



## CASE STUDY

# The Safeguard That Passed Every Test and Failed Open

*A probabilistic safety control passed its full test suite, then produced a crisis intervention on one run and an ordinary answer on the next. Same input. Opposite outputs.*

## CONTEXT

A safety control governs a system that handles free-text inputs across a wide risk spectrum — ordinary questions at one end, high-stakes content requiring immediate intervention at the other. The control's job: classify each input, flag the high-risk tier, and route accordingly. A green test suite. Deployed. Running in production.

## THE TRIGGER

A user filed a report: this worked two days ago. Same input produced a crisis intervention on one run, an ordinary answer on the next. Reported as a regression. The report was specific. The output delta was stark. That combination — persistent, specific, not random noise — is the kind of signal that deserves a real investigation, not a rerun hoping it resolves.

## INVESTIGATION DISCIPLINE

The first verdict came from a screenshot. The second from a git log scoped to a single file — used to defend a claim about a change that touched four. Neither conclusion had been measured. Both were inference wearing a verdict's uniform. The third pass ran the exact input through every safety check at both code versions and read the failure-handling path line by line. That one held.

A safety verdict requires running the actual input through the actual code. Screenshots and scoped git logs are hypothesis generators, not measurement.

## ROOT CAUSE

The high-risk input had no deterministic coverage. Its only net was a probabilistic classifier — a coin flip on ambiguous phrasing, catching the same input on some runs and missing it on others. And on any error, any non-OK response, any timeout past two seconds: the control returned "no finding" and let the input through. The failure branch was commented // FAIL SAFE. It was the opposite. It failed open.

There was no regression. Nothing had changed. The two contradictory outputs were the control working exactly as built: a coin flip in fair weather, absent entirely in foul. A green test suite proved the control was configured — never that it was reachable when its dependency failed.

## RESOLUTION



Move the highest-risk input class onto deterministic pattern checks that fire before any network call — turn the coin flip into a guarantee. Demote the classifier to defense-in-depth. Make failure polarity a named decision: fail open, fail closed, or fail to a safe default with explicit rationale. A mislabeled branch nobody read is not a decision — it is a gap wearing the uniform of one.

## OUTCOME

---

### RESULT

#### **Identified a fail-open control with no backstop on critical paths.**

Nondeterminism eliminated at the highest-risk input tier. Deterministic checks fire first, before any external dependency. Classifier retained as secondary defense. Failure polarity documented and signed.

---

## TRANSFERABLE PRINCIPLES

---

### 01

A green test suite proves configuration, not behavior under failure. Test the path where the dependency dies. Happy-path coverage on a safety control is not safety coverage.

---

### 02

Measure, don't infer. On a safety verdict, run the actual input through the actual code, and scope the conclusion to the whole change — not the file you happened to read.

---

### 03

Treat a persistent, specific anomaly report as signal. The user who said "this worked two days ago" was wrong about the cause and right about everything that mattered.

---

Blue Jacket Consultancy

bluejacket.io · joseph@bluejacket.io · (551) 277-5775