Best Practices for Programming with AI

Ian Ross

OSG User School 2026

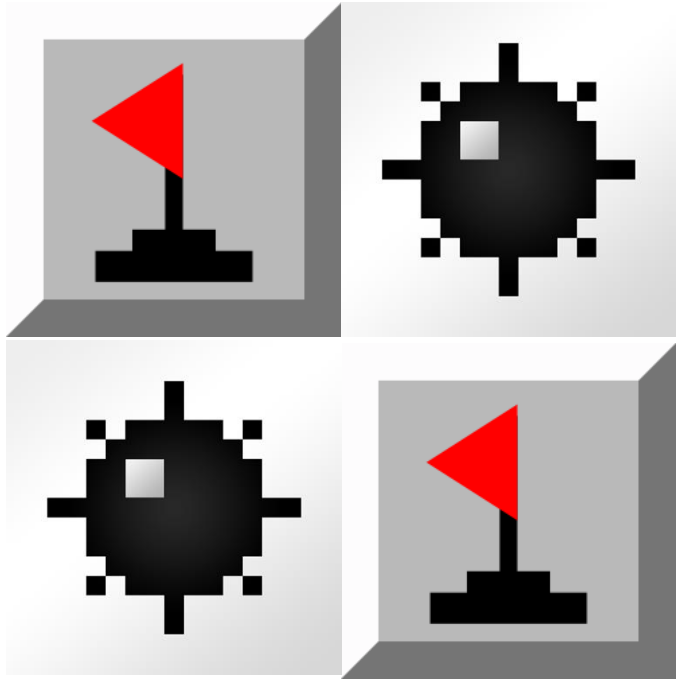
Demo

- Using Claude Code to write a Dockerfile for a conda environment a collaborator gave us

- Using Claude Code and a pre-written plan document to upgrade an *ancient* pytorch stack to support running on new hardware.

Apologies. It's hard to avoid using words in line "thinking" and to not accidentally give these tools more "humanity" than they deserve. And term collision is a serious problem. And so is defining boundaries between the different concepts…

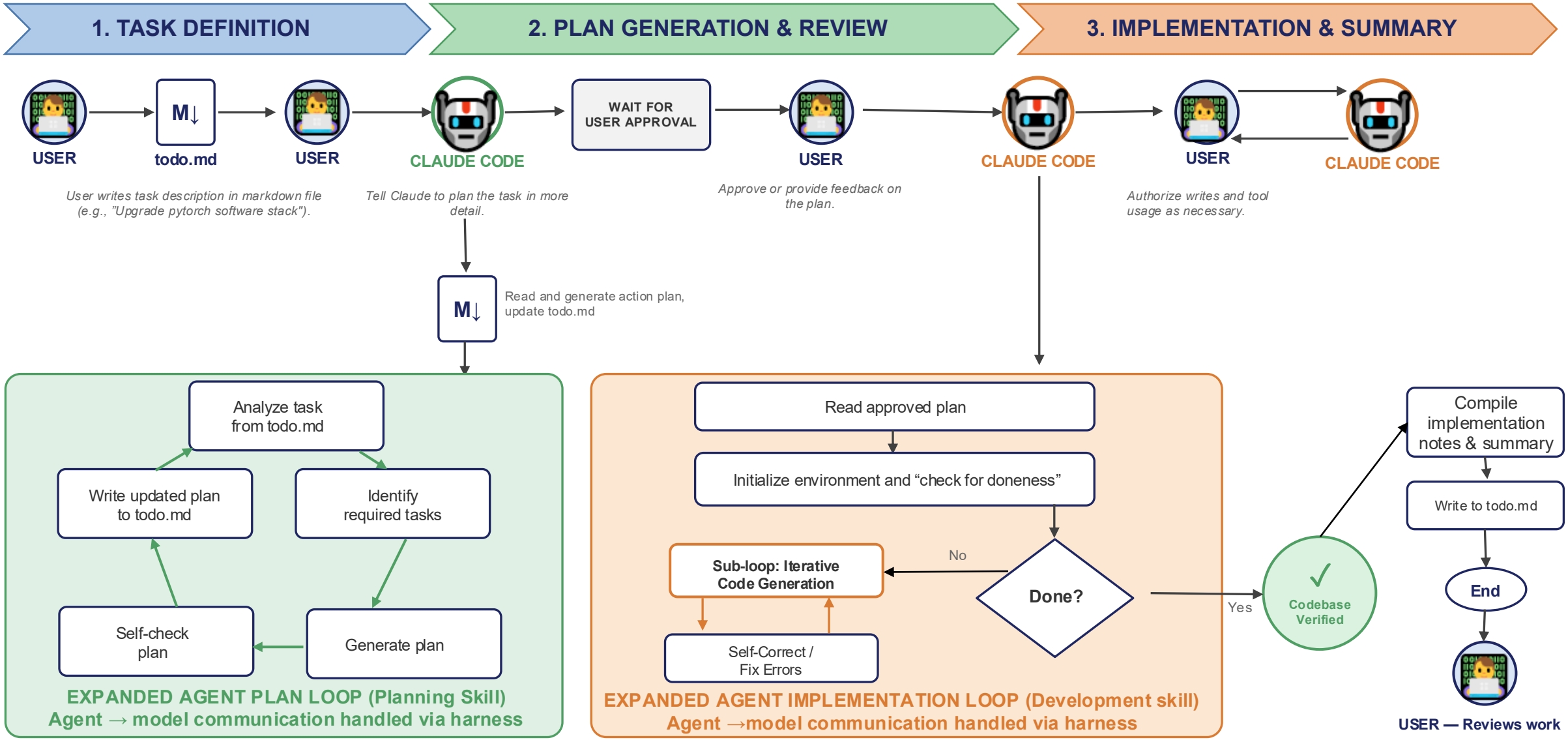
Disclaimer. As a result, this is all "as I view the world" and not absolute truth. If you ask 10 people to define terms and describe action loops, you'll get 14 answers…



A few definitions

- **Model:** the underlying “brain in the sky*” (*: not necessarily cloud: could be local)
  - **Context**: Everything that the model receives as input
- **Harness:** interface and scaffolding around a model. Can take actions or either a standalone tool (Claude Code, OpenCode, Pi) or part of an IDE (VSCode Agents, Cursor). Can invoke agents or execute tools directly
- **Agent**: A specific *autonomous workflow loop*.
- **Model Context Protocol (MCP)/Plugin/Skill** - capabilities that Agents can be given in order to pull in targeted context or execute actions
- The **model** is the “intelligence”, the **harness** is the environment and scaffolding, and **agents** are workflow loops that are leveraged in tasks, with the ability to utilize **MCPs, Plugins, and Skills** as needed to gather relevant **context, take direct action, or spawn (sub)agent tasks.**

So what happened in our demo?



Guiding Principles

Guiding principles

- LLMs act like an ambitious junior project member or an eager-to-please puppy
  - Will happily go off, try their hardest, look up relevant info, and miss the mark entirely…
  - No institutional knowledge or experience or history with project specifics beyond what you provide
- LLMs speak with confidence
  - AI has a tendency to over-state success (and flatter you…)
- Keep it small: Iterate, review diffs, single task per session, etc.
  - Agents will do their own iterative loops until getting to an ‘answer’, but keeping interactions focused improves overall quality

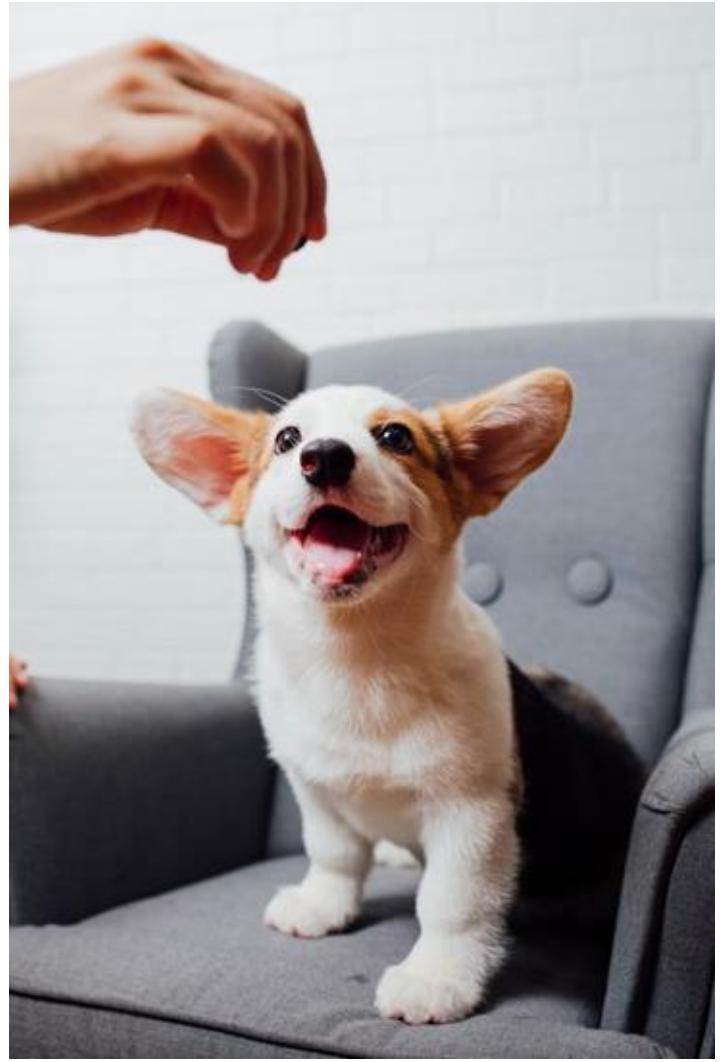
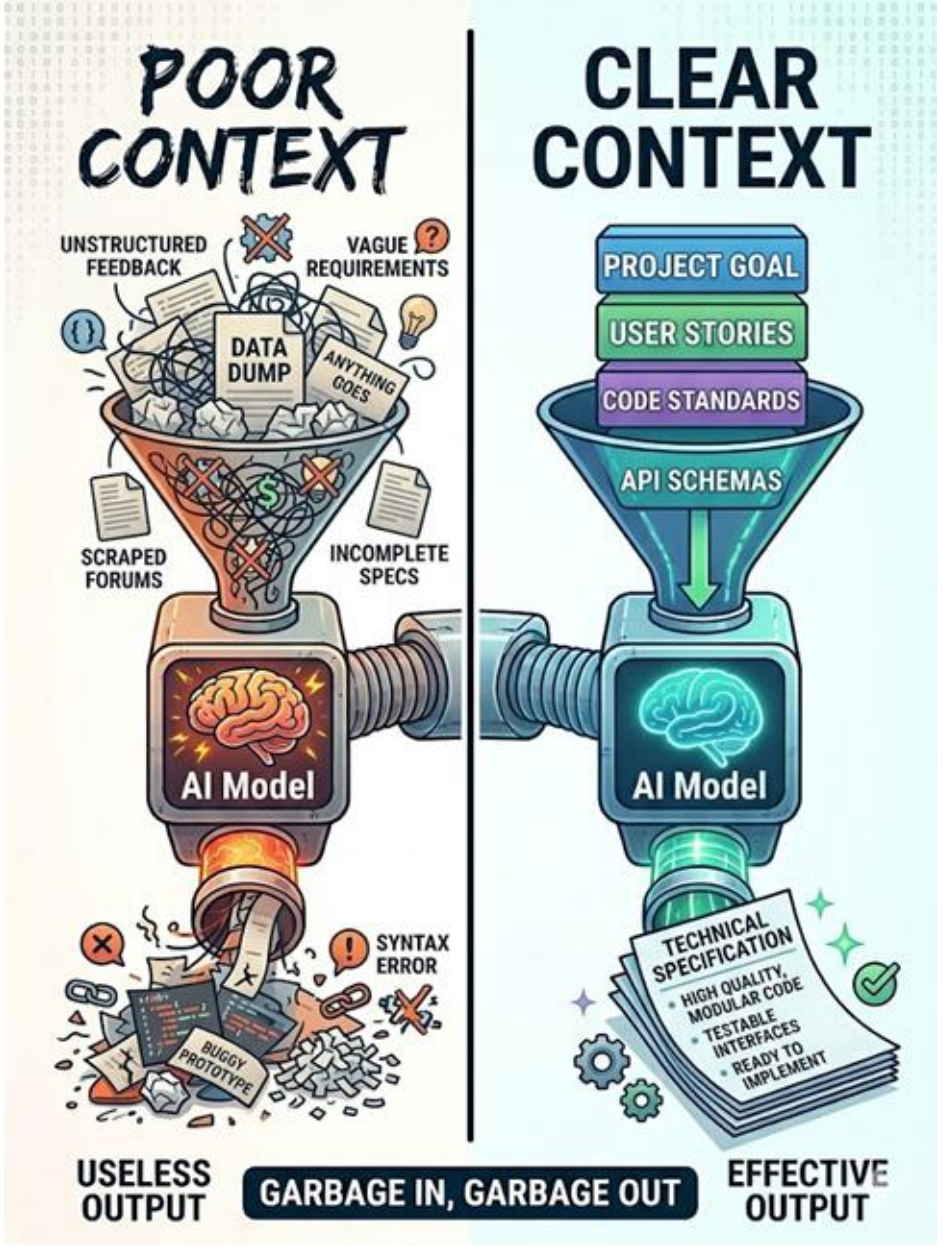


Photo by Hanson Lu on Unsplash

More guiding principles

- Context is key
  - The more relevant information an agent can see and pull in, the better the response from the upstream model and the better the result will be
- Constraints are also key
  - Providing structure and guidance will help it move into your preferred solution space.
  - Not sure what that is? Talk it through! Prompt it to talk through the decision-making and planning processes with you and gather any information it thinks it lacks, rather than making assumptions
- **You are responsible**
  - No different than writing the code itself: you are responsible for any line of code you push or deploy, whether you or an agent wrote it.



Obligatory gemini-generated image

 Neat! How do I get going? 


Models and harnesses

- Hard to say what works best because
  - 1. It's a moving target and
  - 2. Often it's a matter of personal preference
- Megacorps often provide both harnesses and models, but the harnesses *might* be made to be model agnostic
- Claude Code and Anthropic's Sonnet (or Opus or Haiku) seem to be a common and effective setup but requires subscription
  - OpenAI's Codex and Github's Copilot are common as well
- Open tools (pi, OpenCode, aider) pointing at open models (e.g. kimi or qwen hosted by NRP)
  - Services like OpenRouter provide interfaces to many hosted models
- IDEs
  - Cursor, VSCode (+ Cline, Copilot integrations, etc.)
- Self hosting
  - A bit harder, but things like Microsoft's Phi SLM can work nicely for smaller tasks. Low latency but less "brains"

One-time* setup

- Harness config, e.g. $HOME/.claude/settings.json
  - Configuration for the *harness*
  - E.g. includes places to define pre-took use hooks, permission allow/deny lists, mcp server and tool definitions, …)
- CLAUDE.md or AGENTS.md
  - Instructions for the *model* to influence what is and isn't allowed and desired.
  - Also provide additional context and technical guidance for the project
    - Objectives, desired languages, style, test-driven development, …
  - *: Can exist at both global user level *and* local project level
- **Read documentation for your tool of choice!** There's no one standard here.

MCPs, tools, skills, plugins, etc

- These are all ways to provide the agents additional information that can be relevant in solving tasks
- MCP: Protocol specifying ways to interact with external services to generate context. Agent (and model) agnostic. Can be run locally (Context7) or be provided remotely (e.g. Github MCP)
- Skills: Markdown instructions (“You are an expert in biomechanical processes. You generate reproducible software, containerized and ready for high throughput systems…”)
- Plugins: New harness features.
- Advice: Start small, experiment.

Planning and spec-driven development

- Many tools have a “planning” mode that can reason through a request to build an internal (or external!) document outlining what to do before writing code itself.
- Extremely useful to explicitly write down the document so you know what the agent was *trying* to do.
- External tools (backlog.md or writing as Github Issues)

Summary: Day-to-day loops-of-loops

- Choose a model and harness.
- Planning loop: Define the task at hand, chart a course to implementation. Optionally use agents to help develop plan. Optionally write down the plan for future reference.
- Development loop: Set the harness loose on the task until agents declare success
  - Review and iteration sub-loop: Review the code changes, understand the code, request changes as necessary
  - Optionally summarize and log the work done in the plan document
- Victory!

Thanks, Gemini!

Bonus slides

My Typical Workflow

- Mostly Claude Code + Sonnet
- Fairly restricted settings.json, pretty minimal and fluid MCP/Plugins collection

- For meaty new features, write a document outlining the new feature (often using Backlog.md, but it's got a lot of unnecessary cruft now), including an "Acceptance Criteria" to define doneness
  - Also pull in external sources (e.g. links to documentation, tools/libraries to consider)
  - When necessary, describe or explicitly define schema or interfaces