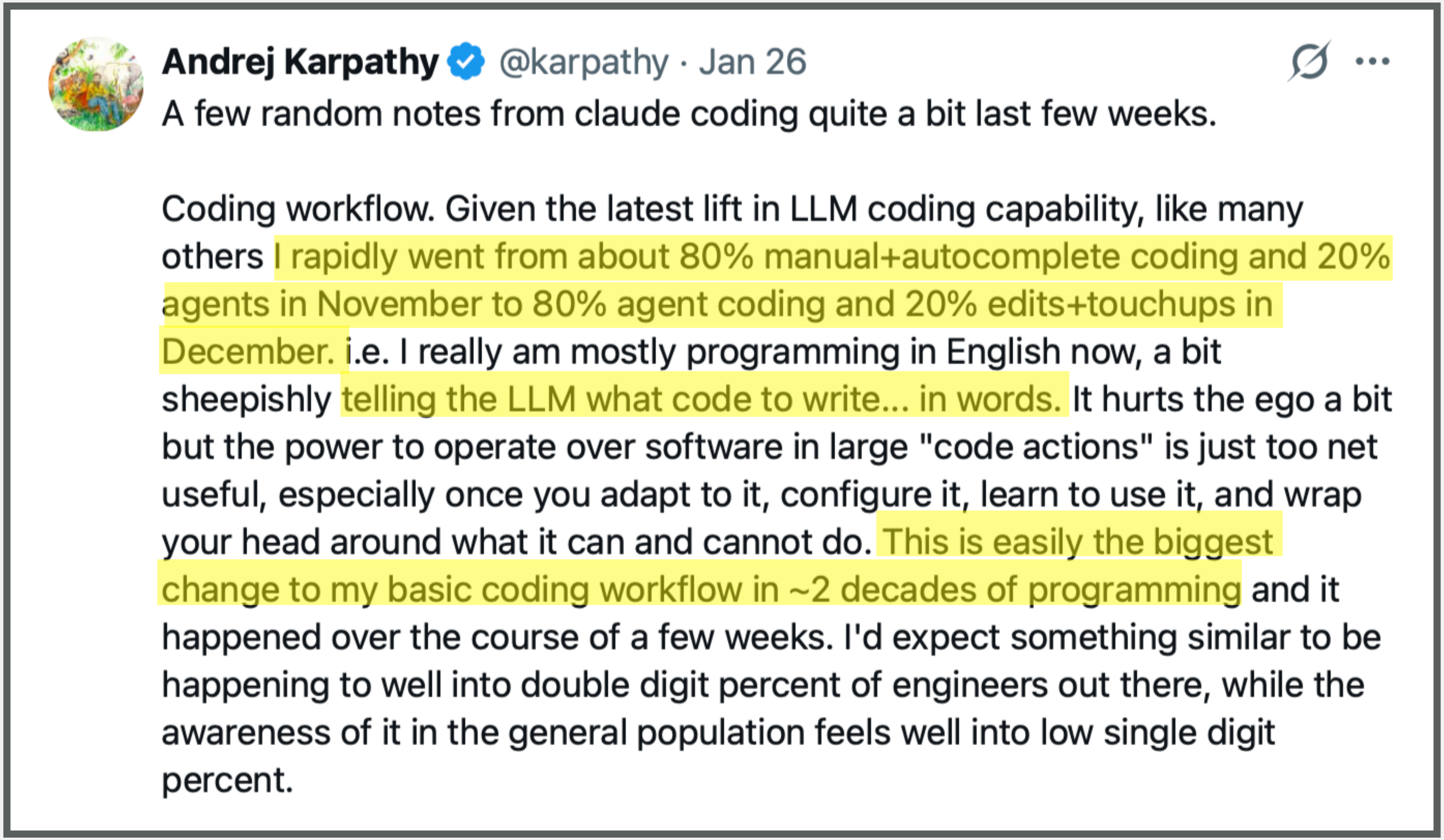
Administrivia

- *SDE Project Part 2*
  - Due Friday March 6th!
  - Two parts:
    - Process & Implementation Snapshot
    - Checkpoint Presentation
- *Assignment 4*
  - Due Monday February 23rd

A Software Engineer's Guide to LLMs

The AI Coding Hype Cycle

Industry Anecdotes

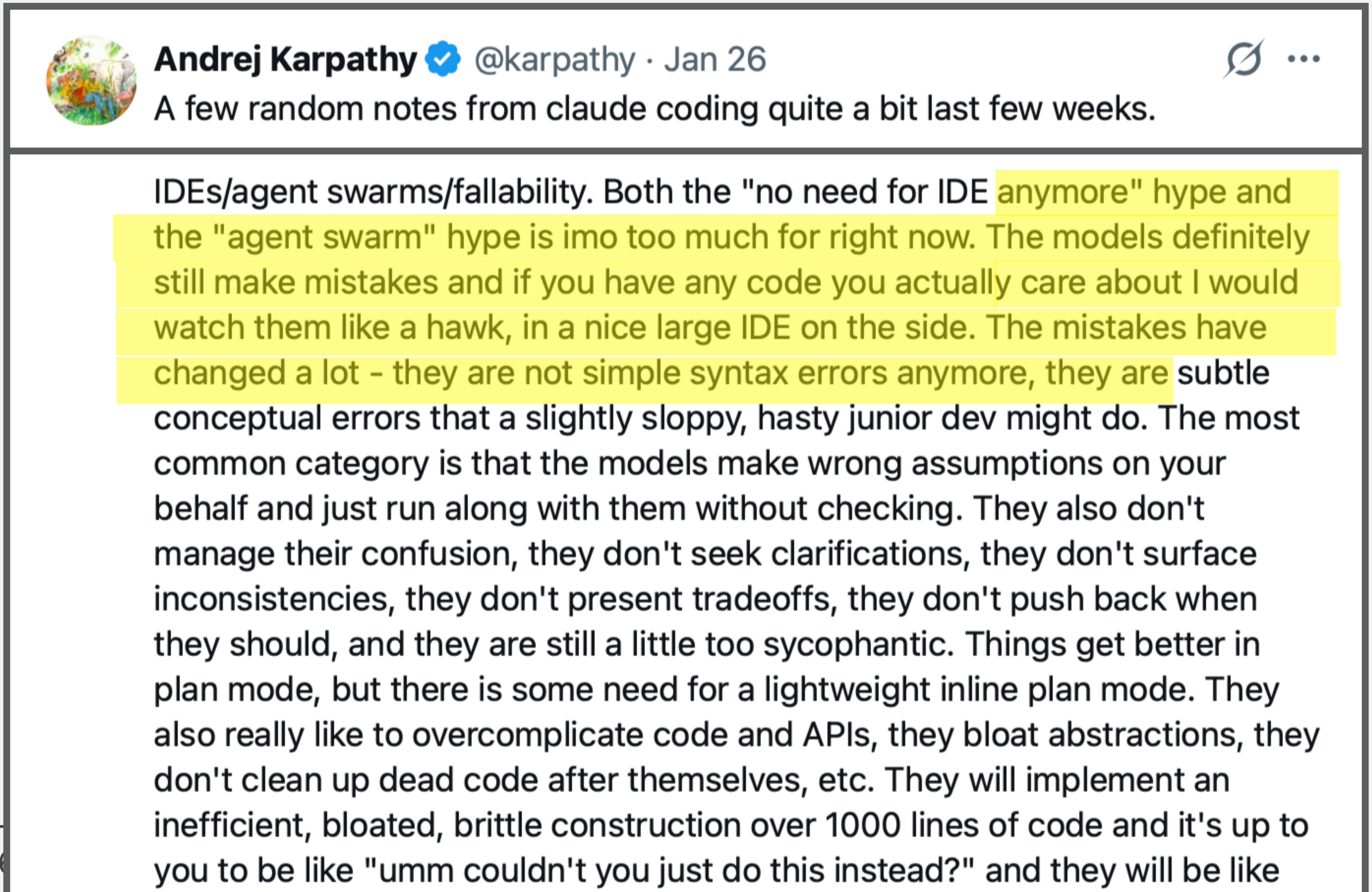


Andrej Karpathy @karpathy · Jan 26

A few random notes from claude coding quite a bit last few weeks.

Coding workflow. Given the latest lift in LLM coding capability, like many others I rapidly went from about 80% manual+autocomplete coding and 20% agents in November to 80% agent coding and 20% edits+touchups in December. i.e. I really am mostly programming in English now, a bit sheepishly telling the LLM what code to write... in words. It hurts the ego a bit but the power to operate over software in large "code actions" is just too net useful, especially once you adapt to it, configure it, learn to use it, and wrap your head around what it can and cannot do. This is easily the biggest change to my basic coding workflow in ~2 decades of programming and it happened over the course of a few weeks. I'd expect something similar to be happening to well into double digit percent of engineers out there, while the awareness of it in the general population feels well into low single digit percent.

Industry Anecdotes

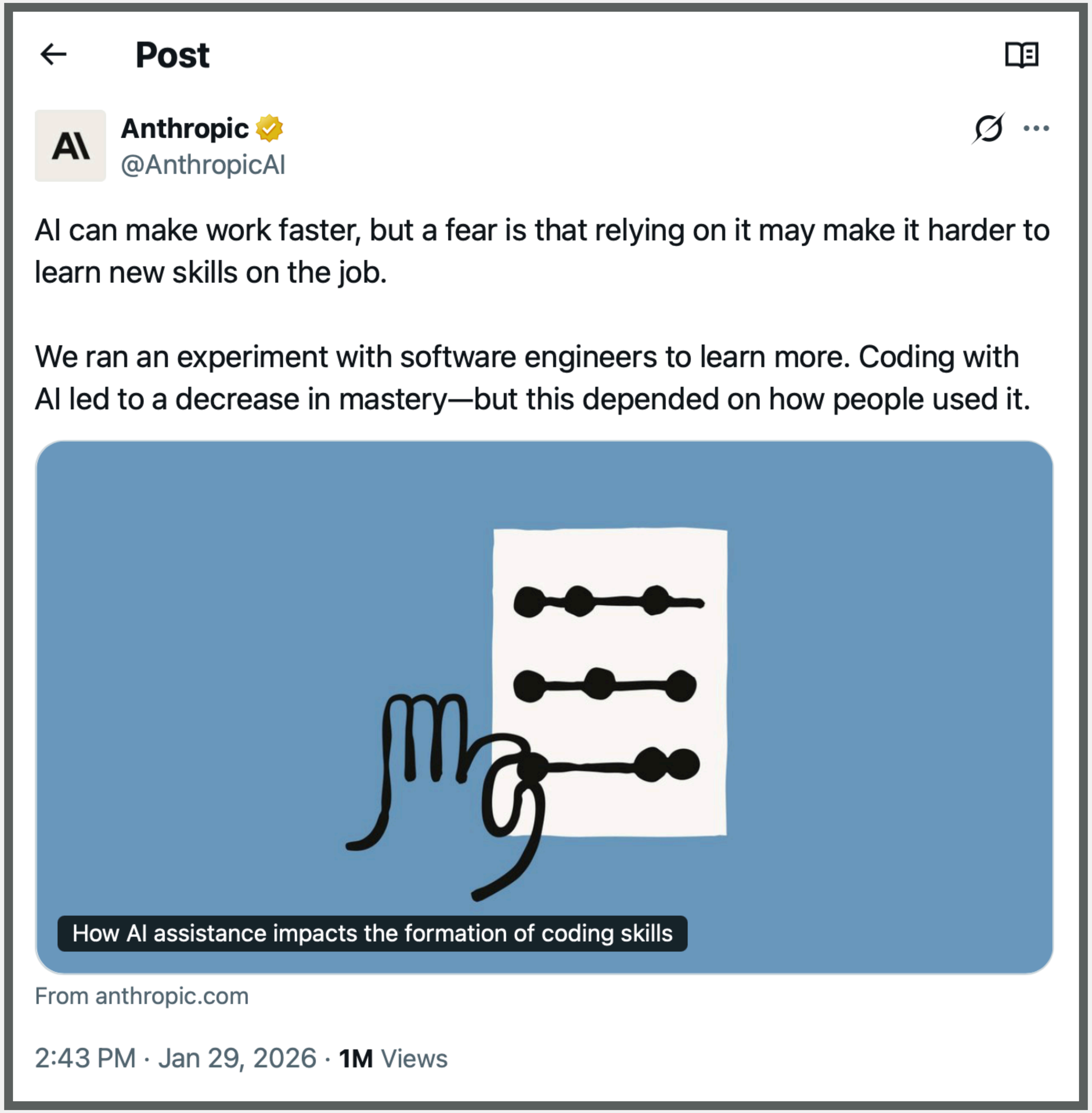


Andrej Karpathy @karpathy · Jan 26

A few random notes from claude coding quite a bit last few weeks.

IDEs/agent swarms/fallability. Both the "no need for IDE anymore" hype and the "agent swarm" hype is imo too much for right now. The models definitely still make mistakes and if you have any code you actually care about I would watch them like a hawk, in a nice large IDE on the side. The mistakes have changed a lot - they are not simple syntax errors anymore, they are subtle conceptual errors that a slightly sloppy, hasty junior dev might do. The most common category is that the models make wrong assumptions on your behalf and just run along with them without checking. They also don't manage their confusion, they don't seek clarifications, they don't surface inconsistencies, they don't present tradeoffs, they don't push back when they should, and they are still a little too sycophantic. Things get better in plan mode, but there is some need for a lightweight inline plan mode. They also really like to overcomplicate code and APIs, they bloat abstractions, they don't clean up dead code after themselves, etc. They will implement an inefficient, bloated, brittle construction over 1000 lines of code and it's up to you to be like "umm couldn't you just do this instead?" and they will be like

Industry Anecdotes



Post

Anthropic
@AnthropicAI

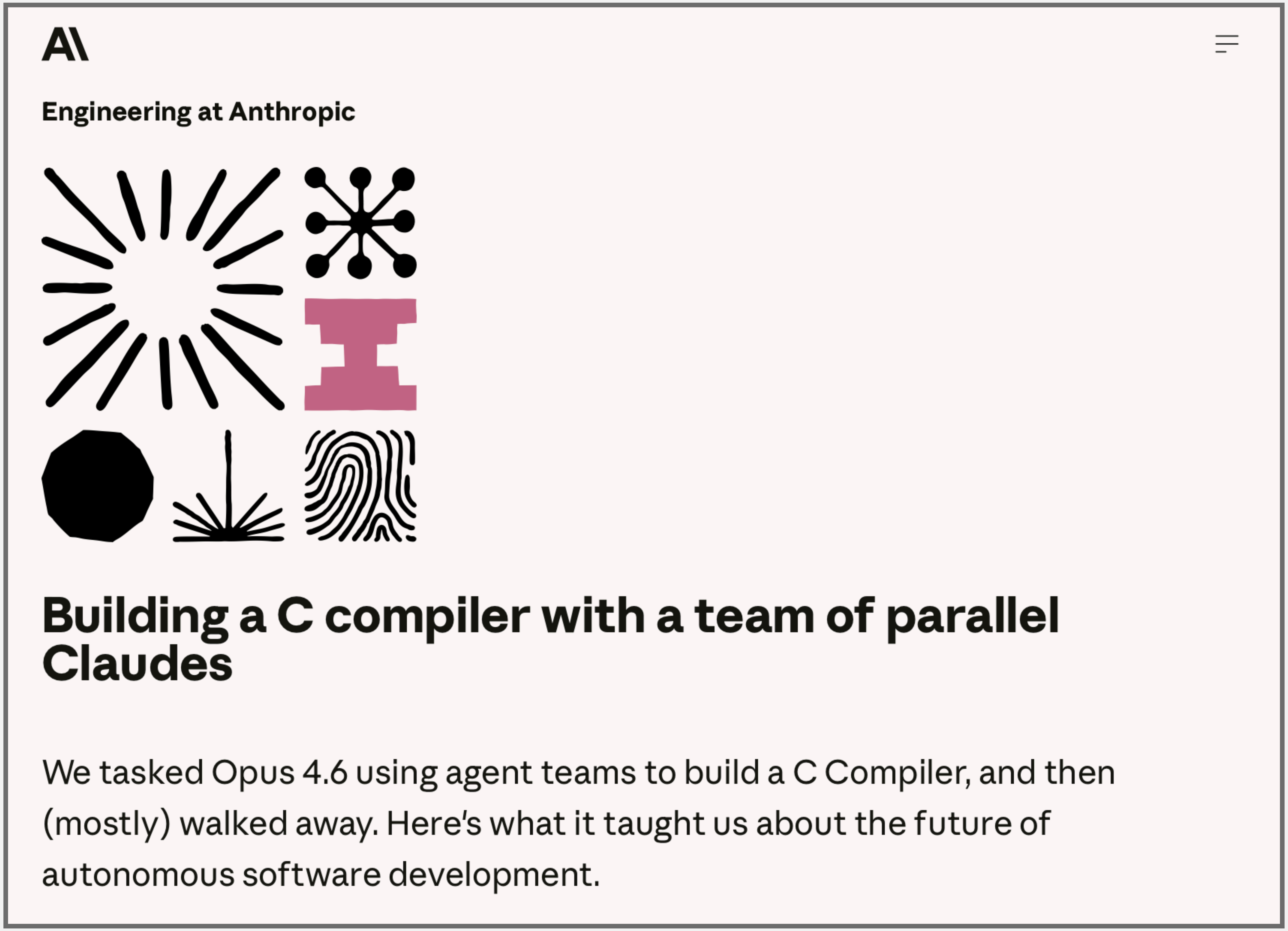
AI can make work faster, but a fear is that relying on it may make it harder to learn new skills on the job.

We ran an experiment with software engineers to learn more. Coding with AI led to a decrease in mastery—but this depended on how people used it.

From anthropic.com

2:43 PM · Jan 29, 2026 · **1M** Views

Industry Anecdotes



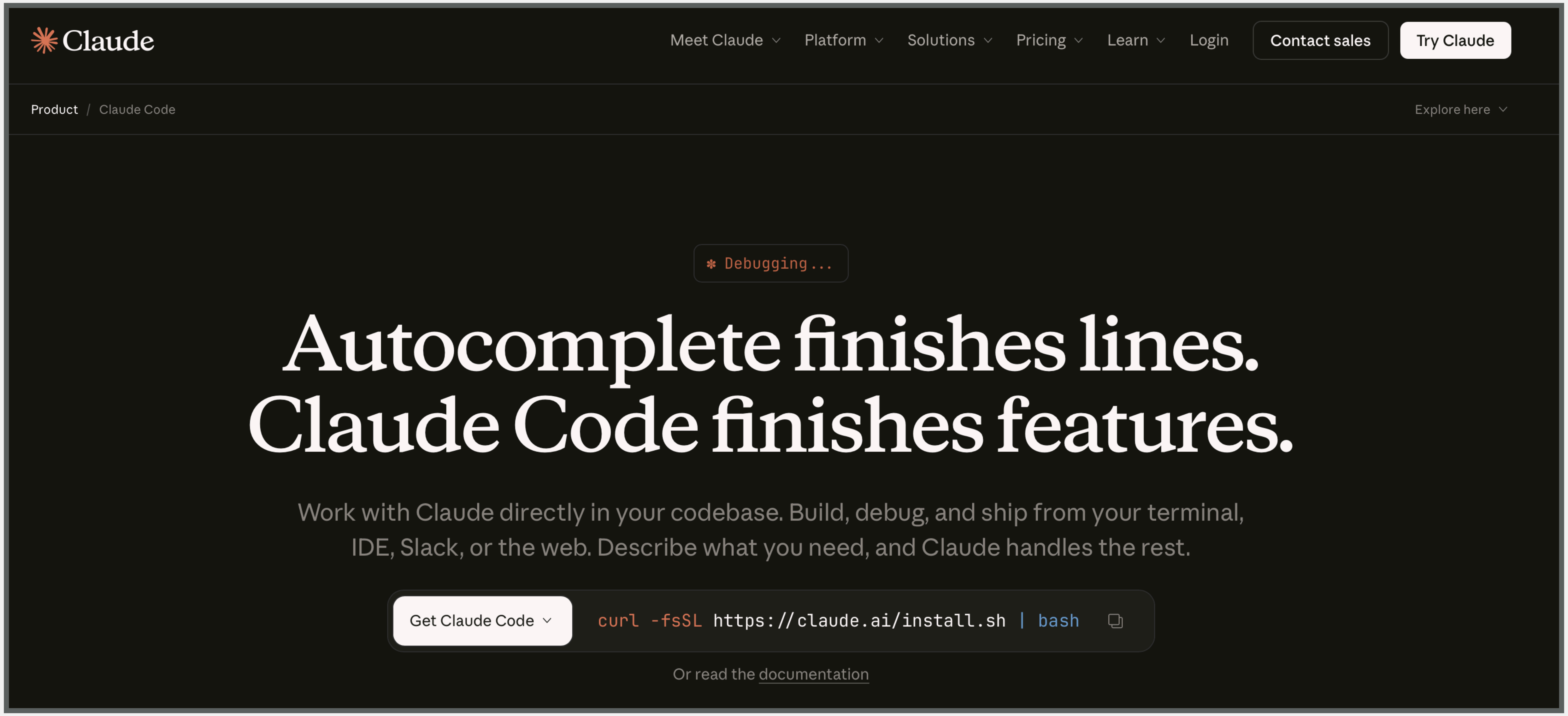
Engineering at Anthropic

Building a C compiler with a team of parallel Claudes

We tasked Opus 4.6 using agent teams to build a C Compiler, and then (mostly) walked away. Here's what it taught us about the future of autonomous software development.

LLM-powered Software Development

Claude Code - Our Subject



Autocomplete finishes lines.
Claude Code finishes features.

Work with Claude directly in your codebase. Build, debug, and ship from your terminal, IDE, Slack, or the web. Describe what you need, and Claude handles the rest.

```
curl -fsSL https://claude.ai/install.sh | bash
```

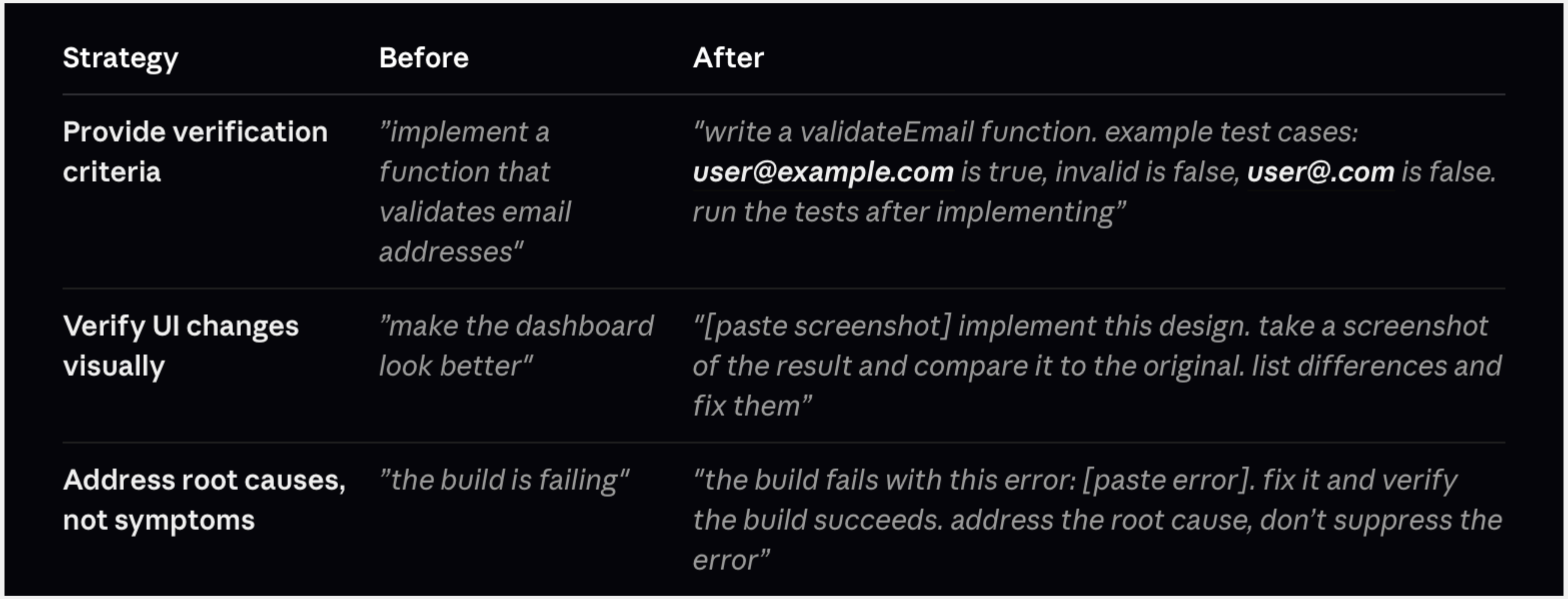
Or read the documentation

Claude Code -Best Practices

- Give Claude a way to verify its work
- Explore First, then Plan, then Code
- Provide Specific Context for your Tasks
- Configure your Environment

Claude Code -Best Practices

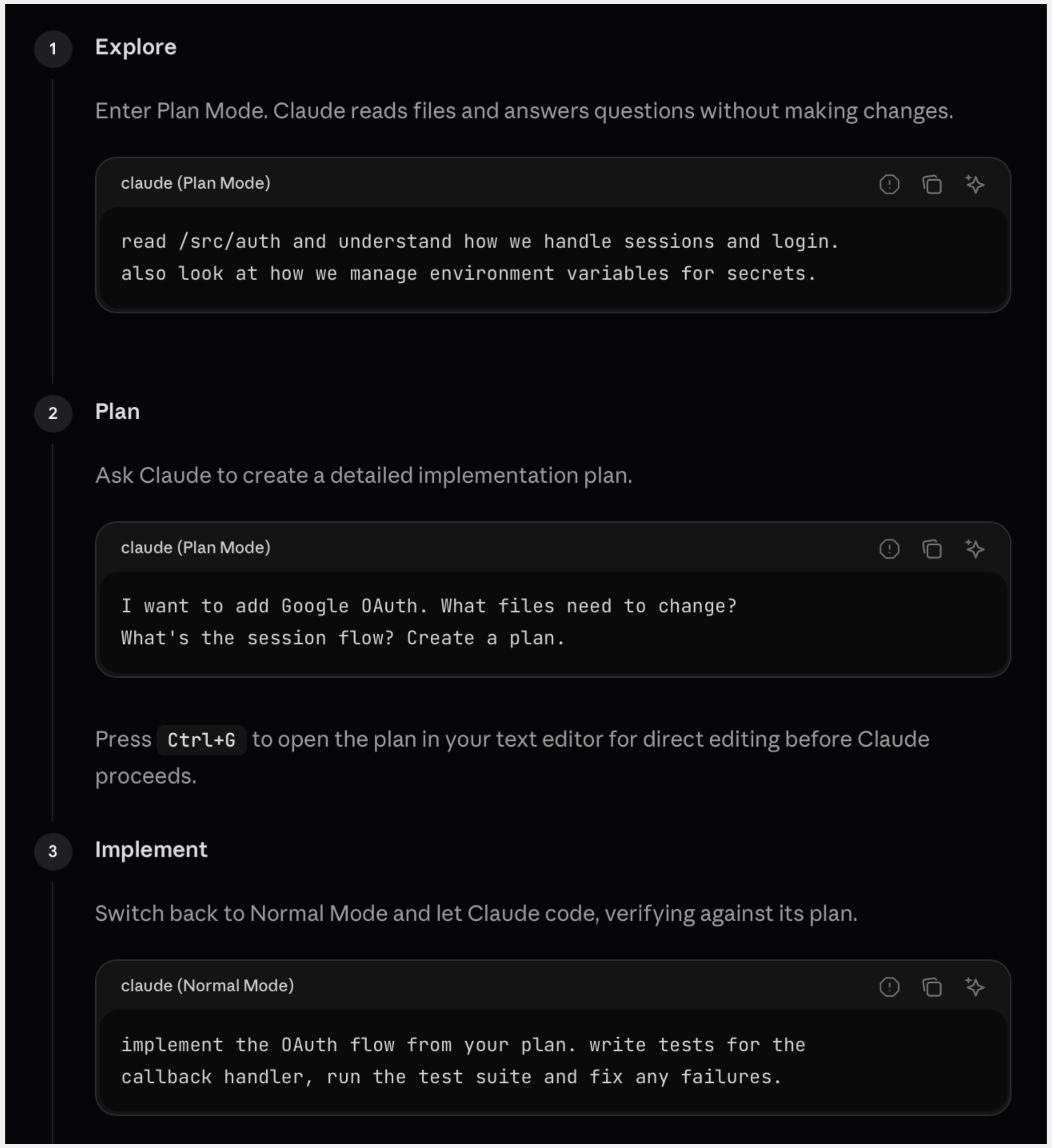
- Give Claude a way to verify its work



Strategy	Before	After
Provide verification criteria	*"implement a function that validates email addresses"*	*"write a validateEmail function. example test cases:* ***user@example.com*** *is true, invalid is false,* ***user@.com*** *is false. run the tests after implementing"*
Verify UI changes visually	*"make the dashboard look better"*	*"[paste screenshot] implement this design. take a screenshot of the result and compare it to the original. list differences and fix them"*
Address root causes, not symptoms	*"the build is failing"*	*"the build fails with this error: [paste error]. fix it and verify the build succeeds. address the root cause, don't suppress the error"*

Claude Code -Best Practices

- Explore First, then Plan, then Code



1 **Explore**

Enter Plan Mode. Claude reads files and answers questions without making changes.

claude (Plan Mode)

```
read /src/auth and understand how we handle sessions and login.
also look at how we manage environment variables for secrets.
```

2 **Plan**

Ask Claude to create a detailed implementation plan.

claude (Plan Mode)

```
I want to add Google OAuth. What files need to change?
What's the session flow? Create a plan.
```

Press `Ctrl+G` to open the plan in your text editor for direct editing before Claude proceeds.

3 **Implement**

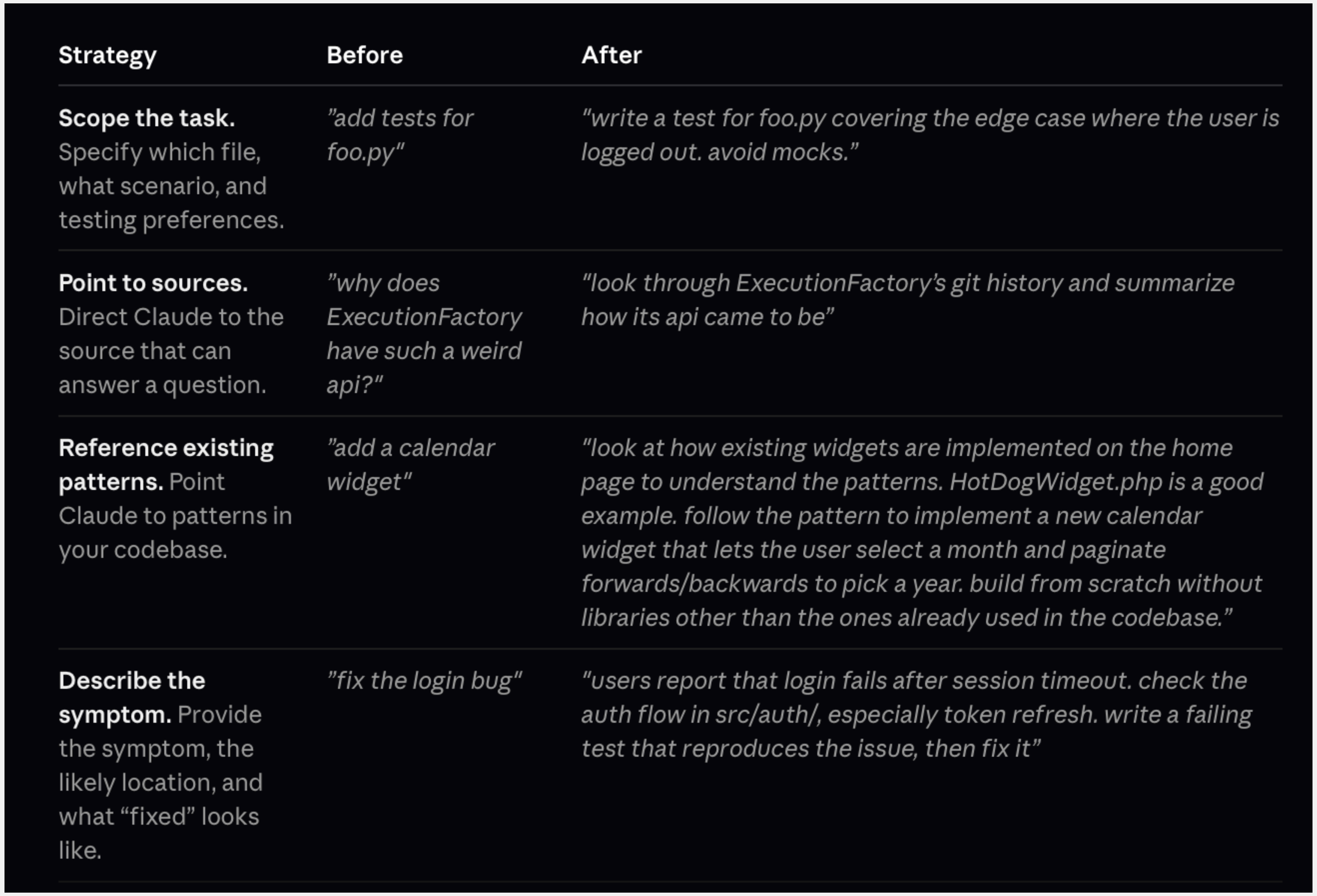
Switch back to Normal Mode and let Claude code, verifying against its plan.

claude (Normal Mode)

```
implement the OAuth flow from your plan. write tests for the
callback handler, run the test suite and fix any failures.
```

Claude Code -Best Practices

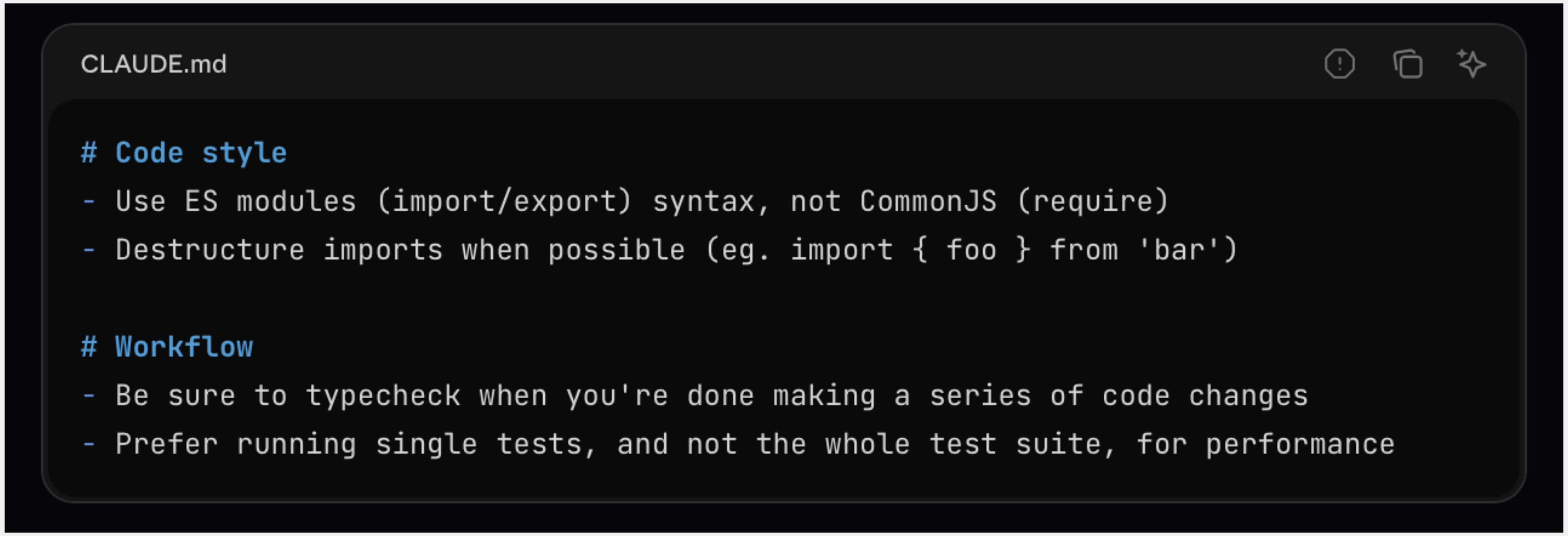
- Provide Specific Context for your Tasks



Strategy	Before	After
Scope the task. Specify which file, what scenario, and testing preferences.	*"add tests for foo.py"*	*"write a test for foo.py covering the edge case where the user is logged out. avoid mocks."*
Point to sources. Direct Claude to the source that can answer a question.	*"why does ExecutionFactory have such a weird api?"*	*"look through ExecutionFactory's git history and summarize how its api came to be"*
Reference existing patterns. Point Claude to patterns in your codebase.	*"add a calendar widget"*	*"look at how existing widgets are implemented on the home page to understand the patterns. HotDogWidget.php is a good example. follow the pattern to implement a new calendar widget that lets the user select a month and paginate forwards/backwards to pick a year. build from scratch without libraries other than the ones already used in the codebase."*
Describe the symptom. Provide the symptom, the likely location, and what "fixed" looks like.	*"fix the login bug"*	*"users report that login fails after session timeout. check the auth flow in src/auth/, especially token refresh. write a failing test that reproduces the issue, then fix it"*

Claude Code -Best Practices

- Configure your Environment



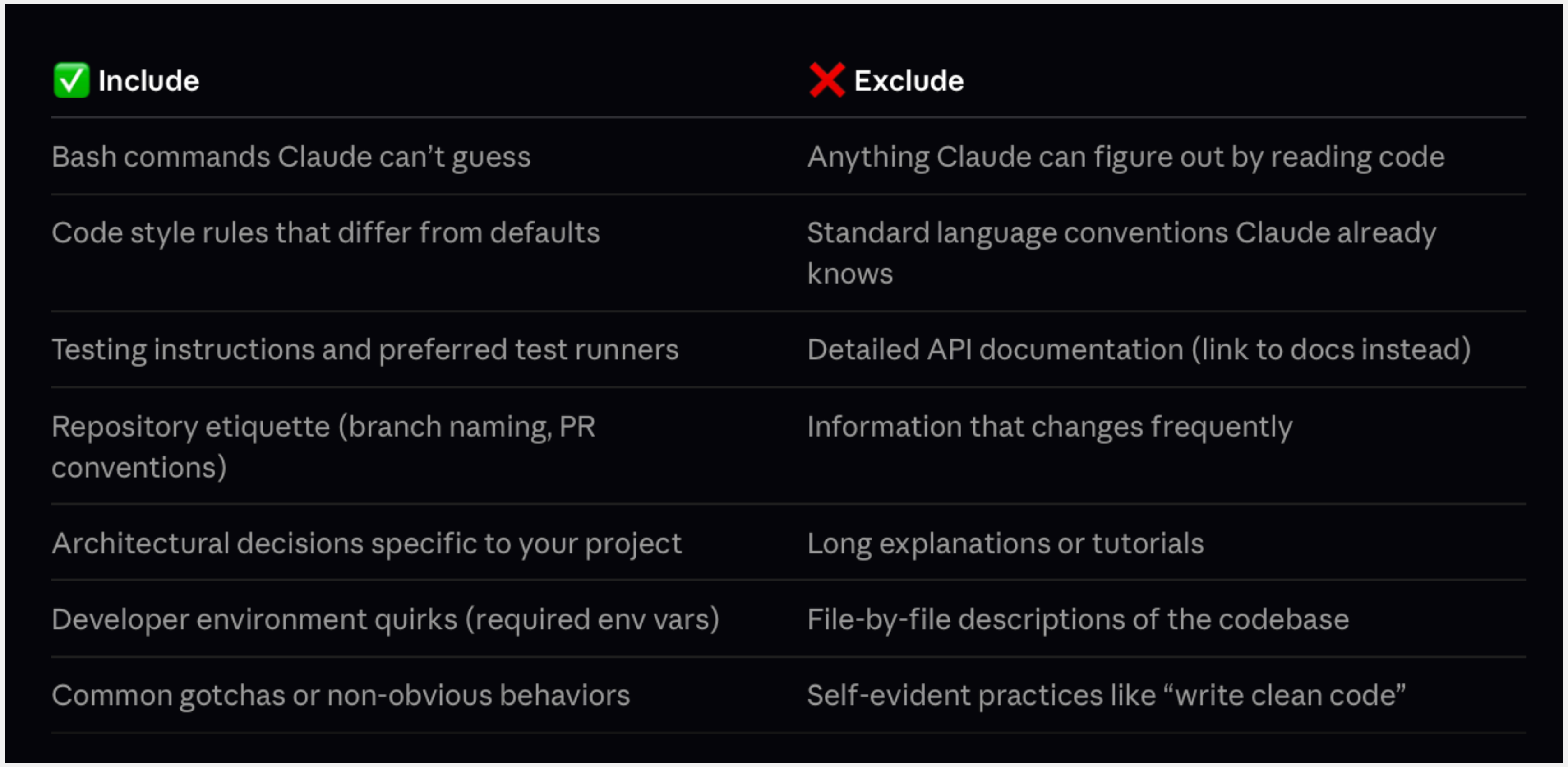
CLAUDE.md

```
# Code style
- Use ES modules (import/export) syntax, not CommonJS (require)
- Destructure imports when possible (eg. import { foo } from 'bar')

# Workflow
- Be sure to typecheck when you're done making a series of code changes
- Prefer running single tests, and not the whole test suite, for performance
```

Claude Code -Best Practices

- Configure your Environment



✅ Include	❌ Exclude
Bash commands Claude can't guess	Anything Claude can figure out by reading code
Code style rules that differ from defaults	Standard language conventions Claude already knows
Testing instructions and preferred test runners	Detailed API documentation (link to docs instead)
Repository etiquette (branch naming, PR conventions)	Information that changes frequently
Architectural decisions specific to your project	Long explanations or tutorials
Developer environment quirks (required env vars)	File-by-file descriptions of the codebase
Common gotchas or non-obvious behaviors	Self-evident practices like "write clean code"

Claude Code - Common Workflows

- Getting Started - Codebase Comprehension
  - `cd /path/to/project`
  - `claude`
    - `> give me an overview of this codebase`
    - `> explain the main elements of the architecture used here`

Claude Code - Common Workflows

- Finding Relevant Code
  - `> find the files that handle user authentication`
  - `> how do these authentication files work together?`
  - `> trace the login process from front-end to database`

Claude Code - Common Workflows

- Fix Bugs Efficiently
  - **Share the error with Claude**
    - `> I'm seeing an error when I run npm test`
  - **Ask for Fix Recommendations**
    - `> suggest a few ways to fix the @ts-ignore in user.ts`
  - **Apply the Fix**
    - `> update user.ts to add the null check you suggested`

Claude Code - Common Workflows

- Refactor Code
  - **Identify Legacy Code for Refactoring**
    - `> find deprecated API usage in our codebase`
  - **Get Refactoring Recommendations**
    - `> suggest how to refactor utils.js to use modern JavaScript features`
  - **Apply the Changes Safely**
    - `> refactor utils.js to use ES2024 features while maintaining the same behavior`
  - **Verify the Refactoring**
    - `> run tests for the refactored code`

Claude Code - Common Workflows

- Create Specialized Subagents
  - **Create Custom Subagents for your Workflow**
    - `> /agents`
    - **Then select "Create New subagent" and follow the prompts to define:**
      - `A unique identifier that describes the subagent's purpose (for example, code-reviewer, api-designer).`
      - `When Claude should use this agent`
      - `Which tools it can access`
      - `A system prompt describing the agent's role and behavior`

Claude Code - Common Workflows

- Use Specialized Subagents
  - **View Available Subagents**
    - `> /agents`
  - **Use Subagents Automatically**
    - `> review my recent code changes for security issues`
    - `> run all tests and fix any failures`
  - **Explicitly Request Specific Subagents**
    - `> use the code-reviewer subagent to check the auth module`
    - `> have the debugger subagent investigate why users can't log in`

Claude Code - Common Workflows

- Use Plan Mode for Safe Code Analysis
    - **When to use Plan Mode**
        - Multi-step implementation: When your feature requires making edits to many files
        - Code exploration: When you want to research the codebase thoroughly before changing anything
        - Interactive development: When you want to iterate on the direction with Claude

Claude Code - Common Workflows

- How to Use Plan Mode
  - **Start a new session in Plan Mode**
    - `claude --permission-mode plan`
  - **Run “headless” queries in Plan Mode**
    - `claude --permission-mode plan -p "Analyze the authentication system and suggest improvements”`
  - **Example: Planning a Complex Refactor**
  - `> I need to refactor our authentication system to use OAuth2. Create a detailed migration plan.`

Claude Code - Common Workflows

- Work with Tests
  - **Identify Untested Code**
    - `> find functions in NotificationsService.swift that are not covered by tests`
  - **Generate Test Scaffolding**
    - `> find functions in NotificationsService.swift that are not covered by tests`
  - **Add Meaningful Test Cases**
    - `> add test cases for edge conditions in the notification service`
  - **Run and Verify Tests**
    - `> run the new tests and fix any failures`

Claude Code - Common Workflows

- Create Pull Requests
  - You can create pull requests by asking Claude directly ("create a pr for my changes") or by using the /commit-push-pr skill, which commits, pushes, and opens a PR in one step.
    - `> /commit-push-pr`
  - **Create your own PR Skill**
    - Summarize Your Changes
      - `> summarize the changes I've made to the authentication module`
    - Generate a Pull Request
      - `> create a pr`
    - Review and Refine
      - `> enhance the PR description with more context about the security improvements`

Claude Code - Common Workflows

- Handle Documentation
  - Identify Undocumented Code
    - `> find functions without proper JSDoc comments in the auth module`
  - **Generate Documentation**
    - `> add JSDoc comments to the undocumented functions in auth.js`
  - **Review and Enhance**
    - `> improve the generated documentation with more context and examples`
  - **Verify Documentation**
    - `> check if the documentation follows our project standards`