System Verification Engineering Manifesto

Version 1.0 | 16 November 2026

Testfold and Carver Agents

The change we face

We are building increasingly complex software while knowing less about the code inside it.

Coding agents can produce implementations faster than engineers can understand their decisions. Multiple agents can change interacting components simultaneously. The resulting systems may themselves contain agents that learn, reason, use tools and act in changing environments.

We expect machine-generated change to grow. AI can produce changes faster than we can establish that they work. Verification is becoming the bottleneck: our capacity to check and correct the system must keep pace with our capacity to change it.

We need to build delivery around verification and engineer how we establish trust.

Code should earn trust through verification. Every change should be checked, and the system should remain under verification as it operates. The question is whether its behavior adheres to the goals we have set, within the conditions and limits we have defined.

Why a new discipline is needed

Much of this work already exists, distributed across organizations.

Developers check implementations. QA tests behavior. Support reproduces failures. DevOps and platform engineers maintain execution and delivery infrastructure. SRE teams investigate production behavior. Reinforcement learning teams design objectives, environments and rewards that shape agent behavior. Agent runtime teams build observability, evaluations and execution controls.

Each sees part of the system. Each uses different mechanisms to judge whether it is working. But the connection between those judgments is often weak. A production failure may never improve a training environment. A support reproduction may never become a protected regression test. A passing evaluator may conceal a gap in the goal it was supposed to represent.

We believe engineering organizations must reorganize around verification outcomes, with explicit roles and teams responsible for the verification and learning system.

The purpose of SVE

We propose System Verification Engineering.

SVE organizes engineering around continuously verifying system behavior, validating it against intended goals, and learning from observed outcomes. It brings specifications, checks, operational evidence and correction into one discipline, improving both the software and the mechanisms used to assess it.

This builds on earlier advances. Agile strengthened feedback around requirements. DevOps connected development and operation. SRE made reliability an explicit engineering responsibility. SVE extends these foundations to systems whose implementation and behavior increasingly exceed what people can inspect directly.

Learning from imperfect goals and checks

It also starts with a limitation we must acknowledge: our goals are incomplete, and our verification mechanisms are approximate.

Specifications omit intent. Tests simplify environments. Rewards can favor the wrong behavior. Evaluators can accept convincing failures. Observability reveals only what we have instrumented.

Real-world failures will expose these gaps. They must improve both the software and the machinery used to judge it.

The organization we foresee

We expect SVE to become a distinct engineering discipline, with practitioners progressing from SVE I and SVE II to senior and staff roles. Product-aligned teams, such as SVE Team — Loan Products, will build and operate verification across their domain. Shared SVE platform teams can supply reusable infrastructure.

Developers, QA engineers, SREs, support engineers and RL and runtime specialists provide foundations for these roles. Some responsibilities will reorganize into SVE teams. Each domain will need its own expertise and stack, with shared evidence and learning connecting the organization.

Our principles

1. **Trust must be earned through verification.** Authorship, plausible explanations and passing local tests are insufficient grounds for confidence. Require evidence appropriate to the behavior and consequences involved.
2. **Verify every change and keep verifying the system.** Apply checks throughout development and operation. Reassess when code, data, dependencies, models, permissions or goals change.
3. **Treat goals and checks as revisable engineering artifacts.** State their assumptions and limitations. Test whether evaluators detect meaningful failures and whether their criteria represent the outcomes we actually want.
4. **Build product-aligned SVE teams.** Organize verification around product outcomes. Teams combine relevant engineering capabilities and specialist stacks, connecting evidence and learning across development and operation.
5. **Turn failures into improvements at the pace of change.** Shorten the path from detection to reproduction, diagnosis, correction and regression verification. If changes arrive at PR speed, our ability to close the gaps they expose must keep pace.
6. **Give the whole loop explicit ownership.** SVE teams own the capability, integrity and capacity of verification and learning for their product domain. Product and domain owners remain accountable for intended behavior and acceptable risk.

Verify every change. Keep verifying the system. Learn from every outcome.

We invite practitioners to develop the methods, tools, career paths and teams that make this discipline work—and to judge it by the outcomes it improves.

Scope of verification

Continuous verification means risk-appropriate checks across development and operation. It does not imply exhaustive proof of every possible behavior. Goals and checks remain approximate, and observed outcomes must improve both.