CODE REVIEW PROCESS PROCEDURE

Document Reference: QMS-DEV-PRO-003

Version: 1.0

Classification: RESTRICTED - INTERNAL ENGINEERING ONLY

Role	Name & Signature	Position	Date
Prepared by			
Reviewed by			
Approved by			

TABLE OF CONTENTS

1. AMENDMENT RECORD

Reviewed at least annually or when tooling, team structure, or compliance requirements change. All revisions go through the same approval gate as the original document.

Date	Issue	Rev	Page	Subject	Revised By	Approved By

2. EXECUTIVE SUMMARY

This procedure defines a structured, repeatable code review process that works across technology stacks and team sizes. It is language-agnostic and scales from a two-person team with no formal tooling to a large engineering organisation with a mature CI/CD platform.

Teams that already have an internal developer guide or engineering handbook should treat this procedure as the formal quality and compliance layer that sits on top of it. Teams that do not yet have such a guide will find everything they need here to get started.

The process covers five core areas:

- Automated checks: linting, SAST, dependency scanning, secret detection, test coverage
- Manual peer review: logic, design, security, documentation, and knowledge transfer
- Feedback resolution: structured comment handling with clear resolution criteria
- Approval and merge: gated approval with a defined rollback posture
- Post-merge loop: deployment verification, metrics capture, and retrospective input

Performance is tracked using metrics that are directly observable in the tools teams already use, not abstract frameworks. Where DORA metrics are relevant they are noted, but every metric in this procedure has a clear owner, a collection method, and a defined action threshold.

3. GENERAL

3.1 Scope

This procedure applies to all code modifications intended for integration into managed repository branches (main, develop, or release). It encompasses:

- New feature development and iterative enhancements
- Bug fixes, patches, and hotfixes (including emergency changes - see Section 4.4)
- Code refactoring and technical debt reduction
- Infrastructure as Code (IaC): Terraform, Pulumi, Ansible, Crossplane, etc.
- CI/CD pipeline definitions and configuration
- Database schema changes and migration scripts
- Security configurations and access control implementations
- Test automation code, fixtures, and test data management
- Third-party integration code and API adapter layers

The procedure applies regardless of team size. Section 4.4 includes a lightweight variant for small teams and a hotfix track for emergency production changes.

3.2 Purpose

The goals of this procedure, in plain terms:

- Ship code that works, is readable, and does not create problems for the next person who touches it
- Catch security issues, logic errors, and performance problems before they hit production
- Spread knowledge across the team so no single person becomes a bottleneck
- Create an auditable trail of who reviewed what, when, and why changes were or were not approved
- Process Efficiency: Replace manual verification steps with automated quality gates to ensure code quality reviews do not introduce delivery latency.

3.3 Guiding Principles

- Review the code, not the person. Feedback is about the change, not the author.
- Small, frequent PRs over large, infrequent ones. Target under 400 lines changed per review.
- Automated Validation: Run all machine-verifiable checks via automated pipeline gates. Human review time must be restricted to architectural logic, design patterns, and security constraints.
- A comment must be actionable. Vague or subjective feedback ('this feels wrong') should include a concrete suggestion.
- Blocking a merge requires a reason. Non-blocking observations are labelled as such (e.g. 'nit:', 'suggestion:').
- Two-pass rule: if a change requires more than two review cycles, the author and reviewer should talk before continuing in writing.

3.4 Responsibility for Implementation

The Engineering Lead (or equivalent senior technical owner) is accountable for this procedure being understood and followed. Day-to-day responsibilities are distributed as follows:

- Engineering Lead: process ownership, reviewer assignment policy, dispute resolution, KPI review
- Quality function (or equivalent): metrics collection, audit coordination, documentation currency
- DevOps / Platform team: CI pipeline configuration, tooling maintenance, automation coverage
- Security team / Security champion: security checklist ownership, SAST/DAST tool configuration, vulnerability triage

3.5 References

- ISO 9001:2015 - Quality Management Systems
- ISO 27001:2022 - Information Security Management
- OWASP Secure Coding Practices Quick Reference Guide
- OWASP Code Review Guide v2
- NIST SP 800-218 (SSDF) - Secure Software Development Framework
- Google Engineering Practices - Code Review documentation (publicly available)
- Accelerate (Forsgren, Humble, Kim) - DORA research on code review and deployment frequency
- Internal: Change Management Procedure
- Internal: SDLC Procedure
- Internal: Incident Response Procedure
- Internal: Developer Guide / Engineering Handbook (if maintained - referenced as QMS-DEV-GUIDE-001 where it exists)

3.6 Roles and Responsibilities

Role	Primary Responsibilities	Backup	Notes
PR Author (Developer)	Write to agreed standards; self-review before opening PR; respond to feedback within 1 business day; keep PRs small	Any team member	Authors should not approve their own PRs
Code Reviewer	Complete review within SLA; distinguish blocking vs non-blocking comments; verify test coverage and security considerations	Senior engineers rotate as backup	Aim for 2 reviewers on critical paths; 1 is acceptable for low-risk changes
Engineering Lead / Tech Lead	Set reviewer assignment policy; resolve disputes; monitor review SLA	Quality Manager	Reviews DORA metrics and review KPIs in quarterly eng review

	compliance; own process improvement		
DevOps / Platform Engineer	Maintain CI pipeline; ensure automated checks run on every PR; alert on tooling failures; manage artifact signing	Cross-trained team member	Pipeline health is a KPI, failures block merges
Security Engineer / Champion	Own security checklist; triage SAST/DAST findings; define severity thresholds; approve security-sensitive changes	Security team rotation	Every team should have at least one trained security champion
Quality Manager (where role exists)	Collect and report KPIs; coordinate audits; maintain procedure currency	Engineering Lead	In smaller teams this role is often absorbed by the Engineering Lead

4. PROCESS DETAILS

4.1 Objectives

- Every production-bound change has been reviewed by at least one peer who did not write it
- Automated checks run on every PR, no human reviewer sees code that a machine can validate
- Security-sensitive changes get a security specialist's eyes before merge
- Review cycles stay short, under 24 hours median, without sacrificing thoroughness
- The process generates enough data to measure itself and improve over time

4.2 Key Performance Indicators

These metrics are grounded in what PR platforms, CI tools, and issue trackers already emit. Each one answers a specific question about review health. DORA Lead Time for Changes is directly influenced by PR Lead Time and CI Pass Rate; the relationship is noted where relevant.

KPI	What it measures	Target	How collected	Why it matters
PR Size (lines changed)	Mean lines added + deleted per PR	< 400 lines median	Git / PR platform stats	Large PRs are reviewed less thoroughly. Small PRs ship faster and catch more bugs per line reviewed.
Time to First Review	Minutes from PR opened to first	< 4 hours (P0/P1	PR platform timestamps	Long wait times stall flow. Tracks reviewer

	substantive reviewer comment	changes: < 1 hour)		responsiveness, not just approval time.
Review Cycle Count	Number of request-changes / re-review loops per PR	≤ 2 cycles on 90% of PRs	PR platform review history	> 2 cycles usually means unclear requirements, missing acceptance criteria, or poor self-review.
PR Lead Time	PR opened to merged (clock time, not business hours)	< 24 hours median; < 4 hours for hotfixes	Git merge timestamp minus PR open timestamp	Core DORA input. High lead time is often a sign of oversized PRs or reviewer bottlenecks.
CI Pipeline Pass Rate	% of PR builds that pass all automated checks on first run	≥ 95%	CI platform build history	Frequent red builds waste reviewer time and erode trust in automation.
First-Pass Approval Rate	% of PRs approved without a request-for-changes round	≥ 80%	PR platform review outcomes	Tracks upstream quality: self-review discipline, PR template adherence, test coverage before opening.
Code Review Coverage	% of merged commits that went through at least one peer review	100% of production-bound changes	Branch protection rules + audit log	Non-reviewed code in production is an audit failure and a security risk.
Test Coverage Delta	Change in test coverage % introduced by the PR (not absolute total)	No net decrease; green-field code ≥ 80% covered	Coverage tool (Jest, PyTest-cov, JaCoCo, etc.) reported in CI	Absolute coverage % is easy to game. Delta coverage tracks whether new code ships with tests.
SAST Finding Rate	New Critical or High severity findings introduced per 100 PRs	Zero Critical; < 2 High per 100 PRs	SonarQube / Semgrep / CodeQL CI output	Tracks whether security standards are being maintained, not just whether scanning is running.
Defect Escape Rate	Production bugs traced back to a specific merged PR	< 1% of PRs produce a production defect	Incident / bug tracker linked to commit history	The ultimate measure of review effectiveness. Slow to collect but the most meaningful.
Reviewer Load Distribution	Gini coefficient or std dev of reviews per person over a sprint	< 0.4 Gini (low concentration)	PR platform reviewer assignment data	Uneven distribution creates knowledge silos and burnout. Not about fairness, about risk.

Comment Resolution Time	Median time from reviewer comment posted to author response	< 8 business hours	PR platform comment timestamps	Stale comment threads are a leading indicator of process breakdown.
Security-Sensitive PR Review Rate	% of PRs touching auth, crypto, secrets, or data handling that received a security-specialist review	100%	Label / tag on PR + reviewer role	Compliance requirement. Cannot be approximated by SAST alone.

4.3 Resources

Select tools appropriate to your stack. The categories below are what matters, specific tool choices within each are a team decision.

Category	Reference Tools
Version Control & PR Platform	GitHub, GitLab, Bitbucket, Azure DevOps - branch protection rules and required reviews configured
Linting & Style	ESLint (JS/TS), Pylint / Ruff (Python), RuboCop (Ruby), Checkstyle / SpotBugs (Java), gofmt / golangci-lint (Go), ktlint (Kotlin)
SAST	SonarQube / SonarCloud, Semgrep, CodeQL (GitHub), Checkmarx - run in CI on every PR
Dependency & SCA Scanning	Snyk, Trivy, Grype, Dependabot, OWASP Dependency-Check
Secret Scanning	GitLeaks, TruffleHog, GitHub Secret Scanning, GitGuardian
IaC Policy / Scanning	Checkov, tfsec, Terrascan, OPA / Conftest, kube-score
Test Coverage	Jest (coverage), PyTest-cov, JaCoCo, SimpleCov, go test -cover, reported as delta per PR
PR Analytics	Built-in platform metrics (GitHub Insights, GitLab Analytics), LinearB, Swarmia, Waydev - for KPI collection
Communication	PR comments (primary); Slack / Teams for escalation; never email for code review feedback

4.4 Process Steps

The standard review flow covers seven steps. Teams should apply the full flow for all production-bound changes. See the lightweight variant note after Step 7 for small teams and hotfix handling.

1. Self-Review & PR Preparation

Owner	Author
Activities	• Run the full local test suite. Do not open a PR against a failing build. • Run SAST and linter locally if available (pre-commit hooks recommended). • Write a PR description that covers: what changed, why, how to test it, and any known limitations or follow-up items. • Check your own diff line by line before opening. If you find things to fix, fix them first. • Apply appropriate labels: feature / bug / hotfix / IaC / security-sensitive / breaking-change. • Keep PRs under 400 lines where possible. If a change is necessarily large, add a walkthrough comment at the top of the diff. • If your org has a PR template - fill it in completely. If it does not, create one (see Appendix A). • Link to the relevant ticket, issue, or ADR. Reviewers should not need to ask for context.
Gate	PR template complete; local tests passing; SAST clean locally; PR size documented if > 400 lines

2. Automated Checks (CI Pipeline)

Owner	CI System (automated)
Activities	• Code style and linting - language-specific (ESLint, Pylint, RuboCop, gofmt, etc.). Fail fast. • Compilation / build verification. • Unit test suite - failure blocks merge. • Integration test suite (where applicable to the change). • Test coverage delta check - fail if new code brings coverage below threshold. • SAST scan, SonarQube, Semgrep, CodeQL, or equivalent. Critical findings block merge; High findings trigger security reviewer assignment. • Dependency vulnerability scan - Trivy, Snyk, Grype, or Dependabot alerts. Critical CVEs block merge. • Secret scanning - GitLeaks, TruffleHog, or platform-native. Any finding blocks merge immediately. • IaC policy check - Checkov, tfsec, or OPA/Conftest for Terraform/Ansible/Helm (where applicable). • PR size warning - automated comment if diff exceeds 400 lines (does not block, but triggers reviewer awareness).
Gate	All automated checks green. Any Critical/High SAST finding or secret detected = hard block.

3. Peer Review - Technical

Owner	Code Reviewer (≥ 1, ≥ 2 for critical paths)
Activities	• Understand the intent of the change before reviewing the implementation. • Logic and correctness: does the code do what the PR description says it does? • Edge cases: error handling, null/empty inputs, concurrency, race conditions. • Design: does this fit the existing architecture? Does it introduce unnecessary coupling or duplication? • Performance: any obvious hot-path issues, N+1 queries, unbounded loops, or blocking I/O? • Testability: is the code structured in a way that makes it testable? Are the tests actually testing behaviour? • Readability: would a new team member understand this in six months without the PR context? • Dependency hygiene: new dependencies should be justified. Prefer standard library where possible. • Label comments clearly - 'blocking:', 'suggestion:', 'nit:', 'question:'. Anything without a label is assumed blocking. • Do not approve with unresolved blocking comments. Do not leave a PR open for > 24 hours without a response.
Gate	No unresolved blocking comments. Reviewer approval recorded in PR platform.

4. Peer Review - Security (conditional)

Owner	Security Engineer or trained Security Champion
Activities	• Required for any PR labelled security-sensitive, or for changes to: authentication/authorisation logic, cryptography, secrets handling, data storage/transmission, access control rules, public API endpoints, IaC with IAM or network policy changes. • OWASP Top 10 check relevant to the change type (injection, broken auth, IDOR, SSRF, etc.). • Input validation and output encoding. • Secrets management: are credentials injected via environment or secrets manager - never hardcoded? • Principle of least privilege: do new roles, permissions, or service accounts have only what they need? • Data classification: does the change handle data at a classification level the author is aware of? • Third-party / OSS component review: licence compatibility, known CVEs, supply chain risk. • For regulated environments (PCI-DSS, HIPAA, SOC 2): explicit compliance check against relevant controls.
Gate	Security reviewer approval recorded. No open security findings above agreed severity threshold.

5. Feedback Resolution

Owner	Author, with Reviewer confirmation
Activities	• Address blocking comments before making any other changes to the branch. • For each blocking comment: either implement the suggestion, or explain why you are not and reach agreement with the reviewer. • Do not mark comments as resolved unilaterally on blocking items, the reviewer should resolve their own blocking comments. • If a comment reveals a systemic issue beyond this PR (e.g. a pattern repeated elsewhere in the codebase), create a follow-up ticket. Do not let it block the current PR unless it is a security or correctness issue. • After addressing feedback, push new commits (do not force-push during review unless agreed with reviewer, it destroys comment threading). • If the change requires a third review cycle, the author and reviewer should move to synchronous discussion (call or Slack huddle) before continuing async. • Non-blocking comments (suggestions, nits) can be implemented at the author's discretion and do not need reviewer re-approval.
Gate	All blocking comments resolved and confirmed by reviewer. Re-run CI if new commits were pushed.

6. Final Approval & Merge

Owner	Engineering Lead or designated approver for the branch
Activities	• Verify all required approvals are present (minimum 1 for standard changes; 2 for security-sensitive or critical path changes). • Confirm CI is green on the latest commit. • Main Branch Merges: Enforce squash-and-merge or linear fast-forward merge strategies based on repository configuration to maintain clean commit history. • Write a clear merge commit message: what was merged and why (the PR title is usually sufficient if the PR description is complete). • Delete the source branch after merge (automated where possible). • Update the linked ticket/issue to 'merged' or 'in staging'. • For changes touching infrastructure or deployment config: notify the on-call or release coordinator.
Gate	Required approvals present; CI green on HEAD; no unresolved blocking comments; change ticket updated.

7. Post-Merge Verification

Owner	Author + DevOps Engineer
Activities	• Verify the change deploys cleanly through the pipeline (dev → staging → production per deployment procedure). • Run smoke tests or synthetic checks in the target environment.

	• Monitor error rates, latency, and relevant SLOs for a defined stabilisation window after production deployment (minimum 30 minutes for low-risk; 2 hours for high-impact changes). • If a regression is detected: follow the rollback plan defined in the deployment runbook. Do not attempt a forward fix without a post-incident discussion. • Capture lessons learned: if the review process failed to catch a production defect, create a retrospective item. • Update documentation: API docs, runbooks, ADRs, or the developer guide if the change introduced a new pattern.
Gate	Smoke tests pass in production; error rate within SLO; no regression detected in stabilisation window.

Lightweight Variant - Small Teams and Hotfixes

For teams of 1-3 engineers or for emergency hotfixes to production, the following compressed flow is acceptable:

- Steps 1–2 (self-review and automated checks) are never skipped, even for hotfixes.
- Peer review (Steps 3-4) can be compressed to a single synchronous review session for hotfixes, with the review documented as a comment on the PR immediately after.
- The merge can be completed by the reviewer rather than a separate approver if team size does not permit separation.
- Post-merge verification (Step 7) is still required. For hotfixes, the stabilisation window starts immediately and should be actively monitored.
- A follow-up full review of the hotfix code should occur within 2 business days in a normal sprint cycle.

Small teams: if you have only one engineer, the minimum viable review is a self-review checklist, CI automation, and a time-boxed review of your own diff 24 hours later with fresh eyes. This is not ideal, it is the floor.

4.5 Process Map

The end-to-end state diagram is provided as a separate artifact in the link below. It shows all states, transitions, automated gates, and feedback loops including the re-review cycle and the hotfix bypass path.

Code Review Process Map

4.6 Retained Documentation

Document Type	Purpose	Retention	Location
Pull Request Record	Complete PR history including description, comments, review decisions, and approvals	Lifetime of codebase	Version Control System
Automated Check Logs	CI build logs, SAST reports, coverage reports, dependency scan outputs	2 years	CI/CD Platform
Security Review Records	Security assessment notes, findings, and sign-offs for security-sensitive PRs	3 years	Secure Document Repository
Code Review Metrics Reports	Weekly/monthly KPI summaries	2 years	Analytics / Reporting Platform
Defect Escape Reports	Production bugs linked to specific PRs; root cause notes	3 years	Incident Management System
PR Template and Checklists	Current version of all review templates and checklists	Current version always; history 2 years	Version Control / Wiki
Training Records	Evidence of code review training and security champion certification	3 years or employment duration	HR / Learning Management System
Process Improvement Log	Retrospective action items, process change decisions	2 years	Team Wiki / Project Tracker

5. RISKS AND OPPORTUNITIES

5.1 Risk Register

Risks are assessed and reviewed quarterly. Mitigations are owned by the Engineering Lead unless otherwise assigned. Risk status is a standing item in the quarterly engineering review.

Category	Risk	Mitigation
Process	Review becomes a rubber stamp - approvals given without genuine scrutiny	Enforce PR size limits; track first-pass approval rate (high rate + high defect escape = rubber stamping); rotate reviewers; spot-check merged PRs in retrospectives
Process	Reviewer bottleneck - one or two people reviewing everything	Track reviewer load distribution (Gini coefficient); define maximum review load per sprint; build review capacity into sprint planning

Process	Review SLA breaches stalling development flow	Automate SLA reminders; escalate to lead after 4-hour breach on P1 changes; treat review availability as a capacity planning concern
Process	Knowledge silos - review concentrated on code authors know well	Deliberately assign reviewers outside their comfort area occasionally; pair review for onboarding; document decisions in ADRs not just PR comments
Technical	Large PRs that are hard to review thoroughly	Enforce pull request size thresholds via automated pipeline warnings; require smaller task allocation during work breakdown; utilize short branch lifecycles to limit code volume per review
Technical	Flaky CI pipeline eroding trust in automated checks	Track pipeline pass rate as a KPI; quarantine flaky tests immediately; treat pipeline health as a production issue
Technical	SAST false positive fatigue leading to ignored findings	Tune SAST rules quarterly; track suppression rate; require justification comment for any finding marked 'wont-fix'; escalate persistent tuning failures
Security	Security-sensitive changes merged without specialist review	Automated labelling for sensitive change types; branch protection rules requiring security reviewer for labelled PRs; audit security review coverage monthly
Security	Secrets committed to version control	Pre-commit hooks and CI secret scanning as hard blocks; periodic repo-wide secret scan; rotate any secret that reaches a commit, even if the commit is reverted
Security	Dependency vulnerabilities introduced via new packages	Automated SCA on every PR; policy against adding dependencies without a documented justification; regular dependency update cycles
People	Author-reviewer friction reducing psychological safety	Establish and reinforce commenting norms (see Section 3.3); managers address repeated hostile review behaviour; blameless approach to defect escapes
Compliance	Incomplete audit trail for regulated environments	Branch protection enforcement; automated check logs retained per Section 4.6; review coverage tracked and reported monthly

5.2 Opportunities

- PR analytics tooling (LinearB, Swarmia): surface DORA metrics and review KPIs automatically without manual data collection. High ROI for teams currently tracking these in spreadsheets.
- AI-assisted review tools (GitHub Copilot PR review, CodeRabbit, Sourcery): can handle first-pass logic checks and flag common patterns, freeing human reviewers for higher-order concerns. Though this requires a clear policy on what AI review can and cannot replace.
- Review culture workshops: one-off sessions on how to give and receive code review feedback have an outsized effect on team dynamics and review quality. Cheap to run, high return.
- PR template iteration: template quality directly affects first-pass approval rate. Teams that review and improve their template quarterly consistently outperform those that set it once.

- Security champion programme: embedding security knowledge into every team, rather than routing all security review through a central team, reduces security review bottlenecks and improves coverage.
- Automated ticket linkage and deployment tracking: linking PR merge events to deployment records and production metrics closes the feedback loop between review quality and production outcomes.

6. CONTINUOUS IMPROVEMENT

6.1 Improvement Cadences

Cadence	What is reviewed	Output
Sprint Retrospective (every 1–2 weeks)	Review friction points, tooling issues, commenting norm violations, SLA breaches	Backlog items; process tweak decisions; immediate tooling fixes
Monthly KPI Review	All metrics in Section 4.2; trend analysis; any metric breaching threshold triggers an action item	KPI report; named action items with owners and due dates
Quarterly Engineering Review	PR analytics trends; reviewer load; defect escape analysis; risk register review; process improvement backlog	Updated risk register; process change decisions; tooling evaluation outcomes
Annual Procedure Review	Full document review; compliance alignment; team structure changes; tooling landscape changes	Updated procedure version; amendment record entry; re-approval

6.2 Defect Escape Review

Any production defect traced back to a merged PR that passed all review gates triggers a lightweight retrospective:

- Was the defect detectable at review time? If yes, why was it missed?
- Was it a logic error that a human reviewer should have caught, or an edge case that required additional test coverage?
- Did the automated checks miss it? If so, can a new rule or test be added to catch it in future?
- Was the PR too large to review effectively?

The output is a single action item (or a decision that no process change is warranted, documented with reasoning). These reviews should take under 30 minutes. They are not blame sessions.

6.3 PR Template and Checklist Maintenance

The PR template and review checklists are living documents. They should be updated whenever:

- A defect escape reveals a gap in the checklist
- A new technology or pattern is adopted that the existing checklist does not cover

- A checklist item consistently goes unticked without consequence, which means either the step is not useful or the process is not being followed

Current templates and checklists are maintained in version control alongside the codebase (or in the team wiki). They are referenced in the PR platform as the default PR template.

6.4 Appendix A - Minimum PR Template

If your organisation does not have a PR template, the following is a starting point. Adapt it to your context.

```
<!-- One paragraph. What does this PR do? -->

## Why
<!-- Link to ticket / issue / ADR. What problem does this solve? -->

## How to test
<!-- Steps to verify the change works as expected. -->

## Checklist (author)
- [ ] I have run the full test suite locally
- [ ] I have run the linter / formatter
- [ ] I have reviewed my own diff
- [ ] Documentation updated (if applicable)
- [ ] No secrets or credentials in this diff
- [ ] PR size is under 400 lines, or I have added a walkthrough comment

## Labels applied
<!-- feature / bug / hotfix / IaC / security-sensitive / breaking-change -->

## Known limitations / follow-up
<!-- Optional. Link to follow-up tickets if this PR is intentionally incomplete. -->
```

6.5 Document Control

This procedure is proprietary internal property. It shall be reviewed and updated systematically upon significant changes to the development toolchain, engineering team restructuring, or during the annual scheduled review cycle.

The procedure is reviewed whenever: team structure changes significantly, a defect escape reveals a process gap, tooling changes affect a step, or the annual review cycle triggers a scheduled review.