08: Agentic workflows

How much autonomy actually works

Why this matters

- A coding agent is a loop that runs until it *thinks* it's done
- Most of the skill is controlling that loop
- Autonomy is a dial you turn one notch at a time
- Only turn it up when your checks catch what you stop seeing
- The productivity evidence is mixed and keeps changing

The agent loop

think → act (call a tool) → observe → think → … until the model decides it's done. Then you verify.

> "Claude stops when the work looks done." … "In practice that makes you the test suite."
> (Claude Code best practices)

A check that returns pass or fail beats asking nicely.

Autonomy is a dial

Level	What it looks like	Who checks each change
1. Autocomplete	The editor suggests the next line	You, as you type
2. Chat	Paste code in, paste answers back	You, line by line
3. Supervised agent	One agent edits and runs commands while you watch	You, per step or diff
4. Parallel supervised	Several agents, each on its own checkout	You, per branch
5. Background or cloud	Hand off a task, get a PR later	You or a reviewer, at the PR
6. Fully autonomous	Specs in, software out	Automated checks only

Yegge's eight stages: 1–4

Stage	Yegge's words	Moving up safely
1	"Zero or Near-Zero AI"	One small, well-tested agent change. Read the whole diff.
2	"Coding agent in IDE, permissions turned on"	Allowlist the safe commands you keep approving, not everything
3	"Agent in IDE, YOLO mode"	A sandbox or auto mode, plus hard deny rules
4	"In IDE, wide agent"	Tests are the contract. Ask for output, not "done."

From "Welcome to Gas Town" (January 1, 2026). The right column is this course's advice.

Yegge's eight stages: 5–8

Stage	Yegge's words	Moving up safely
5	"CLI, single agent. YOLO."	Review every PR, never skip it. Add an independent reviewer.
6	"CLI, multi-agent, YOLO."	One worktree per agent, non-overlapping tasks, a task list
7	"10+ agents, hand-managed."	Tracking and merge discipline, or step back to 6
8	"Building your own orchestrator."	Treat it as production software: tests, budgets, kill switches

Reading the chart

1. The stages measure delegation, not quality
2. Yegge assumes YOLO from stage 3 up; sandboxes get you most of the speed
3. Moving down is allowed

Part 2: In practice

The workflow that holds up in daily use

The core loop

explore → plan → implement → verify → commit

Step	What you ask for
Explore	Read-only. "Don't change anything yet."
Plan	A plan you read before any edit (plan mode)
Implement	A failing test first, then make it pass
Verify	The command it ran and the output
Commit	Small commits: the test, then the fix

Prototype first

> "plan to throw one away; you will, anyhow."
> (Fred Brooks, *The Mythical Man-Month*, 1975)

A throwaway version took weeks then. Now it takes minutes.

Level	What you build	Feedback you get
Application	Clickable mockup, single-page app, Artifact or Claude Design prototype	"Not that button, and where's the export?"
Code	A spike: two API shapes, two module layouts	Which code you'd rather read and keep

Prototyping well

1. Timebox it and mark it as a draft: a `spike/` branch, a README banner
2. Write what you learned into a short spec
3. Rebuild or harden it on purpose; never let it slide into production

A prototype *looks* finished. That's the risk.

Tests are the agent's contract

The contract only works if the agent can't quietly rewrite it.

- Write or approve the tests yourself; commit them separately
- Make each test fail first
- Check the test diff: `git diff --stat main -- tests/`
- Enforce with a hook or deny rule, not just a `CLAUDE.md` line
- Keep some checks out of the agent's reach

Safety nets and steering

- One branch per task; small diffs that keep the build green
- `git add -p` and `git diff --stat main`: what did it touch?
- Checkpoints (`/rewind`) work *alongside* git, not instead of it
- Watch the first minute; press `Esc` the moment it drifts
- After two failed corrections, `/clear` and write a better prompt

Parallel agents need worktrees

A git worktree is another checkout of the same repo, in its own folder, on its own branch.

- One task, one agent, one worktree, one branch
- Pick tasks that touch different files
- Keep a written task list you and the agents can see
- Review every branch before merging
- Mind the disk and the bill; remove merged worktrees

Before you walk away

Five questions before any unattended run:

1. Does it know what done looks like?
2. Can it check its own work?
3. What is it allowed to touch?
4. What stops it?
5. Where does the result go, and who signs off?

Failure modes and counter-moves

Failure	Counter-move
Vague task	Name the file, the symptom, and what "fixed" means
Hallucinated APIs and packages	Typecheck every step; verify new dependencies by hand
Fake or tautological tests	Approve tests first; make them fail first
Over-engineering	"Smallest change. No new abstractions or dependencies."
Sycophancy	Ask it to argue the other side, and for evidence
Context exhaustion and drift	One task per session; `/clear` between tasks
Declaring victory early	Require the command and its output
Reward hacking	Hidden tests and evaluators the agent can't edit

Part 3: How much autonomy works?

The evidence on both sides, and what the successes share

The case for, and the case against

For	Against
METR time horizons: doubling every ~7 months, faster lately	METR trial, early 2025: experienced developers 19% *slower*
C compiler: 16 agents, ~2 weeks, just under $20,000	Told not to cheat, still reward hacking in 70–95% of runs (one task)
Browser: over 1 million lines in about a week	Long tasks drift, then declare the job done
StrongDM: no human writes or reviews the code	The last 30% is the hard part
Anthropic: over 80% of merged code by Claude (May 2026)	Human review becomes the bottleneck

What the successes have in common

Project	Automated oracle	Greenfield?	Harness built by	Cost
C compiler	GCC as reference, compiler test suites	Yes	Carlini: prompt, loop, tests, CI, spec	~$20,000
Cursor browser	Web standards specs, builds, tests	Yes	Planner, worker, judge; several failed designs first	Trillions of tokens
StrongDM	Holdout scenarios, digital twins	Their own products	A whole team	$1,000+ per engineer per day
Gas Town	Yegge's tests and judgment	His own projects	Yegge: four orchestrators in 2025	Lost work; multiple subscriptions

Autonomy scales with the strength of the automated oracle.

On prototypes

> **Bryan's take:** I like draft-driven or prototype-driven development, and you can do it with AI now. From a coding point of view or an application point of view, it's easier to comment on something you can see than to conceptualize it from scratch.

On Yegge's chart

> **Bryan's take:** On Yegge's chart I'm one or two from the top, roughly stage 6–7: several CLI agents running in parallel, each on its own task, managed by hand. I haven't built my own orchestrator. I still decide what gets merged, though honestly that's emergent to an extent. There are projects I mostly just vibe code. For work, I actually read the code.

On full autonomy

> **Bryan's take:** I don't believe fully autonomous agentic workflows work reliably today. Maybe I'm just too dumb; I haven't had good results, but some people claim they work. Gas Town, in particular, really did not work for me.

Try it

On your own, fix a verified bug, then go up one notch:

1. Fix a small, real bug with the full loop. Save the evidence.
2. Review it in a fresh session. Mark each finding real or not.
3. Start a second agent in its own worktree. Where did your attention go?

As a class or team, climb the autonomy ladder: hidden acceptance tests, three conditions, score the results.

Discussion

1. Yegge's stages measure delegation. What would a chart of *outcomes* look like?
2. What's the closest thing to an oracle in your codebase?
3. METR's developers felt 20% faster and measured 19% slower. Why the gap?
4. When should you move *down* a stage?

Read more

- Module 08: Agentic workflows
- Anthropic, Best practices for Claude Code
- Steve Yegge, Welcome to Gas Town (January 1, 2026)
- Becker et al. (METR), the early-2025 developer productivity trial (2025)