IMR by Xurrent™ Incident Management & Response

The Major Incident Portmortem Playbook

Stop Fixing the Same Incidents Twice

FOR SRE & DEVOPS TEAMS | FREE GUIDE

The incident is resolved.

The Slack channel finally went quiet at 4:17am.

Someone typed “all clear.”
You closed your laptop.
And for about eleven minutes, you felt okay.

Then the calendar invite landed.

Post-Incident Review — Thursday, 10am.

You already know how this goes.

Someone pulls up a half-finished doc.
The same three people talk. Someone gets defensive.
Action items get written down and never opened again.
And three months later — same incident, different Tuesday.

It doesn't have to be this way.

The IMR platform already builds the timeline.
It drafted the doc. It reconstructed every event.

But automation can't run the room.
It can't ask the question that changes everything.
It can't turn "whose fault was this?" into "what in our system let this happen?"

That part is still yours. This guide is for that part.

6 stages. 7 mistakes. One template that actually works. Let's go.

The Stakes + Roles

The Postmortem That Actually Matters

Not every incident needs a full postmortem.
A P2 that one team resolved in under an hour? A quick async doc is fine.

But a P0 that woke up your VP, hit thousands of users, and showed up in a customer email?

That one deserves more than a template and a rushed hour on Thursday.

Major incidents are complicated. More people were involved. More decisions were made under pressure. More things went wrong in sequence — and usually for reasons that aren't obvious until you slow all the way down.

A bad major incident postmortem doesn't just waste everyone's time. It makes the next one more likely.

$5,600/min
average cost of production downtime

32% of customers churn after one major outage

3 months
before most teams hit the same incident again

Know Your Roles Before the Room Fills Up

The worst postmortems start with nobody knowing who's driving.

Incident Commander

- Owned the incident.
- Owns the postmortem.
- Sets the tone.
- Often the author.

Communications Lead

- Owned stakeholder updates during the incident.
- Speaks to the customer impact timeline.

Operations Lead

- Coordinated responders in the war room.
- Walks through the technical sequence.

Subject Matter Expert

- Called in mid-incident for their domain.
- Explains decisions made under pressure.

Scribe

- Documented in real time.
- The timeline source of truth.

On leadership in the room:

Managers can attend. But if they're driving the conversation, blameless culture collapses in real time.
Their job is to listen, support, and unblock. Nothing more.

The 6 Stages

01 **Secure the Timeline** — within 24 hours

02 **Schedule the Meeting** — 48–72 hours out

03 **Run the Blameless Walkthrough**

04 **Find the Contributing Factors**

05 **Write Action Items That Actually Close**

06 **Distribute and Archive**

Secure the Timeline

You have 24 hours. After that, the picture starts to blur.

Slack messages get buried. People move to the next fire. Within 24 hours: pull your incident timeline export, grab the deployment logs, monitoring snapshots, git commits. Write down every person who made a decision or took an action.

What most teams do instead

Wait until the postmortem meeting to build the timeline and then spend half the session arguing about what actually happened.

Schedule the Meeting

48–72 hours. The window matters more than you think.

Too soon: people are still running on adrenaline and not ready to reflect honestly.

Too late: the urgency fades, the details go with it, and half the room cancels.

60–90 minutes for most incidents. Two hours for complex multi-team events. And if at all possible, **the facilitator should be someone who wasn't the primary responder.** They ask better questions.

What most teams do instead

The incident commander facilitates their own postmortem. Unconsciously, they protect their decisions. The real learning gets filtered out.

03 Run a Blameless Walkthrough

This is the whole game. Everything else is setup.

The goal isn't to find out who made a mistake.

It's to understand why every decision made sense at the time it was made — given what that person knew, the tools they had, and the pressure they were under. That's a completely different question. And it leads to completely different answers.

The problem is that people don't naturally do this. The instinct in the room is to find the moment someone got it wrong and stop there. Good facilitation is what interrupts that instinct.

Questions that change the room

☑ ***"Walk us through what you were seeing at that moment."***
Grounds discussion in perception, not outcome.

☑ ***"What information would have led to a different call?"***
Separates the decision from the result.

☑ ***"Had you seen this behavior from this system before?"***
Surfaces the knowledge gaps nobody knew existed.

☑ ***"What did monitoring show you vs. what was actually happening?"*** Finds the observability holes.

The question isn't who broke it.
It's ***what in our system made it breakable***.

04 Find Contributing Factors.

...Not a root cause.

Most major incidents don't have one root cause. They have a chain — technical, process, human — that lined up badly at the wrong moment.

The language shift matters more than it sounds. "Root cause" points at one thing. One person. One decision. "Contributing factors" points at the system — which is almost always where the real answer lives.

☒ Blame

"Jake didn't check the deploy before pushing."

☑ Learning

"The pipeline doesn't enforce pre-push checks, and there's no peer review gate for weekend deploys."

One of these closes the conversation.
The other one prevents the next incident.

Still reading? Good.
Because this is the part where most postmortems go to die.

Not in the room. After it.

In the ticketing backlog. In the "we'll get to it next sprint" conversation.

In the quiet moment six weeks later when the same alert fires again.

Stage 05 is what separates teams that learn from teams that repeat.

05 Write Action Items That Actually Close

Here's the uncomfortable truth: a postmortem with ten vague, unowned action items is worse than no postmortem. It creates the illusion of progress while guaranteeing nothing changes.

Every action item needs exactly four things:

1. **A specific outcome** — not "improve monitoring." "Add alerting for DB connection pool above 85%."
2. **One named owner** — not "the team." A person's name.
3. **A real due date** — not "next quarter." A date someone can be held to.
4. **A link to the factor it fixes** — if it doesn't trace back to something in your contributing factors list, it shouldn't be on the list.

Tier	What it fixes	Timeline
Immediate	Stops exact recurrence	This sprint
Short-term	Reduces similar incidents	30–60 days
Strategic	Addresses systemic patterns	Quarterly

Aim for three to five sharp items over ten vague ones.

Review open items at every incident cycle.
If they're not closing, ask why.

06 Write and Distribute

A postmortem document has two audiences:

1. **Engineering team** — needs technical depth, contributing factor analysis, and honest action items.
2. **Leadership and stakeholders** — needs business impact, confidence it won't recur, and a clear executive summary.

Write it to serve both. Don't make leadership read through your deployment logs to understand what happened.

Distribution checklist

- ☑ Share with participants for factual review — 24-hour window
- ☑ Post to your postmortem archive (searchable, tagged by service + severity)
- ☑ Send executive summary to VP Eng, CTO, Customer Success if customer-facing
- ☑ Link back to original incident record in your IMR platform
- ☑ If customer-impacting — align with Customer Success before external comms

7 Mistakes That Quietly Kill Postmortem Culture

You've probably seen most of these. Maybe run a few of them.

01 Running a blame session with a postmortem agenda

It looks structured. But the questions are pointed, and everyone in the room knows who's in the crosshairs. People stop contributing. Within a few incidents, postmortems become something people dread instead of trust. → *Fix: Train your facilitators. Name the rule out loud at the start: we name systems, not people.*

02 Writing "human error" as a contributing factor

It's not a factor. It's where analysis stops. What you actually need to know is why the system let that human error cause this much damage. → *Fix: Keep asking why until you reach a process, a gap, or a design decision.*

03 Leaving the room with twelve action items

Ten of them will never get done. And everyone knows it before the meeting ends. That's the problem — postmortems that generate lists instead of commitments. → *Fix: Three to five items, real owners, real dates. Everything else is a backlog item for later.*

04 Skipping "what went well"

Every major incident has things that worked. The escalation that fired correctly. The runbook someone actually followed. Naming them isn't cheerleading — it's how you reinforce behaviors worth repeating. Skip this section and you've confirmed the postmortem is about punishment, not learning. → *Fix: Make it non-negotiable. Find the wins before you leave the room.*

05 Writing an executive summary in engineering language

The exec summary gets forwarded. It lands in front of your CTO, Customer Success, sometimes legal. If the first paragraph mentions MTTA and connection pool exhaustion, you've lost them. → *Fix: Write it last. Business impact first. Plain language. One paragraph.*

06 Waiting more than five days

After five days, the details that would've made this postmortem sharp are already softening. The urgency has dissolved. The incident feels distant. People cancel. → *Fix: Book the meeting before you close the incident. For P0s, that's not a suggestion.*

07 Not building a searchable archive

One postmortem is a lesson. Fifty postmortems in a searchable archive is a map of your system's failure modes. Without it, every incident feels novel — even when it isn't. → *Fix: Archive everything. Tag by service, severity, and factor category. Make it findable.*

Template Bridge

The Template. The Examples. The Checklist.

Writing a good postmortem from scratch — especially after a major incident, when everyone is exhausted and just wants it done — is genuinely hard.

We built a companion notion doc so it doesn't have to be.

The Xurrent IMR Postmortem Companion Doc

- → Fully annotated template — every field explained with example entries
- → A completed sample P0 postmortem (real-world database outage scenario)
- → 14-point facilitator checklist — print it, run the room with it
- → Five-why worksheet for contributing factor analysis
- → Action item tracker with owner, priority, and status fields
- → Blameless culture quick-reference card for first-time facilitators

Access the Companion Doc ↗

Free Google Sheet. *No second form. Your download unlocks both.*

IMR by Xurrent

> Our most important KPI is uptime of all production websites — and without Xurrent IMR, we cannot do this.
>
> — Vinay Singh, DevOps Manager, IndiaMart

60% MTTR reduction | Zero downtime in 2 years

The incident is behind you. The next one is preventable.

Xurrent IMR automates the timeline, the draft, the distribution. You just got the playbook for everything else.

Start free — no credit card needed ↗

See postmortem automation in 5 minutes ↗