Code Review in the AI Era

Verified 2026-11-02

Reviewing and being reviewed when agents write much of the code — PR size, checklist, agent-diff traps, comment labels, turnaround, AI reviewers.

Contents

Mental model 2

The approval standard 2

Size and stacking 3

Reviewer checklist, in order 5

Reading an agent's diff 6

Comments and severity labels 8

Turnaround 9

Preparing your own change 10

AI reviewers 12

Further reading 13

Mental model

Review exists to verify **intent and risk**: does this change do what it claims, and what breaks if it is wrong? Everything a linter, formatter or type checker can say, automate, so humans spend attention only where tools cannot. With agents writing much of the code, the bottleneck has moved from typing to review — which makes small, well-described changes and fast turnaround the highest-leverage habits a team has.

The approval standard

Question	Decides
Does it improve overall code health?	Approve, even if imperfect
Is it a feature we do not want?	Reject early, kindly
Is it style the guide does not cover?	Author's call
Is it only polish?	Prefix `Nit:` and approve

Google's senior rule: approve once a change **definitely improves the code health of the system**, even if it is not perfect. There is no perfect code, only better code; demanding polish on every line slows improvement down. Technical facts and data overrule opinions, and on style the style guide is the authority — anything it does not cover is personal preference.

```
Blocks:     correctness, security, data loss,
            no tests for new behaviour
Does not:   "I'd have named it differently",
            formatting a tool can fix
```

Size and stacking

Size	Reading
~100 changed lines	Usually reasonable
~1000 lines	Usually too large
Refactor + feature mixed	Split into two
Generated or mechanical	Separate, say so in the description

Small changes are reviewed faster and more thoroughly, are less likely to hide bugs, and are easier to roll back. Google gives no hard limit, but treats 100 lines as usually reasonable and 1000 as usually too large. When work depends on unmerged work, **stack** it: a chain of small dependent PRs, each reviewable alone, instead of one big one.

```
#1  Add RefundPolicy type + tests     (80)
#2  Use policy in RefundService       (120)
     based on #1
#3  Expose refund limit in admin UI   (90)
     based on #2
```

> **Gotcha:** Agents make big diffs cheap to produce but not cheap to review. A 900-line agent PR still costs 900 lines of reviewer attention — ask for it to be split before reading it.

Reviewer checklist, in order

Order	Check
1	Should this change exist? Read the description
2	Correctness of the main file(s) and design
3	Tests: new behaviour covered, nothing weakened
4	Security and data: input, auth, migrations, PII
5	Operability: logs, metrics, flags, rollback
6	Readability: names, comments, complexity

Order matters because an early finding can make later work moot. Take a broad view first and reject a change that should not happen before reading every line. Then find the main part and send design comments immediately, even if you have not finished the rest. Readability comes last because it is the cheapest to fix and the easiest to automate.

```
description: why, what, how tested?
main file first — design comments now
tests: would they fail without the fix?
migration reversible? secrets? authz?
how do we know it broke in prod?
then names, comments, nits
```

Reading an agent's diff

Look for	Because agents
Calls to APIs that do not exist	Invent plausible signatures
Changes outside the task	Drift into "while I was here" edits
Deleted, skipped or loosened tests	Turn red to green by removal
Duplicated helpers	Miss the existing one
Confident description vs real diff	Summarise intent, not result

Read agent output as you would a capable stranger's: the code is fluent, so fluency proves nothing. Verify every external call against the library's real API and check that the test diff only adds or tightens assertions. LinearB's 2026 benchmark of about 8.1 million pull requests found AI-generated PRs merged within 30 days 32.7% of the time versus 84.4% for unassisted ones, and waited about 4.6 times longer for review (vendor survey data).

```
- expect(total).toBe(108.5)
+ expect(total).toBeGreaterThan(0)
  ^ assertion weakened: block, ask why
- it('rejects over-refund', ...)
+ it.skip('rejects over-refund', ...)
  ^ test disabled: block
```

> **Warning:** “All tests pass” from an agent is a claim, not evidence. Check CI on the actual head commit, and check that the tests it cites exist and assert something.

Comments and severity labels

Prefix	Means
(none)	Must address before merge
`Nit:`	Polish; author may ignore
`Optional:` / `Consider:`	Good idea, not required
`FYI:`	Informational, for the future

Comment on the code, never the person, and explain why — a reason lets the author fix the next case without you. Labels stop authors from treating every remark as mandatory and burning a round trip on a nit. Say when something is good, too, with the reason.

```
Bad:  Why did you use threads here?
Good: The concurrency model adds complexity
      but I see no throughput gain here —
      can we keep this single-threaded?
Nit:  `cnt` -> `retryCount` reads better.
```

> **Tip:** When only nits remain, approve with comments (“LGTM with comments”) instead of requesting another round.

Turnaround

Rule	Source of the rule
Respond within one business day, max	Google
Don’t break focus to review	Google
Multi-approval, hours-long review	DORA pitfall
Prefer synchronous review when ready	DORA

Slow review makes authors batch changes into bigger PRs, which are slower to review: a spiral DORA names as a top obstacle to trunk-based development. Review when you finish a focused task rather than mid-flow, but treat the first response as due the same day. If you cannot do a full pass, send the high-level comments now.

```
Team review SLA (example)
first response   < 4 working hours
P0 / revert      immediately
> 400 lines      author asked to split
```

Preparing your own change

Include	Why
First line: imperative summary	Stands alone in history
Why, and decisions not in code	Reviewers and future readers
How you tested it	Evidence, not assertion

Include	Why
Risk and rollback	Reviewer can focus there
A self-review pass first	Catch your own noise

The description becomes permanent history; future engineers will find the change by searching for it. Write what changed and why, not “fix bug”. For agent-written changes, say so and say what you verified yourself.

```
Limit refunds to the original charge amount

Refunds above the charge were accepted and
paid out (INC-2291). Adds RefundPolicy and a
422 for over-limit requests.

Tested: unit tests for equal/over/under;
replayed INC-2291 request on staging.
Rollback: revert; no migration.
```

AI reviewers

Use them for	Do not use them for
A first pass before a human	The approval decision
Spotting missing tests, typos	Judging intent or product fit
Summarising a large diff	Trusting “no issues found”

An AI reviewer is one more linter with a wider range and more false positives. Run it before requesting human review so the author fixes the obvious, then let a human who understands the intent own approval. DORA 2025 found AI raises throughput but still correlates with lower stability; review is part of the control system that keeps the extra change volume from becoming incidents.

```
pipeline: lint -> tests -> AI review
           (author fixes) -> human review
human approves; AI comments are input,
never a required or sufficient check
```

Further reading

- Google Engineering Practices — Code Review
- Google — Small CLs
- Google — How to write code review comments
- DORA — Trunk-based development
- 2025 DORA Report announcement