Appendix: Verification Triangle Metrics

This appendix defines the metrics for each vertex of the Verification Triangle.

The goal here is narrow:

- what each metric is
- what it measures
- how to measure it
- what research or public practice it is grounded in

This appendix intentionally does **not** cover rollout sequence, meeting cadence, ownership, or operating rhythm.

ONE SHARED DEFINITION

Accepted change means a change that survived review, passed evaluation, reached production, and stayed there without rollback or incident inside the team's defined observation window.

Several metrics below use accepted change as the denominator because merged PR overstates successful delivery.

Size normalization: each PR counts as `max(1, ceil(lines_changed / 500))` normalized changes. A 50-line fix is 1 change. A 1,500-line PR is 3 changes. The 500-line unit sits just above the 400-line threshold where review effectiveness collapses (SmartBear/Cisco), so a 405-line PR is not penalized as two changes. Without normalization, teams that merge large PRs appear more cost-efficient per change than teams that split work into reviewable units.

RESEARCH FOOTING

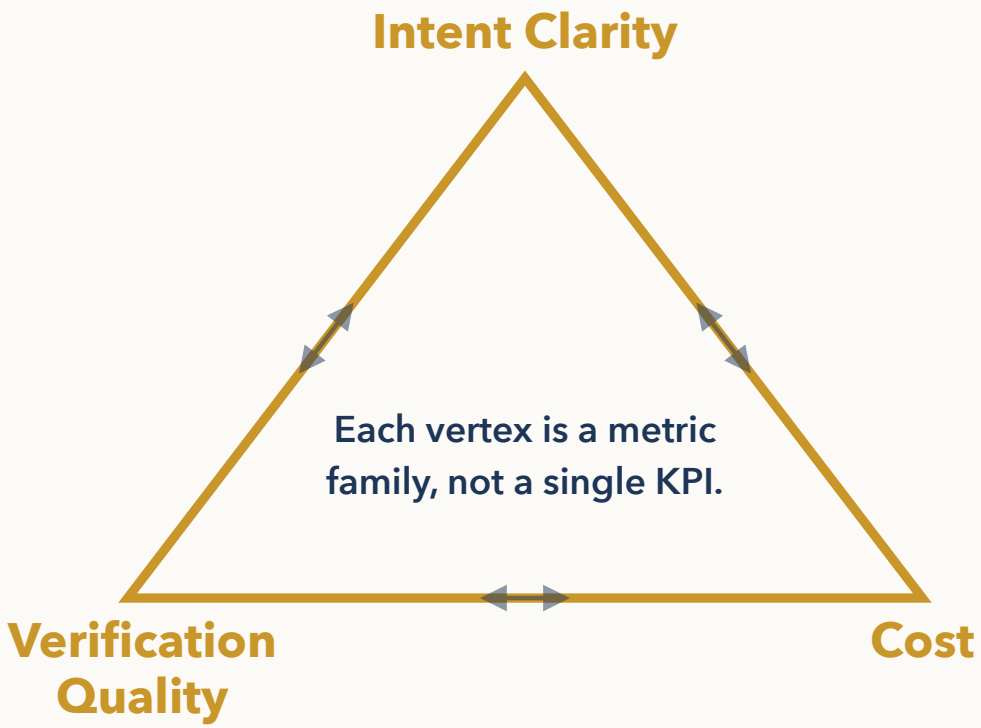
Not every metric below has the same status.

- **Established:** long-standing software quality or delivery metrics with strong prior use
- **Adapted:** established metric logic, adjusted for AI-assisted delivery

- **Synthesized:** practical management metric derived from adjacent research and current public engineering practice

That distinction matters. The appendix is strongest when it is explicit about which metrics are inherited from established literature and which are useful new management constructs.

The Verification Triangle



Intent Clarity

Are we converging on the right thing?

first-pass acceptance rate · post-merge rework rate · increment frequency

Verification Quality

Are they working?

change failure rate · deployment rework rate · machine catch rate · reviewer-minutes per accepted change · review cycle count · time-to-merge

Cost

What does it cost?

cost per accepted change · lead time to accepted change · review, rework, model, and infra cost share

Surrounding constraints

Scope · Permissions · Human Review · Identity · I/O Controls · Credentials · Network · Sandbox · Exit Conditions

Verification Triangle with metric families at each vertex: intent clarity, verification quality, and cost

VERTEX 1: INTENT CLARITY

Intent clarity answers a single question:

How quickly and reliably is the team converging on the right thing?

Do not measure intent clarity by scoring the spec. Measure it by the downstream signals — they are the ground truth for whether the thinking happened.

First-pass acceptance rate

Type	Synthesized
Formula	increments_merged_without_major_rework / total_increments
What it tells you	Whether the team converged before review had to do the thinking
How to measure	Count merged increments with no substantive CHANGES_REQUESTED cycle, no follow-up fix PR, and no revert inside the observation window. Teams with richer review data can segment by "major scope correction" versus small nits.
Source basis	Built from established code review signals (review round count, changes requested) and post-merge rework logic. Closest adjacent sources: Graphite and Jellyfish review-cycle metrics; DORA deployment rework rate; review-effectiveness literature from SmartBear/Cisco.

Post-merge rework rate

Type	Adapted
Formula	shipped_changes_requiring_fix_or_revert / shipped_changes
What it tells you	Whether approved increments were actually right once they met reality
How to measure	Count reverts, follow-up fix PRs, reopened tickets, or equivalent corrective deployments inside a fixed observation window, usually 14-30 days.
Source basis	DORA's deployment rework rate (added in 2024) and existing platform measures of corrective change activity.

Increment frequency

Type	Adapted
Formula	shippable_increments / week
What it tells you	The clock speed of the intent feedback loop
How to measure	Count increments that reach a shippable state each week. Use accepted changes where possible; otherwise use the team's stable definition of "shippable increment" and keep it fixed.
Source basis	Adapted from deployment-frequency logic in DORA and continuous-delivery practice. The Triangle uses it as a convergence-speed metric rather than a throughput KPI.

Supporting diagnostics

The three metrics above are the headline measures. The diagnostics below explain why they moved.

Abandonment rate

Type	Synthesized
Formula	prs_closed_without_merge_after_significant_work / total_prs

What it tells you	Whether teams are starting work before intent is stable enough to finish it
How to measure	Count PRs closed without merge that were open > 7 days and had > 50 lines changed. Filter out quick "wrong branch" or "duplicate" closes.
Source basis	Jiang et al. (ACM TOSEM, 2022): 265,325 PRs studied, abandonment driven by complex PRs, novice contributors, and lengthy reviews. Code Climate: average 8% abandonment, ~$24K/year/developer wasted.

Review cycle time

Type	Synthesized
Formula	Median calendar time from PR creation to final approval, segmented by review round count
What it tells you	Whether review is doing clarification late that should have happened earlier in the loop
How to measure	Count CHANGES_REQUESTED events per PR and measure calendar time between them. Graphite and Jellyfish track this natively.
Source basis	Fan et al. (2021) published models for predicting review rounds. Mantyla & Lassenius (2009) found 75% of review-found defects are evolvability issues, not functional bugs.

Rework rate

Type	Synthesized
Formula	changes_requiring_significant_rework / total_changes
What it tells you	A composite waste signal across pre-merge, post-merge, churn, and abandonment
How to measure	Count changes that required significant correction: post-merge reverts, follow-up fix PRs, re-opened tickets, and PRs with 3+ rounds of substantive CHANGES_REQUESTED review. Use a fixed observation window, usually 14-30 days for post-merge signals.
Source basis	DORA deployment rework rate; code churn research from Nagappan & Ball (ICSE 2005); industry benchmarks from LinearB.

Measurement notes

- The post-merge rework observation window must stay fixed. Thirty days catches immediate quality failures without mixing in unrelated later work. Sensitivity analysis in the companion study showed rework detection increases from 14 to 30 days but shows diminishing returns beyond 30 days.
- Abandonment is not always bad — some experimentation naturally leads to abandoned PRs. Watch for trends, not absolute numbers.
- Review cycle time should be decomposed: response time (push → review, measures queue pressure) vs iteration time (review → next push, measures change quality).

VERTEX 2: VERIFICATION QUALITY

Verification quality answers a single question:

Are defects being caught early enough, and are machines absorbing enough of the verification load?

Change failure rate

Type	Established
Formula	failed_deployments / total_deployments
What it tells you	How often deployments cause production degradation requiring remediation
How to measure	Link deployments to incidents, rollbacks, hotfixes, or customer-visible degradation using a stable failure definition. Most teams already have tooling for this – it is a DORA core metric.
Source basis	DORA core metric. Basis: DORA metrics guide

Measurement note: non-degrading escapes

Not every defect that reaches production causes service degradation. A cosmetic bug, an edge-case behavior, or a minor regression may escape your gates without triggering an incident. These do not count toward change failure rate, but you should track them separately. Each non-degrading escape is a gating opportunity: something your pipeline should have caught and didn't. If you are not tracking these, you cannot see where your gates have gaps. Review non-degrading escapes weekly and ask: could a deterministic check have caught this? If yes, build the gate.

Deployment rework rate

Type	Established
Formula	deployments_requiring_correction / total_deployments
What it tells you	How much shipped work required follow-up corrective work
How to measure	Count reverts, roll-forwards, hotfix deployments, or equivalent corrective releases inside the team's observation window
Source basis	DORA added this to make rework first-class rather than implicit. Basis: DORA history guide; DORA metrics guide

Machine catch rate

Type	Adapted
Formula	defects_caught_by_automated_gates / total_defects_found_before_production
What it tells you	Of all defects found before production, what percentage were caught by automated gates (linting, type checks, contract validation, test failures, static analysis) rather than by human reviewers
How to measure	Count defects caught by CI gates (lint failures, test failures, type errors, contract violations, secret detection hits). Count defects caught by human reviewers (changes-requested in review, reviewer-identified bugs). Divide machine-caught by total.
Complement metrics	Human save rate = defects caught by human reviewers / total defects found. Production escape rate = defects that passed both gates and review and reached users / total defects. All three rates sum to 100% of defects found. Production escapes feed directly into change failure rate.
Source basis	Adapted from decomposed reliability measurement. Uses the same 14-day observation window as the rework detector. Basis: Measuring Agents in Production; Towards a Science of AI Agent Reliability

Reviewer-minutes per accepted change

Type	Adapted

Formula	total_reviewer_minutes / accepted_changes
What it tells you	Whether human verification load is scaling or being overwhelmed
How to measure	Pull review timestamps or estimate reviewer time per accepted change using code-review system data
Source basis	Operationally important in public Anthropic and OpenAI write-ups about review bottlenecks, but not yet a canonical industry benchmark. Basis: Anthropic Code Review; Anthropic work-at-Anthropic; OpenAI Codex; OpenAI Harness Engineering

Review cycle count

Type	Adapted
Formula	review_rounds_before_merge
What it tells you	How much pre-merge friction each change creates
How to measure	Count distinct review rounds per PR from GitHub API (gh pr view --json reviews). A round is a review submission (approved, changes requested, or commented). Multiple reviews in the same round count once.
Source basis	Standard code review metric. In AI-assisted delivery, review cycle count is the clearest signal of intent clarity: well-specced changes pass review in fewer rounds because the intent is unambiguous. Basis: SmartBear/Cisco code review study; Graphite PR analysis

Time-to-merge

Type	Adapted
Formula	merged_at - created_at
What it tells you	How long changes spend in the verification pipeline before acceptance
How to measure	Pull PR creation and merge timestamps from GitHub API (gh pr view --json createdAt,mergedAt). Report as median per period, not mean, to avoid skew from long-lived PRs.
Source basis	Derived from DORA lead time. Distinct from lead time to accepted change because it measures only the review-and-merge phase, not the full spec-to-production cycle. Shorter time-to-merge with stable change failure rate means the pipeline is getting more efficient. Shorter time-to-merge with rising change failure rate means the pipeline is getting sloppy. Always read alongside change failure rate. Basis: DORA metrics guide; Graphite PR analysis

Measurement notes

- Pre-production defects include defects first found in CI, automated gates, staging, or human review.
- Production defects include incident-linked bugs, customer-reported correctness failures, and hotfix-worthy bugs discovered after release.
- Style comments, formatting nits, flaky CI timeouts, and duplicate findings should not count as defects.
- Review cycle count and time-to-merge are available from the GitHub API with zero custom instrumentation. They are the lowest-cost verification quality signals to adopt.

VERTEX 3: COST

Cost answers a single question:

How much does each trusted change actually cost to deliver?

Cost per accepted change

Type	Synthesized
Formula	(model_cost + infra_cost + engineering_cost + review_cost + rework_cost) / accepted_changes
What it tells you	The full unit cost of trusted delivery
How to measure	Combine model/API spend, infra/orchestration spend, human engineering time (discussion, whiteboarding, spec writing, prompting, context preparation), human review cost, and rework cost over the reporting window, then divide by accepted changes
Source basis	Management synthesis designed to connect engineering effort to business outcomes. Conceptually, it is the FinOps cost-to-serve pattern (CloudZero, FinOps Foundation) moved upstream from runtime cost to production cost — a business-meaningful denominator with hidden labor and rework in the numerator.

Lead time to accepted change

Type	Adapted
Formula	time_from_work_start_to_accepted_change
What it tells you	Whether trusted delivery is getting faster, not just cheaper
How to measure	Use issue start, branch start, or first commit as the start point; end when the change is in production and survives the observation window
Source basis	Derived from DORA lead time logic, but uses accepted change rather than deployment or merge as the end state. Basis: DORA metrics guide

Review cost per accepted change

Type	Synthesized
Formula	reviewer_minutes_per_accepted_change * fully_burdened_reviewer_rate
What it tells you	The direct labor cost of human verification
How to measure	Multiply reviewer-minutes per accepted change by the team's burdened reviewer cost
Source basis	Practical cost breakout, not a research-standard metric; useful because review often becomes the hidden cost center

Rework cost share

Type	Synthesized
Formula	rework_cost / total_delivery_cost
What it tells you	How much of delivery cost is waste rather than forward progress
How to measure	Estimate rework hours and related cost, then divide by total delivery cost for the same period
Source basis	Derived from rework and delivery-cost decomposition rather than a standard benchmark

Model/tool cost share

Type	Synthesized
Formula	(model_cost + tool_cost) / total_delivery_cost
What it tells you	Whether visible AI spend is a small or large part of actual delivery cost
How to measure	Use billing data for model/tool spend and divide by total delivery cost
Source basis	Useful because model cost is usually the easiest cost to see and the easiest to overemphasize

Infra/orchestration cost share

Type	Synthesized
Formula	infra_and_orchestration_cost / total_delivery_cost
What it tells you	How much hidden agent infrastructure costs relative to the rest of delivery
How to measure	Include compute, retrieval, orchestration, queueing, tracing, and other support costs where possible
Source basis	Operational cost metric; useful because AI workflow cost is rarely just API spend

Measurement notes

- Review cost and rework cost can begin as shadow estimates. Directional honesty matters more than false precision.
- If accepted-change measurement is not yet mature, use a clearly labeled proxy such as merged-to-main without revert within 14 days.
- Cost metrics are weakest when they are reported without quality metrics. Cost gets more useful when read next to spec and eval metrics.
- **Cost per accepted change is size-dependent.** Specs add overhead to small changes (writing a spec for a 30-line fix costs more than the fix) but dramatically reduce friction on large ones (a 500-line unspec'd change may take 5 review rounds and 48 hours to merge; the same change with a spec takes 2 rounds and 8 hours). Report cost per accepted change bucketed by PR size to see where specs pay for themselves and where they are overhead. This prevents a misleading global average that hides the size-dependent ROI.

SOURCE BASIS

These metrics are grounded in the following source families:

Established delivery and quality metrics

- DORA, DORA's software delivery performance metrics
- DORA, A history of DORA's software delivery metrics
- Capers Jones, Software Defect Removal Efficiency

Requirements quality and ambiguity

- Montgomery et al., Empirical research on requirements quality: a systematic mapping study
- Albayrak et al., Incomplete software requirements and assumptions made by software engineers

- Suri et al., CodeScout: Contextual Problem Statement Enhancement for Software Agents

AI-agent reliability and production measurement

- Measuring Agents in Production
- Towards a Science of AI Agent Reliability

Public engineering practice from leading labs

- Anthropic, Code Review
- Anthropic, How AI is transforming work at Anthropic
- OpenAI, Codex is now generally available
- OpenAI, Harness Engineering: Leveraging Codex in an Agent-First World

The strongest interpretation is:

- use **established** metrics as anchors
- use **adapted** metrics to make the anchor metrics AI-relevant
- use **synthesized** metrics to make the system operational for leadership