Lessons from Red Hat Engineering on AI-first software development

Jeremy Eder · Red Hat · June 2026

Jeremy Eder

Distinguished Engineer

Red Hat

BACKGROUND

- 16 years at Red Hat. Performance engineering, managed services, now AI engineering
- Developed reference design concepts the AI-SDLC pipeline leverages
- PyTorch Foundation board. Led llm-d bring-up program

Jeremy Eder

Distinguished Engineer

Red Hat

Lesson 1: Red Hat Agentic SDLC Overview

Agents running inside the actual SDLC we ship code from.

Lesson 2: Our first Issue-to-PR bot

Patterns are portable.

Lesson 3: Evaluating Agents, Creating Datasets

EvalHub, Harbor, MLflow

Lesson 4: Build the System that Builds the System

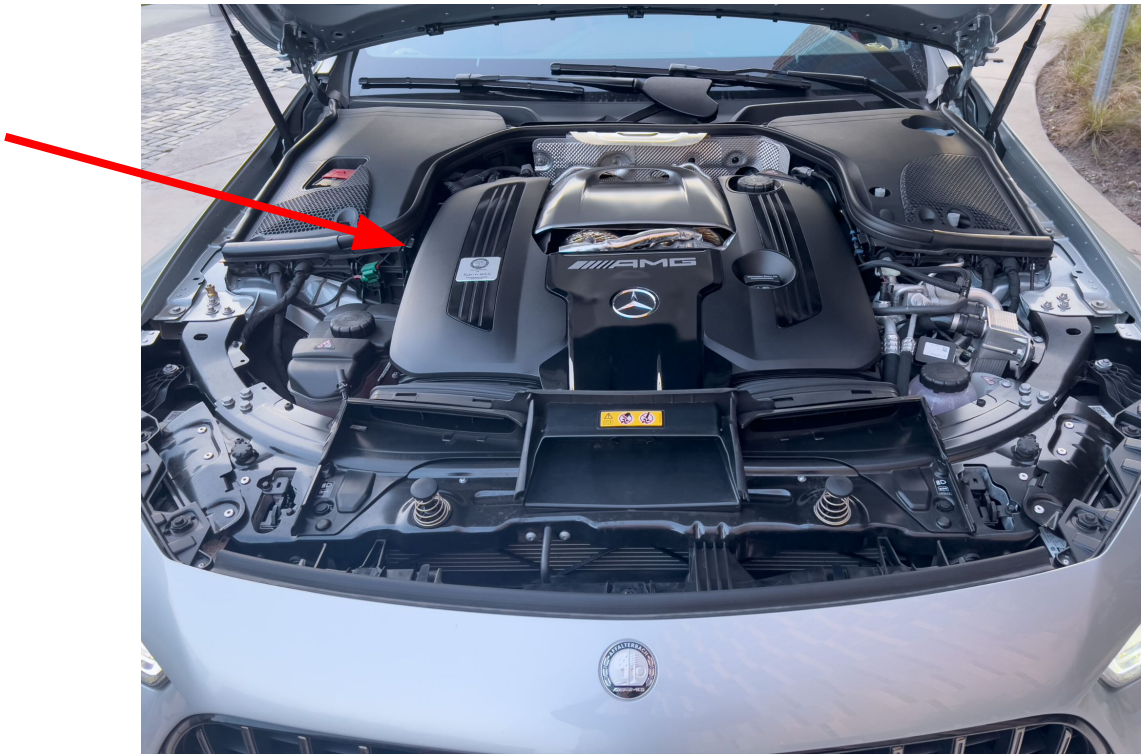
How ready are we?

LESSON 1

Red Hat Agentic SDLC Overview

CODE OWNERSHIP

AI can write code. It cannot own outcomes.



Signed-off-by

CODE OWNERSHIP

AI can write code. It cannot own outcomes.

AI changes who writes code — not who owns production systems. Every change still needs a human accountable for reliability, maintainability, security, and operational behavior.

NO LONGER

“Who wrote this code?”

THE QUESTION NOW

“Who is willing to own this code?”

Accountability cannot be delegated to a model.

CODE REVIEW

The bottleneck is no longer coding. It's judgment.

As code generation gets faster, review becomes the primary quality gate.

REVIEW NOW EVALUATES

- Correctness
- Architecture alignment
- Security implications
- Operational impact
- Long-term maintainability

WHERE JUDGMENT GOES

- AI-generated PRs look polished and complete — a false sense of confidence. Good reviews challenge assumptions and look for what the AI missed.
- Invest in review tooling, automated checks, and eval frameworks so humans focus on judgment, not mechanics.

PATCH SIZE

Who are we optimizing for?

We used to optimize patch size for human reviewers. In an AI-first world, are we optimizing for humans, agents, or both?

Large changes

Easier for an agent to generate — harder for humans to review, validate, and trust.

Small, focused changes

Easier to reason about, easier to roll back, easier to evaluate.

The goal isn't to maximize lines generated. It's to maximize confidence per change.

PROVING AI GUIDANCE

Trust the evidence, not the author.

A passing build is not enough.

AI CODE MUST BE

- Explainable
- Testable
- Reviewable
- Aligned with engineering standards

REQUIRE EVIDENCE

- Tests
- Benchmarks
- Design rationale
- Security validation
- Evaluation results

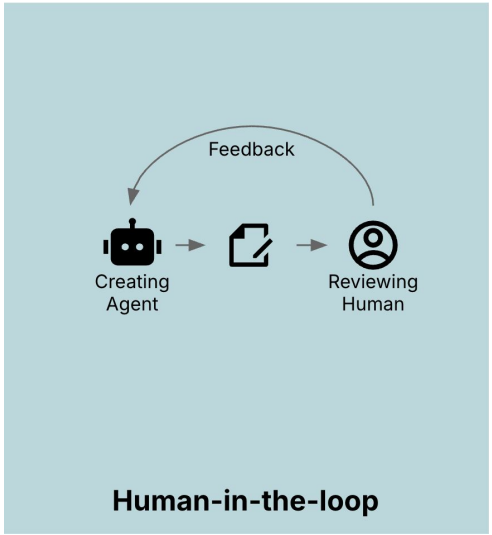
The trajectory that produced a change can matter as much as the diff itself.

Prove the change deserves to be trusted — not just that AI wrote it.

Different Team Topologies

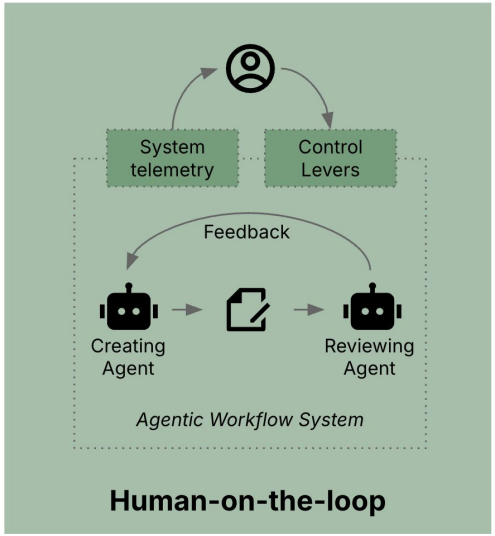
CO-PILOT PATTERN

We are the driver



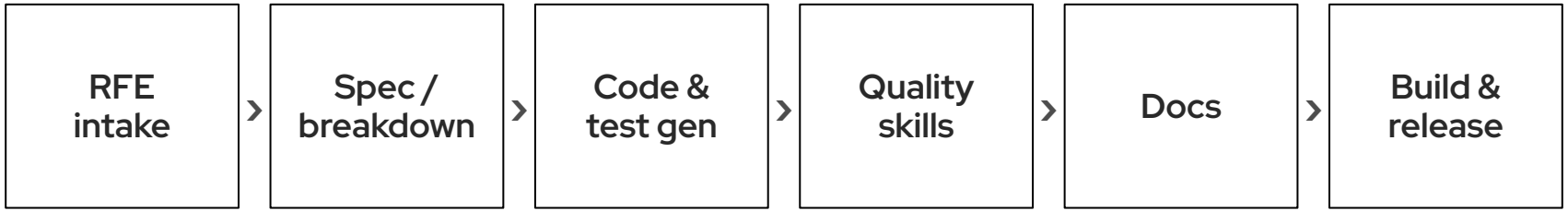
FACTORY PATTERN

We build cars now



We turned the SDLC into a squad of agents.

Each stage hands structured work to the next. Some stages run today and some are still being built. Each has a learning loop.



Live today: some stages are in use by PM and engineering. Others are still being built.

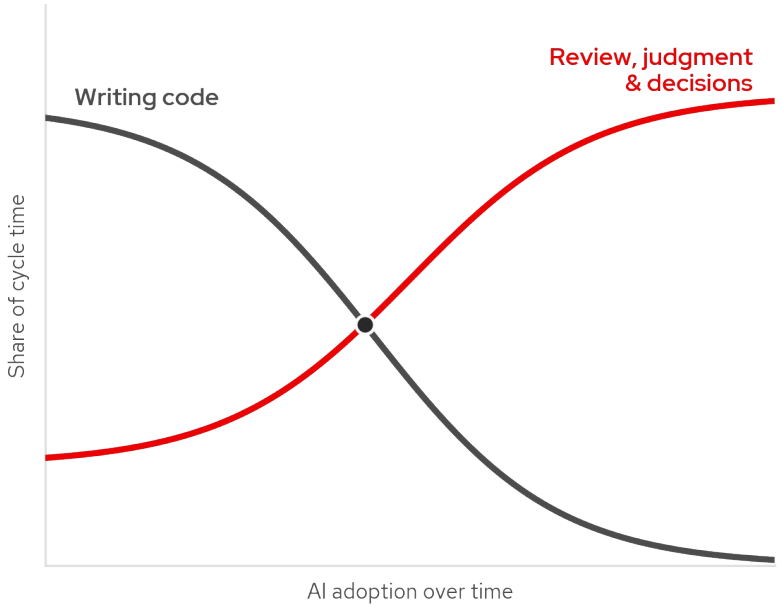
Coding to deciding.

The Shift in Focus

The bottleneck became reviews, and figuring out what can **auto-merge/mergify**.

Modern Workflow

We use the **CodeRabbit CLI** for tight-loop, local reviews — avoiding round trips to GitHub.



Illustrative — the shape, not measured shares.

LESSON 2

The first issue-to-PR bot

An autonomous bugfix agent, watched in the open.

13% → **43%**

PR merge rate, over ~1000 bugs

57%

still pushed back by review

- **78 Auto-fix issues filed**
- **50 PRs merged** (71% merge rate, 88% in the last 2 months)
- Median time from issue filed to merged PR is **~22 minutes**
- 53% completed with zero human intervention

How do we scale this? Evaluations

- Beginning to use Evals for codegen: EvalHub - OpenShift AI 3.4

LESSON 3

Evaluating Agents, Creating Datasets

EVALUATION FUNDAMENTALS

What is an eval?

A structured test that measures how well an AI system performs a specific

THREE INGREDIENTS

1 **Dataset**

Real tasks with known-good answers.

2 **Runner**

Execute the model against each task.

3 **Judge**

Score the output — automated, human, or LLM-as-judge.

Eval flow – datasets in, trajectories out

Evaluate codegen, code reviews, RAG pipelines, Reasoning, Tool use, Instruction following

rhai-datasets

THE TASKS

drawn from real work

›

agent-eval-harness

THE HARNESS

Harbor: runs + trajectories

›

EvalHub

ORCHESTRATION

on OpenShift AI

›

MLflow

THE LEDGER

analysis + tracking

The dataset is the definition of quality

Mine

CONNECTORS

GitHub PRs/issues + Jira tickets

Score

5 DIMENSIONS

Weighted: clarity, verifiability, difficulty, relevance, completeness

Detect

LLM-AS-JUDGE

Flags AI-generated content (optional)

Package

OUTPUT

Harbor · SkillsBench · MLflow

Bootstrapping an Eval dataset

DATASET BOOTSTRAP

Bootstrapping an eval dataset

1

Architecture context

What components exist, how they connect, what languages and frameworks they use. Without this, tasks reference code that doesn't exist or miss constraints that matter. This is what architecture-context provides.

2

Real engineering artifacts

Merged PRs, closed bugs, resolved incidents — work that humans actually did with known-good outcomes. These become your ground truth. Synthetic tasks test synthetic skills.

3

Scoring criteria

How will you judge the output? Automated tests, LLM-as-judge, human review, or all three. Define this before generating tasks — otherwise you build tasks you can't score.

Architecture Context - what's in there?

architecture-context is a single source of truth for what RHOAI is made of – structured markdown and JSON for each component, versioned per release, machine-readable by agents and humans alike.

57
Components documented

22
RHOAI versions (2.6 through 3.5)

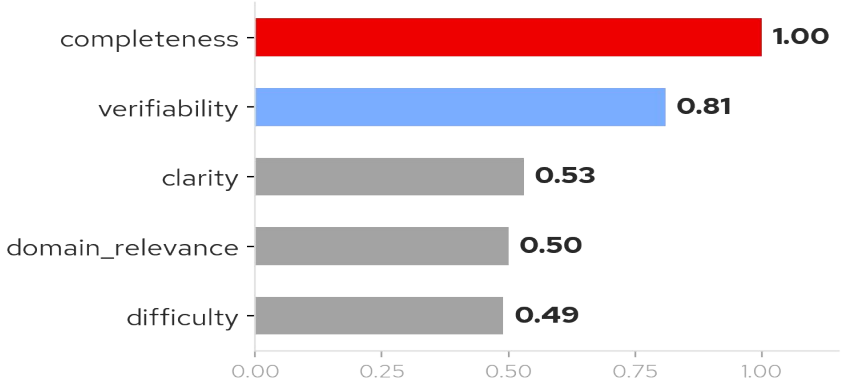
PER-COMPONENT CONTEXT

- Repo, branch, version, languages
- Purpose, architecture, sub-components
- Upstream + downstream dependencies
- API contracts, ports, CRDs

Candidate Selection

50	0.67	74%
Candidates scanned	Avg suitability (0.54 – 0.79)	Good quality (37 of 50 ≥ 0.6)

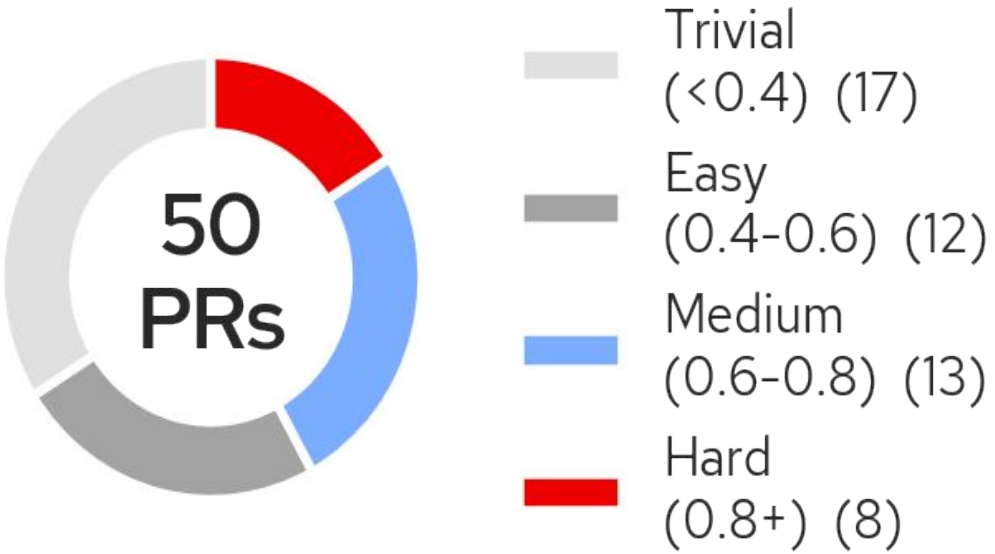
SCORING DIMENSIONS (AVG)



KEY FINDINGS

- Completeness is perfect (1.0) across all 50 candidates – every PR has enough context for evaluation.
- Difficulty and domain relevance are weakest (~0.5), dragged down by trivial chore PRs.
- All 50 candidates classified as agent_coding eval type.

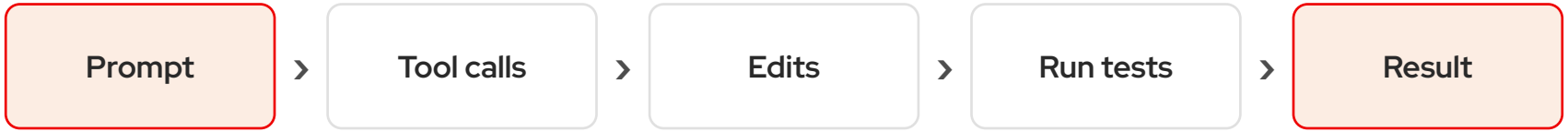
What the scanner found



Highest-scoring PRs for evaluation

Score	Difficulty	PR Title
0.79	0.63	test: add E2E tests for Tenant CR lifecycle
0.78	0.90	feat: mask Authorization header in logs
0.78	1.00	chore: promote stable to rhoai
0.78	1.00	chore: promote main to stable
0.76	0.91	feat: detect rogue AuthPolicies on MaaS auth surface

Earning Trust:
Score the trajectory of the agent loop



EVALUATED END-TO-END

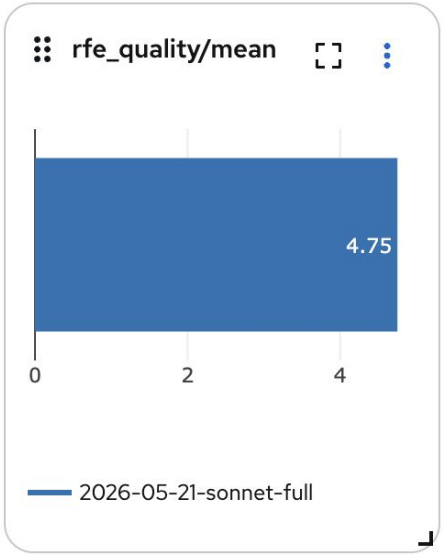
A passing diff that came from a reckless path is still a fail. The trajectory is the evidence + debug

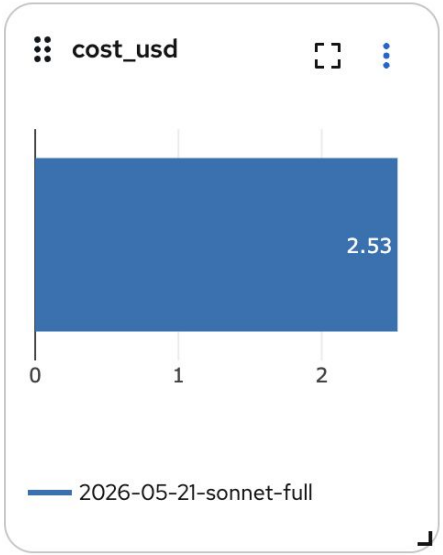
2026-05-21-sonnet-full

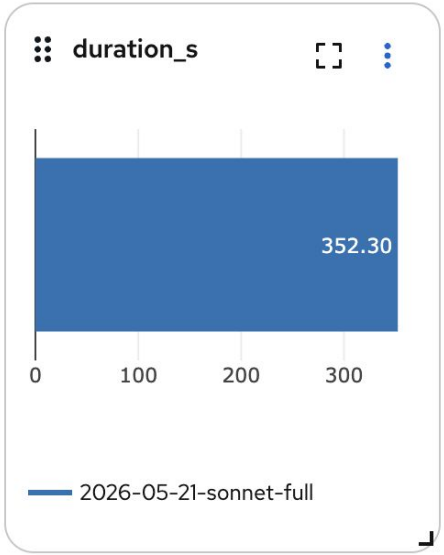
Overview | Model metrics | System metrics | Traces | Artifacts

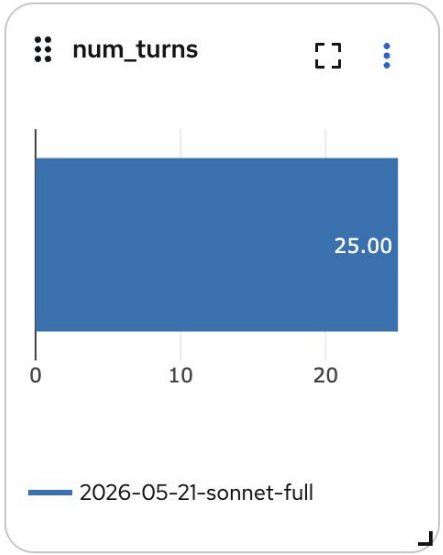
Search metric charts

rfe_quality (4)









Cost Fitting based on Evals

The same harness, scored case by case across three models.

CASE	OPUS	SONNET	HAIKU
task-0008	0.75	0.75	–
task-0010	0.75	0.75	0.75
task-0031	✓	✓	✓
task-0034	0.75	0.75	0.75

✓ perfect pass – no result

Evaluation-driven development with EvalHub

Stop guessing. Start measuring.

June 2, 2026 | William Caban Babilonia, Matteo Mortari

Related topics: AI inference, Artificial intelligence, Developer productivity

Related products: Red Hat AI, Red Hat OpenShift AI

Table of contents:

If you have shipped software, you probably know test-driven development (TDD). Write a failing test. Write the code to make it pass. Refactor. Ship with confidence. The red-green-refactor cycle is elegant because it is prominently deterministic in nature: the test either passes or it doesn't. Every state is unambiguous.

AI systems do not work that way.

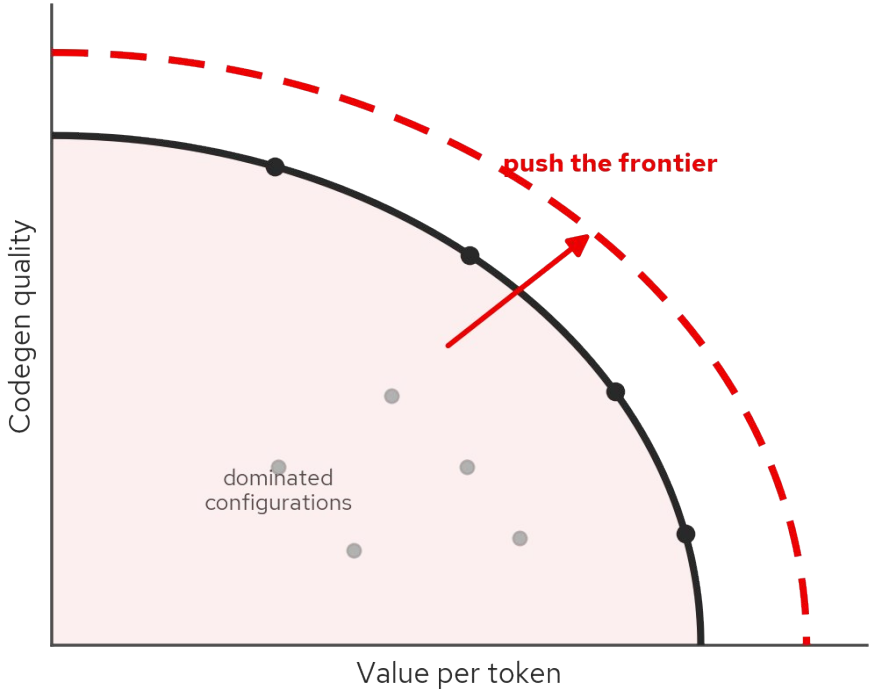
A language model answering a customer support question might give ten different responses to the same query across ten runs—all of them arguably correct, none of them identical. "Does it work?" is the wrong question. The right question is how often does it work, for whom, in what contexts, and how we know. Programmatic pass/fail can't answer that. Threshold-based, multidimensional, gradient scoring can.

https://developers.redhat.com/articles/2026/06/02/evaluation-driven-development-evalhub

Harness Engineering Teams

Goals - a new frontier

Codegen quality trades against cost. Plot it against value per token and the efficient set is a frontier.



Jobs the harness has to handle.

Setup & Ops

Orchestration, scale, shared knowledge. The platform itself.

SDLC Accelerators

Agents for spec, code, test, docs, release.

Governance

Guardrails, IP exposure, embargo, compliance, security.

FinOps

Value per token. Watch what the agents cost — cost-aware engineering as a practice.

Domain Accelerators

Vertical agents where the work isn't generic.

The harness is somebody's full-time job.

WHAT WE DID FIRST

Shared components maintained on the side, between feature work.

EVOLVING TO

A dedicated team that maintains the shared agentic factory as its charter.

What do we triage now?

1. Issue-to-PR bot session review
2. Tool call failures & skill misses (OTEL traces)
3. Session failure triage
4. Skill-creation opportunities
5. Proposed memory additions
6. CodeRabbit review calibration
7. Event/Log stream triage

Context Engineering

01

Make context explicit

Most interrupts are missing context. Write down what the agent keeps asking for — tracing.

02

Log every interrupt

You can't reduce what you don't count. Treat each interrupt as a defect in the system — MLflow.

03

Push context into the system

Encode it where the next agent finds it — architecture-context and style-guide repos.

Context Poisoning

Revisiting Team Norms

Approval gates from old outages

Handoff boundaries

Drawn where handoffs were expensive

Status meetings Evolve

Culture Shifts

Movements vs Mandates

Movements involve building trust, being transparent and consistency

Mandates get people moving and are sometimes useful

Are your repos “AgentReady”?

Assess git repos for AI-assisted development readiness

```
$ uv pip install agentready
$ agentready assess org/repo
```

- AgentReady scores your repo across 13 dimensions and tells you exactly what to fix — no guesswork.
- Agents need the same things good engineers need
  - clear docs
  - working tests
  - consistent structure
  - one-command setup path.

🏆 AgentReady Leaderboard

Community-driven rankings of agent-ready repositories.

🥇 Top 10 Repositories

#1 opendatahub-io/opendatahub-tests
Unknown Unknown
83.5 GOLD

#2 redhat/rhel-ai/wheels/builder
Unknown Unknown
78.6 GOLD

#3 ambient-code/agentready
Unknown Unknown
78.4 GOLD

#4 opendatahub-io/odh-dashboard
Unknown Unknown
76.5 GOLD

Leading the AI-First Transition

01 6-phase Red Hat Agentic SDLC

02 The Constraint is Now Judgment

03 Evals Define Trust

04 Are your repos ready?

Thank you!