AI-Enhanced Pull Request Handbook

Table of Contents

1. Executive Summary

- **Why now:** AI accelerates code creation, which amplifies inconsistency and review fatigue if we don't raise the floor on PR quality.
- **What this handbook provides:** A consistent, evidence-based PR flow using **rules + template + AI gate checks + CI enforcement**, with **human review** focused on design, correctness, and risk.
- **Scope:** PR automation only (not all dev practices). Sub-rules (architecture, security, testing, NASA safety rules, etc.) live in their own docs and can be applied at implementation time **and** as PR gates.
- **Delivery models:** Works for **trunk/topic** and **batch/release trains** (adds a release-level checklist).

Outcome: Faster, clearer, auditable PRs; less nitpicking; higher confidence in production changes.

2. Introduction

AI can help us write code faster, but currently, and in the near future, it does not carry responsibility for what runs in production. That accountability remains with the developer and the team.

When throughput increases, small gaps in our review process become significant risks: inconsistent PRs, checklist fatigue, missing links to tests/scans/observability, and late integration surprises, especially when features are shipped in batches. AI can unintentionally exacerbate this by enabling more partially validated changes to reach review more quickly.

This handbook proposes **suggested practices** for PRs that raise the floor without slowing delivery:

- **Rules serve as guardrails:** Core rules apply everywhere, while supporting and domain-specific rules (e.g., architecture, observability, NASA's "Power of 10") layer on when relevant. They guide **implementation** and act as **PR gates**.
- **Evidence over assertion:** A **PR template** that requires links to tests, coverage, security scans, observability, and rollback plans.
- **AI + CI assist, humans decide: Copilot/Claude** surface gaps and propose fixes; **CI** enforces Core sections; **human reviewers** focus on design quality, business alignment, and risk.

- **Delivery-model aware:** Works for **trunk/topic** teams and **batch release** teams (with a release manifest, integrated validation, coordinated migrations, and staged rollout).

The goal is to standardize **within reason** and establish clear, **paved paths** that teams can adopt, ensuring quality, security, performance, and resilience are consistent across the organization.

3. Why PR Automation Matters

When throughput accelerates, weaknesses in the review process surface quickly. Consider some of the pain points many teams have seen:

- **Inconsistent PRs.** One developer writes detailed descriptions with links to test runs and security scans, while another writes "fixed a bug" without providing any evidence. Review quality suffers from this variability.
- **Checklist fatigue.** Reviewers spend cycles asking the same questions: "Did you add tests?" "Where's the migration plan?" "Is there an observability change?" This slows feedback loops and frustrates both sides.
- **Missing context.** Without explicit links to CI runs, coverage reports, or observability dashboards, reviewers are forced to trust or manually hunt for evidence. This creates blind spots.
- **Integration risk.** When multiple features are shipped together in batch releases, problems often emerge in staging or production, typically because no structured release-level validation existed.

Without intervention, these issues compound in an AI-enabled environment. As AI makes it easier to produce code, it also makes it easier to **flood the system with half-finished or insufficiently validated changes**. The result: higher risk of defects, regressions, and production incidents.

The goal of PR automation is not to slow developers down with bureaucracy, but to:

- **Standardize expectations.** Every PR should demonstrate a minimum baseline of quality.
- **Automate validation.** Machines (AI + CI) should check mechanics, freeing humans for meaningful review.
- **Require evidence.** Links to tests, scans, dashboards, not just verbal assurances.
- **Scale with delivery models.** Whether a team ships continuously from trunk or in coordinated release trains, the system should adapt.

- **Create auditability.** Every PR should leave a traceable record of what was validated, by whom, and with what evidence.

In short, automation raises the floor, not the ceiling. By making PR quality systematic, teams protect developers' time, reduce review fatigue, and lower the risk of production issues, without slowing down the velocity that AI has enabled.

4. Rules Files (Core, Supporting, Domain-Specific)

- `.ai/core-rules.md` baseline
- Supporting rules (`architecture-rules.md`, `observability-rules.md`, etc.)
- Domain-specific rules (e.g. NASA Power of 10)

Rules Reuse: Implementation vs. PR Gate

- **Core rules** are universal.
- **Supporting and domain rules** are overlays:
  - During **implementation**, they guide developers (e.g., observability patterns, NASA's safety rules).
  - During **PR reviews**, they become extra gates when relevant.

- The system is extensible: AI assistants and CI can load extra rule files automatically if present.

5. Delivery Models (Trunk vs. Batch)

Most teams practice **topic- or trunk-based delivery**, which involves small, frequent merges from short-lived branches into the trunk. Other teams, however, still work in a **batch release model**, where multiple features are grouped and deployed together as a "train."

The **Core and Supporting Rules** in `.ai/core-rules.md` apply in both models.

Trunk-Based Delivery (Topic Merges)

- Each PR represents a single feature or slice of work.
- All Core and Conditional rules must be satisfied at the PR level.
- AI + CI checks ensure readiness before merge.

Batch Release Delivery (Trains)

- Multiple PRs/topics bundled into a release branch.
- Each PR meets Core/Conditional rules.
- Additional release-level checklist before deployment:
  - Release manifest with owners, risks, and rollback steps.

- Integrated regression + performance tests.
- Migration coordination across features.
- Staggered rollout/canary with auto-abort rules.
- Release rollback plan documented.

6. Core vs. Conditional Rules

- **Core (always required):**
  - Outcome
  - Risk & Rollback
  - AI Assistance disclosure
  - Unit + Integration Tests
  - Coverage
  - Secrets Scan
  - SAST (static analysis)
  - SCA/License checks
  - Compatibility (backward/forward)
  - Observability (metrics/traces/logs)
  - Provenance & SBOM
  - Approvals
- **Conditional (when relevant):**
  - Mutation Testing (critical paths)
  - AuthN/AuthZ tests
  - Performance benchmarks
  - DB/Cache review
  - Migration plan
  - Dashboards/Alerts
  - Feature Flags
  - DAST/Smoke tests

7. Pull Request Template

```
### Outcome (Core)
What problem and outcome does this change deliver? Link to
ticket/issue.

### Risk & Rollback (Core)
- Risk level: Low | Medium | High
- Rollback strategy: revert/flag/config flip

### AI Assistance (Core)
- Was AI (e.g., Copilot, Claude) used? Where/how?
```

```
- Any external code copied in? License cleared?
- Insecure patterns reviewed: ☑/❌

### Tests
- Unit tests (Core): ☑ [link]
- Integration/Contract tests (Core): ☑ [link]
- Changed-lines coverage (Core): __% (bar = __%) [link]
- Mutation testing on critical paths (Conditional): score __ (target ≥
60%) [link/waiver]

### Security
- Secrets scan (Core): ☑ [link]
- SAST (Core): 0 High/Critical ☑ [link]
- SCA/License (Core): 0 High/Critical ☑ [link]
- AuthN/AuthZ tests (Conditional): ☑ [link]

### Performance
- Hot-path micro-bench/regression (Conditional): [link]
- DB/cache queries reviewed (Conditional): ☑/❌ [comment or link]

### Compatibility & Data
- Backward/forward compatibility (Core): ☑
- Migration plan (Conditional): [migrations/PLAN.md + job link]

### Observability
- New metrics/traces/logs (Core): [code/dashboards links]
- Dashboards/alerts updated (Conditional): ☑ [Grafana/Prom link]

### Feature Flags (Conditional)
- Flag/key + default state [link]
- Kill switch described

### Provenance/SBOM (Core)
- Build artifact signed + SBOM attached [link]

### DAST/Smoke (Conditional)
- External surface tests passed [link]

### Approvals (Core)
- Two reviewers
- High-risk changes reviewed by security/arch
```

8. GitHub Action: PR Rules Check

```
name: PR Rules Check
on:
  pull_request:
```

```
    types: [opened, edited, synchronize]

jobs:
  validate-pr-template:
    runs-on: ubuntu-latest
    permissions:
      pull-requests: write
    steps:
      - name: Check PR body against Core Rules
        uses: actions/github-script@v7
        with:
          script: |
            const prBody = context.payload.pull_request.body || "";
            const requiredSections = [
              "### Outcome (Core)",
              "### Risk & Rollback (Core)",
              "### AI Assistance (Core)",
              "Unit tests (Core)",
              "Integration/Contract tests (Core)",
              "Changed-lines coverage (Core)",
              "Secrets scan (Core)",
              "SAST (Core)",
              "SCA/License (Core)",
              "Backward/forward compatibility (Core)",
              "New metrics/traces/logs (Core)",
              "Provenance/SBOM (Core)",
              "### Approvals (Core)"
            ];
            let missing = requiredSections.filter(s =>
!prBody.includes(s));
            if (missing.length) {
              core.setFailed(`Missing sections: ${missing.join(",
")}`);
            }
```

9. Developer Quick Start (1-Pager)

1. Open PR with the template.
2. Fill Core sections with evidence links.
3. Mark Conditional items N/A with rationale if not relevant.
4. Run AI Gate Check (Copilot or Claude /check-pr).
5. Fix gaps, push updates.
6. CI enforces Core rules.
7. **Human reviewers evaluate design, business alignment, risk, and maintainability.**
8. Merge when all gates are satisfied.

10. Release Rules (`.ai/release-rules.md`)

```
# Release Rules (Batch Delivery / Trains)

These rules apply in addition to `.ai/core-rules.md`.

## Release Manifest
- Must exist (`release-manifest.md`) listing PRs, owners, risk,
rollback steps.

## Integrated Testing
- Regression + performance suite must pass at release level.

## Migration Coordination
- Cross-feature migrations coordinated and reversible.

## Staged Rollout
- Canary rollout with gates and auto-abort rules.

## Release Rollback Plan
- Documented rollback for entire release.

## Observability
- Dashboards/alerts validated for combined release.

## Release Notes
- Summarize features, risks, rollback, owner, on-call.
```

11. Release Manifest Template (`release-manifest.md`)

```
# Release Manifest

## Release Overview
- Release ID/Name:
- Date:
- Release Owner(s):
- On-Call Contact(s):

## Scope of Included Work
| PR/Topic ID | Title | Owner | Risk | Rollback Step/Flag |
|-------------|-------|-------|------|--------------------|
| #1234 | Feature A API | @alice | Medium | Flag `featureA` |

## Integrated Testing
- Regression results: [link]
- Perf test results: [link]
- Smoke/DAST results: [link]
```

```
## Migration Coordination
- Expand/Backfill/Contract checkboxes
- Backfill monitoring link
- Rollback/Forward-fix strategy

## Staged Rollout Plan
- Initial canary %
- Rollout gates (error, latency, saturation)
- Auto-abort thresholds

## Release Rollback Plan
- Mechanism:
- Responsible team(s):
- Estimated time:

## Observability & Monitoring
- Dashboards updated [link]
- Alerts validated [link]
- Post-release smoke tests [link]

## Release Notes
- Summary of features delivered
- Known risks & mitigations
- Owner(s) & on-call contacts
```

12. Claude Integration (`CLAUDE.md` + Commands)

Example `CLAUDE.md`:

```
# Project Assistant Policy (Claude)

Always enforce:
- Core rules: `.ai/core-rules.md`
- Supporting: `.ai/architecture-rules.md`, `.ai/observability-
rules.md`, `.ai/feature-flag-rules.md`

Behaviors:
- Run /check-pr using these rules.
- Prefer proposing diffs over prose.
- Link evidence (CI runs, coverage, dashboards).
- Mark Conditional rules as N/A with rationale if not relevant.
```

Example `.claude/commands/check-pr.md`:

```
Use `.ai/core-rules.md` + supporting rules to review the PR.
- Pass/fail each Core rule with evidence links.
```

```
- Suggest diffs/tests for missing items.
- Mark Conditional rules as N/A with rationale.
- Summarize readiness to merge.
```

13. Evidence Links Reference Table

Rule	Example Evidence
Unit / Integration Tests	GitHub Actions run URL
Coverage	Codecov/Coveralls report
Mutation Testing	CI artifact report
Secrets Scan	Gitleaks output
SAST	CodeQL/Semgrep report
SCA/License	Dependabot/Snyk summary
AuthN/AuthZ Tests	Test file permalink or CI run
Performance	Benchmark job output
DB/Cache Review	PR comment with query plan
Migration Plan	migrations/PLAN.md
Observability	Grafana dashboard link, trace span
Feature Flags	LaunchDarkly/Unleash link or equivalent
Provenance/SBOM	SBOM artifact, attestation
DAST/Smoke	OWASP ZAP run output

14. PR Workflow Diagram

```
Developer (open PR w/ template)
   ↓
AI Gate Check (Copilot / Claude)
   ↓
Fix gaps (add tests, evidence, observability)
   ↓
CI/CD Check (GitHub Action)
   ↓
Human Reviewer (focus on design, business alignment, risk,
maintainability)
   ↓
Merge to Trunk (production-ready)
```

15. Day in the Life of a PR (Example Walkthrough)

1. You finish coding your feature branch

   - Added a new API endpoint /v1/payments/refund.
   - Wrote unit tests for the happy path, but not yet full coverage.
   - The CI pipeline has already run unit tests and integration tests on your branch.

2. You open a Pull Request in GitHub

```
The PR template auto-fills:
### Outcome (Core)
What problem and outcome does this change deliver? Link to
ticket/issue.

### Risk & Rollback (Core)
- Risk level: Low | Medium | High
- Rollback strategy: revert/flag/config flip

### AI Assistance (Core)
- Was AI (e.g., Copilot, Claude) used? Where/how?
- Any external code copied in? License cleared?
- Insecure patterns reviewed: ☑/❌
```

You fill in:

```
### Outcome (Core)
Adds a new refund API endpoint to allow customers to request refunds.
Ticket: [JIRA-1234](https://jira/ticket/1234)
```

```
### Risk & Rollback (Core)
- Risk level: Medium
- Rollback strategy: Disable via feature flag `refunds.api.enabled` or
revert commit.

### AI Assistance (Core)
- Claude Code suggested test case scaffolding.
- Copilot generated initial data validation logic.
- Insecure patterns reviewed: ☑
```

3. You run an AI Gate Check
    a. With GitHub Copilot
        - In VS Code Copilot Chat, you type:
        - "Check my PR against .ai/core-rules.md."
        - Copilot replies:
            - ☑ Outcome provided
            - ⚠ No coverage % listed
            - ⚠ No security scan link provided
    b. With Claude Code
        - In the terminal or Claude interface, you run:

```
/check-pr
```

Claude replies with a checklist:

- Tests: unit test link missing
- Security: SAST scan results not linked
- Observability: no metrics/traces for refund API
- Suggests: "Add request count metric to API handler.

4. You fix gaps before review

    You add:

```
### Tests
- Unit tests: ☑ [link to GitHub Actions
run](https://github.com/org/repo/actions/runs/1111)
- Integration tests: ☑ [link to Pact test run](https://ci/job/222)
- Changed-lines coverage: 87% (bar = 80%) [Codecov link]
- Mutation testing on critical paths: score 65% [Mutmut report]

### Security
- Secrets scan: ☑ [Gitleaks run]
- SAST: 0 High/Critical ☑ [CodeQL scan]
- SCA/License: 0 High/Critical ☑ [Dependabot report]

### Observability
```

```
- New metrics: `refunds_api_requests_total` (Prometheus counter)
- Trace span: `RefundAPI.RefundRequest`
- Dashboards updated: [Grafana dashboard link]
```

5. **You push your update**
   - GitHub Action (PR Rules Check) runs.
   - It parses your PR body.
   - ☑ All Core sections found.
   - ⚠ Placeholder [link] replaced with real URLs.
   - CI posts a comment:

```
## 🔍 PR Rules Check Report
☑ Outcome
☑ Risk & Rollback
☑ AI Assistance
☑ Tests
☑ Security
☑ Observability
☑ Approvals
```

6. **Reviewer opens your PR**

   Human Reviewer(s)
   - Confirm the *design and architecture* make sense.
   - Check for *business alignment*: does the feature solve the right problem?
   - Assess *risk and tradeoffs* that automation can't evaluate.
   - Give feedback on *readability, maintainability, clarity*.

   Instead of asking, *"Did you test this?"*, they see:
   - Links to unit + integration tests.
   - Codecov showing coverage %.
   - CodeQL scan results.
   - Observability metrics + Grafana dashboard.

   They can now focus on:
   - Is the refund API design correct?
   - Is the error handling robust?
   - Is the business logic aligned with product requirements?

7. **Merge to trunk**

   Because AI + CI caught gaps early, review is fast.

   - Reviewer approves.
   - PR merges.
   - Feature flag keeps rollout safe.

- You sleep well knowing you've provided evidence for quality, security, and observability.

Takeaway

As a developer, your job is easier:

- Follow the **PR template**.
- Run **AI Gate Checks**.
- Provide **evidence links**.
- Let **CI enforce Core rules**.

- Reviewers spend less time nitpicking, more time on real design

16. Where Human Review Fits

- AI + CI: enforce mechanics (tests, coverage, scans, observability)
- Humans: evaluate design, correctness, business fit, maintainability
- Explicit callout that both are required

17. (Optional) Release Manifest GitHub Action

Full preserved and expanded content for 18. (Optional) Release Manifest GitHub Action goes here.