BLAMELESS

The Blameless Complete Guide to Incident Management

PART 1

Incidents are inevitable. As your service expands and becomes more complex, you're more likely to encounter outages, slowdowns, errors, and other disruptions to healthy operation. At the same time, as your service becomes more popular and relied on by users, the cost of incidents becomes higher. A bad incident could impact all of the following and more:

- Loss of revenue from a service being unavailable or substandard
- Customers churning to more reliable competitors
- Potential customers abandoning the product during evaluation
- Delay of feature work that could provide a competitive advantage

All of these factors directly and negatively impact your business's bottom line. Studies have shown that the cost of downtime is high, and growing fast in the digital-first world. Since you can never fully prevent incidents, it's important to resolve them as efficiently as possible.

This eBook will break down what to do when things go wrong. We'll cover:

What to do in the heat of an incident

How to prepare for incidents by building resources

How to learn from incidents to become more resilient and robust

Let's dive in and level up our incident management skills!

During an incident

The most important thing to know is what you'll do while things are going wrong. Preparing for incidents and learning from them is necessary, but there will always be things you aren't ready for. Here's some guidance for what to do in the thick of incident response when you don't have everything in place.

Don't hesitate to declare an incident

incident. There might be some overhead in the incident response process if you aren't set up with good tooling and automation, such as creating retrospective documents, setting up collaboration channels, or making formal tickets in tracking systems. On-call engineers might not be sure if it's worth dealing with this overhead for a given problem.

In general, the benefits to invoking your incident response process – like getting the necessary help and creating documents to learn from – are worth any cost to your time.

> **If you're not sure, remember there's a reason you're concerned enough to consider an incident. Even if there's nothing wrong, it's still an opportunity to collaborate and learn.**
>
> *Jake Englund, SRE at Blameless*

Diagnose and solve with deliberation

Coming up with the solution for a mysterious incident is like following the scientific method. You come up with a hypothesis of what is causing the problem, you test it, and adjust and retest based on what you see. Sticking with this model will keep you grounded and focused. It can be tempting to try to consider every aspect of your system before you act, but continually experimenting will keep you learning and moving closer to the truth. Pick a theory and follow it through, without worrying about figuring out the entire story of the incident at once.

Use past incidents or guides as starting points for what to try. As you prepare for incidents and learn from them, you'll build up more of a library of things to try during fixes.

Keep communication flowing

As your team tries different approaches, it's important to share what you learn and avoid redundant work. Continually communicate in a central area, such as a Slack channel set up for the incident. Share what you're trying and planning on trying, and what results you've gotten so far. It can be helpful to hop into a Zoom call while working to brainstorm ideas more freely. Just remember to also record what came out of these conversations in text formats for documentation later.

Communicating with other stakeholders is also essential during major incidents. Talk with your team to make sure someone is relaying updates to management, impacted customers, and even other engineering teams that might be called in to help.

Escalate and ask for help

Asking for help is never a bad thing. It can be tough to admit that you aren't sure what to try, or that you don't have the resources you need to continue. But getting help is essential to solving some incidents, so do your best to ask when it's necessary. It isn't always about escalating to a higher layer of management – it can be helpful to have an "incident buddy" who you're comfortable calling on to help out.

> **On our on-call rotation, we always have two or three engineers on standby as on-call buddies so we're never alone.**
>
> *Alyson Van Hardenburg, Engineering Manager at Honeycomb*

If an incident is proving more extensive and impactful than previously thought, escalating up the chain of command can be necessary to get the proper attention and authorization. In situations where an incident is proving hard to diagnose, it may be necessary to call on specific subject matter experts for the relevant areas. Escalation shouldn't be linear, but an adaptive process that brings in just the right people for each incident.

Learning from incidents

Just as important as preparing from incidents is learning from them. Every incident should be considered an opportunity to improve the reliability of your system going forward. Look for systemic changes that can be made to prevent similar situations from happening again.

Incident retrospectives

Incident retrospectives, also known as postmortems, are documents created for each incident. They let you dive deep into the causes of the incident and serve as a guide for followup tasks. Here are some important things to include in a retrospective:

Summary

This should contain 2-3 sentences that give the reader an overview of the incident's contributing factors, resolution, classification, and customer impact level. The briefer, the better, as this is what engineers will look at first when trying to solve a similar incident.

Customer impact

This section describes the level of customer impact. How many customers did the incident affect? Did customers lose partial or total functionality? If this retrospective will be public-facing, it should also assure customers of what didn't happen. Explicitly mention if data wasn't lost or exposed, or if security was breached or compromised.

Follow-up actions

This section is incredibly important to ensure that accountability around addressing incident contributing factors looks forward. Follow-up actions can include upgrading your monitoring and observability, bug fixes, or even larger initiatives like refactoring part of the code base. The best follow-up actions also detail who is responsible for items and when the rest of the team should expect an update by.

Contributing factors

This section is incredibly important to ensure that accountability around addressing incident contributing factors looks forward. Follow-up actions can include upgrading your monitoring and observability, bug fixes, or even larger initiatives like refactoring part of the code base. The best follow-up actions also detail who is responsible for items and when the rest of the team should expect an update by.

Who are the characters and how did they feel and react during the incident? What were the plot points? How did the story end? This will be incomplete without everyone's perspective.

Make sure the entire team involved in the incident gets a chance to write their own part of this narrative, whether through async document collaboration, templated questions, or other means.

Timeline

The timeline is a crucial snapshot of the incident. It details the most important moments. It can contain key communications, screen shots, and logs. This can often be one of the most time-consuming parts of a post-incident report, which is why we recommend a tool for automation. The timeline can be aggregated automatically via tooling.

I would emphasize that such stories should be written in the present tense, as if it were a story unfolding.

Matt Davis, SRE Advocate at Blameless

Technical analysis

Technical analyses are key to any successful retrospective. Afterall, this serves as a record and a possible resolution for future incidents. Any information relevant to the incident, from architecture graphs, to related incidents, to recurring bugs should be detailed here.

Here are some questions to answer with your team:

- Have you seen an incident like this before?
- Has this bug occurred previously, and if so, how often?
- What dependencies came into play here?

Qualitative analysis

On top of looking at the technical details of an incident, it's important to ask questions and gather data about the qualities of a given incident. This will provide a holistic context to make sense of the incident.

Here are some questions to answer with your team:

- What other incidents were happening at the same time as this one?
- Were there any abnormalities in the environment that impacted the response process?
- Were the processes implemented typical for your response process, or did you have to improvise?

Incident management process analysis

At the heart of every incident is a team trying to right the ship. But how does that process go? Is your team panicked, hanging by a thread and relying on heroics? Or, does your team have a codified process that keeps everyone cool? This is the time to reflect on how the team worked together.

Here are some questions to answer your team:

- What went well?
- What went poorly?
- Where did you get lucky and how can you improve moving forward?
- Did your monitoring and alerting capture this issue?

Messaging

Communication during an incident is a necessity. Stakeholders such as managers, the line of business (i.e. sales, support, PR, etc.) C-levels, as well as customers will want updates. But communication internally and externally might look very different. Even communication internally might differ between what you would send a VPE, vs. your sales team.

Your retrospective template might differ based on the type of incident. More severe incidents might require sections for PR and messaging to management. It's essential to make your retrospective easy enough to complete that engineers won't avoid it. Tooling can help automatically gather and format data, and prompt responders with questions.

Reviewing your retrospectives is important to make sure your systemic changes are being enacted. Schedule regular meetings to go through recent incident retrospectives. Share invitations to these meetings broadly, as people across teams can still learn about your system in these moments.

Patterns in incidents

On top of digging into specific incidents with retrospectives, you can study your system health more generally by looking at patterns of incidents. As you deal with incidents, gather statistics about each of them. Just looking at this quantitative data won't tell you exactly what to do, but it can highlight trends, outliers, and other things that motivate discussion. It can also help people who aren't in the trenches of incident response, like management teams, understand system health better.

Here are some examples of things you can observe from patterns:

Find the product areas causing the most problems

By looking at which product areas have the most incidents, you can know where to invest your time and energy. Product areas with lots of incidents could have accumulated technical debt or bugs.

See if your incident response practices are paying off with "Mean Time to X" data

You can gather statistics about the timelines of different incidents. How long does it take on average to detect an incident, start working on it, come up with a fix, etc? Look for trends in this data as you implement different policies. You have to make sure you're comparing similar types of incidents before and after changes. There will always be outliers, which you can study separately. You just need a general sense if things are getting better or worse.

Prepare on-call schedules around frequency patterns

Your service's health will fluctuate, with incidents cropping up more frequently at certain times of the day, days of the week, or times during the year. These patterns will be apparent as you look at incident statistics over time. Then you can schedule on-call teams to account for these influxes.

By looking at quantitative data about lots of incidents and diving into particular incidents through retrospectives, you can get a complete picture of where to invest in system health. This will be your guide for knowing where runbooks should be built, and what should go in them. There's no better way to be prepared for future incidents than having a handle on past incidents.

Your journey to incident excellence is underway!

You've gotten a handle on how to stay afloat during an incident, and how to learn once the incident has finished. Now it's time to prepare for incidents – with classification systems, runbooks, roles and checklists, and a toolstack that can reduce incident response toil.

Join us in part 2 to complete the incident response cycle!

Blameless drives reliability across the entire software lifecycle by operationalizing Site Reliability Engineering (SRE) practices. Teams share a unified context during incident response, efficiently communicate, and resolve faster. Detailed Retrospectives give teams a data-driven approach to learn and Service Level Objectives (SLOs) inform teams where to prioritize work and innovate at velocity. Customers include brands such as Procore, Under Armour, Citrix, Mercari, Fox, and Home Depot who embrace a blameless culture, team resilience, and greater reliability to protect their customers.

Blameless is a 2021 Gartner Cool Vendor recipient and is backed by Accel, Lightspeed, Decibel and Third Point Ventures.

BLAMELESS

