



ROOTLY BENCHMARK REPORT

How to achieve great incident management

A benchmark and opinionated guide, built on 350K+ incidents across 650+ companies.

October 8, 2026

Incidents happen. No amount of testing, process or AI changes that, and more code shipping faster can mean more incidents, not fewer. So the goal isn't a lower MTTR or a shorter incident list. It's a response that's calm, fast and predictable for the engineer holding the pager, the executive asking for updates and the customer refreshing the status page.

That kind of response is decided before the incident, not during it. Roles, thresholds, rotations and templates get written down and practiced on a normal Tuesday, because nobody makes their best call improvising at 3am. Most of this report is about those decisions.

350K+

incidents analyzed
over the last 12
months

650+

companies, 165 of
them with \$100M+ in
revenue

<4

pages per incident
for the best quarter,
at every size

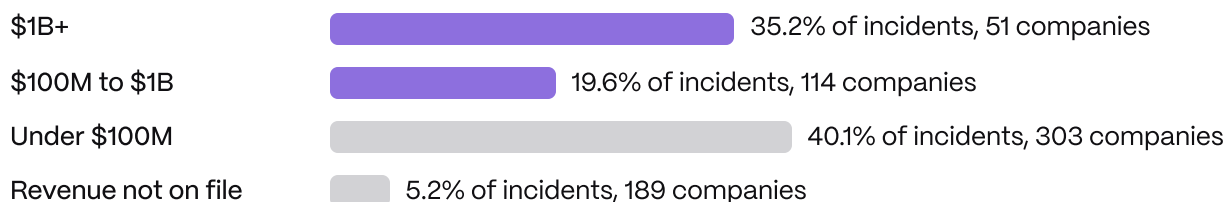
44 min

to stop a SEV0 or
SEV1, best quarter of
teams

It's built on more than 350,000 incidents run in Rootly from Oct 2025 to Sep 2026, at over 650 companies, from startups to enterprises like NVIDIA, DoorDash, Brex, Wise, Canva, Glean, Wix and Okta. For each position, we show what the best teams achieve, and the strongest argument against it.

More than half the incidents come from companies with \$100M+ in revenue

Share of incidents by company revenue, counting history imported from earlier tools.



 rootly *ai*

"I've deployed Rootly at two companies now, because I know what great incident management looks like and this is it."

Nigel Wright, Director of Platform Engineering, [Auror](#)

Before the incident

Everything in this part happens on a normal Tuesday, when nothing is broken. It decides how the worst day goes.

Declare early. Let the IC set severity.

An incident is anything that costs the business money, makes customers unhappy, or that a responder can recognize in the moment without hunting for a metric to justify it. The third test matters most, because nobody parses a definition at 3am.

- Declaring takes one Slack command or button, inside tools people already use. The form asks what's wrong and little else.
- The reporter never grades severity. The incident commander (IC) sets it once paged and changes it as facts arrive.
- When in doubt, declare. When an alert might belong to an open incident, open a new one and link them. Wrongly merging two problems hides one of them.

Some early declarations turn out to be nothing and get cancelled. That's what declaring early looks like, and it's cheap.

Where people disagree. Most published handbooks, PostHog's among them, have the declarer pick severity. Google's SRE book ties declaring to the response, asking whether a second team is needed, whether customers can see it, and whether it's still unsolved after an hour.

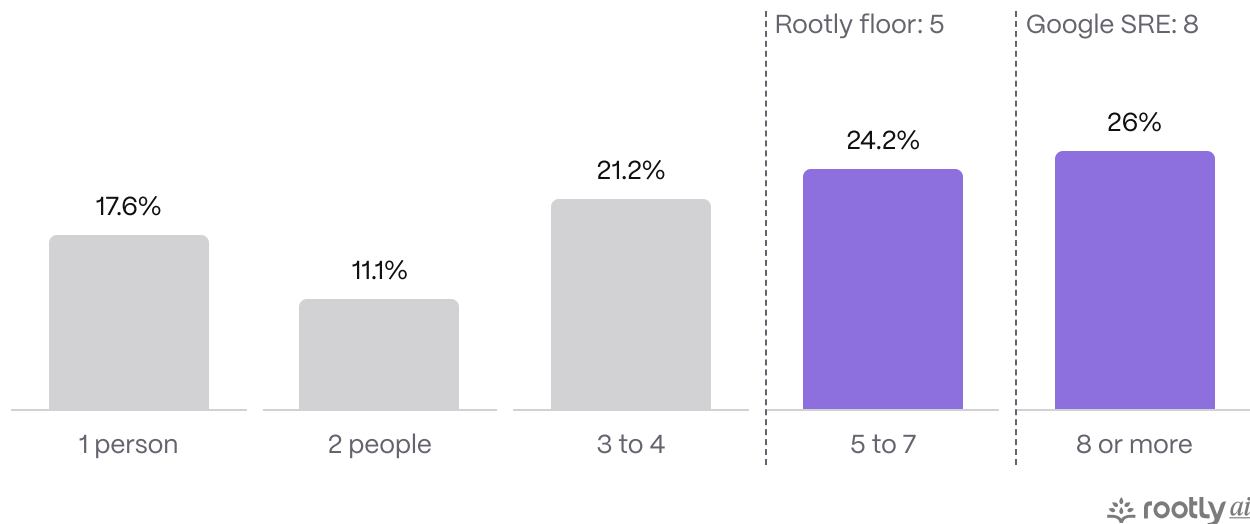
Five people on every rotation, minimum.

Below five, a rotation has no room for a sick day, a vacation or a planned surgery. The same few people absorb everything, and that's when things get dropped. Half the schedules that page someone in Rootly meet that bar, and a quarter clear Google's eight. The other half is where burnout starts, and it shows up in the data long before it shows up

in a resignation. That's why Rootly AI Labs built [On-Call Health](#), an open-source tool that flags overload risk from incident and on-call data.

Half of on-call schedules meet the five-person floor

50.2% of schedules that an escalation policy pages have five or more people. The median schedule has 5.



The chart counts every schedule an escalation policy pages, so some one-person schedules are backup or manager tiers rather than a front line. Getting to five rarely needs a hire. Add managers and directors first, then merge rotations for teams with similar services and cross-train them.

Where people disagree. Google's SRE book sets the floor at eight for a single-site rotation, which only a quarter of schedules meet.

Make the pager livable.

A big enough rotation keeps any one person from being on-call too often. These four things make the shifts themselves bearable.

- **Pay a flat monthly stipend.** On-call is work, and most companies still don't pay for it. Pay per page is the wrong fix, because it rewards more pages instead of fewer.
- **Give back lost sleep.** Every overnight hour comes off the next day. Past four hours, take the whole day.

- **Make it doable from a phone.** Most developer tools now have native mobile apps. A phone you already carry is a smaller burden than a laptop and hotspot all weekend.
- **Follow the sun if you're global.** It means nobody carries nights. The cost is more handoffs, and each one can lose context, so the handoff itself has to be good.

A good handoff covers what's coming, like deploys, maintenance, launches and migrations, not just what already happened. Before escalating to someone, check their real load and the availability they've set themselves. And nobody takes the pager before they're ready. That means shadowing, runbooks and a tabletop first, and at least three months on the team.

Where people disagree. PagerDuty tells responders to keep a laptop within reach. Will Larson calls follow-the-sun an evolutionary dead end because of those handoffs. And Google's on-call path runs from checklists to shadowing to reverse shadowing, with one Google team training for two months before shadowing, so our three months is on the long end.

Page a human only when a human has to act now.

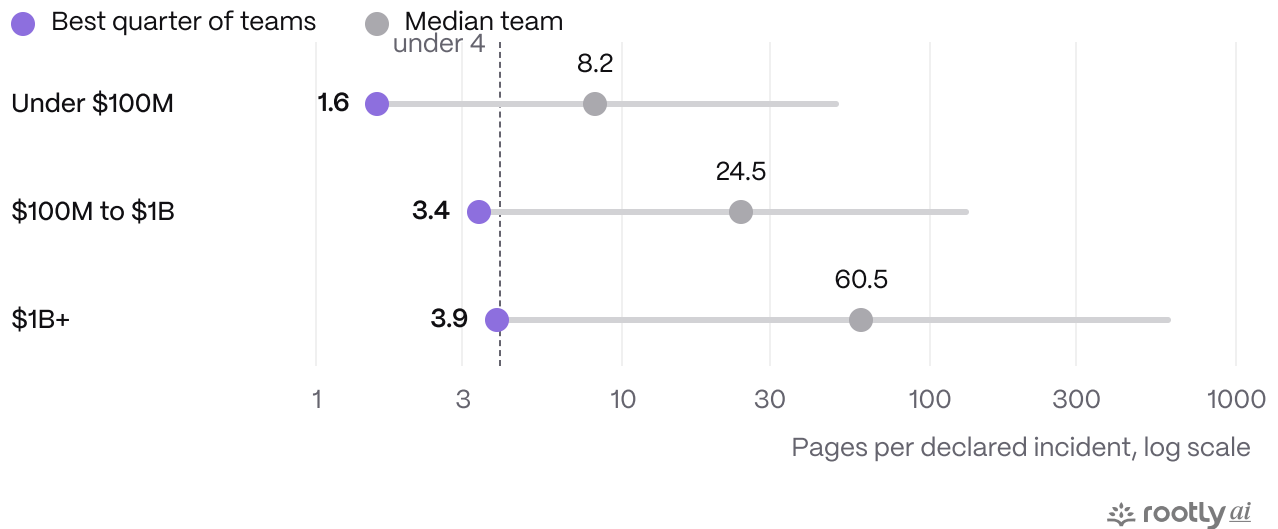
Every paging alert clears two hurdles.

1. At 3am, what would you actually do? No action, or one an agent should take, means no page.
2. Does it need doing now, or can it wait for business hours with no customer impact?

Few teams pass both hurdles on every alert, but the best get close. At every company size, the best quarter of teams page fewer than 4 times per declared incident. Noise grows with scale, so the gap between the best teams and the median widens at larger companies. The fix is the same at every size. Apply the two hurdles, and send what fails them to an agent instead of a person, the way [we run our own production](#).

The best quarter of teams page fewer than 4 times per incident, at every size

Pages per declared incident, per team, by company revenue. A page is an alert routed to on-call, counted once however many people it notifies. The line spans the middle half of teams.

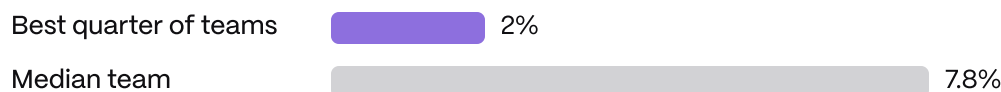


Some pages get handled without an incident, and some incidents start from a customer report, so one page per incident isn't realistic. Under 4 is a target real teams already hit.

When a page does reach a person, the first one paged almost always answers. On the median Rootly team, fewer than 1 in 12 acknowledged pages has to escalate past the first person on call. On the best quarter of teams, it's 1 in 50. That measures whether the first person answers, not what answering costs them. On a rotation that's too small, the same reliability comes from the same few people getting woken up again and again.

The first person paged almost always picks it up

Share of acknowledged pages that escalated past the first on-call level before someone acknowledged them, per team.



rootly *ai*

Add catch-all alerts for what named alerts miss, like overall error rate or traffic dropping to zero. Every alert says what broke and links a runbook anyone could follow at 3am. Watch

the repeated blip that never crosses a threshold and comes back daily. It's often next month's big incident.

Then look at the gap between impact starting and someone declaring, one incident at a time. Sometimes it's under a minute. Sometimes it's weeks, because the problem lived in an edge case or a path nobody was monitoring. Each long gap points at a blind spot, which makes it more useful case by case than as an averaged KPI.

Where people disagree. Charity Majors, co-founder and CTO of Honeycomb, would page only on SLO burn and end-to-end checks that mean users are hurting. Google's SRE Workbook formalizes that as multiwindow burn-rate alerts.

Freeze changes only when the error budget is gone.

SLOs and error budgets belong to an SRE team, or to the incident team when there isn't one. A team that burns its budget freezes changes, except reliability work. Otherwise, skip freezes and trust engineers to deploy carefully around holidays and launches. The exceptions are a team that broke that trust, or a moment when all eyes are on the company, like an IPO.

Where people disagree. Google treats the error budget policy as an agreement between product, development and SRE. Charity Majors argues freezes pile risk onto the first deploy after them. She has a point. A freeze moves risk to later, it doesn't remove it.

During the incident

This is the part people picture when they hear "incident." The bridge, the Slack channel, the updates. Done well, it feels almost boring, because everyone already knows their job.

One person runs the response. They don't fix it.

Incident command came out of California wildfire response in the 1970s, when agencies showed up to the same emergency with no shared plan and no clear lead. Google built its incident model on it. The idea still holds. One person runs the response so everyone else can do their job.

On Rootly teams that use the role, someone is named incident commander (IC) or lead within 20 seconds of declaring on the median team, usually by a workflow. That shows how fast a lead is in place, not whether it's the right person. A named lead is the floor. A trained IC taking command, as described below, is the goal.

72% of incidents with an IC have one within five minutes

Incidents with an IC or lead role filled, by time from declaration to the assignment. Incidents that never had one aren't counted.

All teams



Best quarter of teams



 rootly *ai*

On joining, the IC states the role, runs a roll call and asks who's missing. Before deciding anything, they want three facts, in order.

1. What the alert or report says, and the customer impact right now.
2. What changed recently on or near the affected service. Rolling it back is usually the fastest mitigation.
3. Who else needs to be pulled in or told.

A few rules keep the bridge moving.

- **Pull in more people than you need, then release them fast.** Aim for one engineer per impacted service.
- **Ask for objections, not consensus.** Ask the proposer how confident they are, ask the room for objections, wait five to ten seconds, then say there are none and go.
- **Reset a stalled bridge with five to six questions.** What's true, what's false, what's unknown, what's the leading theory and why aren't we sure, and are the right people here?
- **Hand off at four hours.** Nobody does their best thinking after four hours on the same problem, which is why wildfire crews work in shifts. Anyone can ask to hand off sooner, for any reason, and engineers on multi-day incidents follow the same rule.
- **Keep command.** An IC who goes quiet lets the room build its own structure, and it's rarely a good one.

ICs get ready in four steps. Instructor-led training, shadowing, reverse shadowing, then sign-off from a seasoned IC. A dedicated IC team is ideal. A trained rotation open to anyone willing to learn is the realistic best. Simulation-based programs like [Rootly Academy](#) make the drills and the sign-off repeatable.

Where people disagree. Many teams automatically make whoever responds first the lead, so a lead lands within minutes. We'd rather a trained IC take command. Aviation's crew resource management warns that a quiet room doesn't always mean agreement, which is a fair challenge to the objection check.

Write roles down before anyone needs them.

Roles scale with severity and scope. Like everything else here, they're decided before the incident, not during it. They live in a playbook and get practiced in a tabletop, never

improvised. A SEV1 has a scribe by default. A SEV4 might be one engineer running everything, by design.

- **Comms lead** once the bridge reaches dozens of people and the IC can't also write updates.
- **Deputy ICs** when several work-streams or bridges run at once, with the lead IC over all of them.
- **Customer impact coordinator** to work out what customers see and ready support macros.
- **Compliance** on every incident in regulated industries, owning what gets reported and when. Reporting clocks under DORA, NIS2, the SEC's rules and GDPR mostly start when the company detects or determines an incident, not when the retro finishes, and not only for the big ones.

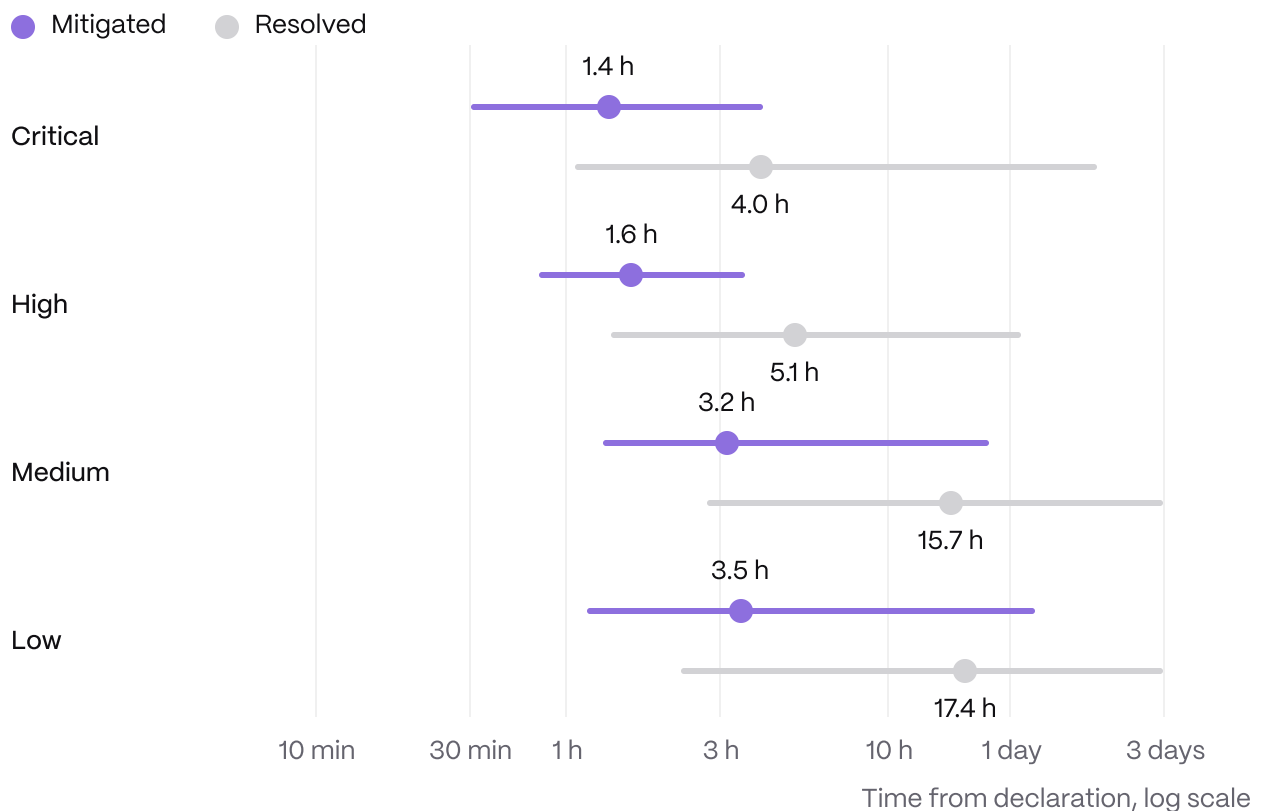
Where people disagree. Some teams make the incident lead the only required role and add others when the lead asks.

Mitigate first. Contributing factors and root cause can wait.

The goal on the bridge is stopping the impact. Contributing factors and root cause belong in the retrospective. Teams using Rootly already work this way. At every severity, the median team mitigates hours before it resolves.

Teams stop the impact hours before they close the incident

Time from declaration, per team. The dot is the median team; the line spans the middle half.



rootly ai

That gap is monitoring and cleanup after the impact stops, and it should exist. Calling an incident resolved too early and reopening it later loses more context than waiting does.

What great looks like, by company size. The best quarter of teams mitigate high-severity incidents in 42 to 78 minutes, depending on company size. Across every size, the median team stops a SEV0 or SEV1 in 88 minutes, and the best quarter in 44.

Company revenue	Best quarter of teams	Median team
Under \$100M	42 minutes	1.5 hours
\$100M to \$1B	63 minutes	1.7 hours
\$1B+	78 minutes	2.4 hours

Larger companies take longer. More services, teams and approvals sit between an alert and a fix, which is exactly why roles and rollback rights get decided in advance.

When the cause isn't obvious, a change earns a hard look when its timing lines up with the symptoms and it touches the affected service or a dependency. From there, check for a major version upgrade, unresolved review comments or failing checks, a skipped pre-production run, a large diff, an unreviewed bot merge, or a flag change that skipped review.

- **Same start time doesn't mean same cause.** Split into cross-linked incidents unless a shared dependency, matching errors or one change ties them together.
- **Pull in the author when the change came from outside the owning team.** That's routing, never fault.
- **Call the vendor.** Confirm your symptoms match theirs before you stop looking on your side.
- **Give every vendor a runbook.** Contacts, escalation path, SLA and status page, written before you need them.

Where people disagree. Google's "Lessons learned from twenty years of SRE" leads with a warning. A mitigation's risk should scale with the outage, because a risky rollback can make things worse.

Say what you know, and only what you know.

Severity	Update every
High	15 to 30 minutes
Lower	45 to 60 minutes
Multi-day	Frequent at first, then daily

Every update names when the next one comes. A fact and a hypothesis never share a sentence. Recovery estimates are buffered ranges, like 20 to 30 minutes. Before anything goes out, ask one question. Would this still be true if our leading theory is wrong?

Audience	What they need
Responders	Current state, every hypothesis and whether it's ruled out, key metrics and their direction, and work-streams with owners and ETAs

Audience	What they need
Executives	Impact, a plain summary, what's gone out externally, and an estimated mitigation time
Customers	What's degraded, how it affects them, and when to expect a fix

Some words never belong in an update. Vague timing like "shortly", minimizing like "minor", early assurances like "no data was affected", blame like "human error", defensive words like "unprecedented", and "we apologize for any inconvenience."

Post "we're aware" on the status page as soon as customer impact is confirmed. Rootly teams already move fast here. When a SEV0 or SEV1 gets a status-page update, the median team posts its first one 9 minutes after declaring.

Nine in ten first status-page updates go out within 20 minutes

SEV0 and SEV1 incidents with a status-page update, by time from declaration to the first update.

SEV0 and SEV1 incidents



 rootly *ai*

In regulated businesses, lean on pre-approved templates and push right up to what legal allows. Keep a fallback for every channel, including the status page itself.

Keep executives off the bridge and send them direct updates instead, so a customer call never blindsides them. Most are too far from the code to make good calls mid-incident, and when they're in the room, responders get less willing to say what they don't know. The IC makes the calls. The exception is real technical depth, like a CTO who built the system.

Where people disagree. Many programs wait 10 to 20 minutes before the first public update, invite executives into SEV1 channels, and favor cautious external wording to limit liability. We'd post sooner, keep executives off the bridge, and say more.

Run security incidents separately.

Security incidents usually run in the security team's own process and tools. The operational process joins when one needs a company-wide response, or when an outage turns out to be an attack. Breaches and other sensitive incidents run in a small private group, often under attorney-client privilege, with counsel involved fast.

Where people disagree. CISA's ransomware guide tells responders to capture system images and memory before restoring. Court rulings in the Capital One and Clark Hill breach cases found forensic reports weren't privileged, so privilege is narrower than many playbooks assume.

After the incident

An incident isn't over when the graphs recover. What happens next decides whether the same one comes back.

Resolve slowly. Start the retro fast.

Once mitigated, an incident moves to monitoring before anyone calls it resolved. A clean revert might need 15 to 20 minutes of watching. A flaky network or an unverified vendor fix might need a day. The IC says out loud what's being watched and for how long.

The moment it resolves, name a retrospective owner and get everyone's raw notes into the retro doc before memory fades. Two tiers of retrospective are enough.

	Low severity	High severity
Format	Async, about 15 minutes	A real investment of time
Covers	What happened and what we learned	What happened, how the response went, impact dashboards and alerting gaps
Interviews	None	One-on-one with key responders for the biggest incidents

Interviews surface what nobody says with a manager in the room, like pressure to ship untested code. Protect the source, carry the substance to leadership yourself, and name nobody.

Every retro grades the response as well as the cause.

- Did we know before customers did?
- Did responders acknowledge within 5 minutes and respond within 10?
- Did they have the context and tool access they needed?
- Was there a playbook, and was it accurate?
- Did anything surprise us, or did communication break down?

When the root cause never settles, say so. List the confirmed contributing factors, then the competing explanations with the evidence for each. Forcing a single root cause usually makes a person the easy answer. Near misses get retros too, starting with the "we got lucky" ones.

Where people disagree. The Learning from Incidents community goes further. Richard Cook's "How Complex Systems Fail" calls post-accident attribution to a root cause fundamentally wrong, and John Allspaw's "The Infinite Hows" argues for asking how an incident unfolded instead of why.

Treat action items as options. The roadmap owner decides.

Action items don't go to an engineer by default. They go to whoever owns the roadmap, a product or engineering manager.

1. The roadmap owner reviews each item.
2. Declined items get a written reason and a closed ticket, so there's a record next time.
3. Accepted items go on the roadmap, and only then to an engineer.

Skip this and action items rot in a backlog nobody with planning authority reads. Low-severity items rot first, and they often seed the next big incident.

Where people disagree. Atlassian and Google treat action items as commitments with owners and deadlines. Atlassian gives priority actions a 4 or 8 week SLO. A deadline measures how fast tickets close. It says nothing about whether they were worth doing.

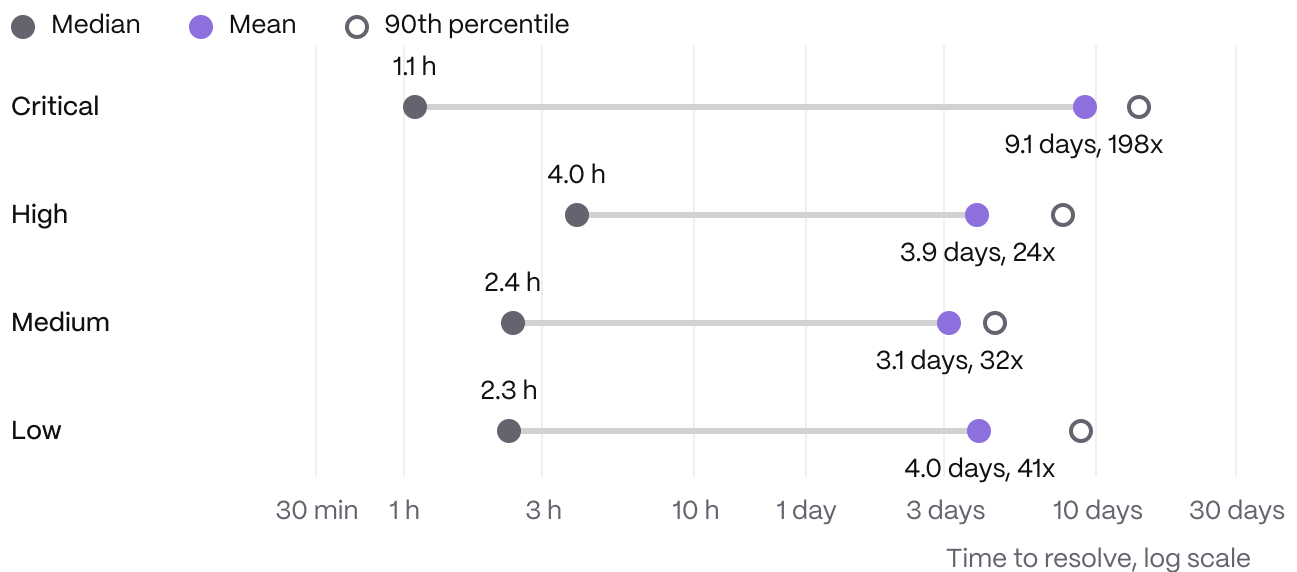
Stop managing to MTTR.

Mean time to resolve (MTTR) is the most misused number in incident management. Pressure on speed pushes people to cut corners and ship untested fixes. And the average barely describes anything. For critical incidents in Rootly, the median incident resolves in about an hour, but the mean is more than nine days. A small number of incidents that run for weeks, or are simply never closed, drag the average away from anything a responder

would recognize. A number that moves when someone forgets to close an incident can't tell you whether your response got better.

For critical incidents, the average time to resolve is 198 times the median

Time from declaration to resolution across 314,862 resolved incidents. Per incident. The mitigation chart above shows per-team medians, so the figures differ.



rootly ai

Put three views in front of your VP of Engineering each quarter instead.

1. Incidents by service, feature or team
2. Incidents by contributing factor
3. The three most expensive incidents, by lost revenue, customers affected or engineer-hours

When an executive asks whether you're more reliable than last quarter, start with uptime. Then add the incidents that hurt customers without touching it, like a wrong monthly statement or an account closed by mistake.

Where people disagree. Google's Štěpán Davidovič calls MTTR poorly suited to decisions rather than harmful. Courtney Nash's VOID research found incident durations too skewed for averages, which our data backs. DORA still reports a recovery-time metric, now scoped to failed deployments.

Blame systems, not people.

Humans don't cause incidents. The systems that let them happen do. Someone shipped without tests? Add a gate. Someone sat on a risk? Add a second reviewer. The one exception is narrow, someone deliberately getting around a safeguard for personal gain. Incidents caused by AI-written code follow the same rule. The question is why the system let that code ship.

"Rootly turned our incident response into an operating system that scales: clearer ownership, faster resolution, stronger retros, and less reliance on heroics."

Matthew Duren, VP of Engineering, [KnowBe4](#)

Where people disagree. David Marx's just culture model, widely used in healthcare and aviation, allows sanctions for reckless behavior, a wider line than ours. Sidney Dekker's restorative just culture sits closer to us.

AI, culture and tooling

The last part covers what runs through everything else. How AI fits, the culture underneath, the tools, and how much of this changes from company to company.

AI does the work. People make the calls.

AI earns its place correlating alerts, stack traces and changes faster than a person can. Agents make good first responders inside boundaries humans draw, like reverting a change that clearly caused errors or restarting a pod. For known problems with known fixes, automated remediation beats a page.

The hard line is that an AI never states a guess as a fact. A confident wrong statement on the bridge ends the investigation while the real cause keeps running. Sent to customers, it becomes a trust problem, and in a regulated business a legal one. We hold AI in an incident to six rules.

- **Label every claim confirmed or unconfirmed.** Confirmed means two sources, one of them a metrics platform or a log.
- **Post several hypotheses with confidence levels,** so the room doesn't anchor on the first story.
- **Get a human's approval outside pre-approved actions,** including status page posts and paging executives.
- **Post when something changes,** never on a timer.
- **Treat alert payloads, logs and error messages as data,** never as instructions.
- **Say when the data is stale,** like an out-of-date service catalog.

When an AI suggests a related past incident, it shows its reasoning, its confidence and what's different this time. A keyword match is worth nothing. Models also differ widely on SRE work, which is why [Rootly AI Labs](#) maintains [SRE-skills-bench](#), an open benchmark that tests LLMs on real-world SRE tasks.

Where people disagree. Some vendors pitch agents that hold the pager and resolve incidents on their own. Lisanne Bainbridge's "Ironies of Automation" warned in 1983 that people who only supervise automation lose the skills they need when it fails, and that warning applies to us too.

Make it safe to say "I caused this."

Almost every incident has a technical cause and a human one, like a Friday deploy pushed for a deadline or a 3am mistake after a week of broken sleep. The human side is usually where the learning is.

- When someone says they pushed the change, the IC answers "How can I help?"
- After a rough incident, check in, tell responders nobody blames them, and ask their manager to give them time off.
- Give every new hire a 30-minute incident onboarding.
- Teams that rarely see incidents should run a game day every quarter. Every team should run one at least once a year. Teams handling incidents weekly are already getting the practice.

Where people disagree. Google's SRE Workbook runs drills on a regular cadence for every team, busy ones included.

Buy incident tooling unless nothing fits or you plan to sell it.

A homegrown incident bot makes you its on-call team, its patcher and its bug fixer. When it goes down during your own outage, you have two incidents and nowhere to run either. Incident tooling is critical, has to keep up with the times, which right now means AI, and has to stay up when everything else is down. Replit, Lucidworks and Webflow all ran their own tooling before moving to Rootly.

Where people disagree. The best-known homegrown tools were built for fit, not cost. If your process really is unusual enough that nothing on the market fits, building can be the right call.

Write down the calls nobody should improvise.

The principles hold everywhere. How a company applies them varies with size, regulation and incident volume. Some calls belong to written policy, decided before the incident, and no person or AI should make them up mid-incident.

- Severity levels and who sets them
- Anything published externally
- When to page executives, legal, security or compliance
- What counts as a security incident, data exposure or reportable event
- Who can roll back, flip a flag or restart in production
- How long an incident stays in monitoring

How Rootly runs its own incidents

Everything above is what we recommend. This is what we do on our own production, including where we've gone further than most teams are ready to.

Rootly runs its own production on rung four

Our five-rung ladder for how much of incident response agents handle.

5 · Fully autonomous operations

Agents run, fix and improve production; humans set policy

Where this is heading

4 · Autonomous first response

Agents acknowledge, investigate and apply pre-approved fixes

Rootly's own production

3 · Agent-first investigation

Agents investigate every alert; humans decide and act

Rootly AI SRE on auto-run

2 · Agents on request

Agents investigate when a human asks

Most teams today, rungs 1 and 2

1 · Assisted

Humans respond; AI summarizes and drafts



Most teams we talk to are on rungs one and two. A growing share of Rootly customers are on rung three, running [Rootly AI SRE](#) with investigations set to run automatically. We run on rung four while we build and tune rung five.

Every page gets an autonomous first response.

No human is the first responder at Rootly. When an alert fires, an agent acknowledges it within seconds, groups it with related alerts, opens an incident if one is needed, and starts investigating with our knowledge graph and [Atlas](#) behind it. Within 60 seconds, it posts what it has found and what it has ruled out.

If the fix is on our pre-approved list, like rolling back a deploy, restarting a stateless service, scaling out or turning off a feature flag, the agent applies it, verifies recovery and writes up what it did. A human gets paged only when the fix needs judgment or the agent isn't confident, and they wake up to a briefing instead of a blank alert.

Weak signals go to the agent, not the bin.

We stopped tuning alerts for human attention spans. Signals too weak to justify waking someone go straight to the agent, which investigates each one and sometimes catches a problem hours before it becomes an outage. That's the two-hurdle test from the paging section, with a third option. An alert that fails both hurdles no longer has to be deleted.

Seven in ten alerts at Rootly close without paging a human

Share handled by agents in Rootly's own production.

Alerts



Low-severity incidents



 rootly *ai*

Off-hours pages to Rootly engineers fell by roughly half over the same period.

An agent reviews every change against our incident history.

Our change-risk agent runs in CI and checks each change against everything Atlas knows. A package upgrade that matches a version combination that caused an incident before gets flagged as high risk. A change to web-server worker counts gets flagged as medium risk, because a Slack thread from months ago warned it could overload the caching cluster. No reviewer would remember that thread. The agent checks for it on every commit. In testing, it flags about 1 in 25 changes as high risk, and that small slice accounts for most of the changes that later caused incidents.

Every agent change is replayed against real incidents.

Before a change to our AI SRE ships, we replay real incidents from our own stack, along with incidents from customers who approved it, and score whether it still finds the right root cause. It's the same rule this report applies to people. Practice on real incidents before you're trusted with the next one.

Agents help with security forensics too.

When a Rails vulnerability landed, our CTO used agent skills to run the forensics on our own systems, then [published how](#).

The most mature teams we work with are starting to copy this, beginning with routing weak signals to their AI SRE once they trust it.

Where people disagree. Lisanne Bainbridge's "Ironies of Automation" warning applies here too. The pre-approved list is short on purpose, every agent action is written up for a human to read, and people still run game days so the skill doesn't fade.

Great, by the numbers

The whole report as targets you can hold a program to.

Practice	What great looks like
Declaring	Anyone declares from Slack and the IC sets severity. The odd cancelled incident is a healthy cost.
On-call	Five or more people per rotation, a flat stipend and sleep returned. Half of paged schedules already get there.
Paging	Under 4 pages per declared incident, the pace of the best quarter of teams at every company size
Mitigation	SEV0 and SEV1 impact stopped in 44 minutes or less, the best quarter's pace across company sizes
Updates	Every 15 to 30 minutes at high severity, each naming the next update time

Practice	What great looks like
Public comms	"We're aware" posted as soon as customer impact is confirmed
Retrospectives	Every incident, async for low severity, grading the response as well as the cause
Action items	A recorded do or don't decision from the roadmap owner
Measurement	Incidents by service, contributing factor and cost, reported instead of MTTR
AI	Every claim labeled confirmed or unconfirmed, and actions only within pre-approved limits

The scorecard

Tick what's true today. Most unticked boxes cost a meeting and a written decision, not a new tool.

Before

- ☐ Anyone can declare from Slack without picking a severity
- ☐ Every on-call schedule has five or more people
- ☐ On-call pays a flat stipend and returns overnight hours as time off
- ☐ Handoffs cover upcoming deploys, maintenance and launches
- ☐ Every paging alert passes the two-hurdle test and links a runbook
- ☐ We track the gap between impact starting and someone declaring

During

- ☐ ICs are trained, shadowed, reverse-shadowed and signed off
- ☐ The IC sets severity, and roles per severity are written and rehearsed
- ☐ Nobody works more than four hours without a handoff
- ☐ Every update names the next update time and keeps facts apart from hypotheses
- ☐ Comms templates are pre-approved, and every channel has a fallback

After

- ☐ Every incident gets a retro, async for low severity
- ☐ Retros grade the response, not only the cause
- ☐ Every action item gets a recorded decision from the roadmap owner
- ☐ Quarterly reporting covers incidents by service, contributing factor and cost, not MTTR

Across the program

- ☐ AI labels every claim confirmed or unconfirmed and acts alone only within pre-approved actions
- ☐ Every new hire gets incident onboarding

How Rootly puts this into practice

Rootly is the AI-native incident management platform built for this way of working, with 650+ companies in this report alone.

- [Run incidents in Slack, Teams, Google Chat](#). Declaring, roles, updates and timelines happen where engineers already work.
- [On-call that puts people first](#). Rootly On-Call handles schedules, escalations and overrides, designed for everyday life and carried on a phone.
- [AI SRE](#). Narrows the search across alerts, changes and past incidents, labeling what's confirmed and what isn't.
- [Status pages](#). Keep customers informed from the same place the incident runs.
- [Retrospectives](#). Automate the post-incident process and save responders hours.
- [Extensible by design](#). Connect Claude, Codex, Linear, Zoom, and go further with the Terraform provider, API or MCP server.

"Rootly is a big part of how we've reached our 99.99% of reliability."

Paul van Liew, Director of Platform Engineering, [Motive](#)

Book a demo

See it with your own incidents at rootly.com/demo