

How we change systems without breaking them

Our discipline for changing, building, and fixing production systems — and how the same method applies to issues in your own environment.

01 — THE PRINCIPLE

Evidence before optimism

Most teams change production on hope: someone has an idea, it looks right, it ships, everyone waits. We run the opposite way. **Nothing changes production until evidence says it should, and nothing we've already disproven gets re-argued.** Every change earns its place or is recorded as a dead end — so the same ground is never covered twice.

02 — THE CHANGE LIFECYCLE

Seven gates. A change clears all of them or it doesn't ship.

- 1 Suggest.** The idea is written as a specific, testable claim — not "make it better," but "changing X should produce more of Y."
- 2 Check history.** Has this been tried and disproven before? We keep a living registry of dead ends, so we never re-run a failed experiment.
- 3 Establish the control.** We reproduce current behavior to the decimal first. We don't measure a change against a ruler we haven't calibrated.
- 4 Test one variable.** Change one thing, across multiple conditions — so when the result moves, we know exactly what moved it.
- 5 Validate.** Promote, reject, or shelve. It has to beat the control where it matters — not just on a lucky average.
- 6 Review.** Anything touching sensitive paths is reviewed against the actual change before it goes out.
- 7 Promote or reject — and record it either way.** It ships through a single verified deploy, or the dead end is logged. Knowledge is captured no matter the outcome.

THE AUTOMATED SAFETY NET

A separate, machine-run regression check proves the change didn't break anything else — on every change, before it can reach production, and it fails closed. The experiment proves the new thing helps; regression proves the rest is untouched. A change passes both.

EVIDENCE BEFORE ASSERTION

"Done," "deployed," and "fixed" are claims backed by real output — never stated from memory. This single habit keeps the other gates honest.

The loop

When we're building rather than tweaking, the same DNA is structured as a loop. The rule that defines it: **never build the next slice until the current one has run on real data and been observed.**



Automations fail on the inputs you couldn't reason about up front. So we build the smallest useful slice, run it on real data, watch what it actually did, and only then move on — the client sees working slices early, and there's never a "we built it all and now nothing works" moment.

Built to change our systems safely. It diagnoses yours the same way.

An issue is reported	Framed as a specific, reproducible symptom — not "it's flaky sometimes."
Before we touch anything	We check history — has this been seen or fixed before? No diagnosing from a cold start.
To confirm it's real	We reproduce it and capture what "working" and "broken" actually look like.
To find the cause	One variable at a time — systematic, no shotgun fixes.
Before it ships	Reviewed, and regression-checked so nothing else in your system breaks.
After it ships	Verified live, and the outcome is recorded so it never has to be re-investigated.

- **No guessing in production.** The fix is proven against a reproduced failure first.
- **No repeat incidents.** Outcomes are recorded, not re-solved from zero.
- **Your existing functionality is protected.** Every change passes an automated regression check.
- **No "works on my machine."** Validated across the conditions that matter.

We don't ship on optimism and we don't diagnose from memory. Whether we're changing a system, building a new one slice by slice, or fixing something in your environment — we reproduce reality first, move one variable at a time, prove it across the conditions that matter, and record the outcome. So the same ground is never covered twice.