Working with AI Coding Agents

Version Nov 2026 · verified 2026-11-02

Supply context, specify the work, verify the output — AGENTS.md, specs, task sizing, hooks, subagents, check loops and reviewing agent diffs.

Contents

Mental model 2

Context files: AGENTS.md and CLAUDE.md 2

Specifying the task 4

Spec-driven development 5

Sizing tasks, and when not to delegate 6

Hooks and permissions 8

Subagents 9

Verification loops 11

Reviewing agent diffs 12

Further reading 14

Mental model

A coding agent knows only what is in its context window: the code it chose to read, the files you pointed it at, and your instructions. It cannot see the meeting where you decided something, and it stops when the work *looks* done. Your job is therefore three things: supply the context it cannot find, specify what done means, and make it prove the result. Instructions are requests the model may weigh; hooks, permissions and CI are rules the harness enforces.

Context files: AGENTS.md and CLAUDE.md

Put in	Leave out
Build, test, lint commands	Directory tours
Decisions not visible in code	Style a linter enforces

Put in	Leave out
Traps: generated dirs, env vars	Anything found by reading code
Links to specs and ADRs	Pasted docs (link instead)

`AGENTS.md` is a README for agents, an open format now stewarded by the Agentic AI Foundation under the Linux Foundation; in a monorepo the nearest file to the edited code wins. Claude Code reads `CLAUDE.md`, falls back to `AGENTS.md` when no `CLAUDE.md` exists, and its docs recommend under 200 lines. Before writing a line, ask: could the agent find this by reading the repo, and would it look?

```
# AGENTS.md
- Test: `npm test`; lint: `npm run lint`
- Never edit `src/gen/`; run `npm run gen`
- Integration tests need `DATABASE_URL`
- Money is integer cents; see docs/adr/0007.md
```

> **Gotcha:** Do not let the agent write this file for you. An ETH Zurich study (2026) found LLM-generated context files slightly *lowered* task success while raising cost over 20%; agents obey every line, so needless instructions make tasks harder.

Specifying the task

Instead of	Write
“fix the login bug”	Symptom, location, failing test first
“add a widget”	The pattern file to copy, the limits
Prose about output	The command that proves it works
An implied scope	“Be ambitious: empty and error states”

A specific prompt gives the agent facts it cannot get elsewhere: the symptom, where to look, what “fixed”

means, and your preference between two reasonable designs. Hand over artifacts — the failing log, the screenshot, the file path — rather than describing them.

```
Users report login fails after session timeout.
Look at token refresh in src/auth/. Write a
failing test that reproduces it, then fix it.
Done = `npm test -- auth` passes; no new deps.
```

Spec-driven development

Phase	Output
Specify	What and why: user stories, acceptance criteria
Plan	Tech choices, design, data model
Tasks	Small ordered steps, each checkable
Implement	Code against one task at a time

For anything larger than a bug fix, separate deciding from doing: have the agent interview you or draft a spec, review it, then build from it in a fresh session with

a clean context. GitHub's Spec Kit scripts this as constitution, specify, plan, tasks, implement; Kiro writes `requirements.md`, `design.md` and `tasks.md`. The spec doubles as the yardstick for review.

```
/speckit-constitution Principles: tests first
/speckit-specify Album view grouped by date
/speckit-plan Vite, vanilla JS, SQLite metadata
/speckit-tasks
/speckit-implement
```

> **Warning:** Thoughtworks' Radar (Vol. 33) rates spec-driven development "Assess" and warns of drift back to heavy up-front specification and big-bang releases. Keep specs per feature, not per quarter.

Sizing tasks, and when not to delegate

Task	Approach
One-file fix with a test	Delegate directly

Task	Approach
Feature across layers	Spec, then one task per session
Unclear problem	Explore together; no edits yet
Novel design, security core	Write it yourself; agent assists
Output you cannot verify	Do not delegate

Small batches are one of DORA's seven capabilities that amplify AI's benefit: a small diff is cheap to review and cheap to throw away. Start a fresh session per task so earlier dead ends do not ride along in the context. If you could not tell a correct result from a plausible one, the agent will not save you time.

```
Task 3 of 7: add rate limit to POST /login
Scope: src/auth/limit.ts and its test only
Check: npm test -- limit && npm run lint
Out of scope: refactors, dependency changes
```

Hooks and permissions

Need	Mechanism
Never touch a path	Deny rule, e.g. `Edit(src/gen/**)`
Never read secrets	Deny `Read(./.env)`
Run a check every time	Hook (e.g. `Stop`, `PostToolUse`)
Allow safe commands silently	Allow rule `Bash(npm test *)`

Anything that must happen every time belongs in configuration the harness enforces, not in a prompt. Claude Code checks hooks and deny rules before allow rules, and a deny wins even when everything else is auto-approved. If you keep adding emphasis to a context-file line and the agent still misses it, make it a hook.

```
{
  "permissions": {
```

```
    "allow": ["Bash(npm test *)"],
    "deny": [
      "Read(./.env)",
      "Edit(src/gen/**)",
      "Bash(git push *)"
    ]
  }
}
```

> **Gotcha:** In Claude Code, `Write(path)` rules never match and `.claudeignore` has no effect. Path rules are checked only for `Edit(...)` (which covers Write) and `Read(...)`.

Subagents

Pays off	Costs
Parallel independent work	Tokens for every subagent

Pays off	Costs
Long output kept out of main context	Time re-gathering context
A reviewer that never saw the code	Same model: not a smarter answer
Fewer tools, e.g. read-only	Only its summary comes back

A subagent runs in a fresh context and returns only its final message. Use one to fan out a wide search, run a noisy test suite, or review a diff with read-only tools enforced by the harness rather than requested in a prompt. Point simple jobs at a cheaper model.

```
---
name: reviewer
description: Reviews a diff against its spec.
tools: Read, Grep, Glob
model: haiku
---
```

```
Compare the changed files to the spec given.
Report only gaps that affect correctness.
Skip style.
```

Verification loops

Check	Strength
“Run the tests” in the prompt	Agent decides when it is satisfied
Stop hook that runs the check	Harness refuses to finish on failure
Fresh-context reviewer	Not swayed by its own reasoning
CI and branch protection	Final gate, outside the agent

An agent with nothing to run that says pass or fail stops when the code looks right, which makes you the test suite. Give it a test, a build exit code, a linter or a fixture diff, and ask for evidence: the command it ran and the output, not “all tests pass”.

```
{
  "hooks": {
    "Stop": [{ "hooks": [{
      "type": "command",
      "command":
        "npx tsc --noEmit 1>&2 || exit 2"
    }] }]
  }
}
```

> **Tip:** Exit code 2 blocks the stop and feeds stderr back to the agent as the reason, so it keeps fixing until the check passes — Claude Code caps repeated blocks to prevent an endless loop.

Reviewing agent diffs

Look for	Why
Invented APIs or options	Plausible names that do not exist

Look for	Why
Scope creep	Unasked refactors hide the real change
Deleted, skipped or loosened tests	The easy way to “pass”
New dependencies	Supply-chain and licence risk
Secrets, logs, broad permissions	Leaks the agent did not notice

Review the diff against the spec, not against the agent’s summary of it. Thoughtworks puts “complacency with AI-generated code” on Hold: agents produce larger change sets that are harder to review, and duplication and churn rise. Read the test changes first — they show what the agent decided “correct” means.

```
git diff --stat main...HEAD
git diff main...HEAD -- '*test*' '*spec*'
git diff main...HEAD -- package.json
```

Further reading

- Claude Code best practices
- AGENTS.md
- Evaluating AGENTS.md (arXiv 2602.11988)
- GitHub Spec Kit
- DORA AI Capabilities Model
- Thoughtworks Technology Radar