ZTHR

ZTHR: Executive Summary

Zero Trust – High Reliability. A pattern language for software built by AI.

Russ Little · v1.0 · © 2026 Russ Little, CC BY 4.0 · DOI: [10.5281/zenodo.23826734]
(https://doi.org/10.5281/zenodo.23826734)
Executive summary · https://zthr.ai

The problem

AI has made writing code cheap. Deciding whether that code is right has not become cheaper. Most teams have bolted AI authors onto a process paced by human code review, and under real volume that process fails in one of two ways: reviewers become the bottleneck that holds delivery back, or they become a rubber stamp that approves what nobody had time to read. Replacing the human reviewer with another copy of the same model does not help. Models that share training data share blind spots.

Waiting for a model good enough to trust repeats the old assumption that quality lives in the author. A better author still needs evidence that a particular change is right, and that evidence has to come from somewhere the author does not control.

The answer in one sentence

Let AI write the code, and build a factory that cannot be fooled about whether the code works.

Two principles

Symmetric Zero Trust. No author, reviewer, model, operator or report gains authority by asserting success. A claim counts only on evidence the claimant does not control. The rule runs in every direction: toward the AI that writes the code, toward the AI running inside the product, toward the automated checks themselves, and toward the automation that operates the pipeline. A test that has never been shown to fail is not trusted to pass.

The Fixture Inversion. Quality moves out of the author's skill and into the environment the author works in. This is the move manufacturing made two centuries ago, when jigs, fixtures and go/no-go gauges let ordinary operators produce parts to tolerances no craftsman could hold. In software, the jigs are a governed foundation and a catalog of qualified parts that limit what can be built. The gauges are checks proven to fail on the defects they exist to catch. The datum, the reference everything is measured from, is user intent: what people came to do, written down where the author cannot change it.

How it works

The AI generates candidates. The fixtures judge them. A rejected candidate goes back for another attempt, as many times as it takes; trying again is cheap, and shipping a defect is not. Only candidates that pass every fixture are admitted, deployed and verified as actually serving.

Every change is reviewed by at least three models from different families before commit: one given the context, one given only the diff, and one checking the code independently. A fourth reviews it in place before full promotion. The strongest checks read the system in different forms (the source, the database, the built artifact, the rendered screen and the running service) because readers of different representations cannot all share the same blind spot.

When a defect slips through, the fix is not only to the code. A new fixture is added so that kind of defect can never pass again. Mutation testing, nightly attempts to break the system, calibrated twins of external services, and re-review by each new generation of models keep finding what nobody thought to test. The factory gets stricter every week, and so does everything it builds.

What changes for people

People stop reading diffs and start doing the work that compounds: defining intent with the users who hold it, building and governing the fixtures, and judging consequential risk. Deterministic changes that can be audited and reverted in one step proceed on their own. Destructive or multi-step changes, any change to the AI itself, and any agent request for wider permissions require a person's sign-off.

Quality and scale are not traded off. Each fixture replaces a person who would otherwise have to watch for that failure, so one builder or a small team can operate at a level conventional teams reach only with far more people.

How to judge it

The headline measure is intent completion: how often the software lets people do what they came to do. It measures what goes right, not only what goes wrong. The comparison that matters is against how the work is done today, on equal terms and normalized for volume. AI will never be perfect, and it does not need to be; the question is whether it is better than the current process, and the fixtures make that measurable.

The honest limits

The largest risk is that the system builds what it is told. A bad specification is enforced as faithfully as a good one, so ZTHR reviews specifications against the existing parts catalog and from first principles before any code is generated. Fixtured software is as correct as its fixtures are faithful. The claim is not perfection; it is that fidelity is visible, measurable and rising.

Status

ZTHR is in production use for the development of greenfield systems in a regulated industry, where it has proven highly effective. Its thirty-three patterns describe how and why rather than a toolset, so they carry over to most software engineering work.

Read further: the five-page brief covers each principle, the six pattern families and governance in more depth. The full paper sets out all thirty-three patterns with their forces and consequences, and is published openly at zthr.ai.