AI Coding Readiness & Guardrail Assessment

Sample Report — Meridian Analytics (Demonstration) · v3.0

> ***About this sample.*** *Meridian Analytics is a fictional but representative company — a composite of governance patterns common at 10–100-engineer software organizations. Every finding below is written exactly as it would appear in your report: observed with a cited source, risk-framed, independently validatable by your team, remediated concisely, and grounded in recognized authority. Your report is produced from your CI/workflow configs and intake questionnaire, by an AI agent pipeline with adversarial review and expert human sign-off — the same gated architecture Section 5 teaches you to build.*

Company profile: Meridian Analytics · B2B data-analytics SaaS · 42 engineers across 6 teams · AI coding tools (Copilot, Cursor) in use ~14 months · monorepo + 3 service repos · GitHub Actions CI.

---

1. Executive Summary

Coverage: 5 dimensions · 34 control points assessed · **7 controls verified in place** (Section 2) · **11 findings** (Section 3). Every verified control and every finding cites its evidence source.

Readiness verdict: Level 1 of 4 — Adopted, Ungoverned. Meridian's engineers use AI coding tools daily and productively. The organization has no structural mechanism that distinguishes AI-authored code from human-authored code at any point between keystroke and production. Velocity gains are real; so is the unpriced risk accumulating behind them.

Dimension	Score (0–4)	One-line basis
Validation coverage	1.5	46% line coverage, no coverage gate, flaky-test quarantine disabled
Review-gate integrity	1.0	AI attribution via ad-hoc PR comments only; single-approver merges to main
Guardrails & permissions	0.5	No protected paths; CI config editable by any PR; secrets exposed to AI editor context in 2 repos
Autonomy posture	2.0 (misaligned)	Team operates at higher autonomy than its validation quality permits
Policy & disclosure	1.0	No written acceptable-use policy; license/provenance handling undefined

Top 3 risks: (1) CI pipeline configs are modifiable by the same agents whose output the pipeline validates — the validator is not independent of the validated (F-07). (2) Test coverage is insufficient to catch the defect classes AI assistants most commonly introduce, at the volume they introduce code (F-01–F-03). (3) Secrets in two repos are readable by AI editor context (F-08).

Top 3 quick wins (this month, low effort): protect `.github/` and CI config paths behind CODEOWNERS; enable branch protection with required checks on `main` in all four repos; migrate the two exposed secrets to the CI secret store.

The one-paragraph version for your CTO: Meridian gets real velocity from AI coding tools and should keep them. But the org is operating at autonomy Level 2-equivalent trust with Level 1-quality validation. The fix is not less AI — it is a 90-day sequence of guardrails and validation upgrades, after which the team can *raise* autonomy safely instead of capping it out of fear.

---

2. Positive Observations — Controls Verified In Place

The assessment verifies what is working with the same evidence standard applied to findings. This section exists at every finding count: a zero-finding report carries the largest version of it, so "no findings" always reads as "controls verified," never "nothing checked."

Validation

- CI executes the full test suite on every PR across all four repos — no skip-paths observed. *(Source: `ci.yml`, all repos.)*
- Production deploys are blocked on green CI; no manual-override deploy path exists in the workflow. *(Source: deploy workflow review.)*

Review & merge

- All changes reach `main` via pull request — zero direct pushes observed in the 200-PR sample. *(Source: PR history sample; repo settings export.)*

Guardrails & permissions

- The CI secret store is in active use for deployment credentials — the two exposed credentials in F-08 are exceptions to an otherwise correct pattern, not the pattern itself. *(Source: workflow environment review.)*
- Organization-wide two-factor authentication is enforced. *(Source: org settings export.)*
- Dependabot security updates are enabled on all four repos. *(Source: repo settings.)*

Culture & readiness

- Engineers voluntarily flag AI-assisted PRs in comments without any mandate (9 observed) — the attribution instinct F-04 formalizes already exists in the team. *(Source: PR history sample.)*

---

3. Findings

Finding format: **Observation** (what is, with source) · **Risk to the Organization** (why it matters here) · **Steps to Validate** (verify it yourself — no finding asks for trust) · **Remediation** (concise; the 90-day plan in Section 5 sequences these) · **References** (recognized authority; a finding without references is opinion).

F-01 — No enforced coverage floor on AI-accelerated code

Observation. Line coverage is 46% overall; `billing-service` sits at 31%. No coverage threshold is enforced anywhere — `ci.yml` contains no coverage gate step. *(Source: coverage artifacts in CI run history;* `ci.yml` *full review; questionnaire Q14.)*

Risk to the Organization. Coverage can decline silently while AI-assisted authorship raises code volume, compounding untested surface each sprint. The revenue-critical billing path carries the weakest safety net, so the highest- blast-radius repo has the lowest defect-detection probability.

Steps to Validate.

1. Run the coverage suite locally per repo (`pytest --cov` / `nyc` per stack) and compare against CI artifacts.
2. Grep all workflow files for a coverage threshold or gate step — confirm absence.
3. Plot coverage over the last 12 months from CI artifact history; check the trend against commit volume.

Remediation. Add a changed-line coverage gate (start 50%, ratchet to 70%) as a required status check; report total coverage as a dashboard metric, not a gate. Prioritize `billing-service`.

References.

- NIST SP 800-218 (SSDF) v1.1 — PW.8: verify code through testing prior to release
- Google Testing Blog, "Code Coverage Best Practices" (2020) — changed-code coverage over repo-total targets
- DORA research program — test automation as a capability predicting delivery performance

---

F-02 — Disabled flaky-test quarantine has degraded the oracle

Observation. The flaky-test quarantine block in `ci.yml` (lines 61–74) is commented out, dated 11 months ago. Engineers report re-running failed CI as routine practice. *(Source:* `ci.yml`*; questionnaire Q17.)*

Risk to the Organization. A red build no longer reliably signals a defect, so humans are trained to override the signal — and every downstream gate (coverage, review, deploy) inherits the noise. This is the single most consequential finding in this report: with a noisy oracle, increasing AI code volume increases escaped defects roughly linearly, and no added gate fixes it until the oracle is trusted again.

Steps to Validate.

1. Pull the last 200 CI runs; count failed runs that passed on re-run with no code change — that ratio is your noise floor.
2. Survey the team: "When CI fails, what do you do first?" Count "re-run it" answers.
3. Confirm the quarantine block's git blame date and the absence of any replacement mechanism.

Remediation. Re-enable quarantine; institute a weekly quarantine-list burn-down owned by the platform team; track re-run-to-green rate as the oracle-health metric with a <2% target.

References.

- Luo et al., "An Empirical Analysis of Flaky Tests," ACM FSE 2014 — foundational taxonomy of flakiness causes
- Google Testing Blog, "Flaky Tests at Google and How We Mitigate Them" (2016)
- Martin Fowler, "Eradicating Non-Determinism in Tests" (2011)

• NIST SP 800-218 (SSDF) — PW.8.2: determine that testing is sufficiently reliable to act on

---

F-03 — Test types don't match AI-characteristic defect classes

Observation. No mutation testing, property-based testing, or contract tests exist on the two service boundaries with the highest AI-assisted refactor traffic. *(Source: questionnaire Q15, Q22; CI config review.)*

Risk to the Organization. AI assistants' documented failure modes — plausible-but-wrong edge-case handling, subtly altered semantics in refactors, insecure defaults — are precisely the classes line coverage does not detect. Peer-reviewed studies find a material fraction of AI-suggested code contains security-relevant weaknesses, and that developers using assistants can be *more* confident while producing *less* secure code. Meridian's test portfolio was designed for human-defect distributions and is now facing a different distribution at higher volume.

Steps to Validate.

1. Run a one-off mutation-testing pass (PIT/Stryker/mutmut per stack) on one hot module; a mutation score far below line coverage confirms hollow coverage.

2. Diff-sample 20 recent AI-assisted refactor PRs at the two service boundaries; check whether any test would have caught an intentional semantic inversion.

3. Confirm absence of consumer-driven contract tests on the two boundaries.

Remediation. Add contract tests (e.g., Pact-style) on the two hot boundaries; run a weekly mutation-testing audit on `billing-service` as a coverage-quality metric (not a merge gate).

References.

• Pearce et al., "Asleep at the Keyboard? Assessing the Security of GitHub Copilot's Code Contributions," IEEE S&P 2022

• Perry et al., "Do Users Write More Insecure Code with AI Assistants?," ACM CCS 2023

• GitClear, "Coding on Copilot" code-quality research (2024) — churn/duplication shifts in AI-assisted codebases

• Fowler, "ContractTest" / Pact documentation — consumer-driven contract testing

• OWASP Top 10 for LLM Applications (2025) — LLM09: Misinformation (overreliance on generated output)

F-04 — AI-authored code carries no structural attribution

Observation. AI attribution exists only as voluntary PR comments — present on 9 of the last 200 PRs, against questionnaire-reported usage of "daily, most code touched." *(Source: PR history sample; questionnaire Q6.)*

Risk to the Organization. Reviewers cannot apply different scrutiny to machine output they cannot identify, so human-trust heuristics ("Sarah knows this subsystem") are silently misapplied to generated code. Incident forensics, license/provenance questions, and any future compliance obligation (customer security questionnaires, ISO/AI-audit requests) all require knowing which code is AI-authored — that record is currently unrecoverable.

Steps to Validate.

1. Count PRs labeled or annotated as AI-assisted over the last quarter; compare against tool-usage telemetry or a team survey.

2. Ask three reviewers whether they knowingly review AI-generated diffs differently; ask how they would know.
3. Attempt to answer, for one production incident, "was the faulting code AI-generated?" — time how long it takes.

Remediation. Mandatory `ai-assisted` / `human-authored` PR labels, enforced by a CI check; record the label in commit trailers for permanence. Fifteen lines of workflow YAML; template included in the delivery package.

References.

- NIST AI RMF 1.0 — MAP 2.3 / GOVERN 1.2: documentation and traceability of AI system outputs
- ISO/IEC 42001:2023 — AI management systems: traceability and accountability controls
- SLSA v1.0 — provenance: knowing what produced an artifact
- NIST SP 800-218 (SSDF) — PS.3: archive and protect provenance data

---

F-05 — Single-approver (and self-approval) merges to main

Observation. All four repos accept single-approver merges to `main`; two permit author self-approval. *(Source: branch-protection settings export.)*

Risk to the Organization. Combined with F-04, one engineer plus one AI assistant can move unreviewed generated code to production with zero independent human eyes on it. Beyond defect risk, this fails the two-person review bar that supply-chain frameworks treat as baseline, and it is a finding customer security reviews increasingly check directly.

Steps to Validate.

1. Read branch-protection settings on each repo (Settings → Branches, or the API export we analyzed).
2. Query merged PRs from the last quarter where author == sole approver; count.
3. Confirm "required approvals ≥ 2" and "dismiss stale approvals" are unset.

Remediation. Disable self-approval everywhere; require two approvers on `ai-assisted` PRs to `main`; enable required status checks and stale-approval dismissal. Configuration baseline included in the delivery package.

References.

- SLSA v1.0 — source requirements: two-person review of changes
- CIS Software Supply Chain Security Guide — §1.1: code-review and branch-protection controls
- OWASP Top 10 CI/CD Security Risks — CICD-SEC-1: Insufficient Flow Control Mechanisms
- NIST SP 800-218 (SSDF) — PW.7: human review of code

F-06 — Review capacity is saturating under AI-accelerated authorship

Observation. Median PR waits 22 hours for first review; PR size has grown 38% year over year. *(Source: repo analytics export.)*

Risk to the Organization. AI raises authorship throughput; review capacity is fixed. The predictable equilibrium is rubber-stamp review — the gate exists on paper and not in practice — at exactly the moment F-04 means reviewers most need to scrutinize what they approve. Industry research on AI-assisted teams shows delivery instability rising

where review and validation practices don't adapt alongside adoption.

Steps to Validate.

1. Recompute median time-to-first-review and PR size trends from your own analytics (we used the export you provided; the queries are in the appendix).

2. Sample 20 approvals: measure time between "review started" and "approved"; sub-5-minute approvals on 400+ line diffs are your rubber-stamp rate.

Remediation. Enforce a 400-changed-line ceiling on `ai-assisted` PRs; set a team review SLA; track review depth (time-in-review vs. diff size) as a monthly health metric.

References.

• Google Engineering Practices documentation — "Small CLs": review quality degrades with change size

• DORA "State of DevOps / Impact of AI" research (2024) — AI adoption's relationship to delivery stability absent process adaptation

• SmartBear / Cisco code review study (2006, replicated since) — defect-detection falls sharply beyond ~400 LOC per review

---

F-07 — The validation pipeline is modifiable by the code it validates

Observation. No protected paths exist: `.github/workflows/`, `ci.yml`, and deployment configs are editable by any PR, including AI-generated ones. No CODEOWNERS files present. *(Source: repo tree review; branch-protection export.)*

Risk to the Organization. The validator is not independent of the validated. A single malicious, compromised, or simply careless PR can weaken or disable the very checks that gate it — the "poisoned pipeline execution" pattern. With AI agents authoring code at volume, the probability of an inadvertent pipeline-weakening change rises even absent an attacker. Your tests are only as trustworthy as the least-reviewed PR that could touch them.

Steps to Validate.

1. Open a test PR that edits `ci.yml` to skip the test job; observe that nothing distinguishes it from a normal PR (revert immediately).

2. Confirm the absence of CODEOWNERS entries for `.github/` and infra paths in all four repos.

3. Check whether workflow files require any additional approval to change (GitHub: Settings → Actions → workflow permissions; branch protection path rules).

Remediation. CODEOWNERS on `.github/`, `**/ci.yml`, `infra/`, and deployment configs, owned by platform leads; branch protection requiring code-owner review on those paths; restrict Actions workflow permissions to read-only defaults. Templates included in the delivery package.

References.

• OWASP Top 10 CI/CD Security Risks — CICD-SEC-4: Poisoned Pipeline Execution

• NIST SP 800-204D — integrating supply-chain security into DevSecOps CI/CD pipelines

• SLSA v1.0 — build integrity: build platform isolation from influence by tenants

• CIS Software Supply Chain Security Guide — §3: build pipeline configuration controls

F-08 — Live credentials in tracked files, readable by AI tool context

Observation. Two repos contain live credentials in version control: a `.env.example` populated with production values, and a hardcoded API key in a test fixture. Any AI editor with workspace context reads both. *(Source: config scan.)*

Risk to the Organization. Committed secrets are exposed to every present and past clone, every CI runner, and — new with AI tooling — every editor agent whose context leaves your boundary under the vendor's data-handling terms. Rotation cost is immediate and certain; breach cost is unbounded. Secrets in git history remain exposed after deletion from HEAD.

Steps to Validate.

1. Run a secret scanner (gitleaks / trufflehog) across all four repos *including full history*; confirm the two hits and enumerate any others.
2. Test whether the flagged credentials are live (a scoped read call).
3. Check your AI tool's context/telemetry settings against its current data-retention terms.

Remediation. Rotate both credentials today; migrate to the CI secret store; add secret scanning as a required status check; add pre-commit scanning to developer machines. History rewrite is optional once rotated.

References.

- CWE-798 — Use of Hard-coded Credentials
- OWASP Top 10 CI/CD Security Risks — CICD-SEC-6: Insufficient Credential Hygiene
- GitGuardian "State of Secrets Sprawl" (annual) — scale and persistence of committed secrets
- NIST SP 800-218 (SSDF) — PS.1: protect all forms of code from unauthorized access
- OWASP Top 10 for LLM Applications (2025) — LLM02: Sensitive Information Disclosure

---

F-09 — AI tool context is unscoped across all repositories

Observation. Cursor is configured with full-workspace context in every repo, including `infra/` and the compliance-documentation directory. *(Source: questionnaire Q9; workspace config review.)*

Risk to the Organization. Least-privilege is absent at the AI layer: infrastructure topology, security posture documents, and compliance materials flow into prompt context without need. This widens the blast radius of F-08, creates data-residency questions your customer contracts may not permit, and makes vendor-side retention terms a single point of policy failure.

Steps to Validate.

1. Open the AI tool's workspace/context settings per repo; enumerate inclusion rules (or their absence).
2. Trigger a completion inside `infra/` and observe context the tool reports using.
3. Map what the vendor's current terms say about retention/training on your tier.

Remediation. Scope AI tool context per directory: exclude `infra/`, compliance docs, and anything customer-data-adjacent via tool ignore files (e.g., `.cursorignore`); standardize the exclusion file in the repo template.

References.

- NIST SP 800-53 rev.5 — AC-6: Least Privilege
- OWASP Top 10 for LLM Applications (2025) — LLM02: Sensitive Information Disclosure
- NIST AI RMF 1.0 — MAP 1.6 / MANAGE 2.2: mapping data flows into AI systems and managing third-party risk
- Vendor documentation: Cursor privacy mode & ignore-file controls (verify current terms at review time)

F-10 — Operated autonomy exceeds earned autonomy (headline finding)

Observation. Engineers report delegating whole tickets to AI tools and reviewing "at the diff level, mostly" — L2 behavior on the autonomy ladder (L0 human-does-everything → L3 AI self-advances within pre-registered criteria). The validation estate documented in F-01–F-08 supports L1 at most. *(Source: questionnaire Q7; findings F-01–F-08.)*

Risk to the Organization. The gap between operated and earned autonomy is where AI incidents live: trust is being extended that the control environment cannot verify. Teams in this state typically respond to their first serious incident by banning tools — destroying the velocity gains — rather than by closing the gap. The strategic risk is not the incident; it is the overcorrection.

Steps to Validate.

1. Survey engineers: "What's the largest unit of work you hand to an AI tool end-to-end?" Map answers to the ladder.
2. Cross-check against the validation criteria in Section 5.1; score each repo's *earned* level.
3. The delta between the two numbers is this finding.

Remediation. Adopt the laddering plan in Section 5.1: make L1 real in 30 days (attribution, two-person review, protected paths), earn L2 per-repo within 90 (coverage gate, trusted oracle, escape-rate evidence). Raise autonomy only against met criteria — never by default, never org-wide.

References.

- NIST AI RMF 1.0 — MANAGE function: risk-based decisions on AI system use and oversight levels
- ISO/IEC 42001:2023 — controls for determining appropriate human oversight of AI systems
- DORA research (2024) — AI adoption outcomes conditional on surrounding engineering practices

---

F-11 — Acceptable-use policy exists in 42 heads, in 42 versions

Observation. No written policy covers what data may enter AI tool context, which licenses are acceptable in AI-suggested code, or what provenance is recorded. Two engineers gave opposite answers on whether customer data may appear in prompts. *(Source: questionnaire Q10–Q12.)*

Risk to the Organization. Individually improvised policy means your real policy is the loosest interpretation currently in use. Exposure spans customer-data handling (contractual breach), license contamination in AI-suggested code (IP

risk on the product itself), and indefensibility in any future audit or customer security review — "we had no policy" is the worst possible answer to a good-faith incident.

Steps to Validate.

1. Ask five engineers the same three questions (customer data in prompts? license check on suggestions? where is this written?); count distinct answers.

2. Search the wiki/handbook for any AI acceptable-use document; confirm absence.

3. Sample 10 AI-assisted PRs for license-relevant inclusions (vendored snippets, unusual idioms) and check whether anything would have caught them.

Remediation. Ship the two-page policy skeleton in the delivery package: permitted data classes, license posture and provenance recording, tool configuration standards, incident path. Enforce the technical half via F-09's scoping and F-04's attribution — policy that isn't mechanized decays.

References.

- ISO/IEC 42001:2023 — AI management system: acceptable-use and organizational controls
- NIST AI RMF 1.0 — GOVERN 1.1–1.4: policies, accountability, and risk tolerances for AI use
- ISO/IEC 5230 (OpenChain) — open-source license compliance program requirements
- NIST SP 800-218 (SSDF) — PO.1: define security requirements for software development

---

4. Risk Register

#	Risk	Findings	Severity	Effort to close
R-1	Validator not independent of validated	F-07	**Critical**	Low — ~1 day
R-2	Secrets exposed to AI tool context	F-08, F-09	**Critical**	Low — ~2 days
R-3	Defect escape scales with AI volume against weak, noisy coverage	F-01–F-03	High	Medium — 90-day program
R-4	Unattributed AI code reviewed with human-trust heuristics	F-04, F-05	High	Low-Medium
R-5	Review saturation converts gates to rubber stamps	F-06	Medium	Medium
R-6	Undefined acceptable use → data/IP/audit exposure	F-09, F-11	Medium	Low — ~1 week

Severity = blast radius × current likelihood. R-1 and R-2 are critical because each is a single-PR / single-prompt event, not an accumulation.

---

5. Tailored Adoption Playbook

5.1 Autonomy-laddering plan

Current earned level: **L1**. Operated level: **L2**. Make L1 real in 30 days; earn L2 per repo within 90. `billing-service` ladders last. **L2 criteria (per repo, all required):** changed-line coverage gate ≥70% · flaky quarantine active, <2% quarantine rate · AI-attribution labeling in force · protected paths enforced · 30 days at <1 defect escape per 50 AI-attributed PRs.

5.2 90-Day Sequence (week-level, owner-assigned, nothing longer than a week)

Weeks	Work	Closes	Owner
1	CODEOWNERS + branch protection all repos; rotate + migrate secrets	F-07, F-08	Platform lead
2	AI-attribution label + CI check; 2-approver policy; disable self-approval	F-04, F-05	Eng managers
3–4	Re-enable flaky quarantine; weekly burn-down cadence	F-02	Platform team
5–6	Changed-line coverage gate at 50%, announced ratchet to 70%	F-01	Platform team
7–8	Acceptable-use policy shipped; AI context scoping applied	F-09, F-11	Eng dir + senior IC
9–10	Contract tests on the two hot boundaries; PR size ceiling + review SLA	F-03, F-06	Owning teams
11–12	Coverage gate to 70%; score L2 criteria per repo; declare where earned	F-10	Eng dir

5.3 What NOT to do

Do not respond to this report by banning or throttling AI tools. The gap is governance debt, not tool risk — prohibition predictably produces shadow usage on personal accounts, which is strictly worse on every dimension measured here.

---

6. Guardrail Templates (Appendix — delivered as files)

CODEOWNERS protected-path config · AI-attribution CI check (~15 lines of workflow YAML) · branch-protection baseline · `.cursorignore` / context-scoping standard · two-page acceptable-use policy skeleton · analytics queries used in F-01/F-06 validation steps.

Methodology & Disclosure

This assessment was executed by an AI agent pipeline — intake clarification, config analysis against a governance rubric, drafting, adversarial review — with expert human quality sign-off before release. That is deliberate: it is the same gated, validation-first architecture Section 5 recommends, operating at the autonomy level its own validation has earned. Analysis is confined to materials you provide; nothing is retained beyond delivery.

Scoring rubric, briefly: each dimension is scored 0–4 against fixed criteria; scores are evidence-gated — a score without a cited finding is structurally invalid in our pipeline and is rejected before a human sees the draft. Findings follow a fixed contract: observation with source, organizational risk, independent validation steps, concise remediation, and references to recognized authority. A finding without references is opinion; we don't ship opinion.

Sample report v2.0 · codegoverned.com · Fixed-scope assessment, $950, delivered in 5 business days. Standard terms at checkout.