LEADDEV BERLIN

The New Tech Debt

How AI is changing the way our systems fail, and what we did about it

Bhavana Hindupur

November 2026

Development velocity has shifted

Adoption

84%

of developers use or plan to use AI tools.

Stack Overflow Developer Survey 2025

Code generation

75%

of Google's new code is AI-generated.

Sundar Pichai, April 2026, via Semafor

Merge velocity

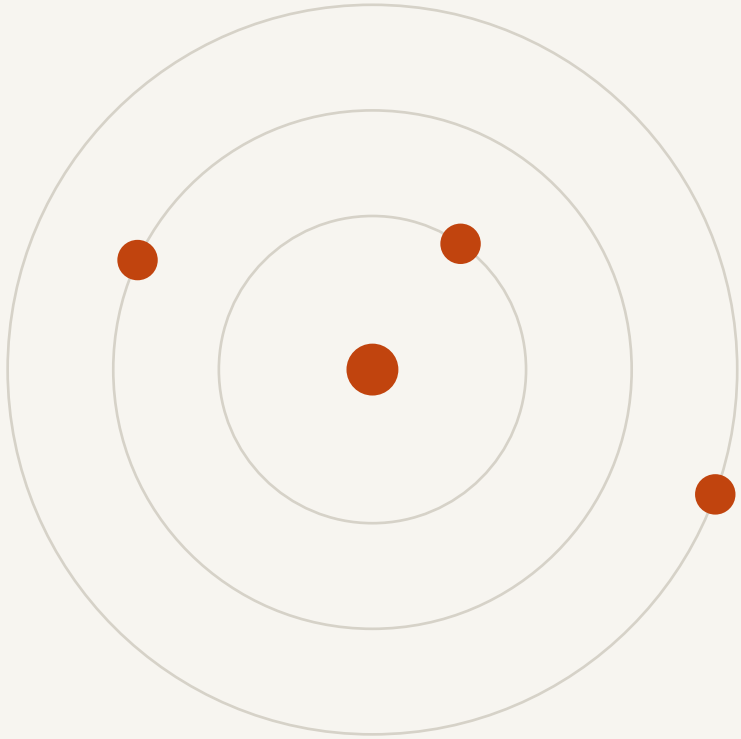
+98%

more PRs merged. Review time up 91%.

Faros AI, 10,000+ developers, 2025

ABOUT ME

I build systems that run at planet scale



01

Amazon

Retail backend

02

Google

Cloud

03

Meta

Security

04

Airbnb

Guest and host experiences

02 — FAILURE MODES & MITIGATIONS

Four failure modes, and how we close them

2.1 Local correctness, global incoherence
2.2 Test AI slop
2.3 Pattern replication at scale
2.4 Review signal degradation

FAILURE MODE 1 OF 4

2.1 Local correctness, global incoherence

Every PR passes review. The system still drifts apart.

What we did: checks that block



2.1 Local correctness, global incoherence

Every PR looks right. The system still drifts.

Bypasses a required layer

Skips the layer everything depends on.

```
// controller.ts
- dataAccess.getProduct(id)
+ db.query('SELECT * FROM…')
```

"Cleans up" load-bearing legacy code

The "redundant" line was load-bearing.

```
// retry logic
- await sleep(50)
  // handled replica lag
```

Picks the textbook answer

Drops the durability the queue gave us.

```
// payments.ts
- queue.enqueue(payment)
+ await payment.processSync()
```

2.1 — what we did Make the hidden rules explicit, then place them

Same rule, different home, different behavior.

Must it happen every time? No ↓ →

Hook
Guaranteed to run.

Can you fix it in the code? No ↓ →

Fix the codebase
Lint, types, tests. Then delete the rule.

Does Claude need it every session? No ↓ →

CLAUDE.md
Core conventions. Under 200 lines.

Only for certain files or folders? No ↓ →

Nested CLAUDE.md
Per folder

Rules file
Per file pattern

Needed only for specific tasks? →

Skill
Reference or workflow

Subagent
Bulky side tasks

2.1 — WHAT WE DID

Rules advise. Checks block.

Agents write at machine speed. What stops bad code must be deterministic.

ADVISORY

Written rules

CLAUDE.md, rules, skills. Context, not configuration. The model decides.

BLOCKING

Deterministic checks

Hooks Agent can't finish until checks pass.

CI PR can't merge until checks pass.

Build rules A forbidden dependency fails the build.

FAILURE MODE 2 OF 4

2.2 Test AI slop

The suite is green. It proves nothing.

What we did: anti-slop tooling

2.2 Test AI slop

Then

Writing a test cost effort, so tests tracked what mattered.

Now

Agents generate tests as easily as code. The suite is green and proves little.

What slop looks like

- Name and body disagree
- Trivial getter tests
- Over-mocking
- Asserts presence, never behavior
- No assertions at all

Why it matters

- False sense of security
- Burns CI and oncall time
- Replicates itself
- Bloats the codebase at AI speed

2.2 what we did We run hooks so the agent catches its own slop

```
claude: ~/shop-api

> add tests for the order service
● Write(src/test/java/shop/OrderServiceTest.java)
  – PostToolUse hook: anti-slop check failed
    ✗ 3 tests have no assertions
    ✗ 4 tests differ only by input
      → use @ParameterizedTest (JUnit 5) or @DataProvider (TestNG)
    ✗ 2 tests only assert not-null
    ✗ OrderRepository is mocked in OrderServiceTest
● Update(src/test/java/shop/OrderServiceTest.java)
  + assertThat(order.total()).isEqualByComparingTo("42.50");
  – PostToolUse hook: anti-slop check passed ✓
```

What the hook can check

- ✓ Every test asserts behavior
- ✓ Same body, new inputs: parameterize
- ✓ No tests for plain getters
- ✓ Never mock the class under test
- ✓ Test name matches what it checks

The hook says what to fix, not just no. Claude corrects the test before any PR.

4.2 what we did Our CI grader fails the build on slop

✓ passed
Build
2m 14s

→

✓ passed
Unit tests
All green

→

✗ failed
Anti-slop grader
Test quality 0.41, threshold 0.70

→

⊘ skipped
Deploy
Blocked

```
anti-slop-grader: 14 of 20 new tests below threshold
tests/cart.test.ts                6 tests with no assertions
tests/cart.test.ts                5 tests assert presence only
tests/payments.test.ts            3 tests mock the code under test
```

Green suite. Failed build.

FAILURE MODE 3 OF 4

2.3 Pattern replication at scale

One bad pattern becomes the template for the next hundred changes.

What we did: detect, then remediate

2.3 How existing code shapes new code

The agent doesn't just read your code. It treats it as the spec.

01
Search
Finds similar code first.

→

02
Infer
Repetition reads as convention.

→

03
Reproduce
Copies the pattern and its hidden assumptions.

Example: loosely typed Java

```
// Billing.java
Object applyDiscount(Map<String, Object> order,
  Map<String, Object> opts) { ... }
// new call, written by the agent
+ applyDiscount(cart, Map.of("percent", 10));
// compiles, but the real key is "rate"
```

No contract, so the agent guesses.

Claude Code's own guidance

Agents follow existing patterns. Good or bad.

2.3 what we did We measured what our agents get wrong

How we measured: agent changes a human edited, or that needed a fix or revert within 14 days, split by file traits

01

Dead code

Agents copy unused patterns as if valid

02

Static typing debt

No contracts, so models hallucinate them

03

Bad test quality

No safety net to verify a change

04

Large file size

Past 10k+ LOC, reasoning degrades

Caveat: our list, not a universal ranking. Measure yours.

2.3 what we did Remediate: deterministic first, agentic second

1 Deterministic codemod

rules/replace-price-formatter-v1.yml

```
id: replace-price-formatter-v1
pattern: formatPriceV1($PRICE)
fix: formatPrice($PRICE)
$ ast-grep scan --update-all
```

No model. Every match gets the same fix, on every file.

2 Agentic codemod

agents/typed-options-migration.yml

```
name: typed-options-migration
schedule: weekly
scope: src/billing/**
task: typed record, not a loose map
checks: same as any PR
review: a human merges
```

Only where a rule can't express the change.

FAILURE MODE 4 OF 4

2.4 Review signal degradation

Every PR looks clean, so review catches less.

What we did: a tiered pipeline



4.4 Review signal degradation

Then

Reviewers spot risk from messy PRs and odd style.

Now

Every PR looks clean. Review catches less, and issues surface later.

WHAT WE DID: PROTECT REVIEWER ATTENTION

Layer 1	Deterministic checks. Non-negotiable.
Layer 2	AI reviewer comments.
Layer 3	Approval tiered by risk.

2.4 what we did Layer 3: tiered approval

Lower risk — Higher risk

Automated only

Checks and AI approve. No human.

Author is a SWE
< 25 LOC, 1 to 2 files

- Error message wording
- Logging statement change

Deferred review

A human reviews after.

< 100 LOC, < 5 files

- UI copy behind a feature flag
- New unit tests only

Standard review

A teammate approves.

Any size, or over 100 LOC

- New endpoint or feature
- Business logic change

Expert review

A domain expert approves.

The riskiest 3% of PRs

- Highest risk scores

Risk score from File incident history · File change traffic · PR size · and more

What these controls bought us

2024 vs 2026

2.4x
PRs

1.4x
Human reviews per reviewer

Tiered review absorbed the gap.

PRs failed for test slop

40% → <10%

January 2026 to September 2026.

Four things to try this week

01 **Block what must never break**
Rules advise. Hooks, CI and build rules block.

02 **Stop test slop early and late**
A hook while the agent writes. A grader in CI.

03 **Measure your own codebase first**
Find what your agents get wrong. Fix it with codemods.

04 **Tier review by risk**
Checks, AI review, then approval by risk.

Thank you

The New Tech Debt: How AI Is Changing the Way Our Systems Fail



Bhavana Hindupur

linkedin.com/in/bhavanahindupur