10: AI code review

Layered review, verified findings, and a human who owns the merge

Why this matters

Agents now write code faster than people can review it.

Signal	Source
Over 80% of merged code authored by Claude; ~8× code merged per day vs 2024	Anthropic, 2026
Review and merging "have increasingly become the bottlenecks"	Cursor, December 2025
AI adoption: higher throughput, *lower* delivery stability	DORA, 2025

Review has to keep up with the extra code.

What code review is for

Reason	Do AI reviewers help?
Finding defects before users do	Mostly, yes
Keeping code maintainable	Somewhat
Sharing knowledge across the team	No
Accountability: a second person signed off	No

How AI reviewers differ

- How much it sees: the diff, nearby files, or an index of the whole repo
- Whether it verifies its own findings before posting
- Whether it only reads, or can run tests and browsers
- Which instruction file it reads, and from which branch

Good at, weak at

Good at	Weak at
Local logic errors: off-by-one, nulls, a missing `await`	Intent: is this the right change?
Edge cases the tests skip	Architecture: where is the system heading?
Mismatches with nearby code or the spec	Effects outside the context it retrieved
Security smells: injection, missing auth, secrets	Organizational knowledge
Reading every line of a 2,000-line PR	Telling cosmetic from critical, untuned
Speed: minutes, not days	Staying quiet when nothing is wrong

Part 2: In practice

Tools, layers, triage, and the human checklist

The tools (as of November 2026)

Tool	Worth knowing
Claude Code `/code-review`	Any plan; fresh-context subagent; `low` effort: fewer false positives
Claude ultrareview	Cloud fleet of reviewer agents; every finding reproduced and verified
Claude Code Review (managed)	GA October 15, 2026; never approves or blocks a PR
OpenAI Codex review	`/review` locally, `@codex review` on GitHub; flags only P0 and P1
GitHub Copilot code review	Not a required approval unless an admin enables Copilot approvals
Cursor Bugbot	Rules in `.cursor/BUGBOT.md`; autofix via a cloud agent
Your own reviewer	A subagent or CI script; the cheapest cross-model review

A layered review workflow

author self-check → CI → independent bot → verify findings → human → merge → watch

Layer	Who	Looks for
1. Self-check	The author agent	Tests, lint, typecheck, diff vs spec
2. Independent bot	Fresh context, ideally another lab's model	Local bugs, edge cases, security smells
3. Human	A named person	Intent, design, risk

Verify findings before acting

```
The reviewer says: "<finding>". Don't fix it yet.
First, write a test that demonstrates the failure,
run it, and show me it fails. If you can't make it fail,
explain why the scenario can't happen and we'll dismiss it.
```

- AI critics hallucinate bugs
- "Fixing" every finding bloats the code

Severity triage

Severity	Examples	Action
High	Data loss, security hole, outage, broken migration	Block the merge until fixed and verified
Medium	Unhandled edge case, missing test, performance cliff	Fix now, or a ticket with an owner
Low	Naming, minor duplication	Author's choice
Noise	Formatter-enforced style, speculative "consider…"	Tune the bot so it stops

Agree on the levels before the bot starts commenting.

False-positive fatigue

Every useless comment teaches people to skim the next one.

- Raise the severity threshold to tune for precision
- Tell it what to skip: "skip style," "skip generated files"
- React to comments; some tools learn from that
- Track the share of bot comments that lead to a change

In field studies, 39% to 74% did.

Configure the reviewer

Start with two or three concise rules (OpenAI's guidance):

```
## Code Review Rules
- Amounts are integers in cents.
  Flag any float arithmetic on money.
- Every migration must be reversible.
  Flag a migration with no down step.
- Skip files under src/gen/; they are generated.
```

Check which branch the rules come from. If a PR can edit its own review rules, that's a prompt-injection path.

Reviewing AI-written PRs: the checklist

Look for	Look for
Intent: does what the issue asked, nothing else	Swallowed errors: empty `catch`, silent defaults
Hallucinated APIs: does every call exist?	Unnecessary abstraction: one-implementation interfaces
New dependencies: real, spelled right, needed, licensed	Dead code: debug output, duplicate helpers
Tests that test nothing: mocks, never seen failing	Security: validation, auth, injection, secrets
Licensing and provenance: copied-looking blocks	Consistency: existing patterns and names
Operability: logs, metrics, reversible migrations	

Spot these on sight

```python
# Swallowed error: "no orders" looks like "database down"
def get_orders(user_id):
    try:
        return db.query(
            "SELECT * FROM orders WHERE user_id = %s", user_id)
    except Exception:
        return []
```

```javascript
// Tautological test: checks the mock against itself
it("fetches the user", async () => {
  const api = { getUser: jest.fn().mockResolvedValue({ id: 1 }) };
  const user = await api.getUser(1);
  expect(user).toEqual({ id: 1 });
});
```

Keep PRs small

PR size	Got findings	Average issues
Over 1,000 changed lines	84%	7.5
Under 50 lines	31%	0.5

- Set a soft size limit for human review
- Use stacked PRs for big features
- Mark generated files so reviewers skip them

The human who merges owns it

- If you can't explain a line, it doesn't merge
- A PR template with "what the agent did vs what I did"
- Branch protection with at least one human approval
- Required checks no one can skip
- Co-author trailers on agent commits
- Incident reviews ask "what did review miss?"

Team metrics

Watch	Don't measure, or don't target
Review latency: time to first review, to merge	Lines of code
PR size distribution	Number of PRs merged
Escaped defects by severity	Percentage of code written by AI
Change failure rate and revert rate	Number of bot comments
Bot comment acceptance rate	
Share of PRs with a substantive human review	

Goodhart's law: when a measure becomes a target, it stops being a good measure.

Part 3: What the research says

Real findings, real noise, and an unmeasured outcome

Industrial field studies

Study	Setting	Key result
Cihan et al., 2024	One company, PR-Agent bot, 4,335 PRs	73.8% of comments resolved; PR closure time *rose* (5h52m → 8h20m)
Aðalsteinsson et al., 2025	WirelessCar Sweden	AI-led reviews preferred, depending on familiarity and severity
Sun et al., 2025	16 GitHub Actions, 22,000+ comments	Concise, code-bearing, hunk-level comments got acted on
Tantithamthavorn et al., 2026	Atlassian RovoDev, one year	38.7% led to changes; cycle time down 30.8%

What the studies agree on

1. Bots find real issues; 39% to 74% of comments get acted on
2. Speed depends on design and context
3. Noise is the main complaint, and precision drives adoption
4. Short, specific comments with code get acted on
5. The outcome is rarely measured; few studies track escaped defects

On multi-agent review

> **Bryan's take:** Multi-agent review is one of the most valuable things you can do with these tools: several agents, and ideally several models, reviewing each other's work. Having a different agent review the work is good. It isn't the only thing to take away from this module, but it's an easy place to start.

On which models review

> **Bryan's take:** For code, Claude and Codex are the two I trust right now, so they review each other: whichever one wrote the change, the other one reviews it. Every finding still gets checked before anyone acts on it. Model rankings change fast; see module 03.

Try it

On your own, a seeded-bug benchmark:

1. Plant five bugs in a copy of a clean 100–300 line PR
2. Run at least two reviewers, from different labs if you can
3. Score recall, false positives, time, and cost
4. Which reviewer would you trust as Layer 2?

As a class or team, run a two-week review-bot pilot against a baseline.

Discussion

1. If bots catch more local bugs, what should humans review instead?
2. One study saw PRs close slower with a bot; another, 30% faster. Why?
3. Should an AI reviewer ever approve a merge on its own?
4. How would you detect rubber-stamping?

Read more

- Module 10: AI code review
- McAleese et al., LLM Critics Help Catch LLM Bugs (2024)
- Cihan et al., Automated Code Review In Practice (2024)
- Bacchelli and Bird, Expectations, Outcomes, and Challenges of Modern Code Review (2013)