Tech Debt Governance Policy

Organizational Policy Template for Managing Technical Debt

For VPs of Engineering and CTOs

TechDebt.fail

https://techdebt.fail/

1. Purpose and Scope

This policy establishes a framework for identifying, tracking, prioritizing, and remediating technical debt across all engineering teams. It applies to all software development activities including new feature development, maintenance, and AI-assisted coding.

1.1 Definitions

- **Technical Debt:** The implied cost of future rework caused by choosing an expedient solution now instead of a better approach.
- **Debt Item:** A tracked instance of technical debt in the backlog.
- **Remediation:** The process of reducing or eliminating a debt item.
- **Debt Ratio:** The percentage of engineering effort spent on debt vs features.

1.2 Policy Owner

This policy is owned by: _____ (VP Engineering / CTO)

Review frequency: Annually or after significant organizational change

Last reviewed: _____

2. Debt Classification

2.1 Categories

- **Code Debt:** Code smells, duplication, complexity violations
- **Architecture Debt:** Design pattern violations, coupling issues
- **Dependency Debt:** Outdated libraries, vulnerable packages
- **Testing Debt:** Missing coverage, flaky tests, no E2E
- **Infrastructure Debt:** Manual deployments, monitoring gaps
- **Documentation Debt:** Missing docs, outdated runbooks
- **AI-Generated Debt:** Unreviewed AI code, inconsistent patterns

2.2 Priority Levels

- **P1 - Critical:** Security vulnerabilities, compliance violations, data loss risk.
  SLA: Remediate within 1 sprint.
- **P2 - High:** Customer-facing impact, blocking other work.
  SLA: Remediate within 1 quarter.
- **P3 - Medium:** Slowing development, increasing maintenance cost.
  SLA: Remediate within 2 quarters.
- **P4 - Low:** Cosmetic, minor inefficiencies.
  SLA: Address opportunistically.

These SLA windows are commitments your organization makes, not findings from any published study. Set windows your teams can actually meet, then hold to them.

3. Engineering Capacity Allocation

3.1 Minimum Allocation

All teams MUST allocate a minimum of _____% of sprint capacity to debt reduction. A common starting point is 15%; set the figure your leadership is willing to enforce.

This allocation is non-negotiable and may not be borrowed for feature work without VP approval.

3.2 Allocation Tiers

- **Healthy (debt ratio < 10%):** 10% capacity for maintenance
- **Moderate (debt ratio 10-25%):** 15% capacity mandatory
- **Elevated (debt ratio 25-40%):** 20% capacity mandatory
- **Critical (debt ratio > 40%):** 30% capacity + dedicated remediation team

The tier boundaries and percentages above are suggested defaults for you to adopt or change. They are policy choices, not industry benchmarks, and no publisher reports them as such.

3.3 Exceptions

- Short-term exceptions (1 sprint) may be approved by Engineering Director.
- Extended exceptions (2+ sprints) require VP/CTO approval and remediation plan.
- All exceptions must be documented with justification and payback plan.

4. AI-Assisted Development Policy

4.1 Approved AI Tools

List approved AI coding assistants: _____

Unapproved tools may not be used for production code.

4.2 Review Requirements

- All AI-generated code requires human review before merge.
- AI-generated PRs must be labeled/tagged for tracking.
- Reviewers must verify: architecture conformance, security, test coverage, no hallucinated dependencies.

Why hallucinated dependencies get their own check: researchers found 205,474 unique hallucinated package names across 576,000 code samples from 16 large language models, with hallucination rates of at least 5.2% for commercial models and 21.7% for open-source models (Spracklen et al., 34th USENIX Security Symposium, 2025). Separately, AI-generated code introduced risky security flaws in 45% of tests across more than 100 models (Veracode, 2025 GenAI Code Security Report, 2025).

https://arxiv.org/abs/2406.10279 and https://www.veracode.com/resources/analyst-reports/2025-genai-code-security-report/

4.3 Quality Gates

- AI code must pass all existing quality gates (lint, tests, coverage thresholds).
- Additional AI-specific checks: dependency validation, pattern compliance scan.
- Monthly audit of AI-generated code for pattern drift.

4.4 Accountability

- The developer who submits AI-generated code is responsible for it.
- Cannot be merged to do what the AI suggested without the developer understanding why.

5. Reporting Requirements

5.1 Team-Level Reporting (Monthly)

- Debt items created vs resolved
- Current debt ratio and trend
- Capacity allocation actual vs target
- Top 5 debt items by impact

5.2 Organization-Level Reporting (Quarterly)

- DORA metrics trending
- Debt cost estimate (using ROI Calculator)
- Cross-team dependency debt
- AI-generated debt percentage

5.3 Executive Dashboard (Quarterly)

- Business impact of debt (incident frequency, velocity, retention)
- Investment vs return on debt remediation
- Risk assessment for critical systems

6. Policy Enforcement

6.1 Compliance

- Teams are audited quarterly for policy compliance.
- Non-compliant teams must present a remediation plan within 2 weeks.
- Repeated non-compliance escalated to VP Engineering.

6.2 Policy Review

- This policy is reviewed annually.

Any team may propose amendments through: _____

- Changes approved by policy owner and engineering leadership.

Signatures

Policy Owner: _____ Date: ___

CTO/VP Engineering: _____ Date: ___

Engineering Director: _____ Date: ___

Template from https://techdebt.fail/ai-governance-framework/