IBM · EARLY IN MY CAREER

I thought I was doing maintenance.

We kicked off a heavy database index build during a maintenance window. CPU spiked. IO saturated. The system went down. For hours, real people could not complete transactions.

> ***The incident did not start when the alert fired.***
> ***It started when we assumed "maintenance window" meant "safe."***

Lessons from 100 P0 Incidents

100

P0 INCIDENTS

20 years · 5 Companies

20 years across telecom, retail, ecommerce, and AI platforms. What the failures taught me about design, observability, response, and leadership.

Dileep Kumar Pandiya • Principal Engineer · ZoomInfo

CORE THESIS

Every P0 has two timelines .

TIMELINE 1 · HOW THE SYSTEM FAILED

- A design choice made months earlier
- An alert threshold set too high
- A "routine" change on a hot path

TIMELINE 2 · HOW THE ORG RESPONDED

- Did we detect it fast enough?
- Was there one incident commander?
- Did leadership reduce confusion or add to it?

Technically similar incidents end very differently. The difference is almost always decided before the pager fires.

MY JOURNEY

Different companies, different domains.
Same four forces, different stacks.

IBM · Infosys

Telecom. Vodafone. National scale system architecture. Core incident response discipline learned early.

Walmart

Global retail scale under extreme transaction load. Complex microservices migration path under pressure.

State of MA

State government. Secure migration to public cloud. Zero downtime limits on critical public systems.

Wayfair

High volume ecommerce scale. Reduced production outage rate over 50% via automated pipelines.

ZoomInfo

AI-native platforms processing millions of events. 99.9% availability targets on distributed system.

Severity is decided at design time.
Recovery speed is decided before deplo y.

🔗 Tight coupling without fallback

⏪ Rollback that has never been tested

⚙️ Unsafe operational defaults

🐒 Game days and chaos testing

THEME 2 · OBSERVABILITY

Green dashboards can still mean customer pain .

Common engineering gaps repeatedly encountered across tech systems:

🔭 **Monitoring what you own, not what you call**

📊 **Technical metrics with zero business signal**

🔔 **Static thresholds that alert after performance degradation.**

THEME 2 · OBSERVABILITY

More dashboards is not more visibility.

Less ambiguity is.

- Track Business-Level Success Rates Daily
- Alert on Behavior Deviations, Not Just Thresholds
- Promote External Dependencies to Top Status

THEME 3 · RESPONSE

Most POs get worse because the team is overloaded , not because the system is incomprehensible .

Cognitive load is the silent second incident.

Name context in one sentence

Ask 'what changed?' first

⏪ Rollback over deep debugging

Keep execution groups small

THEME 3 · RESPONSE (OWNERSHIP)

When ownership is unclear, everyone responds and nobody decides.

What unclear ownership looks like

- Two teams both assume the other is handling it Alert fires, nobody knows whose runbook to follow
- 12 people in the war room, no decision made
- Post-incident blame because accountability was never defined

What actually works

- Every service has one named owner before anything breaks
- Ownership boundaries defined in architecture review
- On-call rotation tied to ownership, not availability
- Cross-team dependencies documented and agreed explicitly

Ownership is a system design decision. Define it in the architecture, not in the middle of a P0.

THEME 3 · RESPONSE (LEADERSHIP)

Leadership during a P0 is not motivational. It is operational.

- Create clarity, not energy
- Absorb blame upward
- 🤫 Updates: status, decision, next
- Protect decision bandwidth

THEME 3 · RESPONSE (BURNOUT)

The incident ends.
The cost to the team does not.

Hero culture and repeated P0s erode teams quietly. Burnout is a lagging indicator of poor incident practice.

Patterns that create burnout

- Same engineers pulled into every P0, no rotation
- No debrief after high-stress incidentsOn-call fires every night with no follow-through fixes
- Heroism celebrated instead of systemic fixes rewarded

What good leaders do differently

- Rotate incident commander deliberately
- Acknowledge stress explicitly after a hard P0
- Track on-call pain as a reliability metric
- Reward the engineer who prevented the incident, not just who fixed it

If your best engineers are the ones most likely to burn out, your incident process is consuming its own foundation.

THEME 4 · LEARNING

Most postmortems produce a document, not a change.

Too much narrative, too little actionable repair

Root cause analysis stops at the trigger

The same failure patterns return later

THEME 4 · LEARNING

A postmortem is only complete when the system is harder to fail the same way again .

Everything else is a memory aid.

Named owners and sprint slots for all corrections

Ensure at least one structural change

Track repair actions in central execution systems

WHY THIS MATTERS MORE NOW

AI changes the tools. Not the fundamentals.

The 4 forces don't disappear in AI systems, they become harder to see.

What AI genuinely improves

- Anomaly detection from learned baselines
- Alert correlation: 50 alerts into 1 incident context
- Faster triage through automated context assembly

New failure surfaces AI introduces

- Silent model drift before any alert fires
- Hidden retrieval dependencies that break without warning
- Soft failures that are harder to detect and debug

WHAT TO DO FIRST

5 things to change before the next P0

01 Fast Rollbacks over Debugging

02 Add Clear Business Metrics

03 Draft Your Commander Runbook Now

04 Manage Operational Shifts Like Deploys

05 Bind All Actions to Active Sprint Slots

CLOSING

That team at IBM recovered. But for months after, every time I touched a production system, I asked myself, if this breaks right now, how fast will we know? Who owns it? Can we roll back?

Those questions felt anxious then. After 100 incidents, they feel like engineering discipline.

The goal was never to stop incidents from happening.

It was to become the kind of team that failures cannot surprise.