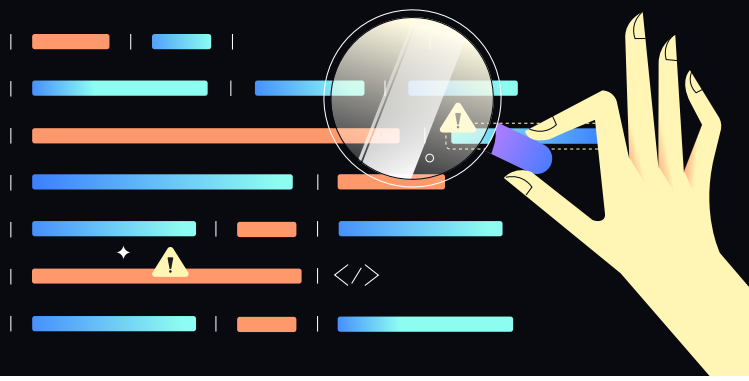
The ultimate code review checklist



1. Functional correctness & logic

Make sure the code actually does what the ticket says it should do.

- [ ] **Requirements met:** Are all acceptance criteria and feature purposes met as described in the ticket/spec?
- [ ] **Edge cases:** Are null inputs, empty collections, and maximum limits handled gracefully?
- [ ] **Validation:** Are all input parameters validated for type, range, and format?
- [ ] **Logic paths:** Are both happy and failure paths covered?
- [ ] **Concurrency:** Does this change introduce race conditions or thread-safety issues?
- [ ] **UX impact:** Are user-facing changes (copy, logging, behavior) clear and correct?
- [ ] **Test alignment:** If logic changed, were the corresponding unit/integration tests updated?

2. Security & vulnerability

Focus on inputs, permissions, and dependencies.

- [ ] **Sanitization:** Is external input sanitized to prevent XSS or SQL injection?
- [ ] **Authorization:** Are permissions checked at the correct layer (not just UI)?
- [ ] **Secrets:** Are keys, tokens, and secrets excluded from source control?
- [ ] **Data safety:** Is sensitive data encrypted at rest/transit and excluded from logs?
- [ ] **Dependencies:** Are third-party libraries up-to-date and vulnerability-free?
- [ ] **Access control:** Does this introduce new access paths that could be abused?
- [ ] **Information leakage:** Do error messages avoid exposing internal system details?

3. Architecture & design

Protect system cohesion and future scalability.

- [ ] **Patterns:** Does the code follow established patterns (e.g., MVC, clean architecture)?
- [ ] **SRP:** Do components/classes stick to a single responsibility?
- [ ] **Configuration:** Are hardcoded values moved to configuration or constants?
- [ ] **DRY:** Is duplicated logic extracted into shared utilities?
- [ ] **Coupling:** Are there tight couplings that make future changes risky?
- [ ] **Complexity:** Is control flow simple (avoiding deep nesting)?
- [ ] **Scalability:** Will this handle increased data volume or user load?

4. Readability, maintainability, & style

Ensure the code is debuggable by future you.

- [ ] **Naming:** Do names clearly communicate intent and follow conventions?
- [ ] **Style:** Does it pass the linter/style guide? Automate this to avoid nitpicks.
- [ ] **Flow:** Can a teammate follow the logic without jumping files constantly?
- [ ] **Function size:** Are complex blocks broken into smaller, testable functions?
- [ ] **Comments:** Do comments explain why something exists, not just what it does?
- [ ] **Cleanliness:** Is dead code, debug output, and commented-out logic removed?

5. Testing & documentation

Proof that it works and instructions for the future.

- [ ] **Coverage:** Do new tests directly cover the changed logic?
- [ ] **Reliability:** Are tests deterministic, independent, and fast?
- [ ] **Performance:** Are loops, queries, and hot paths optimized?
- [ ] **Context:** Does the PR description explain the “Why”?
- [ ] **Docs:** Are READMEs, API docs, and internal guides updated?
- [ ] **Metrics:** Will this affect deployment or DORA metrics?