# Pattern Elimination System

Facilitation guide · team, one facilitator

**What this is:** a short, facilitated lab you run when the same kind of failure keeps coming back. Instead of fixing incidents one at a time, the team lays several recent incidents side by side, extracts the one pattern beneath them, names it, and installs a prevention so the whole category ends. The mechanism is three beats — Tally, Extract, Lock — and the capture sheet carries them. It takes about 45–60 minutes once the incidents are gathered. Bring the incidents before the session — don't spend meeting time searching for them.

**The one line to hold:** your job isn't to fix the latest incident — it's to make the team see the pattern under all of them. Ten incidents that look different are usually one recurring incident. **You don't eliminate recurring failures by fixing incidents. You eliminate them by finding the pattern.**

· WHEN DO I USE THIS?

## Use this when

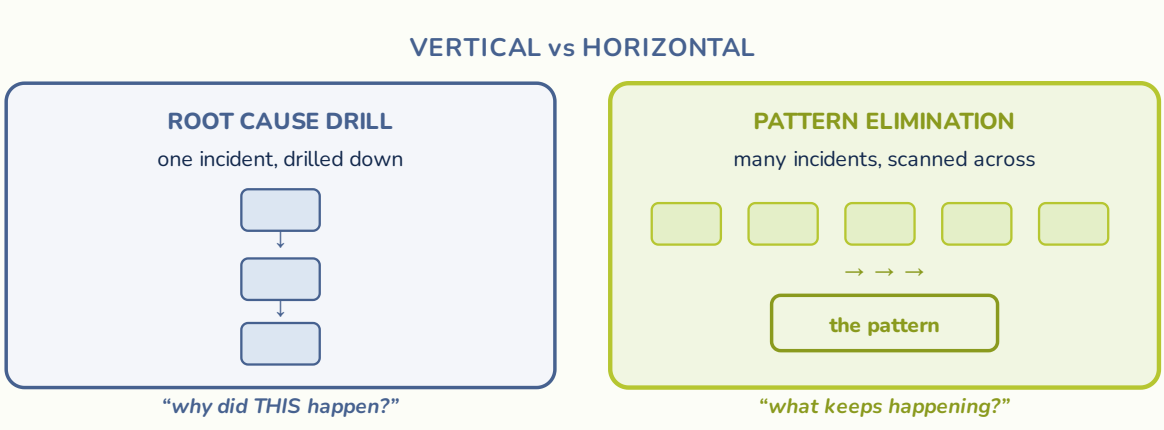
**GOOD FIT**

The same kind of failure keeps returning in different forms; your retros generate good actions but the recurrence never stops; you're firefighting the same category month after month; you sense a connection between "unrelated" incidents but haven't pinned it down. You need at least three related incidents to run it.

**WRONG TOOL**

You need to understand why *one* incident happened › Root Cause Drill. You're trying to prevent a *future* plan from failing › It Already Failed. You already know the pattern and just need to install one prevention › If–Then Lock.

**The boundary that matters most:** this is not Root Cause Drill. That tool goes *vertical* — one incident, drilled to its root. This goes *horizontal* — many incidents, scanned across for what repeats. Drill one problem; scan for the pattern behind many.




© Witmer

## PART 1 RUNNING THE LAB

team, one facilitator

1 · WHAT IS THIS REALLY DOING?

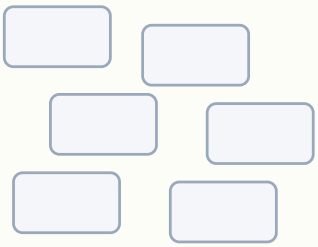
# The one idea

**THE ONE IDEA**

Your retrospectives feel useful — good discussion, lots of actions — and then the same kind of incident happens again six weeks later. The reason is that you fix incidents one at a time, and treated independently each incident looks unique. That isolation is exactly what hides the pattern: nobody lines the incidents up, so no one sees that ten "different" problems are one condition appearing in different ways. This lab lays them side by side, forces out the shared condition, names it, and locks it — so the category ends instead of reloading.

### TEN "DIFFERENT" INCIDENTS, ONE PATTERN

**TREATED ONE AT A TIME**



*each looks unique — pattern invisible*

→

**SEEN TOGETHER**

**ONE NAMED PATTERN**

"we have a Hidden Dependency pattern"

*one condition, many forms*

© Witmer

2 · WHAT DO I NEED?

## What you'll need

| What | Why it's there |
|---|---|
| **At least three related incidents (core)** | You can't extract a pattern from one or two — that's a coincidence. Bring the last several incidents of the kind that keeps recurring; three is the minimum, five or more is better. |
| **The capture sheet (core)** | It carries the whole sequence — the incident tally, the extracted pattern name, the If–Then lock, and the monitoring check — so the pattern becomes a record, not a whiteboard that gets wiped. |
| **A facilitator** | Lays out the incidents, forces the extraction beat instead of jumping to a fix, holds the "three incidents minimum" rule, and makes sure you leave with an installed prevention. |
| **The team closest to the incidents** | The people who lived the incidents can see the shared condition fastest — and will own the prevention that comes out of it. |
| **Someone who can validate the pattern** | Someone willing to check that the proposed pattern actually explains *each* incident — otherwise teams convince themselves too quickly. |

Resilience · Learn and Improve · about 45–60 minutes once the incidents are gathered, depending on how many you're scanning and how quickly the pattern comes into focus.

3 · HOW DO I RUN IT?

## The run-sheet

**Gather the incidents first — at least three of the kind that keeps recurring.** Then run the three beats. The whole lab turns on one discipline: force the extraction. Don't let the room jump from "here are the incidents" straight to "here's the fix."

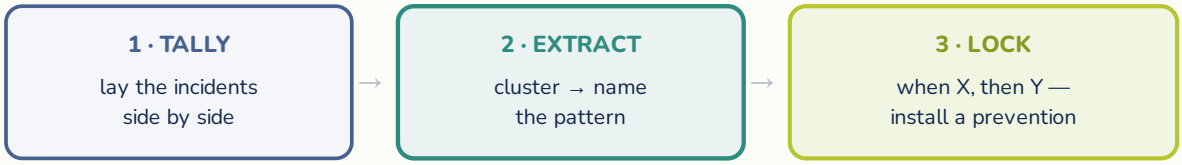
> **Extract is the hard beat — this is the whole tool.** Everything else is setup and follow-through; the value lives in forcing the abstraction. Before you start extracting, run one diagnostic: *if you fixed every one of these incidents individually, would the same type of incident still happen again?* If yes — you're dealing with a pattern, and it's worth finding.

| | | |
|---|---|---|
| **1. Tally the incidents** | **10 min** | *Lay out the last several incidents — at least three — side by side on the capture sheet, one row per incident.* Enough detail that the team can compare them, not so much that you drown in it. |
| **2. Cluster the similar ones** | **10 min** | *Group the incidents that feel like they share something.* Don't name it yet — just get the ones that rhyme sitting together. Some incidents won't cluster; that's fine. |

© Witmer

| | | |
|---|---|---|
| **3. Extract the common condition** | **10 min** | *For each cluster, ask what single condition sits beneath all of them:* "we're looking for one condition that sits under all of these — what do they share? Not 'various causes' — one thing they have in common." Keep pushing until one condition explains the cluster. |
| **4. Name the pattern** | **5 min** | *Write it as a sentence: "we have a _____ pattern."* Only name it if it shows up in at least three incidents, then check the name holds against every incident in the cluster. A specific name — "Late Requirement Drift" — is what the team recognises months later. |
| **5. Install the If–Then lock** | **7 min** | *Bind a prevention to a trigger: "when X happens, then Y."* Make it concrete enough that it fires on its own, without someone remembering to look. This is the step that makes it elimination, not discovery. |
| **6. Set the monitoring check** | **3 min** | *Decide which next incidents will tell you it worked.* If the next three of this kind don't recur, the pattern's eliminated; if they do, the name or trigger was wrong — come back and revisit it. |

**TALLY → EXTRACT → LOCK**




**EXTRACT is the hard beat — force the abstraction, don't skip to the fix.**

*Naming is discovery. Installing the lock is elimination.*

**FACILITATOR TIP — FORCE THE EXTRACTION**

The room will want to skip beat two. Someone will see the incidents laid out, spot the fix for the most recent one, and push to "just sort that." That's the firefighting reflex — and it's exactly what has kept the pattern alive. Your one job is to hold the team in the extraction: cluster, find the common condition, name it. Don't let anyone propose a prevention until the pattern is named. If someone jumps to a fix, ask: "what pattern does that fix?" — it immediately forces the abstraction back open. If the team can't find a condition shared by three incidents, you don't have a pattern yet — and that's a finding, not a failure.

© Witmer

PART 2 **GETTING IT RIGHT** why it works & common mistakes

4 · WHY DOES THIS WORK?

## Why it works

Firefighting feels productive — you're rewarded for fixing today's incident fast — so prevention always feels less urgent, and each fix stays isolated. But isolation is exactly what hides the pattern: you can't aggregate what you never lay side by side. Laying the incidents out forces the shared condition to surface; naming it makes it something the team can recognise and reuse; and binding a prevention to a trigger turns "we learned something" into "we installed something." The category ends because you stopped treating its symptoms as separate problems.

5 · WHAT IF IT GOES WRONG?

## Common mistakes

| What happened | The fix |
|---|---|
| **You named a pattern from two incidents** | Two points make a line, not a pattern. Require at least three independent incidents — five is better — or you'll name coincidences and chase ghosts. |
| **The room jumped straight to fixing** | The firefighting reflex. Hold them in the extraction: no prevention proposed until the pattern is clustered, named and validated. |
| **The pattern explained only some incidents** | If the proposed pattern doesn't explain *every* incident in the cluster, it isn't the right pattern yet. Re-extract until one condition covers them all. |
| **The pattern name was vague** | "Recurring issue" eliminates nothing. Push for a specific, reusable name the team will recognise months later — "Silent Ownership Gap," not "process problem." |
| **You ended on a discussion, not a lock** | "We learned something" isn't the output. Bind a prevention to a concrete trigger — "when X, then Y" — or you ran Pattern Discovery, not Pattern Elimination. |
| **The lock relied on someone remembering** | A prevention that needs a human to remember will fail the same way the incidents did. Make the trigger fire on its own — a check, a gate, a default. |
| **You never checked whether it worked** | Without monitoring, you don't know if the pattern's gone. Watch the next three similar incidents: none recur, it's eliminated; they recur, the name or trigger was wrong. |

© Witmer

**Before you close the lab**

- ✓ The pattern was drawn from at least three independent incidents
- ✓ It's named as a reusable sentence — "we have a _____ pattern"
- ✓ A prevention is bound to a concrete trigger — "when X, then Y"
- ✓ You agreed which next incidents will confirm the pattern is gone
- ✓ Everyone can explain the pattern in one sentence

6 · HOW DO I MAKE THIS STICK?

## Take it back to work

THE TEST

After the next three similar incidents, ask:

**"Has the _____ pattern happened again?"**

If the answer is no, you eliminated a category — not just closed a ticket. If it's yes, the lock didn't hold: the name or the trigger was wrong, so come back and re-extract. Either way, the pattern name is now shared vocabulary the team will reach for again.

## Appendix · Reference

You should not need this to start. Read it before, or if a run goes sideways.

7 · A WORKED RUN

## Watch a pattern get named

A team brought five incidents from the last quarter that felt unrelated: two outages, a missed launch, a data bug, and a failed deploy.

| Stage | What happened |
|---|---|
| Tally | Five incidents laid side by side on the sheet, each with its trigger and what broke. |
| Cluster | Four of the five shared something: each involved a service or dataset that no single person owned. The fifth didn't cluster — left aside. |
| Extract & name | The common condition: work crossing an unowned boundary. Named in a sentence: "we have a Silent Ownership Gap pattern." Checked against all four — it held. |
| Lock | If–Then: "when a service or dataset has no named owner, then it can't go to production." A gate that fires on its own. |
| Monitor | Agreed: watch the next three ownership-boundary incidents. A quarter later, none had recurred — the category was gone. |

© Witmer

8 · WHERE DOES THIS FIT?

## Where this sits in the Gym

| The real need | Use this |
|---|---|
| Eliminate a failure that keeps recurring across many incidents | **This tool — Pattern Elimination System.** Scan horizontally, name the pattern, install a lock. |
| Understand why **one** incident happened | **Root Cause Drill** — drills *vertically* into a single incident. This scans *horizontally* across many. The sharpest boundary. |
| Install one prevention for a problem you already understand | **If–Then Lock** — answers “how do I stop this one thing?” This answers “what keeps happening?” first, then uses a lock. |
| Prevent a *future* plan from failing | **It Already Failed** — imagines future surprises. This eliminates recurring history. |

**The move underneath:** stop treating repeated symptoms as separate problems — aggregate them until the shared condition is undeniable, then make that condition impossible. That same move — see across, name the pattern, install the prevention — is one of the Gym’s most compounding, because every category you eliminate is one you never firefight again.

© Witmer