The New Code Review Culture: 9 Rules for High-Speed, High-Quality Engineering

qodo

1 Preserve strong ownership over everything that ships

No matter who generated the code, a human remains accountable for correctness, reliability and long-term maintainability. AI can suggest code, but only engineers can take responsibility for what reaches production.

Best practice:

- Treat AI as an assistant, not an author.
- Hold AI-generated changes to the same review bar as handwritten code, including tests, edge cases, and failure modes.

2 Limit PR size for fast, reliable reviews

AI can create large batches of code, but human attention does not scale linearly. Very large PRs show diminishing returns in review time and effectiveness compared to smaller changes.

Best practice:

- Break work into small, coherent PRs that each do one clear thing, end-to-end.
- Avoid "AI dumps" of large, mixed-purpose changes that slow review, hide defects, and increase time-to-merge.

3 Review the quality of decisions, not just the code

As AI becomes part of the creation process, code review increasingly evaluates how well the contributor directed and curated AI output.

Best practice:

- Look for thoughtful structuring, clear intent, and selective acceptance of AI suggestions.
- Encourage reviewers to assess both the change and the underlying reasoning.

4 Review changes with full context, not just the diff

AI-generated changes can appear correct in isolation while conflicting with broader system behavior or intent. High-quality review for brownfield changes should validate the preservation of intended behavior and should fit cleanly into existing system boundaries.

Best practice:

- Review changes in the context of the repository, upstream/downstream dependencies, and real execution paths, not just the local diff.
- Confirm that the change aligns with documented intent, non-functional requirements, and any domain-specific constraints or assumptions.

5 Adopt team-specific workflows and generation patterns

Different domains, stacks, and problem types require different prompting styles and development rituals. Effective teams standardize domain-aware prompting, test strategies, and review checklists for each area of the codebase.

Best practice:

- Document generation and review patterns that work well for each domain (e.g., migrations, APIs, infra, etc).
- Maintain local guidelines or templates—prompt snippets, test expectations, and review questions—that reflect each domain's risks and nuances.

6 Use layered governance: global standards, domain rules, team rules

Quality at scale depends on clear guardrails, not ad hoc judgment. Clear standards reduce cognitive load on reviewers and make quality less dependent on individual heroics.

Best practice:

- Define non-negotiable org-wide rules (e.g., testing requirements, security constraints, review coverage) that apply to all code, AI-generated or not.
- Add domain-level patterns where needed.
- Allow teams to maintain their own working conventions within these boundaries, so governance supports speed instead of fighting it.

7 Split greenfield and brownfield AI workflows

Greenfield and brownfield work demand different AI strategies and review checklists. Exploration is cheap in new code, but established systems benefit more from mechanical, tightly-scoped, test-driven changes.

Best practice:

- In greenfield areas: leverage AI for rapid iteration and exploration.
- In brownfield areas: use structured, mechanical, test-driven transformations for safety and consistency.

8 Prevent code quality drift as models change

Model versions, prompts, and tools evolve, and small shifts can quietly erode quality if review workflows stay static. Teams that periodically re-audit AI-generated changes and patterns catch regressions earlier and maintain trust in automation.

Best practice:

- Regularly sample and re-review AI-generated code, migrations, and common patterns to verify they still meet standards as tools evolve.
- Use AI-assisted review to flag risky patterns, unexpected diff shapes, or deviations from house style before they reach human reviewers.

9 Make excellence visible

Code review culture is built more by what gets celebrated than by what gets mandated.

Best practice:

- Publicly recognize review behaviors that prevented incidents, improved reliability, or clarified architecture, not just "fast approvals."
- Share effective rules, checklists, and workflows across teams
- Use visible quality signals to spark healthy standards-raising competition.