BLAMELESS

Incident Response for Resilient Socio-Technical Systems

Table of Contents

Incident response challenges for modern teams

Sociotechnical systems. We've heard this term time and time again, yet what does it really mean, and what does it have to do with teams responding to incidents? As the name implies, sociotechnical systems are systems that exist with both human and technical components working in harmony. Or at least ideally. As Gordon Baxter and Ian Sommerville state in the abstract to *Interacting with Computers, Volume 23, Issue 1*, "The underlying premise of sociotechnical thinking is that systems design should be a process that takes into account both social and technical factors that influence the functionality and usage of computer-based systems."

Our technical systems are built by, maintained by, and crafted for humans. When incidents happen, it's humans who respond to fix the system. For the relationship to be symbiotic, the system should support those who build and maintain it. Yet many organizations struggle with this balance.

Difficulties stem from many sources:

- Repetitive tasks necessary to maintain the system and resolve incidents add toil to responders' processes.
- Alerts can be noisy. Even if they are signaling something important, the alert may be ignored due to alert fatigue.
- Especially while working remotely, communication during incidents is crucial. Teammates must communicate with one another, to executives, and to their customers.
- Engineers experience burnout due to the high frequency of incidents and the amount of reactive work necessary to keep the system functional.
- Lessons learned are rarely fed back into the software development lifecycle, and are often left forgotten in backlogs or tickets, causing repeat incidents.

But, there are ways to overcome these challenges and build stronger sociotechnical systems. Teams have adopted Site Reliability Engineering (or SRE) to bolster resilience, ease the pressure on the humans operating these systems, and create a blameless environment of psychological safety.

In this whitepaper, we will:

- Describe why incident resolution is harder than ever
- Discuss communication, cognitive capacity, and customer needs
- Share how SRE can help teams respond better under pressure
- Detail SRE best practices which can help your team both during and after incidents
- Provide a comprehensive incident response blueprint
- Share how Blameless can help teams with their incident resolution needs

Why incident response is harder than ever

Incident response is getting more complex. With the adoption of microservices, it can be difficult to identify what broke and how. Additionally, teams are automating their software systems, but forgetting that incident response is itself a system, and often a toilsome one. Incident response automation can sometimes be left out of planning.

As for the human side of incidents, communication during incidents requires more intentionality and planning than the informal swarming which could take place in person. As teams strive to staunch the flow of incidents, burnout becomes a pernicious problem. And all the while, customers are expecting more from the services they rely on.

We'll share why these issues are important before we address how SRE best practices can help teams overcome these challenges.

Honing in on the technical difficulties of incident response

Making the switch to microservices

Most systems are no longer single blocks of code where dependencies are clearer. As teams moved to microservices, dependencies became opaque and determining what broke became more difficult. Yet teams continue to adopt microservices for the range of benefits they provide, such as increased agility and improved scalability.

According to an O'Reilly survey, "About 28% of respondents say their organizations have been using microservices for at least three years; **more than three-fifths (61%) of the respondents have been using microservices for a year or more."**

And for the most part, teams state they've found success with this process. As O'Reilly states, "A minority (under 10%) reports 'complete success,' but **a clear majority (54%) describes their use as at least 'mostly successful'** and 92% of the respondents had at least some success."

Teams have had to work with the issues that come along with microservices, however. Some of these issues include:

- **Service sprawl:** You'll need to set up monitoring for each of these services, and map key user journeys.
- **Distributed data:** When your data is spread across multiple sources, it can be difficult to figure out exactly what caused the issue.
- **Service communication:** How do all these services work together? How do they talk to one another? You'll have to solve this problem for microservices to be effective.

Navigating microservices isn't the only problem teams are facing right now, though. There's also the subject of automation.

To automate or not to automate?

Automation is tricky, especially in incident response. There's no way (so far) to totally take the humans out of incident response, nor would you want to—remember, in sociotechnical systems, the human is just as important as the technology. With this in mind, you'll need to determine what is worth automating.

Automation takes time and resources. If engineers spend a sprint automating parts of incident response that only manage to shave off a few seconds' worth of response or mitigation time, was that initiative worth it? Many teams will need to pick and choose what they will automate and make tradeoffs. Below are some common processes that are automated in incident response:

- Data aggregation and timeline curation
- The connection from incident alerting to incident creation
- Scripts for detection or remediation
- Automatic rollbacks

The necessity for automation can also be dictated by the amount of reliability your customers require. For life-critical systems, downtime has to be minimal. Automatic rollbacks might be a worthy investment, even if they are difficult to create. For a social media site sharing pet images, automation requirements might be lower as customers allow more leeway for downtime.

However, automation is difficult to perfect. And no matter what, you still need humans in the loop to make the tough calls, to run the right scripts, and more. Finding the balance between human input and machine automation is crucial. For example, in the article "Tesla's problem: overestimating automation, underestimating humans" the author notes "the recent drive for full automation has overlooked the importance of adaptability. Humans are still far more able to adapt to change than artificial intelligence (AI)."

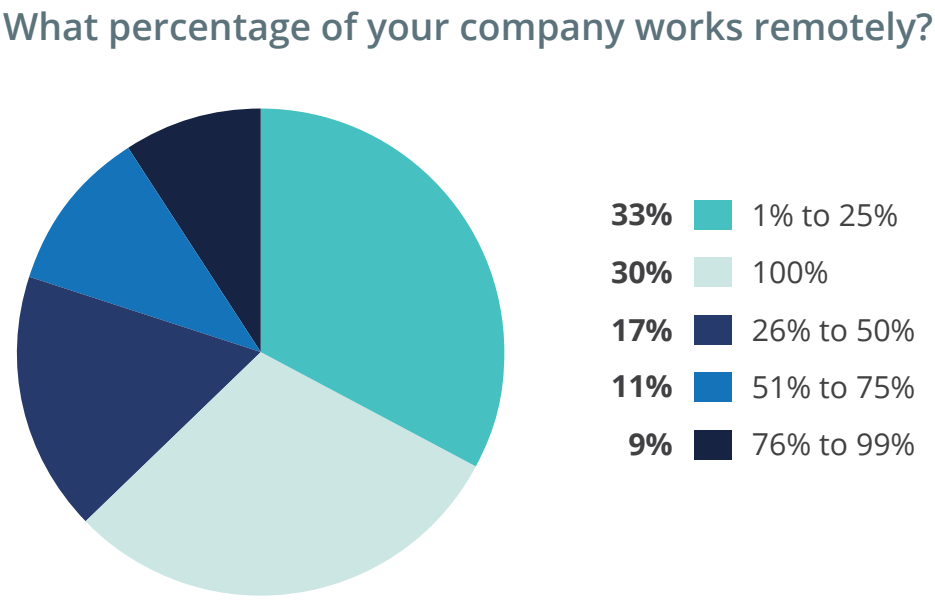
Human adaptability is what makes systems resilient. Automation should help the humans who operate the system rather than seek to replace them.

Balancing communication, cognitive capacity, and customer needs

Understanding the effect of remote work

Remote work has become the norm. While many people took up this way of working as a response to COVID-19, remote work is here to stay. According to a survey, "**More than 80% of company leaders surveyed by research and advisory firm Gartner said their organizations plan to permit employees to work remotely at least part of the time upon reopening from the COVID-19 pandemic.**"

This is very different from even just a few years ago when, according to Buffer's State of Remote Work Report, the largest segment of companies (33%) allowed only 1-25% of their employees to work remotely.



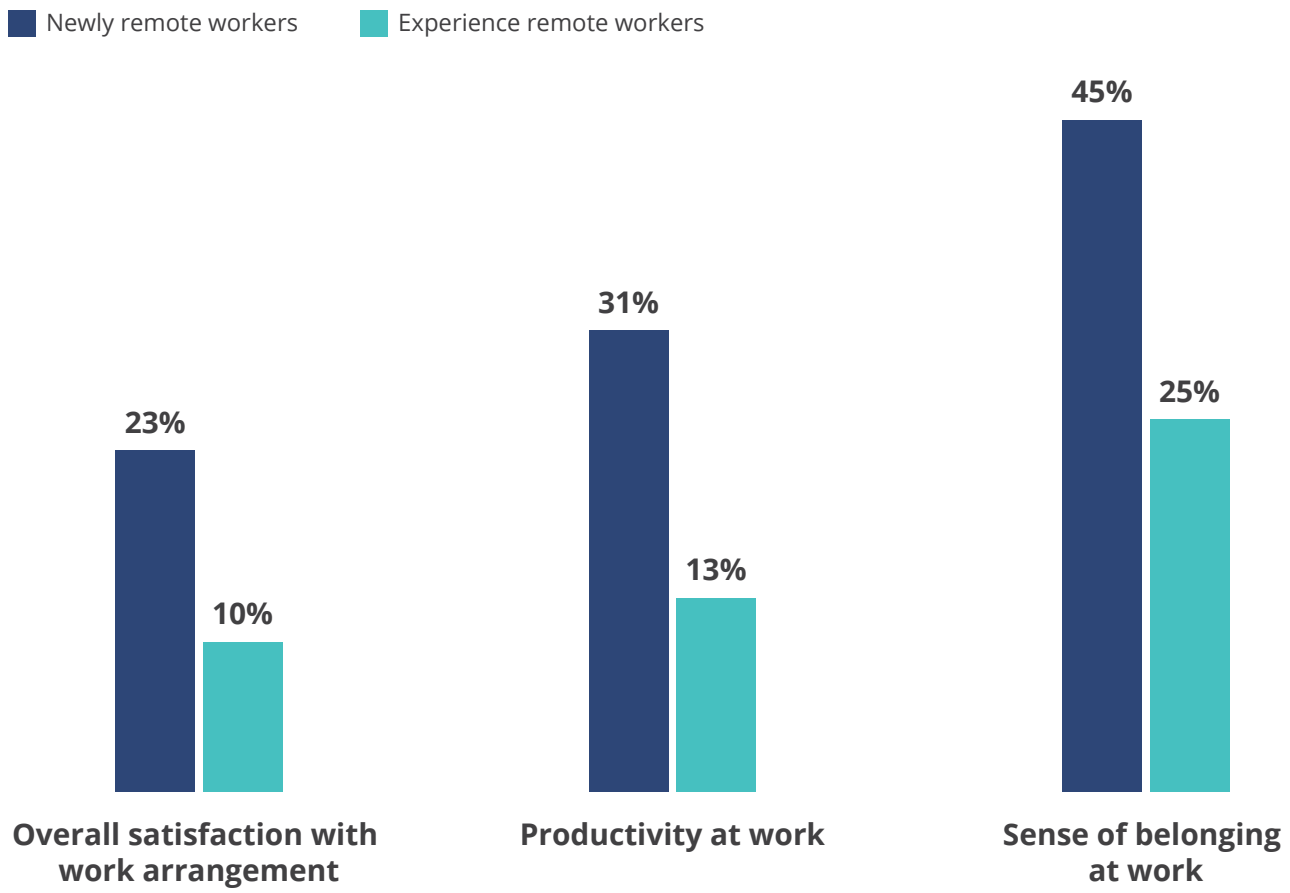
Source: State of Remote Work Report, 2019

While working remotely comes with many benefits (no commutes, the ability to workout during lunch, never having to change out of pajamas), there are many challenges as well. This means that, while many people are waiting for the day they can return to the office, many others are hoping to work remotely the majority of the time post-COVID.

Microsoft's "New Future of Work Report" states "In a March survey of US-based Microsoft software engineers and program managers, 34% said their productivity had decreased while working from home, and 34% said it had increased. In a July survey of employees in Puget Sound, 32% agreed that returning to the office would make them more productive and 30% disagreed [109]."

Perhaps the success of the person working remotely is based partially on previous remote experience. In Slack's "Remote work in the age of Covid-19 report," employees with little to no experience working remotely had lower overall satisfaction, productivity, and sense of belonging compared to those who had previously worked remotely (displayed in the graph below).

Experienced vs. newly remote workers who say working from home is worse for the following indicators



Source: Remote work in the age of Covid-19 report by Slack

Additionally, Zoom fatigue, extended hours, and the blurring of lines between work and home weighs heavily on teams. Communication can suffer, and teammates can begin checking out mentally.

Burnout has become a much larger problem as the lines blur between work and home. While COVID-19 increased burnout rates, this issue is not unique to it. In a survey by FlexJobs it states, "75% of people have experienced burnout at work, with 40% saying they've experienced burnout specifically during the pandemic."

Teams facing burnout, Zoom fatigue, and more pressure to keep services operational need additional resources to mitigate possible attrition.

And don't forget the customer!

In addition to all these challenges, customers are more reliant on technology than ever. And, as systems get better at operating with few interruptions, they become more accustomed to services being nearly constantly available. Disruptions (especially frequent disruptions or ones that last for long periods of time) can damage customer trust.

Reliability has become feature No. 1, as any other feature is made irrelevant if customers cannot access it. Yet 100% uptime is not possible. Incidents will happen. You need to understand what level of reliability your customers will expect. Then, you need to optimize for meeting those standards.

With too much reliability, you're wasting resources that could otherwise be spent on developing new features. With not enough reliability , your customers will be dissatisfied and leave you for your competitor. This balancing act is difficult. However, this problem (as well as the ones mentioned before) can be solved with SRE best practices. Let's take a look at how.

How SRE can help you improve your incident resolution process

SRE, or site reliability engineering, is not a silver bullet. It's a cultural and procedural change that takes time and organizational buy-in. However, the benefits are worth it. As teams adopt SRE best practices, they'll experience heightened alignment, better reliability, and happier customers. Below are some ways that SRE best practices can help solve the sticky problems making incident response difficult.

Aligning on what your system needs with SRE

Managing microservices

As teams move toward a microservice model, they need to make changes about how they document and monitor their services. SRE encourages a culture of detailed documentation. This limits tribal knowledge, or knowledge that exists only within the minds of those who work with the service or product.

With thorough documentation, teams can help ease the burden of incident response. For instance, runbooks that describe common incidents should point to related services that might also experience downstream effects of an outage. They could also list services that interact with the one experiencing the outage that could potentially be a contributing factor.

These runbooks can also contain scripts or tests to run which help teams deduce what happened and how to fix it. Runbooks can also be used beyond the crisis of incident response to document common activities your team completes. For instance, if upgrading a service changes the versioning, it's great to know what microservices could also be affected by this change.

Another documentation best practice is service registry or mapping. As the number of services you have grows, it's important to keep track of them and how they interact with each other. Mapping these connections out can help teams create a visual representation of their dependencies.

Automation to help the human, not replace them

The humans who adopt SRE best practices and follow these processes should feel supported by these procedures. Rather than thinking, "*How can I take the humans out of the equation?*" think, "*How can I help the humans responding to the incident?*". This shift in thinking will help you determine what to automate.

Here are several things to consider:

- Will this automation reduce the time the service is down?
- Will this limit the amount of context switching my team will need to do?
- Will this lower the cognitive load required in this process, especially if the responder is woken up in the middle of the night?

When kicking off an incident, automating the incident command framework and any ticket or other communication artifacts can decrease the time teams spend getting organized before actually working to resolve the issue. Runbooks can help teams avoid context switching.

And, after the incident, when teams are working on their retrospectives, data aggregation and timeline creation can lower the cut-and-paste toil. Teams can start working on problem solving and deep thinking rather than sifting through messages and incident communications.

Keeping an eye on what matters most: the people

Use your data to help your team recover

As burnout rates increase and teams are under more pressure to keep systems operational, it can seem like only a matter of time before you hit a wall. Managers shouldn't be the last to know when their team is struggling. Use your data to get ahead of the curve.

SRE looks to create systems that are resilient, and people are part of that system. There's no way (or reason) to engineer a *person* to be prepared and alert around the clock. But you can build a team that's well rested enough that it's ready for any incident. This requires paying attention to data that you may already have and yet overlook.

For instance, here are a few things to track that can help you recognize when burnout is on the horizon:

- Time spent per person on call
- Alert noise (non-actionable alerts)
- Incidents per on-call shift
- Incidents occurring off hours (nights and weekends, separately)
- Incident duration

This data alone may not be very indicative. However, an important part of SRE is *using data* to *make the decisions* that best support system reliability. Metrics alone don't make decisions. We still require humans to analyze them.

If you look at this data, you may realize that some of your teammates are at greater risk for burnout. For instance, let's compare two people. Person A had two back-to-back on-call shifts with 3 incidents a piece. They were all Sev 3s and resolved within 15 minutes. Person B has a single on-call shift and only 2 incidents. However, one was a Sev 2 that lasted an hour and one was a Sev 1 that lasted for 4 hours.

If this schedule repeats and Person B continues to take on-call shifts during times with high incident volume, they'll be at risk for burnout faster than Person A, despite being on call only half the time. This is important to know to provide accurate on-call relief.

By making sure that each team member is cared for, you build a more resilient team and system.

Using SLOs and error budgets as a barometer for customer happiness

How do you know which areas of your service are most critical to your customers, and how much failure they will tolerate from your system? Well, it takes a bit of trial and error to get right, and lots of inter-team communication. An SRE best practice, SLOs (service level objectives) can help.

Teaming up with your product team and your Customer Support team, you can identify critical paths in your service, or user journeys that customers care about. For instance, if you have an ecommerce website, your checkout service might be an important user journey to look at.

Next, you determine SLIs (service level indicators) for these user journeys. As the Google SRE book states, SLIs are "a carefully defined quantitative measure of some aspect of the level of service that is provided." This measure is determined by the equation of good events/valid events and typically measures some of these aspects, defined by the Google SRE workbook:

- **Availability:** The proportion of requests that resulted in a successful response.
- **Latency:** The proportion of requests that were faster than some threshold.

- **Quality:** If the service degrades gracefully when overloaded or when backends are unavailable, you need to measure the proportion of responses that were served in an undegraded state.
- **Freshness:** The proportion of the data that was updated more recently than some time threshold. Ideally this metric counts how many times a user accessed the data, so that it most accurately reflects the user experience.
- **Correctness:** The proportion of records coming into the pipeline that resulted in the correct value coming out.
- **Coverage:** For batch processing, the proportion of jobs that processed above some target amount of data. For streaming processing, the proportion of incoming records that were successfully processed within some time window.
- **Durability:** The proportion of records written that can be successfully read.

Once you've determined your SLI and looked at historical data of previous reliability performance, you can set your SLO. Think of this as a goal (and one that will need to be changed and iterated upon). This SLO should provide you wiggle room to fail, yet still keep your customer happy-ish.

For example, perhaps you have set an SLO for 99.5% availability. This keeps your customers happy. You don't want to be more available than this, nor less. This means that per month, you can have 3.6 hours of downtime. It's not a perfect 100%, and it doesn't have to be! If you're meeting your SLO each month, customers are still happy and your team has some "error budget" remaining to make mistakes, learn, and grow.

While this can take some of the pressure off incident responders to know they don't have to be perfect, it can also help in other ways. You can base alerts and severities off SLOs as well.

For instance, if a minor incident is consuming a very small fraction of your error budget, you know you don't need to rouse the on-call team from sleep to fix an issue. It can wait until the morning, and your phone can stay quiet. However, if you're burning through your error budget at a rapid rate, you know it's a higher severity.

SLOs are a fantastic decision making tool, both in peacetime and during incident response. Just like the rest of these SRE best practices, they help teams care for both the humans and the system. While SLOs are a more advanced best practice, coordinated incident response is one that many teams can get started with right away. Below is a template you can use to help you set up processes and guidelines for incident response.

Your incident resolution blueprint

Purpose

The purpose of this blueprint is to help teams lay the groundwork for incident response procedures. This blueprint provides examples of documenting severities, roles and responsibilities, communication guidelines, incident phases, and more. If you want to create a set of guidelines for your own team, feel free to make a copy of this template of policies.

Severities

Severities should correlate to customer impact. This should take into consideration a few things: how many people are affected (and for how long, predictably), historical service performance, incident duration, and potential SLA violations. For many teams, incidents that result in a breach of SLA are categorized as a Sev 0 or Sev 1. This is because there is a direct business impact; the organization will likely need to pay for this lapse in service.

However, the other three criteria provide a little more wiggle room. These may not be tired directly to business impact, but they are tied to customer happiness, which eventually determines overall company success. It's important to make sure that you understand how you weigh the needs of different customers. For some high-touch customers, incidents which affect them will automatically receive higher severities. For low-touch customers with more lenient expectations, you might be able to downgrade a severity.

Let's use an example. In this example, product owners have determined that the users of this service are greatly concerned with availability. The users will tolerate slower loading speeds, but they want the site to work when they visit it.

Additionally, customer satisfaction surveys have shown that, during quarters where the downtime has exceeded 3.65 hours per month (99.5% availability, as an SLO), customers are unhappier. Anything underneath that amount of downtime has no negative impact on survey results. This is the error budget, or the allowable amount of failure. The team has determined severity by the likelihood of the incident consuming a disproportionate amount of the error budget.

For example, if an incident continues unresolved, based on current burn rate, how quickly would the budget deplete? Within an hour? A day? 30 days? If the incident is a slow burn and unlikely to consume large portions of the error budget, even over an extended period of time, it can be marked as a lower severity. If it will eclipse the error budget within the next 30 minutes, you're probably facing a Sev1.

The criteria will look very different based on the service in question. However, the response dictated by severity should be consistent across services.

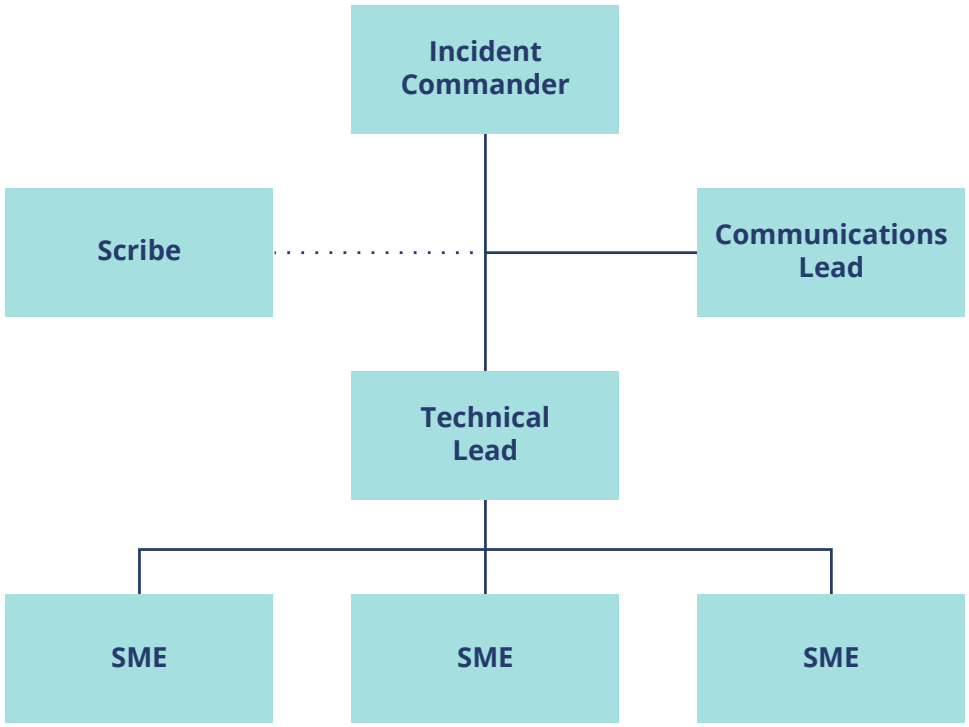
For example, you might use something like this table to determine responses necessary.

Severity	Trigger Criteria	Response
Severity One	Complete customer outage	All hands on deck
Severity Two	> 1 region affected	Senior engineering teams and management alerted
Severity Three	1 region affected	Senior engineering teams alerted
Severity Four	Partial outage of a major product feature	Relevant engineering teams alerted
Severity Five	Limited or minor outages for select customers or of a non-critical feature	Incident logged into ticketing systems, but no immediate escalation or alerting necessary

Roles and responsibilities

Incidents can be chaotic. They're even more difficult when teammates are unsure of the roles they each play in the resolution process. It's important to determine who will complete what tasks. Here is a typical incident command chart.

Keep in mind that smaller teams may not have highly differentiated roles, and smaller incidents may not need as much differentiation. Some roles may be combined and completed by a single person. You can adjust this chart to reflect your team size and skillset.



Below is a table with common roles depicted above, and their responsibilities. Beside it is room for you to fill out the name of the person who will complete these tasks.

Role	Responsibilities	Person
Incident Commander	Person running the incident. Their ultimate goal is to bring the impact to completion as fast as possible and coordinate with other participants.	Dwight Schrute
Scribe	Person (or bot) transcribing key information during the incident and documenting the process.	Angela Martin
Communications Lead	Person in charge of communications. For smaller incidents, this role is typically subsumed by the Incident Commander.	Kelly Kapoor
Technical Lead	Person who is knowledgeable in the domain in question, and helps to drive resolution by liaising with Subject Matter Experts. In smaller incidents, this role might be performed by the Incident Commander as well.	Jim Halpert
Subject Matter Expert	Person (or multiple people) working in coordination with one another to resolve the incident and reporting progress to the Technical Lead.	Meredith Palmer

Communication guidelines

Your team should also know where, when, and what to communicate during an incident. How often should you update one another? Where should discussions take place? At what point do you escalate the issue? Each of these questions can be addressed in communication guidelines. Below is an example that you can adjust for your own purposes.

Incident communication standards

During an incident, these are the channels of communication we use, as well as example requirements for updates.

Chat and screen shots: Ensure that all text and screenshots pertaining to the incident are captured in the dedicated incident channel which the incident commander creates. All communications should occur within this channel. Please refrain from using direct messaging during an incident as key context can be lost that would benefit the entire team. Add any pertinent screen shots to the channel as well so they can be captured and included in the incident retrospective after the incident is resolved. Chat is our primary communication channel.

Video conferencing: Please use the dedicated video conferencing room established by the incident commander. This ensures that conversations aren't happening in places inaccessible to other teammates. Video conferencing is our secondary communication channel.

In the event of a communication tooling outage: If either primary or secondary communication methods are unavailable, the incident commander will create a single email thread which the team will reply all to for the duration of the incident. Please keep all communications within this thread.

Timing (policies can vary by severity levels):

- **SME:** Update the technical lead every 15 minutes with a brief progress update and any resources needed to help resolve this issue. Bubble up relevant discoveries as they happen and ask questions to the team and the technical lead as needed.
- **Technical lead:** Update the communications lead and incident commander every 15 minutes with relevant updates from the SMEs. Bubble up discoveries and questions as they develop.
- **Communications lead:** Update people internally and externally on the hour. Internal updates and external updates should reflect the amount of information each audience requires and use the terminology which is meaningful to each audience. You can refer to the CAN (conditions, actions, needs) framework to craft your communications with intention.
- **Incident commander:** In the case of a combined role of incident commander and communications lead, follow the communications lead instructions. In the event of a longer incident (<8 hours), this person would be replaced by a secondary on-call incident commander. If the organization has follow-the-sun changes of staff, this person should find 30 minutes to give a detailed handoff before removing themself as commander and reassigning the role to the incoming commander.

Escalation policies

What happens when the on-call team member is not able to resolve an incident? Or when they're slightly out of their area of expertise? Who can the team look to for help to get their service back to normal faster? According to Atlassian, there are three types of escalations teams should consider. It's important to know which one is best for the situation at hand, and who to hand off to.

- **Hierarchical escalation:** This is when teams escalate an issue to a more senior person. This escalation method can be especially helpful if the incident is of high severity (Sev 1 or Sev 1) and requires judgement calls that only more senior team members would be equipped to handle.
- **Functional escalation:** This is when teams escalate an issue to a person with deep subject matter expertise. It's especially important during incidents dealing with edge cases, rarely-occurring incident types, or services that require niche knowledge.
- **Automatic escalation:** This method is more related to tooling and acknowledgement procedures than the others and is often used in conjunction with either above method. Automatic escalation is when an incident is escalated to a secondary or tertiary on-call team member if the primary on-call is unable or fails to recognize an alert in their alerting system. This ensures that incidents don't fall through the cracks.

Below is an example escalation policy using a combination of all three escalation methods.

1. In the event of the primary on-call team member being unavailable to respond to the incident, our paging system will alert the secondary on-call team member. In the event the secondary on-call team member is also unavailable, this incident will escalate to the engineering team manager.
2. Once the on-call team member has determined a severity, the team will follow this escalation policy:
    a. Dev 4+ should be resolved by the no-call team member. No role assignment is necessary for these incidents.
    b. Sev 2 and 3 should be resolved by the on-call team member, looping in a secondary SME as needed. You may not need to assign roles for these incidents.

c. For Sev 1 and Sev 0, the engineering team manager should join the incident response team. The on-call SME should then assign a role of incident commander to the person who will be coordinating the incident, likely the engineering manager. For these severities, you must also assign the roles of communications lead. Technical lead duties may remain with the incident commander.

d. For Sev 0, engineering leadership (such as the CTO or VPE) should also be notified. For this severity, all roles (incident commander, communications lead, technical lead) should be assigned.

3. In the event that the incident is not able to be resolved within the below timeframes, a service-specific subject matter expert should be consulted.

   Sev 0 → 1 hour

   Sev 1 → 2 hours

   Sev 2 → 4 hours

   Sev 3+ → 8 hours

On-call SMEs for additional services

When escalating, especially when using functional escalation policies, it's very handy to have a list of SMEs for other services. All teams should consider creating a list of their SMEs, specializations, and on-call dates for the month. These documents should be publicly available to other teams within the IT organization in the event of an escalation.

Incident phases

These are some common incident response phases, though they may look different from the ones referred to in your organization.

Identify

So, there's an incident. In this stage, your team figures out that something is wrong with your system. Maybe your monitoring triggers an alert that your servers are overloaded. Maybe you have alerts tied to SLOs (service level objectives) and your error budget is burning at an unexpectedly high rate. Or, maybe your customers are reaching out to your Customer Support team with complaints. However you find out, the "identify" stage begins.

Assess

During this phase, the on-call person begins to ask the question “What’s wrong and how bad is it?” Maybe code was pushed to production with a bug that caused some areas of the product to load slowly or be unusable. Or maybe all you know is that a concerningly large portion of your servers have stopped working, and customers are unable to use the product at all.

It’s time to assess severity. In the case of a bug in production, your on-call team member might be able to roll back the update, or turn on a feature flag that hides the bug from users. Based on your classification model, you might determine that this is a Sev 3. Your team can handle it

Or, in the case of the unknown server issue, you may classify this as a Sev 1, as many customers are unable to use the product. Perhaps you’re also very close to breaching your SLA this month with an affected customer. You need to call in some help, escalating on both the hierarchical policies and the functional ones.

In this stage, it’s important to understand the resources you’ll need to get on top of the incident. Afterall, waiting to determine severity or escalate to the necessary people will only result in a delay of resolution. Acting fast can save time and money.

The on-call team member will then loop in other people as needed and assign roles. Some SRE tools even have checklists that walk incident responders through this process and automatically create the incident channel and war room, lowering the toil required to kick off an incident.

Respond

This phase is typically the one most people think of when they imagine being in an incident: actually attempting to fix whatever is broken. However, there are ways to maximize your team’s ability to respond faster. One such way is by writing and using runbooks.

Runbooks are documents that walk you through a certain task with specific steps. For example, a runbook for spinning up a new server might ask some questions about the purpose of the server and its estimated load, then lead you to the appropriate instructions and settings. Runbooks ease the cognitive load of these common tasks by clearly outlining the process for each.

During these moments where the pressure is high, it helps to have a codified set of processes that walk teams through the incident. Additionally, these are evolving documents. Your team should be encouraged to include how the runbooks used could be improved during the review portion of the incident. You should also set time to periodically review runbooks to ensure they’re still accurate, especially when they’re for a particular incident type that you would run infrequently.

Below is an example runbook that we use at Blameless. It can be used both during an incident (say for instance if one of your MongoDB clusters went down) or during peace times when spinning up a new cluster.

Description: This runbook explains the steps to create a new cluster in mongoDB Atlas.

1. **Jira ticket:** We use Jira to track work items. Make a ticket if you haven't, or use an existing one. This step is required when creating a new region for an instance. Using the Jira ticket name in git will also attach your branches and PRs to the ticket.

2. **Choose a workspace:** This terraform resource can be run from your local laptop or the respective Bastion for the region (or any of them, really). A couple of things are needed: the latest version of [terraform-infra] (https://dundermifflin.atlassian.net/wiki/github.com/dundermifflin/terraform-infra) and a valid sa-key.json file at the root of your terraform-infra repo.

3. **Run The Script:** When you are ready, run the terraform_workflow.sh script in the root of terraform-infra. We use the following naming format for regional MongoDB clusters, dundermifflin-<region>. This will run a script that will create a new deployment directory, generate new terraform, and attempt to apply it. This is all based on the defined flags in the command.

4. **Update The Connection String:** Finally, in order for this to be usable, terraform needs to know about this new cluster. We can update the mapped variable in terraform to point to our new cluster when this provider and region are specified. These individual replicas can be found within the MongoDB Atlas Web Console, under the "Overview" section. We need the full path of each replica here, in a comma separated list. Take a look at the other regions for an example. Make a PR for your change, as usual.

During the response process, you'll also need to focus on the communication aspect. This often happens simultaneously, but requires a different thought process and often a different person to accomplish it.

Communication

As we noted above, communication during an incident is key. This part of the process is often done by the communications lead or by the incident commander if the roles are combined. There are two important communication types to consider within an incident. The first is how to handle communication to executives and stakeholders (such as legal) within the company. The second is how to handle communications externally to customers.

For all groups of people, it's important to set communication expectations. Letting someone know up front how often you'll be updating them on the situation can help ease concerns about a communication breakdown. It will also limit the amount of people reaching out to you or your team. If the executive team knows that you'll update them every hour on the hour, they shouldn't ping you every fifteen minutes. Likewise, if your status page states that it'll post hourly updates, it's less likely that your Customer Support or help desk teams will be inundated with calls.

However, what you communicate to each group can differ. Let's look below at what the different groups might look for during a communication.

Severity	People to loop in on communications
Sev 0	All team members (including managers), legal and finance if SLAs are likely to be impacted. PR or marketing if the incident is large enough to require a public-facing statement.
Sev 1	All team members (including managers), legal and finance if SLAs are likely to be impacted.
Sev 2	Team manager, all team members.
Sev 3+	On-call only, no need for internal updates.

Executive: Does everyone in the company need to know when an incident is happening? Probably not. In fact, some people may not want to know what's happening in an incident if it's a low enough severity. Your CTO is not likely to be concerned with a brief Sev 4 incident, afterall. But executives, legal, finance, and perhaps even marketing may need regular updates on the incident, if it's a high enough severity.

You'll also need to know what to communicate to these people, as high-level technical jargon might confuse them, leading to more questions. Instead, consider structuring your regular communications as such (using the CAN framework previously mentioned):

- Brief description of the incident (ex: servers overloaded during routine maintenance)
- Any recent updates (ex: we discovered that we promoted a new feature launch at the exact time the servers were updating, causing the servers to be overloaded with traffic while they're already functioning at lower than normal rates)

- Any necessary resources needed to resolve the incident
- An update on customer impact

This high-level update keeps everyone on the same page while easing the burden on the communications lead. This is especially important as the communications lead also needs to focus on external communications, either communicating directly with external stakeholders, or delegating to a person or group such as PR that is responsible.

Technical: For technical teams, you'll want your updates to be more in-depth. Technical teams who require alerts may have critical context or resources that can help bring the incident to a close. Make sure to include all the same information you would for the executive updates and expound as needed in the description and updates section. In fact, in this case the more thorough the update (as thorough as the comms lead has time for) the more helpful it will be to the entire technical team. Consider adding graphs, logs, screenshots, and anything else that might be pertinent to other SMEs.

External: For the purposes of external communication, we'll consider all users of the service, even if they're internal users. Updates for external communications can take a few different forms. Teams often employ a combination of these methods.

- **Status page updates:** These updates are the most common, and the ones your users will likely check first. Status pages come in a variety of formats. Some of the most common ones include a list of customer-facing components, usually with a signifier beside them indicating status. Additionally, some status pages have space for text updates as well, so that a short message can be entered describing an incident. It's important to make sure your status page is updated, either manually or via automation.
- **Email updates:** These are more personal, and aimed at current affected customers. Sending out a bulk email update once an hour pertaining to the incident at hand can help Customer Support teams from being overwhelmed with incoming requests. It also helps build trust with the customer. They know that you're aware of the problem and working to resolve it.
- **Social media updates:** Your marketing team may also want to post social media updates if the outage is wide-spread enough. This also helps alleviate the burden on Customer Support teams who might be inundated with calls. These updates can be short and simple, perhaps linking to the status page and letting people know that the team is working to address the issue.

Depending on the severity of the incident, some of these external communication methods might be unnecessary. For instance, a Sev 3 or 4 is unlikely to require social media updates and may not even require a customer email.

After the incident concludes, make sure to update your users to let them know that the service has resumed normal operation. You can even consider publishing a public-facing incident retrospective once the review process has taken place.

Review

After each incident, teams should incorporate a review as part of the incident response process. This allows teams to learn more from each failure and feed their learnings back into the development cycle, potentially even preventing similar incidents from happening the same way later on.

Reviews are called many things: postmortems, RCAs, After-Action Reviews (AARs) and, our personal favorite, incident retrospectives. Whatever your team decides the term is, your review should contain some or most of the below information to help you get the most out of your incidents. Here are some guidelines you can use while writing your retrospectives.

Summary: This should contain 2-3 sentences that gives a 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 for 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? Adding tags can be helpful here as well to help with future reporting, filtering and search.

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: With the increase in system complexity, it's harder than ever to pinpoint a root cause for an incident. Each incident might have multiple dependencies that impact the service. Each dependency might result in action items. So there is no single root cause.

Narrative: This section is one of the most important, yet one of the most rarely filled out. The narrative section is where you write out an incident like you're telling a story. 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.

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.

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.

More important than a rigorous procedure is ***blamelessness***. Teams and individuals need to feel psychologically safe to fail. Nobody is to blame for an incident. "Human error" is actually an indicator of a systemic problem. Getting to the bottom of this problem can be tricky, however. There are usually many contributing factors when it comes to incidents rather than a single root cause.

Rather than taking the easy way out and blaming a person's actions, dig in. There is almost always more than meets the eye. For instance, maybe someone deleted an entire database. It's too easy to simply blame the person, and it really doesn't solve any problems.

As you look deeper into the incident, perhaps you realize that the person who made this mistake had just finished a tough on-call shift and worked from 2 AM to 6 AM on a separate incident. That's a sign that you need better on-call relief.

Maybe you should also question why there were no (or at least no *effective*) guardrails to act against this issue. Do you need better documentation? Runbooks? There are many considerations here, and none of them should be hastily brushed off with blame.

Blame also actively works against problem solving. Teams worried about punishment become risk-averse. They won't bubble up issues or contribute meaningfully to discussions if they fear becoming a scapegoat for the incident. Important knowledge remains buried. By setting standards and processes around your retrospectives, you can make it harder for blame to have a seat at the table.

Templates like this one can help teams respond to incidents faster, learn from failure, and can streamline and codify the process. However, they are not the only resource teams can look to. Tools can also make a difference in how prepared you are for responding to incidents. Here's how tooling helped Iterable.

How Iterable sees a 43% reduction in critical incidents

Iterable's growth marketing platform enables organizations to deliver seamless, personalized customer experiences across channels, including email, SMS, mobile push and more. In less than seven years, the platform has scaled to **billions** of cross-channel messages sent *per month*.

The company's mission— as implied by its name — is enabling companies across the digital maturity spectrum to iterate on messaging that maximizes customer engagement. Scaling its technology as quickly as possible while protecting reliability has been core to Iterable's explosive growth.

The SRE team at Iterable is focused on optimizing reliability of the platform. A key initiative the team is championing, for example, is working in lockstep with the go-to-market team to improve stability and predictability in scaling its database investments. By using projections to right-size its Elasticsearch indexes, the SRE team can reduce the risk of reliability issues, especially in the case of large customers with vast amounts of data (a whopping **tens of billions of data points** for their largest customer!).

The Challenge: Shallow Incident Learning and Platform Instability

In 2018, before adopting the Blameless platform, Iterable lacked a defined incident management process. Incidents were created on an *ad hoc* basis, and postmortem reviews were conducted only for the highest severity issues, largely due to the toil of creating postmortems. The VP of Engineering spent **2-3 hours generating each postmortem timeline**. He was also spending a disproportionate amount of time on calls with unhappy customers.

The team recognized the need for automated incident coordination and streamlined learning to prevent repetitive issues. At the time, the team had been considering building similar functionality in-house, but realized the effort would cost multiple months of full-time engineering work, as well as ongoing maintenance. **As a result, according to Staff Site Reliability Engineer Tenzin Wangdhen, the investment in Blameless was a "no-brainer."**

Pain Points Before Blameless

- Arduous process (2-3+ hrs) to build postmortem timeline
- Overall platform instability, resulting in time spent appeasing disgruntled customers
- Lack of feedback loop to address incident contributing factors, creating a “treadmill” of burnout

Goals

- Shift to a culture of learning from incidents
- Automate incident coordination and scale response processes
- Avoid building a solution internally

According to Tenzin Wangdhen, Staff SRE, “Blameless is one of those solutions where you forget it’s there because it’s so well-ingrained into our system. It’s really blended into our day-to-day toolkit due to its ease of use and intuitiveness.”

The Solution: Centralized Coordination and Actionable Insights

After adopting Blameless, incident creation and coordination now takes place in seconds. One team member contrasted that with previous organizations where simply creating an incident could take upwards of an hour.

Furthermore, Blameless has helped the Iterable SRE team facilitate a culture of learning. The team conducts weekly incident meetings — one with Customer Success and one with Engineering — led by the Incident Commander for the week. The team reviews all the Blameless incidents from the week before, and will bubble up questions and discuss next steps on follow-up items. **Blameless’s centralized charting uncovers actionable insights on where to focus their reliability efforts.**

Reliability Toolchain

- Blameless
- Slack
- JIRA
- Datadog

The Business Impact

In the two years since using Blameless, Iterable has seen incident frequency shift from high-severity incidents (Sev0s & 1s) to lower-severity incidents (Sev2s & 3s), which is a positive signal. This means that **incidents used to affect a larger blast radius of customers, but as they now trend smaller; impact can be isolated to a smaller subset of users**. Lower-severity incidents also enable the team to more easily encapsulate impact and identify proactive opportunities to improve platform stability.

- 43% reduction of Sev1s and Sev0s (over a 6-month timeframe)
- Automated postmortem timeline creation, compared to 2-3 hrs previously
- Flexible reporting, especially around impacted services, impacted customers, and contributing factors
- Fewer repeat incidents

What's Next

Longer term, the Iterable engineering team is interested in integrating Blameless' SLO solution to provide even more granular visibility with its customers around platform reliability. SLOs create a concrete data point to validate when to work on stability instead of new features. The eventual goal is to surface real-time SLO metrics (e.g. success rate for an API endpoint) by each customer.

According to Tenzin, **there is very strong interest across the company in understanding incident and platform stability trends, reflecting the mission-critical role that embedded and core SREs play in enabling Iterable's success as a digital-first business.**

Ultimately, in partnership with Blameless, Iterable can continue focusing on scaling the platform to serve the world's leading marketing organizations.

"Blameless is a sticky product, and without it, there would be much more manual work for everyone involved. The SRE team would be more thinly stretched, and we wouldn't have as much bandwidth to work on key reliability initiatives."

The Blameless solution for incident resolution

Blameless helps teams like Procore, Eventbrite, Under Armour, and more with their incident resolution needs. Blameless Incident Resolution allows teams to minimize the costs of coordination in critical moments by automating key incident tasks, centralizing context, and capturing details in real time so distributed teams can focus on essential decision-making.

Ready to see how Blameless can help your team? Try Blameless today or schedule a demo here.

About Blameless

Blameless is the first end-to-end Site Reliability Engineering platform, trusted by leading teams such as Home Depot, Mercari, and Citrix. With integrated service level objectives, incident resolution automation, toil-free learning, reliability insights, and more, teams are empowered to deliver amazing software experiences, and optimize innovation velocity without sacrificing reliability. Headquartered in San Mateo, Calif., Blameless is backed by Lightspeed Venture Partners and Accel.