Agentic Coding Playbook

Umay ŞAMLI

February 23, 2026

Contents

1 Introduction

This playbook is designed for people who want to learn the best practices of agentic coding. We will be defining **different types of agentic** coding styles and how to choose which style to adopt. We will use **Cursor and Claude Code** for agentic coding examples. Hopefully, this resource will be helpful for those who want to use AI in their coding journey.

2 What is agentic coding?

In the current landscape, agentic coding refers to a person who completely vibe-codes but with well-defined hooks, tests, and paradigms. For this resource, agentic coding will be defined as the process of using AI agents to assist in coding tasks. The level at which it assists can vary. However, if an environment has a deployed agentic coding system, it means it is engaging in agentic coding in some capacity. This approach gives us the ability to manage different styles of using AI in coding.

3 Types of agentic coding

We will be defining 3 different types of agentic coding styles. These are:

- **Agentic Vibe Coder**: This is also known as the person who writes, "hey opus, write me GTA 6 and make no mistakes." This approach lacks proper hooks or tests; the developer just vibe-codes with the agent. This is the most basic form of agentic coding and is not recommended for anything other than testing the capabilities of the models.
- **Classic Agentic Coder**: This is a person who makes the necessary configurations but still heavily relies on the code generated by the agent. Some big tech companies are currently trying to practice this. It is not the best practice but could be extremely useful for small tool or script development.
- **Agentic Developer**: This is a person who uses AI agents to assist in coding tasks but also writes their own code and tests. They also review the code created by the agent. This is the most recommended approach for most developers.

We do not recommend the **Agentic Vibe Coder** style for any production code. This approach is highly susceptible to serious bugs and incidents. Never forget that AI agents are not perfect and can make mistakes. They can hallucinate and may also be biased.

The **Classic Agentic Coder** approach could be useful, especially for security researchers. However, it is not recommended for any software developer writing code for a commercial product.

The **Agentic Developer** approach is the best practice for developers. It can boost productivity and help write better code, but it must be applied properly.

4 How to configure my agent?

When you decide to adopt an agentic approach, you need to configure your agent properly. Proper configuration will significantly increase the code quality and capabilities of your agent. We will take a look at how to configure an agent.

4.1 Skills

Skills are mainly helpers for the agents. They give context to the agent to perform tasks better. There are a lot of skills that you can use. You can find skills at `https://skills.sh/`. There are also other sites that host skills for agents.

Skills are essentially modular capabilities or plugins that you can install to extend your AI agent's functionality. They provide specific context, tools, or instructions tailored to particular domains, such as penetration testing, web development, or data analysis. However, because skills can execute code or interact with your local environment, they introduce potential security flaws. Malicious or poorly written skills could execute arbitrary commands, leak sensitive workspace data, or introduce vulnerabilities into your codebase.

When downloading skills from repositories like `https://skills.sh/`, it is crucial to review their source code and permissions. Security tests on these platforms often involve static analysis of the skill's instructions, checking for unauthorized network requests, and sandboxing execution to ensure the skill does not access restricted files or environment variables.

```
+-------------+       Installs       +-------------+       Extends       +-------------+
|             | -------------------> |             | -------------------> |             |
|    User     |                      | Skill Repo  |                      |  AI Agent   |
|             | <------------------- | (skills.sh) | <------------------- |             |
+-------------+       Downloads      +-------------+       Provides      +-------------+
      |                                    |                                    |
      |                                    |                                    |
      v                                    v                                    v
+-------------+                       +-------------+                       +-------------+
|             |                       |             |                       |             |
| Local Env   |                       | Security    |                       | New Tools   |
|             |                       | Sandbox     |                       | & Context   |
+-------------+                       +-------------+                       +-------------+
```

4.2 Commands

Commands are predefined instructions or shortcuts that trigger specific agent behaviors or workflows. They allow developers to quickly execute repetitive tasks without writing lengthy prompts every time. You can use commands to scaffold projects, run test suites, or format code according to your project's style guide.

Here is an example of a classic command without inputs:

```
/format-workspace
```

This command might instruct the agent to run a linter and formatter across all files in the current directory.

Here is an example of a command that takes an input:

```
/generate-test --file=src/auth.ts
```

This command directs the agent to specifically read the `src/auth.ts` file and generate a corresponding unit test file.

```
+-------------+       Triggers       +-------------+       Executes      +-------------+
|             | -------------------> |             | -------------------> |             |
|    User     |                      | Command     |                      | AI Agent    |
|             | <------------------- | Definition  | <------------------- |             |
+-------------+       Returns        +-------------+       Provides      +-------------+
      |                                    |                                    |
      |                                    |                                    |
      v                                    v                                    v
+-------------+                       +-------------+                       +-------------+
|             |                       |             |                       |             |
| Input Args  |                       | Predefined  |                       | Workspace   |
| (--file=..) |                       | Workflow    |                       | Actions     |
+-------------+                       +-------------+                       +-------------+
```

4.3 Rules

Rules (often defined in files like `.cursorrules` or `.clauderules`) are persistent instructions that dictate how the agent should behave within a specific workspace. They are used to enforce coding standards, specify architectural patterns, or restrict certain actions. To use them, you simply create a rules file in your project's root directory and define your constraints in plain English. The agent will automatically read these rules and apply them to all subsequent interactions, ensuring consistency across your codebase.

```
+-------------+       Defines        +-------------+      Applies       +-------------+
|             | -------------------> |             | -----------------> |             |
|    User     |                      | .rules File |                    | AI Agent    |
|             | <------------------- |             | <----------------- |             |
+-------------+       Updates        +-------------+       Reads        +-------------+
      |                                     |                                  |
      |                                     |                                  |
      v                                     v                                  v
+-------------+                       +-------------+                     +-------------+
|             |                       |             |                     |             |
| Constraints |                       | Persistent  |                     | Consistent  |
| & Standards |                       | Context     |                     | Code Output |
+-------------+                       +-------------+                     +-------------+
```

4.4 Subagents

Subagents are specialized, autonomous AI instances spawned by a primary agent to handle complex, multi-step tasks. Instead of the main agent trying to do everything at once, it delegates specific subtasks (like researching documentation, searching the codebase, or running a series of terminal commands) to a subagent. The subagent works independently and reports back its findings to the main agent, which then synthesizes the final output for the user.

```
+-------------+       Delegates Task       +-----------------+
|             | -------------------------> |                 |
| Main Agent  |                            |    Subagent     |
|             | <------------------------- |                 |
+-------------+       Returns Result       +-----------------+
      ^                                             |
      |                                             |
      | Interacts                             Reads/Executes
      |                                             |
      v                                             v
+-------------+                            +-----------------+
|             |                            |                 |
|    User     |                            | Workspace/Tools |
|             |                            |                 |
+-------------+                            +-----------------+
```

4.5 Hooks

Hooks are automated triggers that execute specific agent actions at predefined points in your development lifecycle, such as pre-commit, post-save, or pre-push. They are used to automatically review code, generate commit messages, or run security checks before changes are finalized. To use them, you configure the agent to listen for specific Git or editor events and define the script or prompt that should run when the event occurs.

```
+-------------+       Triggers       +-------------+      Analyzes       +-------------+
|             | -------------------> |             | ------------------> |             |
| Git Commit  |                      | Agent Hook  |                     | Source Code |
|             | <------------------- |             | <------------------ |             |
+-------------+    Returns Review    +-------------+    Finds Issues     +-------------+
      |                                     |
      |                                     |
      v                                     v
+-------------+                       +-------------+
|             |                       |             |
| Commit Log  |                       | Auto-Fixes  |
|             |                       |             |
+-------------+                       +-------------+
```

4.6 MCP

The Model Context Protocol (MCP) is an open standard that enables AI models to securely connect to local or remote data sources and tools. It works by establishing a standardized client-server architecture where the AI agent (the client) communicates with an MCP server that exposes specific resources, prompts, or tools. To use it, you configure your agent with the connection details of an MCP server (e.g., a database connector or a GitHub integration). The agent can then query this server to fetch real-time context or perform actions outside its default capabilities.

```
+-------------+      MCP Protocol      +-------------+      Fetches       +-------------+
|             | ---------------------> |             | -----------------> |             |
|  AI Agent   |                        | MCP Server  |                    | Data Source |
|  (Client)   | <--------------------- |             | <----------------- | (DB, API)   |
+-------------+      JSON-RPC          +-------------+      Returns       +-------------+
```

4.7