A CRASH COURSE

Loop Engineering

Designing systems that prompt your agents for you — and staying the engineer who reads what they ship.

15 concepts | 6 parts | Claude Code + OpenCode

THE WHOLE IDEA, UP FRONT

You stop holding the tool.

You've learned to drive a coding agent: you give an instruction, it edits, you check. One turn, then the next. **You are holding the tool the whole time.**

Now imagine you let go — and build a small system that does the driving for you.

EVERY MORNING, ON ITS OWN, IT…

1. **Looks at what changed overnight**
2. **Decides what's worth doing**
3. **Gives each job to an agent**
4. **Checks the result**
5. **Calls you only for real decisions**

You built it once. After that, it prompts itself.

PREREQUISITE · A 2-MINUTE RECAP

What you already know

These are the raw materials. A loop gives each one a new job.

Plan mode

The agent proposes a plan before changing anything; you approve first.

The rules file

Short, permanent project notes read at the start of every session.

Skills

A saved, reusable instruction loaded only when the task matches.

Subagents

A helper with its own context that does a job and hands back the result.

Connectors / MCP

The standard way to plug an agent into outside tools: GitHub, Slack, a database.

Context management

Keep the conversation lean — the model gets worse and costs more as it fills.

WHERE THIS CAME FROM · MID-2026

"I don't prompt Claude anymore. I have loops running that prompt Claude… my job is to write loops."

— Boris Cherny, creator of Claude Code

Peter Steinberger: *"you should be designing loops that prompt your agents."* Addy Osmani then named the pattern and listed its parts.

None of them say the work got easier. They say the valuable skill moved — from the prompt to the loop.

THE MINDSET SHIFT, IN ONE PICTURE

From holding the tool → designing the loop

The leverage point moves from the prompt to the loop.

PROMPTING

You hold the tool

⌨️ You type

↓

🤖 Agent replies

↓

👀 You read & decide

↺ forever

→

LOOPING

The loop runs itself

Discover → Implement → Verify → Commit

↻ prompts itself

You sit at the human gate
Approve only the risky calls.

IT IS NOT "A BIGGER PROMPT"

A different shape of work

Same agent. Completely different relationship to it.

PROMPTING · WHAT YOU KNOW	LOOPING · WHAT THIS COURSE ADDS
You start each turn	A schedule or an event starts each turn
You read the output and decide what's next	A checker checks it; the loop decides what's next
Stops the moment you stop typing	Keeps running while you sleep
One task, one session, your full attention	Many small runs, your attention only at the gate

⚠️ This is harder than prompting, not easier. A loop running on its own is also a loop making mistakes on its own. The reward is leverage.

WHAT THIS COURSE COVERS

The map

Six parts. The middle four are the anatomy of a loop, one part each.

1 **The Shift**
What a loop is, its six parts, two roads

2 **The Heartbeat**
Making it run on its own — four kinds

3 **The Body**
Isolation, knowledge, action, maker–checker

4 **The Spine**
State that survives between runs

5 **A Loop, Twice**
One full triage-to-PR loop, both tools

6 **Staying the Engineer**
Cost, checking, and the traps that grow

01

PART ONE

The Shift

What a loop is, the six parts it's made of, and the two roads to building one.

CONCEPT 1 · PART 1

From prompting to looping

A loop replaces *you, the operator,* with a system. So where does the value go? Not away — it splits to the two ends a loop can never automate.

🎯

Intent

Saying precisely what you want — clearly enough that the result can be **checked**.

🤝

Accountability

Standing behind what comes out. You own what ships, however it got made.

The loop automates the **middle** — the steps.

The **two ends** stay yours.

You're paid for intent and judgment — not for ignoring how the work got made.

CONCEPT 2 · PART 1

Anatomy of a loop

Five working parts — and one spine that remembers.

1

Heartbeat

A schedule or event that fires the loop.

2

Worktree

Isolation, so parallel agents don't collide.

3

Skill

Project knowledge, written down once.

4

Sub-agents

One writes, a different one checks.

5

Connector

MCP — act in your real tools, not just suggest.

6 **State / Memory — the spine**

The model forgets between runs. The spine is how today's run knows what yesterday's did. **No spine, no loop.**

CONCEPT 3 · PART 1

Two roads to the same loop

The shape of the loop is identical. Only the wiring differs.

Claude Code

Ships the parts

The heartbeat, the run-until-done checker, the isolation, the event intake — all **built-in commands** now.

`/loop` `/goal` `Routines` `--worktree` `Channels`

Cloud **Routines** run on Anthropic's servers — even with your laptop closed. Price: per-account daily caps, and it's a research preview.

OpenCode

The layer below

No built-in cloud scheduler. OpenCode is the **worker** you call; **you** bring the heartbeat from the OS or CI.

`opencode run` `cron / launchd` `GitHub Actions` `serve --attach`

More wiring — but full control, it runs on machines you already have, and it needs **no vendor cloud**.

WHERE WE'RE GOING

The whole loop, early

Six plain steps. Every concept ahead is one line of this picture.

Heartbeat	every weekday at 9am
Spine	read progress.md — what did I do last time?
find	overnight CI failures + open issues
Worktree	for each: draft a fix in its own checkout
Skill	using the project's triage skill
Maker–checker	a separate reviewer grades it → PASS / FAIL
Connector	PASS → open a PR · risky → leave it for a human
Spine again	update progress.md

02

PART TWO

The Heartbeat

What turns one run into a loop. Four kinds — from "you hold it" to "runs without you at all."

PART 2 OVERVIEW

Four kinds of heartbeat

A spectrum from "you hold it" to "runs entirely without you."

← you hold

runs without you →

Concept 4

In-session

Re-run on a timer while the session is open. Stops when you close it.

Concept 5

Run-until-done

Repeats until a **checked** condition is true, then stops on its own.

Concept 6

Scheduled

Runs on a clock — laptop off. Routines, cron, GitHub Actions.

Concept 7

Event-driven

Reacts the moment something happens — a PR, an issue, a message.

💡 **Most real loops use the last two. The first two are how you rehearse and build trust before you let one run unattended.**

CONCEPT 4 · PART 2

In-session loops

Repeat a prompt on a timer *while you watch*. Good for: a deploy, a long test run, a CI job.

Claude Code

The bundled `/loop` skill — give it an interval and a prompt.

```
/loop 5m check if the deploy finished and tell me what happened
```

Runs every 5 min while the session stays open. Close the terminal and it stops — on purpose.

OpenCode

No /loop. The shell is the heartbeat; `opencode run` is the beat.

```
while true; do opencode run "check the deploy" sleep 300 # 5 min
done
```

Same idea, one layer down. Start serve once and --attach to skip the startup cost each beat.

CONCEPT 5 · PART 2

Run-until-done

"Keep going until this is true." The loop must **not** let the agent that did the work decide whether it's done.

Claude Code

Give `/goal` a condition a **command can prove**.

```
/goal All tests in test/auth pass and `npm run lint` is clean.
```

A separate, smaller model checks "are we done?" after each turn — not the one that wrote the code.

OpenCode

A capped shell loop. A **real command** decides whether to stop.

```
for i in $(seq 1 8); do opencode run "fix the tests" if npm test; then break; fi done
```

The test runner is the most honest checker there is — a command can't talk itself into "good enough."

⚠️ **Always give a loop two stops: a success condition *and* a ceiling (max tries, minutes, or spend). A goal with no ceiling will spend your whole budget chasing a goal it can never meet.**

CONCEPT 6 · PART 2

Unattended schedules

The heartbeat that makes loop engineering matter: *"Every weekday at 9am, sort through overnight CI failures."*

Claude Code

Cloud Routines — laptop can be off

Create at claude.ai/code/routines . Bundles a prompt, repos, connectors, and a trigger; runs on Anthropic's servers.

Headless in your own cron

```
0 9 * * 1-5 claude -p "summarize CI failures" >> ~/cron.log
```

OpenCode

Your machine — cron

```
0 9 * * 1-5 opencode run "summarize CI failures"
```

The cloud — GitHub Actions

```
on: schedule: - cron: "0 9 * * 1-5"
```

No machine of yours needs to be on.

CONCEPT 7 · PART 2

Event-driven

Not "check every hour" — *"react the moment X happens."* A PR opens, an issue is filed, a message lands.

Claude Code

Routines + GitHub webhook

A routine can run on a push or a new PR — not only on a clock.

Channels Telegram · Discord · iMessage

Push messages from outside straight into a running session — the event-driven partner to scheduling.

OpenCode

Install once: `opencode github install`

After that it reacts to repository events inside your Actions runners:

pull_request **issues** **/oc comment**

For a pull_request with no prompt, OpenCode reviews the PR by default.

03

PART THREE

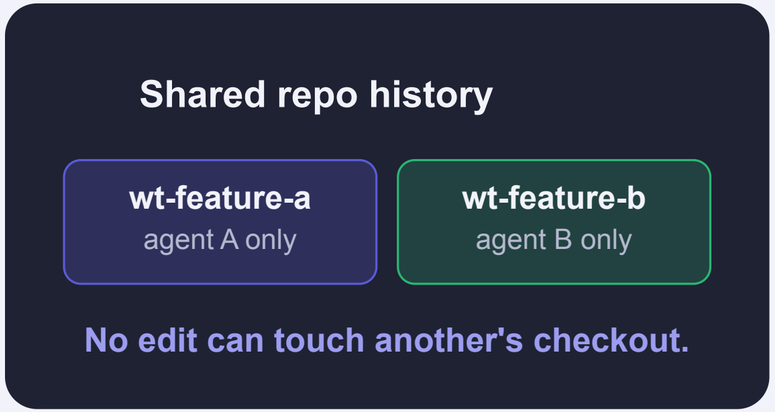
The Body

What the loop *does* on each beat: isolation, knowledge, action, and the maker–checker split.

CONCEPT 8 · PART 3 · ISOLATION

Worktrees

Two agents editing at once overwrite each other — like two people editing the same lines without telling each other. A **git worktree** gives each its own checkout.



Claude Code

Built in. Use `--worktree`, or set `isolation: worktree` on a subagent — it cleans itself up after.

OpenCode

No single flag — use git's own worktrees and point a run at each one. Same isolation, made plain.

CONCEPT 9 · PART 3 · KNOWLEDGE

Skills, so no run is "day one"

A loop runs cold every beat. A **skill** is your project knowledge written down once, in a SKILL.md the agent reads on every run.

The rule

Anything you would otherwise re-explain on every run belongs in a skill.

The triage steps. The project habits. The "we don't do it this way because of that one incident." All of it lives in the skill — so the loop builds on itself instead of restarting.

Without a skill: a wall of instructions pasted into a schedule nobody keeps up to date. Re-figured out every beat. Wasted tokens.

↓

With a skill: the scheduled prompt becomes one line — "run the daily-triage skill" — and the skill holds the detail.

Same format in both tools: a folder with a SKILL.md of instructions, plus optional scripts and references.

CONCEPT 10 · PART 3 · ACTION

Connectors — the loop acts

A loop that can only read your files can only *talk*. Connectors (built on **MCP**) let it *do*.

A loop that only reads

"Here is the fix."

…and that's all it can do.

→

A loop with connectors

Opens the PR, links the ticket, posts to the channel once CI is green.

- **Open a PR**
- **Update a Linear ticket**
- **Post to Slack**
- **Query a database**
- **Call a staging API**

Both tools speak MCP, so the protocol transfers — but packaging & auth (local vs hosted, OAuth, permissions) often need tool-specific wiring.

Maker–checker: sub-agents

The agent that writes the work must not be the agent that approves it.

✍️

The maker

Explores and implements. Often a strong model. Sure it got it right.

→

🔍

The checker

Different instructions, often a different (sometimes cheaper, read-only) model. Catches what the maker was sure about.

Claude Code: define subagents in .claude/agents/ as a team — one explores, one implements, one checks against the spec & tests.

OpenCode: give the checker its own read-only model; the maker calls it with an @mention . A strong model implements, a focused one checks.

CONCEPT 11B · PART 3

Codify the body: dynamic workflows

Package the whole orchestration of a beat — find work, fan it out, grade it — into one **rerunnable script**.

Claude Code

Ask in plain words, trigger with effort `ultracode`. Press s in /workflows to save a run as a rerunnable command.

Two guardrails: agents are capped (~16 at once, 1000 per run), and a run's memory lives only within that run.

OpenCode

No /workflows — the script you write **is** the workflow. The capped for loop and the &/wait fan-out are a hand-rolled version of the same idea.

Full control, no agent cap — at the price of maintaining the orchestration yourself.

⚠️ **A workflow is the body of one beat — not the loop.** It runs once and forgets. The loop is the composition: a **heartbeat** fires the beat, the **workflow** is the body, and a **progress file** is the spine the next firing reads.

04

PART FOUR

The Spine

State that survives between runs — the one part people forget, and the one that makes a loop a loop.

CONCEPT 12 · PART 4

State that survives between runs

The model forgets everything between runs. **No spine = the same first step, forever.**

The rules file CLAUDE.md / AGENTS.md

Steady habits read on every run. Keep it short — you pay for it on every beat.

A progress file progress.md

What was tried, what passed, what's still open. The real spine. Tomorrow's 9am run opens it and picks up where today's stopped.

The habit: read it at the start, update it at the end — every run.

```
# progress.md — the loop's memory

## Done
- 2026-06-22: fixed flaky test in test/auth (retry on token refresh)

## In progress
- Dependency audit: 3 of 7 patched; lodash bump blocked by an API change

## Open / needs a human
- CVE in image lib — fix changes the output format, escalating to a maintainer
```

05

PART FIVE

A Loop, Twice

One full morning-triage-to-PR loop, with real files — built in both tools.

BEFORE YOU LET ANY LOOP RUN ALONE

The minimum safe loop checklist

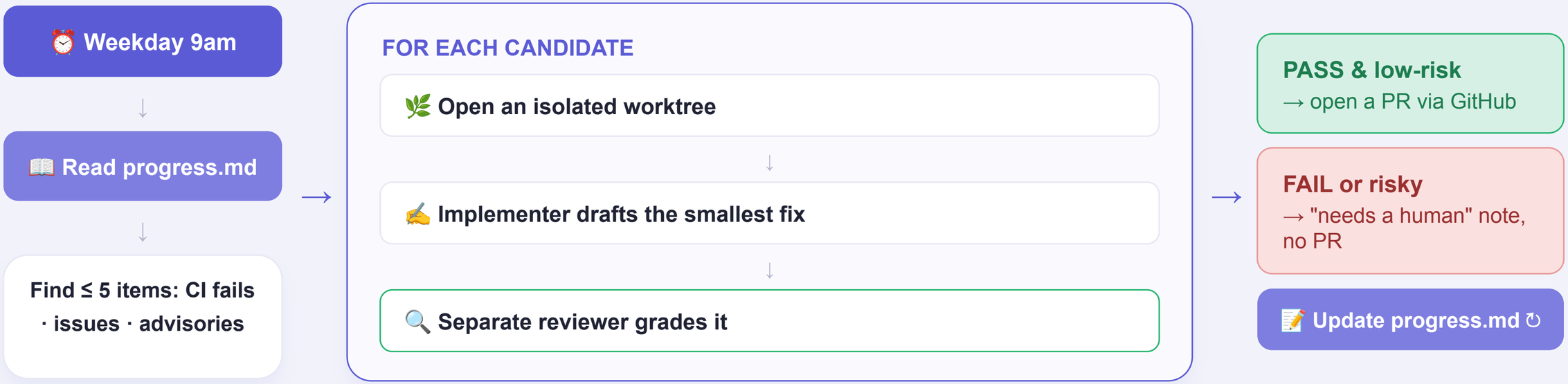
Seven things. Miss one and the loop is unsafe, forgetful, or invisible.

- ✓ **Success condition** — how it knows it's done
- ✓ **Ceiling** — max tries, minutes, or spend
- ✓ **Isolated branch / worktree**
- ✓ **Read-only checker** — grades, can't edit
- ✓ **State file** — the spine
- ✓ **Human gate** — risky work goes to a person
- ✓ **A log or notification** — so a failure overnight is visible, not silent

PART 5 · THE SHAPE, BOTH TOOLS

The morning-triage loop

Same design in both tools. Only the heartbeat and where it runs differ.



PART 5 · THE TWO FILES THAT DO THE WORK

The shared skill & the reviewer

One skill file works in both tools. The reviewer is the maker–checker split, made real.

daily-triage / SKILL.md

The steps, written once

1. Read your memory first — progress.md
2. Find the work — CI fails, issues, advisories (≤ 5)
3. Work each in an isolated checkout; send the diff to the reviewer
4. Decide from the verdict — PASS & low-risk → PR; else → flag
5. Update your memory last

Rules: never > 5 PRs/run · never touch main · when in doubt, escalate.

reviewer.md

The checker — read-only

```
You are a strict, read-only reviewer. You never edit files. 1. Run the tests & linter yourself. 2. Check against conventions + spec. 3. Reply PASS or FAIL with reasons.
```

A change that only "looks fine" is not a PASS. Tests must actually pass.

Runs on a cheaper model — different from the maker.

PART 5 · THE ONLY REAL DIFFERENCE

Wiring the heartbeat

Everything in the middle was the same. Here's where the two roads diverge.

Claude Code → a Routine

Create one at claude.ai/code/routines : weekday-9am, your repo, GitHub + Slack connectors. Point its prompt at the skill:

```
Run the daily-triage skill. Read progress.md first; update it last. Reviewer subagent grades each fix; PR only on PASS.
```

Runs at 9am whether your laptop is open or not — within your plan's daily cap.

OpenCode → GitHub Actions

Runs in the cloud, no machine of yours awake. The Action is the heartbeat; opencode run is the worker.

```
on: schedule: - cron: "0 9 * * 1-5" prompt: Run the daily-triage skill. Invoke @reviewer; PR only on PASS.
```

Want it on your machine? The same prompt runs from a cron line — only the heartbeat changes.

PART 5 · WHAT YOU WAKE UP TO

One real morning

You designed all of it once. Here's a single run (illustrative).

[09:00] daily-triage fires
→ reads progress.md: 1 item still "in progress", nothing new flagged
→ finds: 2 CI failures overnight, 1 new npm-audit advisory

CI #1 (flaky auth test): drafts claude/fix-auth-retry
reviewer → **PASS** → opens PR #142, links the issue

CI #2 (type error in report.ts): drafts claude/fix-report-types
reviewer → **PASS** → opens PR #143

advisory (image library): safe fix changes the output format
reviewer → **FAIL** (public behaviour change)
→ writes it to "Open / needs a human", opens no PR

→ updates progress.md, exits

[you, 09:30] two PRs to review, one flagged item to decide on. You typed nothing.

06

PART SIX · THE MOST IMPORTANT PART

Staying the Engineer

Three problems get *bigger* as your loops get better — not smaller.

Token cost — cadence is the lever

Same loop, ~$0.21 a beat. The money is in how *often* it runs, not which command you typed.

Cap every loop
Max tries, minutes, or spend. Always.

Match model to job
Strong model to plan & check, cheap one to do. The single biggest saving.

Run it less often
Hourly beats every 5 min — and is ~12× cheaper.

CONCEPT 14 & 15 · PART 6

The two traps a smooth loop grows

Concept 14

Checking is still your job

The maker–checker split makes "it's done" mean *something* — but "done" is a **claim, not a proof**.

Trust the loop to do the work. Check the work before it counts.

Concept 15

Don't stop understanding

The faster a loop ships code you didn't write, the wider the gap between your project and what you actually understand.

Designing the loop keeps you engaged *when you do it with care* — and lets you stop thinking *when you do it to avoid the work* . Same act, opposite result.

Build the loop — but build it like someone who plans to stay the engineer, not just the person who presses go.

PART 6 · BEFORE YOU TRUST IT OVERNIGHT

When a loop fails while you're asleep

An unattended loop fails unattended too. Make it observable *first*.

Send output where you'll see it
A log, a Slack/Discord message, the Triage inbox — not the terminal you closed.

Write a line every run
Even on failure. A silent failure is the worst kind.

⏮ **Keep runs replayable**
`opencode export` / a Routine's run history in the web UI.

Fail loud at the ceiling
Leave a clear "needs a human" note, don't just stop.

Earn the nightly slot
Run it hourly and watched for a few days before you let it run nightly. When something looks wrong, read the spine first.

PRACTICE · BUILD THEM YOURSELF

Five loops, easy → hard

Two rules every time: **use a throwaway repo**, and **set a ceiling first**.

1	**A watch loop** — notice a long task finished, say so once. (Concept 4)	easy
2	**Make the tests pass, then stop** — a command decides "done." (Concepts 5, 11)	easy–med
3	**The morning brief with a memory** — run twice; the 2nd builds on the 1st. (Concepts 6, 12)	medium
4	**A fix loop with a real checker** — a planted bad fix must get FAIL. (Concepts 8, 9, 11)	med–hard
5	**Your own daily loop** — all six parts; runs unattended for a week. (Capstone)	capstone

THE ONE-LINE SUMMARY

Stop prompting your agent turn by turn. Design the loop that prompts it for you — a heartbeat, four working parts, and a spine that remembers.

🫀 Heartbeat · 🌿 Worktree · 📘 Skill · 🔍 Maker–checker · 🔌 Connector · 🦴 Spine

…and stay the engineer who reads what it ships.