The VOID REPORT 2021

How We Learn From Incidents and Outages

VOID

In Partnership with

indeed®

"Incident analysis is not actually about the incident, it's an opportunity we have to see the delta between how we think our organization works and how it actually works."

—Nora Jones, Jeli.io [1]

Table of Contents

Introduction

04/22/21

Dozens of major businesses came to a near halt around the globe.

These businesses included those in retail, banking, financial services, and major news outlets. Just six weeks earlier, nearly the same thing happened. The source of each outage was identified as a Content Delivery Network (CDN), which helps other organizations' websites and online services run faster. The two CDN companies involved—Akamai and Fastly, respectively—resolved the issues relatively quickly, and issued statements apologizing for the issues while promising plans to prevent them from happening again.

Many of the impacts were merely minor inconveniences, people had to wait to make online purchases or play their favorite video games. Others were more consequential: major airline delays, unresponsive banking apps, and hampered government services—for the British government, this limited access to critical public services, such as their portal for booking coronavirus testing.

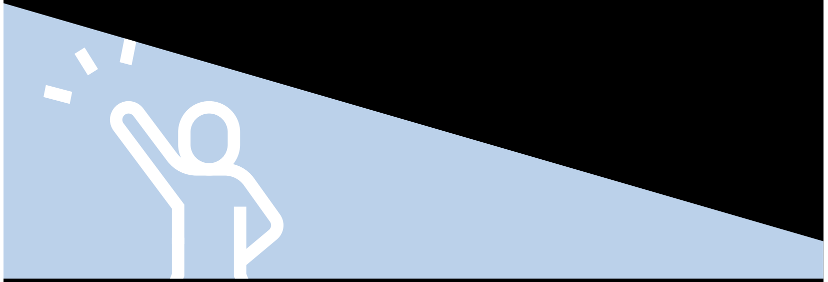
While many of us have gotten used to software taking an ever growing place in our lives, it may be less obvious the extent to which it has become safety-critical.

Slack and Microsoft Teams are used for coordination and communications in emergencies unrelated to the software industry; Facebook, Twitter, and other social media are fundamental to public policy; malfunctioning developer toolchains can take down entire industries; and a worldwide pandemic has made virtual communication tools central to global economies.

Modern online systems are expected to run 24 hours a day, 365 days a year. These increased pressures—combined with software models of interrelated, increasingly automated services that run in the cloud—have accelerated the complexity of these systems.

Incidents arise from software systems that are inherently complex *sociotechnical* systems. They comprise code, machines, and the humans who maintain them. The humans operate via their own mental models of an environment shaped by multiple competing pressures, many of which are completely invisible. Given the complexities of and pressures on these systems, it is impressive that the vast majority of the time, they function effectively. Yet as any operator knows, they do fail, and often in unexpected and unusual ways. And now, these failures affect us all.

What is the VOID?



The Verica Open Incident Database

The Verica Open Incident Database (VOID) collects public software-related incident reports and makes them available to everyone in a single place.

Collecting these reports matter. Software has long since moved beyond hosting pictures of cats online to running transportation, infrastructure, power grids, healthcare software and devices, voting systems, autonomous vehicles, and many critical societal functions.

Much the same way airline companies set aside competitive concerns in the late 1990s and beyond in order to improve the safety of their industry, the tech industry has an immense body of commoditized knowledge that we can share in order to learn from each other and push software safety forward. Expertise in incident analysis exists as its own skill set, distinct from software engineering. If we want to improve as an industry, we can't just stumble around in the dark or imitate large players without understanding why their approach may not work or which caveats apply.

An open database of incidents is the first step to improvement. By casting a light on our industry and practice, we can gain the visibility required to establish more solid foundations.

Executive Summary

As of the time of publishing, the VOID contains **1,818** reports

for

599 organizations.

The reports span from September 2021 back through 2008, and include the following formats:

- social media posts
- status pages
- blog posts
- conference talks
- news articles
- comprehensive retrospectives/postmortem reports

It is an inherently incomplete dataset, in that such organizations are not mandated to investigate, write up, or share incident reports. (We discuss some of the limitations of these kinds of public reports later on).

Key Findings

- Over Half of Incidents Are Externally Resolved in Under Two Hours
- MTTR is a Misleading Metric
- Root Cause Analysis is Rarely Used
- Nearly No Near Misses Are in the Dataset

Over Half of Incidents Are Externally Resolved in Under Two Hours

We looked at the distribution of duration data for reports that provided this information (which was about half of the total reports in the VOID). Of that subset of reports with duration data, over half (53%) reported that the incident was resolved in under two hours.

MTTR is a Misleading Metric

Based on the distribution of the data we see in the VOID, measures of central tendency like the mean, aren't a good representation of positively-skewed data. The mean will be influenced by the skewed spread of the data, and the inherent outliers. Central tendency measures of incident duration are inherently too noisy to be measures of success for incident response.

Root Cause Analysis is Rarely Used

Only about ¼ of the reports in the VOID follow some form of Root Cause Analysis (RCA), or at least explicitly identify a "root cause" of the incident. We are interested in RCA because, like MTTR, it is appealing in its decisiveness and apparent simplicity, but it too is misleading and can foster a culture that focuses more on attribution of cause and less on understanding how the system ended up failing.

Nearly No Near Misses Are in the Dataset

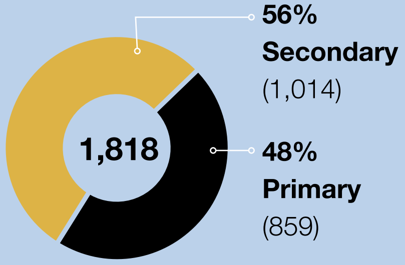
A near miss is an incident that the organization noted had no noticeable external or customer impact, but still required intervention from the organization (often to prevent an actual public incident from eventually happening). As of today, there are only seven near-miss reports in the VOID—not even a half of a percent of the total entries. We're interested in near misses because they often include much richer information about sociotechnical systems, including breakdowns in communication, dissemination of expertise, and cultural or political forces within those systems.

The Data:
What We Know (and What We Can’t Know)

Methodology

Total VOID Reports

The total number of primary and secondary reports in the VOID is 1,818. Some records have both sources for the same incident, making the total of primary + secondary more than the total number of reports.

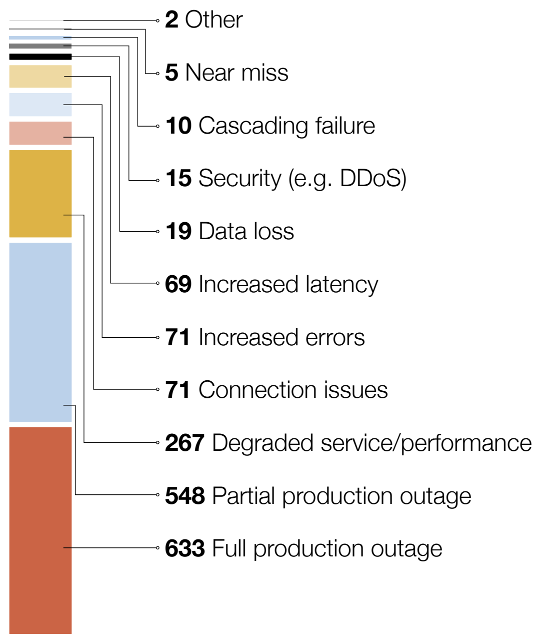


The VOID includes everything from tweets to status page updates, conference talks, media articles, and lengthy in-depth company post-mortems. We track these by whether they are primary or secondary sources: primary sources are authored by someone at that organization, and anything external to the organization is considered a secondary source.

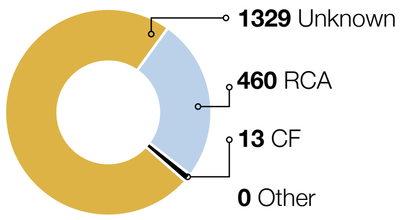
Nearly all the information in the VOID comes directly verbatim from the report artifacts themselves, along with a set of metadata that we've collected based solely on the report content. The metadata collected include:

- **Organization**
- **Date of incident**
- **Date of report**
- **Report type**: Primary and secondary
- **Duration:** If available, either directly from the report, or calculated based on information in the report
- **Technologies involved:** This reflects what technologies were listed as contributing to the incident, if present in the report
- **Impact type:** We tag incidents based on language in the report (when available), and there can be multiple tags per report. These are intended to serve as a jumping off point for exploration, and do not represent a formal classification system. See the current set of tags to the right.
- **Analysis format:** If noted, we track what kind of analysis is used in the incident report. See the current set of analysis methods to the right.

Impact Tag



Analysis Format



A Caveat

“Generalizations may be possible in retrospect, given enough detailed data and enough understanding of the data. But this means understanding details of the task, the context, the environment, and its constraints. Simply constructing taxonomies is grossly insufficient and it permits only counting of incidents that fall under phrase a, b or c of the taxonomy.”

—Charles Billings

Generalizations and Quantitative Analyses

It is tempting to think we can look at this set of reports as a corpus of incidents that we can directly compare and contrast with each other. However, as Charles Billings (the architect of NASA’s Aviation Safety Reporting System) notes,

> “Each incident is unique and not easily classified or pigeonholed.”

The VOID contains reports spanning single-digit-person startups to massive, global, multinational, publicly traded corporations with hundreds of thousands of employees.

Their business models, products, strategies, funding, legal liabilities, and execution all vary wildly—and so do the data in these reports. They reflect everything from increased error rates for a matter of minutes to multi-hour (in some cases, multi-day) outages that impact downstream organizations and their customers and users as well.

Taxonomies can also perniciously turn normative, where the richness of some incidents’ attributes are lost into having to fit them into existing categories. We discuss this in more detail in the section on what we can’t conclude from the reports in the VOID.

Finally, because of their voluntary nature, we can’t determine the incidence or prevalence of a problem solely from public software incident reports. Due to the myriad ways complex systems fail, there is too much noise and variance in the data from which to deduce meaningful significance (we demonstrate this in the Key Findings section later). As such, most of the findings currently available in this report are more patterns and directional signals for further investigation.

The nature and degree of a "disruption or reduction" will not be the same across organizations.

Inconsistent Definitions

Generally speaking, a software-related incident has some kind of undesirable impact on users, customers, or other individuals external to the organization. We'd broaden this loose definition to include the fact that organizations have internal-only incidents (e.g. downtime for developer tooling; discovering that production database backups are corrupt) which we almost never hear about. While these aren't visible outside the company, they can be quite critical and urgent in an all-hands-on-deck way.

Referring to the more general safety community, OSHA defines an incident as "an unplanned, undesired event that adversely affects completion of a task." In the context of software, "completion of a task" can be broadened to cover anything that users, customers, operators, or other stakeholders expect that service to do. In it's incident response handbook, Atlassian defines an incident as the "disruption to or a reduction in the quality of a service which requires an emergency response." And in Chapter 28 of Seeking SRE, John Allspaw and Richard Cook use the term *incident* as "a pointer to a set of activities, bounded in time, that are related to an undesirable system behavior."

These definitions contain a lot of language left open to interpretation. The nature and degree of a "disruption or reduction" will not be the same across organizations, nor will the extent, timing, or nature of what an "emergency response" constitutes. Some organizations have formally-defined Service-Level Agreements (SLAs) and/or Service-Level Objectives (SLOs) that determine when a system is operating outside it's desired boundaries that can inform what constitutes an incident, while many do not. Additionally, who decides whether something is or is not an incident also varies across organizations.

Public Incident Reports Are Inherently Incomplete

It can be tempting to use these reports as records of *what really happened* in an incident. But most of the time, public incident reports are heavily redacted and remove many details from what an internal report would ideally contain.

This is in large part due to who the intended audiences are for such writeups. This audience comprises a mix of customers, partners, investors, analysts, and the media. We do also see a portion of reports that are written for other engineers, either in the spirit of sharing learnings, and/or for recruiting purposes. Take note if a more technical incident report contains a link to the company's hiring page!

John Allspaw provides an excellent breakdown of the differences between internal and public incident reports. From his post:

> "Internal reports that focus on learning from an incident tend to include:
>
> - An **informative description** of how the various mechanisms, composition, and dynamics of the technical systems involved in the event work–what they're expected to do, what their relevant history entails, etc.
> - A better understanding of what **operational vulnerabilities, fragilities, or surprises** can show up in their systems.
> - A **much richer picture** about what to look out for in designing or operating similar systems in the future.
> - A better understanding of **what was difficult** for people as they wrestled with what was happening in the event, and what made those difficulties… difficult, including context and history about how a system ended up the way it was.
> - **A greater appreciation for what pitfalls or minefields exist to lead them to mistaken understanding(s),** what options they might generate for taking actions to remedy the situation, what options are not available to them, what steps may make the situation worse, etc.
>
> External (public) reports, on the other hand:
>
> - Are very often **cherry-picked from the stream of incidents** the company experiences, and typically are written only about severe and/or high-profile events.
> - **Rarely link to other public posts about incidents published in the past.** To make a connection from one incident to another (for example, a "repeat" incident) would undermine a core purpose of public posts: to assure the audience that the incident is sufficiently understood and future prevention is "ensured."
> - **Serve more to reassure** and apologize than to enlighten or educate."

Given these factors, it's important to highlight a few key things that the VOID can't tell us.

What We Can't Know

"A central question facing us is not really how many [reports] there are, but how many is enough."

—Charles Billings[3]

How Many Incidents are Happening

No organizations are exempt from incidents, but most don't write them up publicly. What's in the VOID is a small sample of a larger population, that likely reflects the more severe or noteworthy events that organizations feel compelled to publish. (The counterpoint to this is the growing trend for organizations to maintain status pages which provide details on everything from very minor issues to large-scale, major events.)

We *can* note major large organizations that are conspicuously absent, like Apple, Facebook, and Amazon (AWS does occasionally publish detailed incident reports for major outages, but they're also hard to find and not clearly indexed on their site). We have entries for these companies, but they come primarily in the form of media articles fed by social media or Down Detector reports, not information provided directly from the organization.

Leading Causes of Incidents

It will be tempting to look at the metadata about the technologies involved or outage types, and look for conclusions like "configuration changes account for X% of software-related incidents." This is problematic for multiple reasons.

First, as Billings noted above, taxonomies that categorize incidents dramatically reduce the detail available for an incident. They only support shallow categorization, which can lead to misleading conclusions. For example, 'configuration changes' can hide organizational pressures or challenging communication patterns that get buried under a technical explanation. We are collecting these metadata because we believe they can be useful to practitioners looking for technologies and tools they are interested in using.

We assume that they are willing to properly research them by pulling information from detailed reviews of incident reports, and complementing them with a wider variety of sources.

Every organization or team has their own local context, rife with their specific technical requirements, internal and external pressures, and social and political systems.

What Organizations Are Learning From Incidents

As we demonstrate in the following section, the data from the VOID paint a general picture of the landscape of software-related incidents, but they don't show whether a very heterogeneous group of organizations is getting better at preventing, responding to, or learning from incidents. Until organizations that rely on software undertake the effort to analyze their incidents with the aim of learning from them and sharing their results, we're still largely operating independently in the dark.

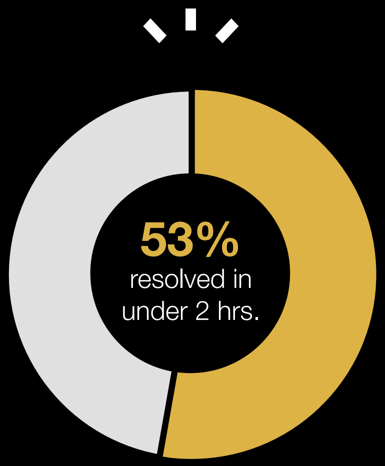
But now with the VOID available to everyone we can establish some baselines to track over time, so let's look at the preliminary results.

Complex systems run as broken systems. The system continues to function because it contains so many redundancies and because people can make it function, despite the presence of many flaws."

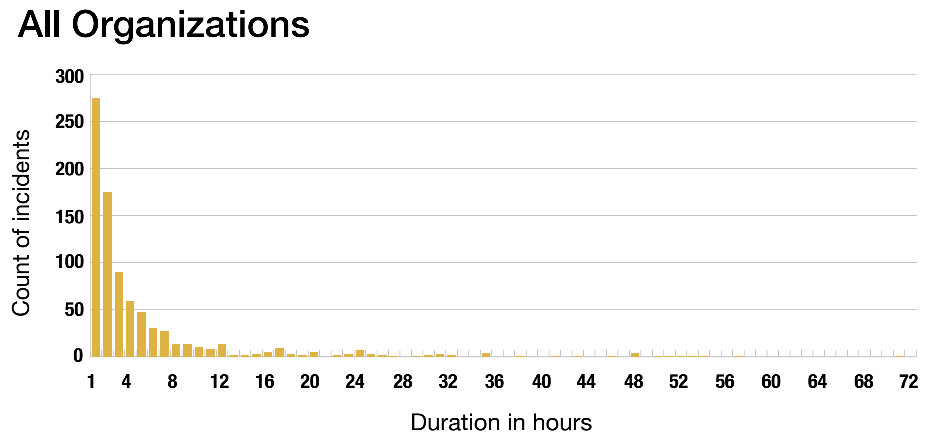
—Richard Cook

Key Findings

Over Half of Incidents are Externally Resolved in Under Two Hours



The expertise that operators bring to bear works relatively quickly more often than not.



We looked at the distribution of duration data for reports that provided this information (which was about half of the total reports in the VOID). Duration in this context is calculated as the time from detection to the time that the organization considered the incident to be externally resolved or remediated. Of that subset of reports with duration data, over half (53%) indicated that the incident was externally resolved in under two hours. An important caveat is that these data generally only reflect the public-facing aspects of incidents, and don't encompass the total amount of time and effort required of the internal team. Additionally, like judgements of the severity of an incident, what constitutes the start and end time are negotiable and vary across organizations. We'll discuss the limitations of these duration data in detail, but first we wanted to look at the distribution of the data.

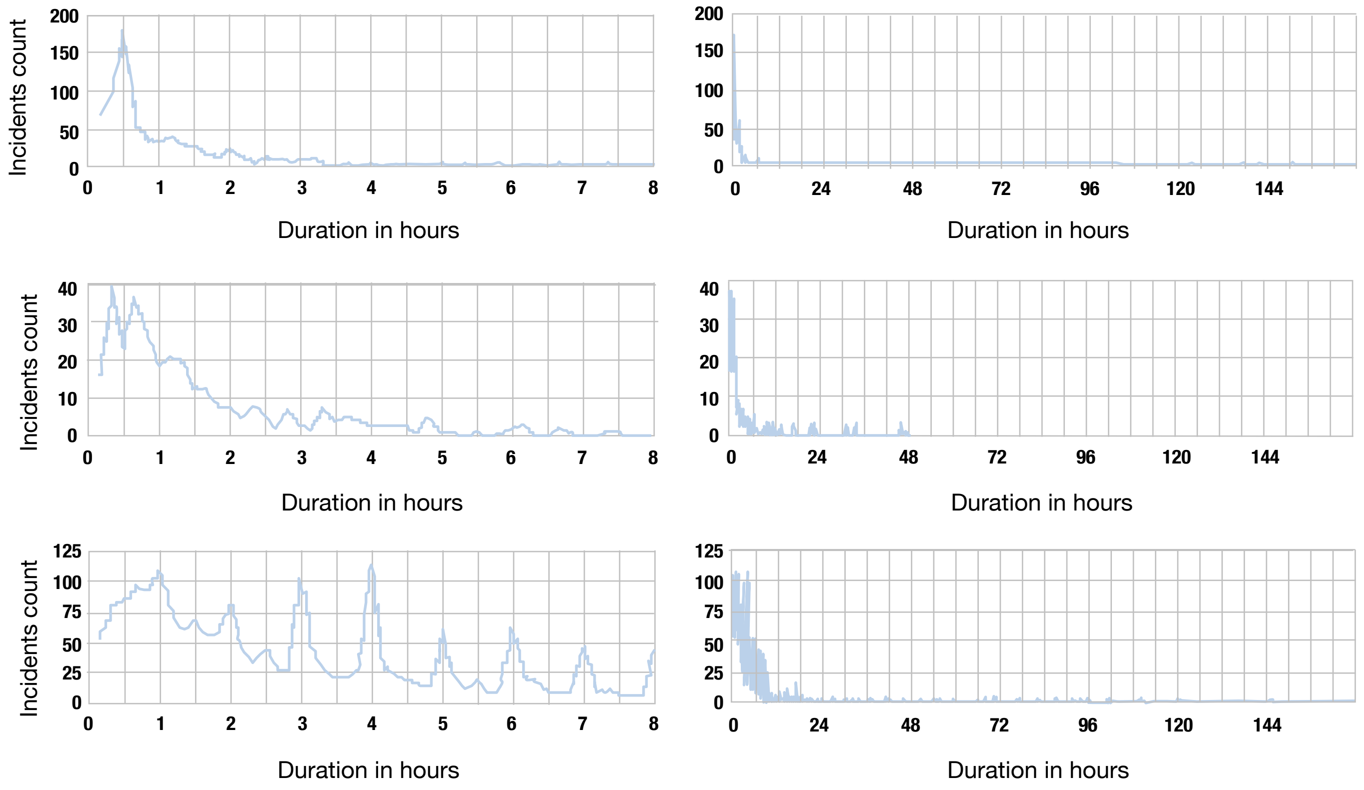
Our initial finding demonstrates

the consistency of a positively-skewed distribution of duration within and across organizations (including the entire corpus of reports in the VOID for which we had a determined duration), a phenomenon also found by Štěpán Davidovič in "Incident Metrics in SRE: Critically Evaluating MTTR and Friends." A positively-skewed distribution is one in which most values are clustered around the left side of the distribution while the right tail of the distribution is longer and contains fewer values.

Davidovič looked at the count of incident duration for three companies, and each one demonstrated a positively-skewed distribution (which appeared to fit a log-normal distribution) where most incidents were resolved in under two hours.

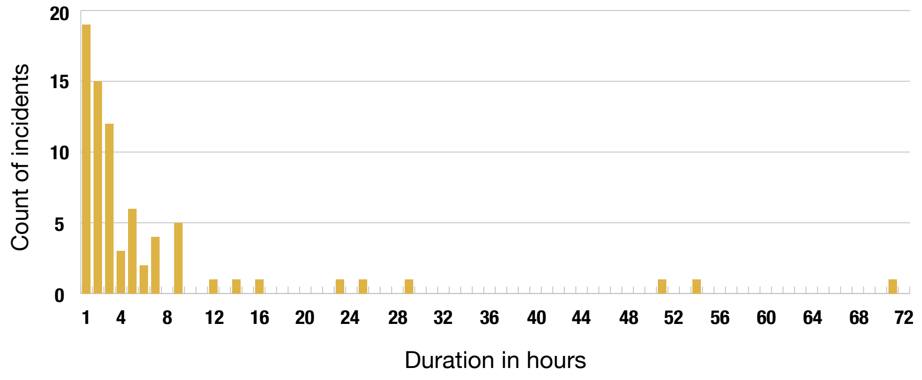
Distribution of Incidents' Durations with Incident Counts

Distribution of incidents' durations with incident counts. Rows are, in order, Company A (N = 798; 173 in 2019), Company B (N = 350; 103 in 2019), and Company C (N = 2,186; 609 in 2019). Columns represent each company over a short and long time frame to show the tail of the distribution. (Source: Google/O'Reilly)

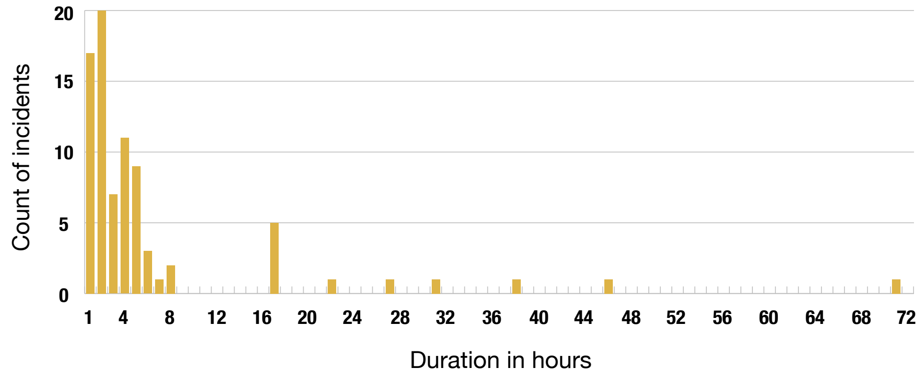


While we do not have over 100 reports for any single organization (yet) in the VOID, we were able to replicate this pattern across a number of them that had at least 40 reports:

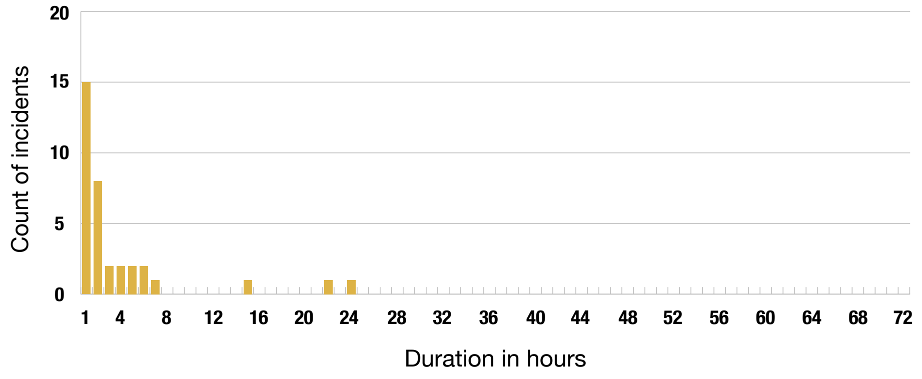
Heroku



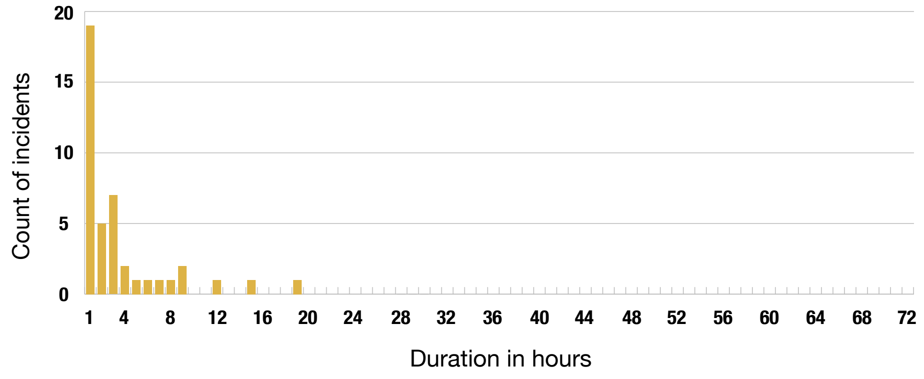
Google



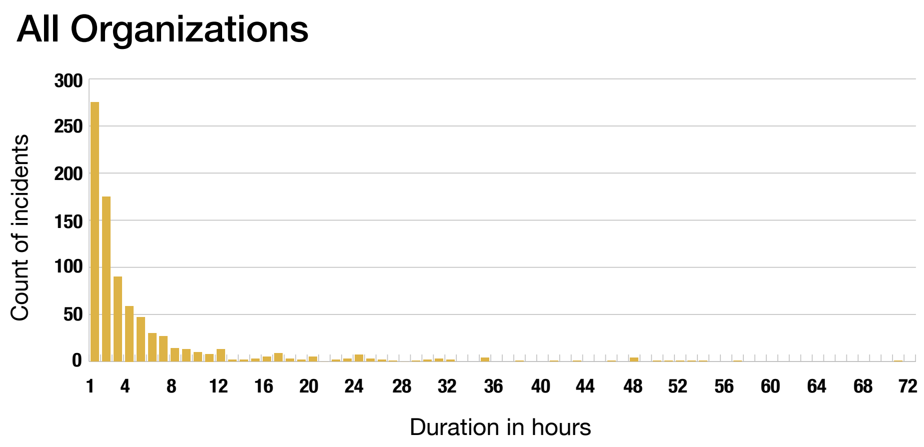
Honeycomb



Slack



Moreover, this distribution persists even if we chart the data from every single organization, regardless of size, frequency, or total number of reports:



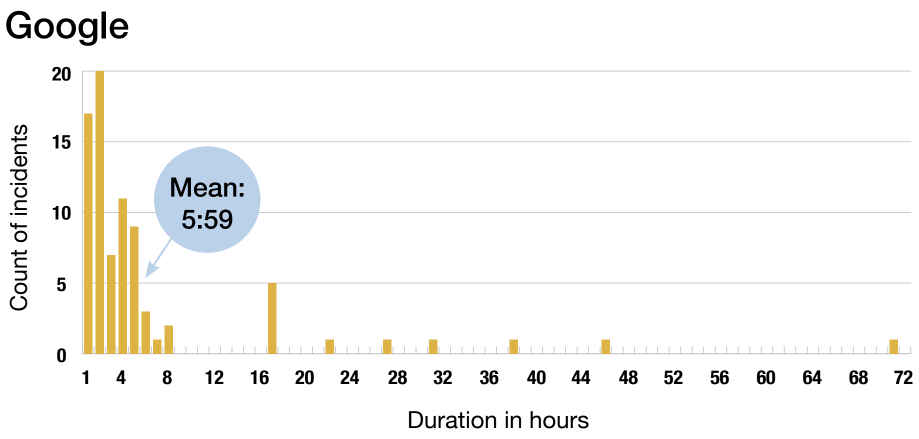
The consistency of this positively-skewed distribution across the wide variety of organizations in the VOID is noteworthy. We can't yet say why this pattern persists, though we suspect it is likely due to standard mitigations—rolling back, reverting a bad deploy, scaling up something that is saturated, etc.—working the vast majority of the time. The expertise that operators bring to bear works relatively quickly more often than not. However, there are limitations to what we can conclude from these duration data, which we cover next.

If the metric you're using isn't itself reliable, then how could it possibly be valuable as a broader measure of system reliability?

MTTR is a Misleading Metric

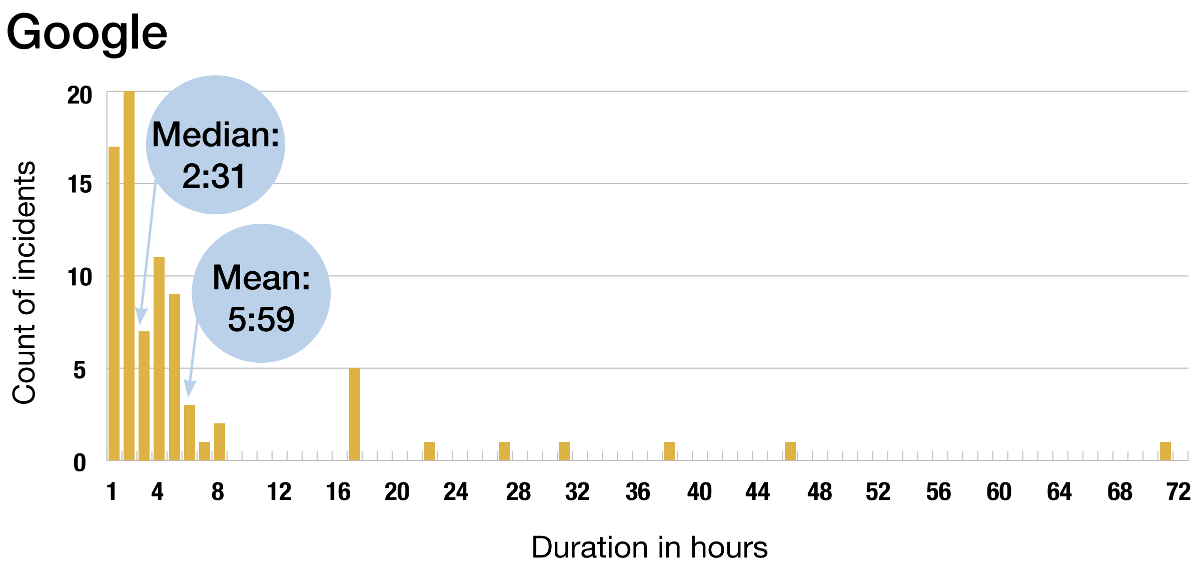
MTTR is trying to simplify something that is inherently complex.

Software organizations tend to value measurement, iteration, and improvement based on data. These are great things for an organization to focus on; however, this has led to an industry practice of calculating and tracking Mean Time to Resolve, or MTTR. While it's understandable to want to have a clear metric for tracking incident resolution, MTTR is problematic for a couple of reasons. The first is solely statistical: based on the data we see in the VOID (which replicate what Davidovič found), measures of central tendency like the mean, aren't a good representation of positively-skewed data, in which most values are clustered around the left side of the distribution while the right tail of the distribution is longer and contains fewer values. The mean will be influenced by the spread of the data, and the inherent outliers. Consider our previous Google data:



The mean is clearly well to the right of the majority of the data points in the distribution—this is to be expected for a positively-skewed distribution with a small set of large outliers.

So next you might consider a different MTTR: Median Time to Respond. The median value will be less influenced by outliers, so it might be a better representation of the data that you could use over time.



What actionable conclusions can you reach based on this information?

That certainly looks more representative, and your putative incident response metric just got 2.5 hours better simply by looking at it this way. Which begs the question: what actionable conclusions can you reach based on this information? Presumably, you calculate and track a metric like this in order to improve (or know when things are getting worse), which means you need to be able to detect changes in that metric. This is where Davidovič was able to demonstrate something really powerful, which is also applicable to these data.

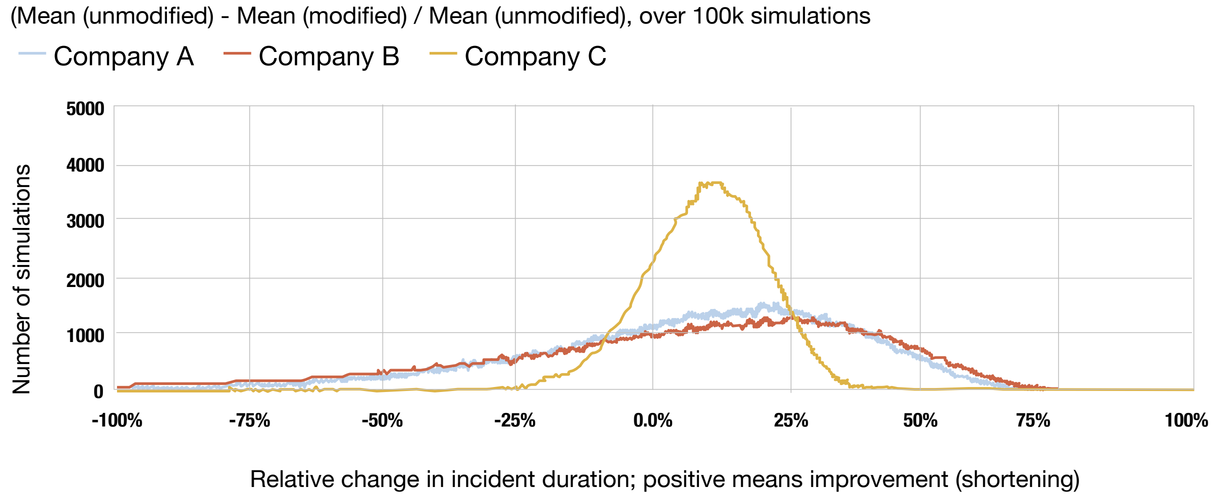
Davidovič demonstrated both with empirical data and Monte Carlo simulations how incident duration data do not lend themselves to reliable calculations of improvement related to any central tendency calculations of incident duration (or overall incident count). If you're not familiar with Monte Carlo simulations, think of it as an A/B test on simulated data instead of real-world production data. Davidovič took two different, equally-sized samples from three companies' incident duration data, and compared the two sets, one which had experimental changes applied to it (e.g. a 10% improvement in incident duration), and the other which had no changes and acted as the control. He then calculated the MTTR difference between the two samples over a series of 100K simulations, such that a positive value indicates an improvement, or shortening of MTTR.

Even more surprisingly, Davidovič ran the same simulation with no changes to either sample group. The results were similar to when an improvement was intentionally added to one of the sample groups. Even when nothing changed, between 10-23% of the time (depending on which company it was), the simulation returned improvements in MTTR between the sample sets. Davidovič concluded,

"We've learned that even without any intentional change to the incident durations, many [simulations] would make you believe that the MTTR got much shorter—or much longer—without any structural change. If you can't tell when things aren't changing, you'll have a hard time telling when they do."

Incident Duration Data Monte Carlo Simulation

Distribution of simulated changes to MTTR if improvement actually happened, as relative improvement.
Source: Incident Metrics in SRE (Google/O'Reilly)



He also went further to demonstrate that other measures—often called MTTx, including the median, geometric mean, absolute duration sum—all fell prey to similar problems with the high variance of the data (this applied for incident total counts as well). Given that our incident report data follow the same distribution, we can presume these effects (or lack thereof, depending on how you look at it) are also present in our data.

The length of an incident yields little internally actionable information about the incident.

MTTx Are Shallow Data

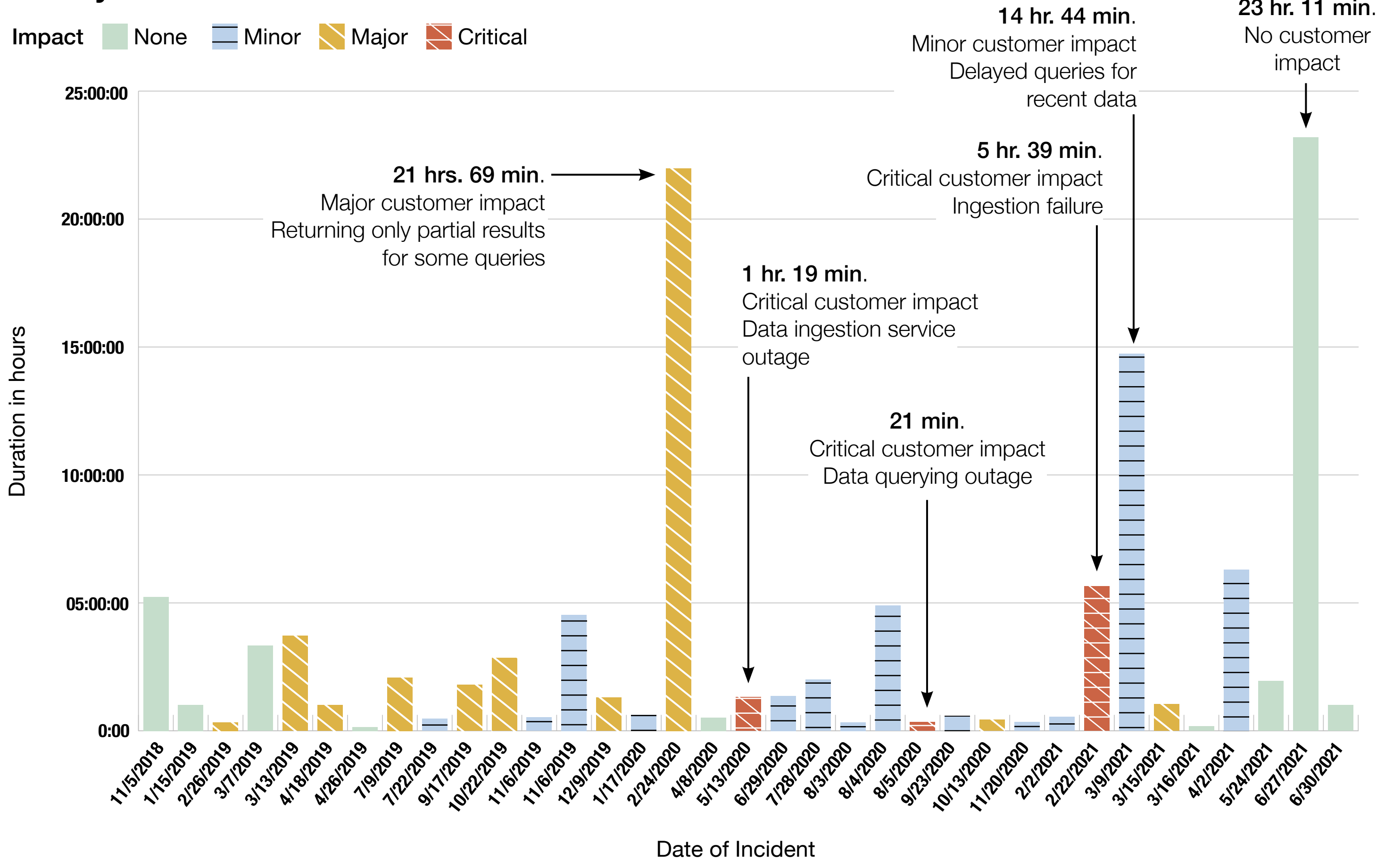
The second problem with MTTx metrics is they are trying to simplify something that is inherently complex. They tell us little about what an incident is really like for the organization, which can vary wildly in terms of the number of people and teams involved, the level of stress, what is needed technically and organizationally to fix it, and what the team learned as a result.

MTTx (along with other data like severity, impact, count, and so on) are what John Allspaw calls "shallow" incident data. They are appealing because they appear to make clear, concrete sense of what are really messy, surprising situations that don't lend themselves to simple summaries. As Allspaw notes, "Two incidents of the same length can have dramatically different levels of surprise and uncertainty in how people came to understand what was happening. They can also contain wildly different risks with respect to taking actions that are meant to mitigate or improve the situation." Conversely, the exact same set of technological circumstances could conceivably go a lot of different ways depending on the responders, what they know or don't know, their risk appetite, internal pressures, etc.

The length of an incident yields little internally actionable information about the incident. Consider two chess games of the same length but very different pace, expertise, and complexity of moves; feature films that generally all clock in at around two hours but can have wildly different plots, complexity of characters, production budgets; War and Peace (~1,200 pages) versus Slaughterhouse Five (~200 pages).

Could we add more to the current view of our data to get a little deeper and learn more? Let's take a closer look at duration over time data for Honeycomb, using their reported impact categories from their status page updates (None, Minor, Major, and Critical) to add more context to a few key incidents.

Honeycomb



This additional information provides a little more context, but it only casts more doubt on duration (or MTTR) as a useful metric. First, we see that duration alone tells us little about external customer or internal team impact: for three of the five longest incidents, the customer impact was either minor or ultimately nonexistent. And two of the three incidents with critical impact were resolved in 21 minutes and 79 minutes, respectively. Could this be because the critical incidents were all hands on deck, while the others allowed the organization time to investigate and resolve the issue with less urgency?

Does that mean that the response was "better" in one case than the other? Additionally, is a "major" incident similar across companies? We can't know these things either. This is the inherently limited nature of shallow data. We can't draw a sharp line around the nature of the impact of the incident, nor can we draw a sharp line around duration.

In the case of Honeycomb, however, we are fortunate to have additional, deeper information.

They chose to write a detailed meta-summary of a number of incidents from early 2021. The summary makes almost no mention of incident durations, beyond noting an especially painful 14-hour incident in mid-March. It also diverges from most public write-ups that we see in that it includes coverage of:

- Internal, business and system pressures (that led to a large-scale architectural migration)
- Detailed technical and organizational background information about decisions, surprises, and optimizations or trade-offs made by the team
- What was confusing, difficult, or surprising about the incidents they encountered, and the shifting performance envelopes that ensued

An increase or decrease in MTTR would not help the team determine the above insights about their systems—those came from a decision to invest the time and resources into investigating the messy details of the incidents, including the non-technical forces at play.

Moving Beyond MTTx

We argue that organizations should stop using MTTX, or TTx data in general, as a metric related to organizational performance or reliability. First and foremost, if you are collecting this metric, chart the distribution of your incident data. If they're positively skewed, then you're not measuring what you think you're measuring with MTTx, and you can't rely on it.

Second, ask yourself what decisions you're making based on those data. Are you prioritizing hiring SREs, or are you trying to justify budget or infrastructure investments? What could you decide given either a decrease or increase in TTx (were that a reliable metric)?

What could you decide given either a decrease or increase in TTx (were that a reliable metric)?

Given that MTTx metrics are not a meaningful indicator of an organization's reliability or performance, the obvious question is what should organizations use instead?

We'd like to challenge the premise of this question. While rolled-up quantitative metrics are often presented on organizational dashboards, quarterly reviews, and board presentations, they generally fail to capture the underlying complexity of software-related incidents.

That said, it may not be possible to immediately or completely abandon metrics that management or executives want to see. Vanessa Huerta Granda, a Solutions Engineer at Jeli, has an excellent post detailing a process of using MTTR and incident count metrics as a way to "set the direction of the analysis we do around our entire incident universe." They can be an excellent jumping-off point to then present the insights, themes, and desired outcomes

“Because we had that [qualitative] data we were able to make strategic recommendations that led to improvements in how we do our work. In addition, having that data allowed us to get buy-in from leadership to make a number of changes; both in our technology as well as in our culture. We beefed up a number of tools and processes, our product teams started attending retrospectives and prioritizing action items, most importantly everyone across the org had a better understanding of what all went behind ‘keeping the lights on.’”

—Vanessa Huerta Granda

If quantitative metrics are inescapable, we suggest focusing on Service Level Objectives (SLOs) and cost of coordination data.

of detailed qualitative incident analyses that highlight knowledge gaps, production pressures and trade-offs, and organizational or socio-technical misalignments, which are also an important form of data:

> “Because we had that [qualitative] data we were able to make strategic recommendations that led to improvements in how we do our work. In addition, having that data allowed us to get buy-in from leadership to make a number of changes; both in our technology as well as in our culture. We beefed up a number of tools and processes, our product teams started attending retrospectives and prioritizing action items, most importantly everyone across the org had a better understanding of what all went behind ‘keeping the lights on.’”

If quantitative metrics are inescapable, we suggest focusing on Service Level Objectives (SLOs) and cost of coordination data. A detailed explanation of SLOs is beyond the scope of this report, but Chapter 17 of Implementing Service Level Objectives covers using SLOs as a substitute for MTTx and other common problematic reliability metrics.

If people or staffing is something you need metrics for, consider tracking how many people were involved in each incident, across how many teams, and at what levels of the organization—these factors are what Dr. Laura McGuire deemed “hidden costs of coordination”that can add to the cognitive demands of people responding to incidents. [4]

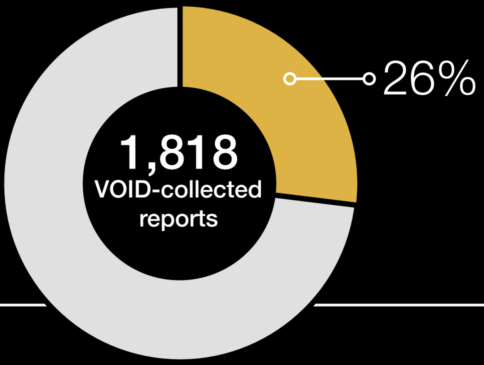
There’s a big difference between a 30-minute incident that involves 10 people from three engineering teams, executives, and PR versus one that an engineer or two puzzles over for a day or so, and is lower impact. (Such metrics also help inform the degree of internal intensity of a given incident, which can be useful when trying to manage and prevent burnout from incident response activities.)

Lastly, Granda notes that your metrics and data will evolve as your incident analysis progresses and matures—the more you tackle the basic issues, the more complex the next set will be, and your metrics and data will need to adjust accordingly.

Root Cause Is for Plants, Not Software

Root Cause Analysis Data

About a quarter of the incident reports either identify a specific “root cause” or have conducted a Root Cause Analysis (RCA)



Roughly a quarter of the incident reports (26%) either identify a specific “root cause” or explicitly claim to have conducted a Root Cause Analysis (RCA). We consider these data preliminary, however, given the incomplete nature of the overall dataset. As we continue to add more reports, we’ll track this and see if it changes.

We’re specifically looking into RCA because, like MTTR, it is appealing in its decisiveness and apparent simplicity, but it too is misleading.

An Artificial Stopping Point

At its core, RCA posits that an incident has a single, specific cause or trigger, without which the incident wouldn’t have happened. Once that trigger is discovered, a solution flows from it, and the organization can take steps to ensure it never happens again. This sequence-of-events mindset (also known as the “Domino model”) has its origins in industrial accident theory[5] and is common in incident reports, both for software and other domains.

As safety researcher Sidney Dekker describes in The Field Guide to Understanding ‘Human Error’ when people choose causes for events, “This choosing can be driven more by socio-political and organizational pressures than by mere evidence

“What you call ‘root cause’ is simply the place where you stop looking any further.”

—Sidney Dekker

found in the rubble. Cause is not something you find. Cause is something you construct. How you construct it and from what evidence, depends on where you look, what you look for, who you talk to, what you have seen before, and likely on who you work for."[6]

Let's approach this first with a software-centric thought experiment from Casey Rosenthal:

"Consider an example taken from a large-scale outage that was not publicized but resembles many high-profile outages.

A configuration change was deployed that brought down a service. The outage was big enough that time-to-detect was almost immediate. The time-to-remediate took a bit longer though, on the order of about 40 minutes. Service was restored by reverting one single line of the configuration file. It's tempting to say that the one line is the 'root cause' of the incident. Consider these possible alternative causes:

- The configuration wasn't wrong; rather, the code that interpreted the configuration file wasn't flexible enough.
- The person who wrote the offending line wasn't given enough context by their team, training, and onboarding process about the configuration to know how it would affect the system.
- The system allowed a configuration change to be deployed globally all at once.
- The parts of the organization closer to operations did not spend enough time with feature developers to understand how the latter might actually try to use the configuration file.
- The person's management chain did not set the appropriate priorities to allow for more testing, exploration, and review before the change was pushed.
- The peers who signed off on the change were so busy because of external resource constraints that they didn't have time to properly review the pull request.
- The company doesn't make enough money to support increasing the deployment costs high enough to run blue/green deployments whereby new configurations are run in parallel with the old until they are verified to be correct.

None of these alternatives are fixed as easily as reverting the one line. None of them suggest that they stem from the same 'root.' And yet, all of these alternatives are more fundamental to resilience than the one line of a configuration file."

Now let's consider another thought experiment before looking at some RCA excerpts from the VOID. Your team has finally shipped a major architectural change to support a new product feature that the Executive team is counting on for revenue this year. What is the root cause of that release? What single factor led to that success?

This is, of course, a ridiculous question. Dekker again: "In order to push a well-defended system over the edge (or make it work safely), a large number of contributory factors are necessary and only jointly sufficient." In the same way that successes and regular daily operations are the product of interrelated systems of people and machines, so too are their failures.

While we plan to delve deeper into more quantitative data (as possible) from the VOID about RCA, for the purposes of this initial, directional report we'll highlight a few examples of RCAs that are not so clear-cut.[7]

Root Cause Matryoshka: Salesforce, May 2021

This was a significant outage that resulted in a high volume of external communications from the company. With a strong theme of reassuring customers, the RCA kicks off with a paradoxically unusual start: a primary root cause that has contributing factors.

"Root Cause Analysis
After extensive investigation across multiple swimlanes, Salesforce has determined that the primary root cause has the following contributing factors, in priority order:

1. A lack of automation with safeguards for DNS changes

2. Insufficient guardrails to enforce the Change Management process

3. Subversion of the Emergency Break Fix (EBF) process

Before we dive into detail on each of these root cause themes, we'll first discuss the technical trigger for this incident with full transparency."

The language is largely focused on technical details, including the supposed trigger, which is a configuration change to the DNS servers. It's a seemingly simple "cause" but there are multiple socio-technical layers beneath it, including:

- The change was requested as part of normal maintenance work
- A script used to make the change that had been running for three years without issue, exposed a race condition under higher-than-usual traffic levels
- Change management and EBF (Emergency Break Fix) procedures that the engineer making the change did not follow

On the surface, this investigation comes across as rather cut-and-dry: Salesforce refers many times to its layers of defense, guardrails, policies and procedures, and regular training of engineers on how to follow them. In this case, the analysis revealed that someone didn't follow procedures, and the incident ensued (and was even exacerbated by "subverting" the EBF process):

"The investigation into this incident also showed areas that need to be hardened in the Change Management system. Our incident analysis leads us to believe the process would have worked in preventing this incident had it been followed, but insufficient guardrails allowed circumvention of the change management process."

But many questions remain unanswered, including:

- Where (or who) did the request for the DNS change come from? Was there a strong business need which influenced the perceived importance or urgency of the request?
- What about that request or the potential

business or production pressures led to making it in the middle of the week, during peak traffic hours, which is not their standard procedure?

- How normal is "subverting" the EBF process? Was it just this one time for this one incident?
- How many changes were happening in the system at this time? How many changes was this person making in the hours leading up to this change?
- What context is available for when and why that script was written, and what assumptions were made about its use?
- Are there other parts of their infrastructure that they've believed safe but potentially aren't given what they learned from this incident?

The conclusion of this report is one all too familiar with researchers in other safety-critical domains. The company vows to increase its defenses in depth, add more automation and guardrails, and thoroughly review and better enforce rules, processes, and procedures:

> "While every engineer takes an annual change management process training and has full awareness of Salesforce policies, this incident exposed gaps in the enforcement of these policies. As part of the Preventative Action plans coming out of this incident, Salesforce will be focused on policy enforcement improvements such as automated workflows and audits, the usage of EBFs across the enterprise, as well as the review and approval procedures for EBFs."

> "Instead of increasing safety, post-accident remedies usually increase the coupling and complexity of the system. This increases the potential number of latent failures and also makes the detection and blocking of accident trajectories more difficult."
> —Richard Cook[8]

Guardrails—notably for humans via additional or more thorough processes—and an emphasis on training are hallmarks of one approach to system safety called High Reliability theory. This theory posits that complex systems can be safely controlled if proper design and management practices are put in place. The irony of this approach is that, like automation in complex technical systems[9], additional guardrails, processes, and formalities add to the complexity and tight coupling of the system, leading to unforeseen tradeoffs which may themselves become contributors to future incidents.

In her book Engineering a Safer World, researcher Nancy Leveson provides an example of this regarding a safety mechanism put in place after a train accident. The proposed safety procedure would keep train doors from opening between station platforms, but that procedure also presents a hazard when passengers are trapped if the train stops between stations, like if it's on fire.[10]

A similar paradox of problematic process shows up in *Drift Into Failure*: "Closed loop communication was introduced as a safety practice on ships to avoid misunderstandings. Closed loop means that both the recipient and one's understanding of a message are explicitly acknowledged, for example by repeating the message. A maritime pilot, however, had observed that under certain conditions in his pilotage area, the changes of course and speed came so fast, that the helmsman's repeating of the pilot's instructions resulted in them talking at the same time (producing overlapping speech). This resulted in more communication, with more overlap. So he abandoned closed loop communication."[11]

The people involved in the incident are often disincentivized from acknowledging the existence of incidents or near-incidents for fear of being blamed.

High Reliability stands in stark contrast to what Charles Perrow termed Normal Accidents theory, which presumes that serious accidents or incidents are inevitable within complex high technology systems. From this perspective, the challenges organizations face when trying to improve safety aren't lack of procedures or change management processes—or people's failure to follow them—but rather things like:[12]

Highly ambiguous feedback. What happened in an incident is not always clear, and even well-meaning leaders may tend to only learn the lessons that confirm their preconceptions, assign success to actions they took (despite not knowing whether those actions would actually lead to success), and fit into their cultural views of the organization. Effectively, this is hindsight bias in action.

Political or organizational forces. Post-incident analyses are often driven less by an actual desire to learn from incidents, but rather to determine "causes" that protect the interests or reputations of the most powerful or influential people. Credit and blame must be assigned and action items listed to "prevent" such things from happening again. In such political situations, organizational learning is very likely to be highly biased, limiting the type of learning that is possible.

Fear of reprisal. Incomplete or inaccurate reporting from field operators (or practitioners) makes it difficult to assess organizational learning or performance. The people involved in the incident (what Sidney Dekker calls the "sharp end") are often disincentivized from acknowledging the existence of incidents or near-incidents for fear of being blamed, and potentially even disciplined. Reprisal can also come in the form of extra unproductive process being dumped on them in the name of safety, without any reduction in production pressure.

Secrecy. This can manifest both in terms of compartmentalization of information within complex organizations and disincentives to share knowledge across organizations. Restricting the sharing of knowledge can severely limit those organizations' ability to learn and implement changes based on what they've learned.

While we can't determine what happened

behind the scenes at Salesforce, many of these factors appear to be in play in between the lines of this incident report.

A Common Root Cause Culprit: Joyent 2014

This incident had many hallmarks of a cascading failure, where the simultaneous reboot of all servers led to unexpected consequences of the system working "as designed."

"Once systems rebooted, they by design looked for a boot server to respond to PXE boot requests. Because there was a simultaneous reboot of every system in the data center, there was extremely high contention on the TFTP boot infrastructure which like all of our infrastructure, normally has throttles in place to ensure that it cannot run away with a machine. We removed the throttles when we identified this was causing the compute nodes to boot more slowly. This enabled most customer instances to come online over the following 20-30 minutes."

Two other technical issues covered in the report included a known source of degraded system performance (transient bug in a network card driver on legacy hardware platforms), and another by-design tricky system behavior regarding how stateful systems are manually recovered that increased the recovery time for their API outage.

Despite these factors, the company was quick to identify "operator error" as the root cause of the incident. The report acknowledges, however, that the tool lacked sufficient input validation, which let the change go through. So really, the failure was not on the part of the human operating the system, but in the design of the system in which they were operating.

"Root cause of this incident was the result of an operator performing upgrades of some new capacity in our fleet, and they were using the tooling that allows for remote updates of software. The command to reboot the select set of new systems that needed to be updated was mis-typed, and instead specified all servers in the data center. Unfortunately the tool in question does not have enough input validation to prevent this from happening without extra steps/confirmation, and went ahead and issued a reboot command to every server in us-east-1 availability zone without delay."

No single one of these latent technical or social system conditions caused the incident, but they were all collectively sufficient for it to have the impact, scope, and timeframe that it did.

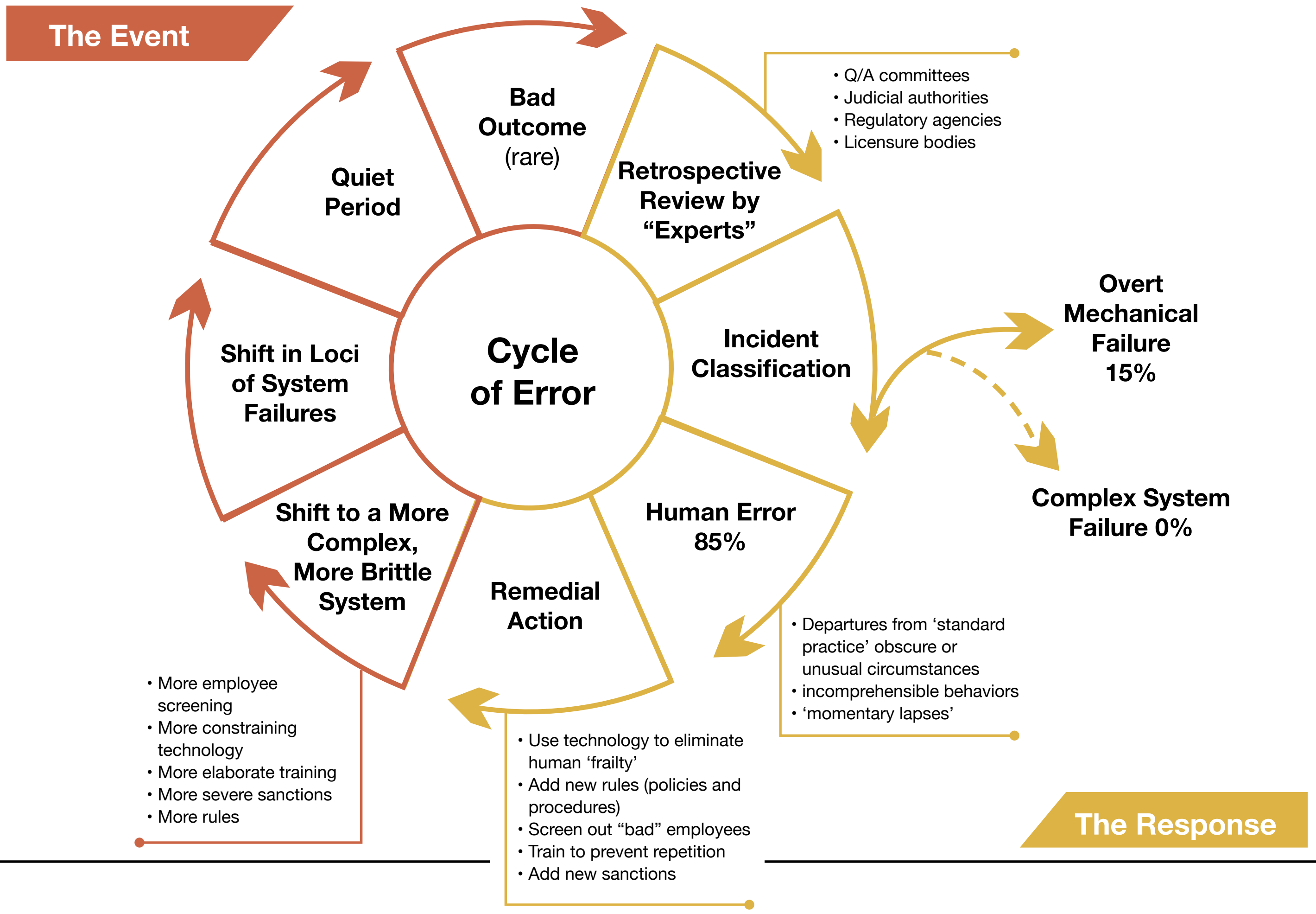
"The complexity of these systems makes it impossible for them to run without multiple flaws being present. Because these are individually insufficient to cause failure they are regarded as minor factors during operations. Complex systems run as broken systems. The system continues to function because it contains so many redundancies and because people can make it function, despite the presence of many flaws."— Richard Cook[13]

These latent conditions of large complex systems mean that applying fixes to certain specific failures are unlikely to prevent future failures. That particular combination of latent contributors is unlikely to recur, but as we noted above, new rules and regulations, technologies, and training are often dictated, all of which add more cost and complexity to the system, making it more brittle than it was before the incident. Time will pass, and because the system is generally reliable overall—largely through the efforts of human operators—it will feel as though those changes have made things better. But then eventually, a different combination of latent factors arises, resulting in another incident.[14]

The Cycle of Error

Attributing system failures to human operators generates demands for more rules, automation, and polkaing. But these actions do not significantly reduce the number of latent failures in the system. Because overt failures are rare, a quiet period follows institution of these new policies, convincing administrators that the changes have been effective. When a new overt failure occurs, it seems to be unique and unconnected to prior failures (except in the label human am), and the cycle repeats. With each pass through the cycle, more rules, policies, and sanctions make the system more complicated, conflicted, and brittle, increasing the opportunities for latent failures to contribute to disasters.

(01993, R.I. Cook, reprinted by permission)


The Event
Bad Outcome (rare)
Retrospective Review by "Experts"
• Q/A committees
• Judicial authorities
• Regulatory agencies
• Licensure bodies
Incident Classification
Overt Mechanical Failure 15%
Complex System Failure 0%
Human Error 85%
• Departures from 'standard practice' obscure or unusual circumstances
• incomprehensible behaviors
• 'momentary lapses'
Remedial Action
• Use technology to eliminate human 'frailty'
• Add new rules (policies and procedures)
• Screen out "bad" employees
• Train to prevent repetition
• Add new sanctions
Shift to a More Complex, More Brittle System
• More employee screening
• More constraining technology
• More elaborate training
• More severe sanctions
• More rules
Shift in Loci of System Failures
Quiet Period
Cycle of Error
The Response

A final example, while not actually an incident report, is this Red Hat report, which conflates configuration changes with "human error" as one of the primary causes of Kubernetes security incidents. From there, the report goes on to conclude that, "The best way to address this challenge is to automate configuration management as much as possible so that security tools—rather than humans—provide the guardrails that help developers and DevOps teams configure containers and Kubernetes securely."

So far we've focused on the fallacy of root cause in incident analysis, and the dangers in particular of pinning that cause on humans. The latter is especially important because it is tied to the other frequent outcome of assigning root cause: blame.

A Branch to Blame

On June 17, 2021, HBO Max tweeted about an accidental empty test email they'd sent out.

"We mistakenly sent out an empty test email to a portion of our HBO Max mailing list this evening. We apologize for the inconvenience, and as the jokes pile in, yes, it was the intern. No, really. And we're helping them through it."

They joked that it was the intern, and the resulting thread is a heartening record of thousands of people sharing their own mistakes in their careers. In the end, it was a rather charming moment (though many are skeptical and suspect it was a—largely successful—PR stunt). But it speaks to the phenomenon we're all too familiar with: when something goes wrong, the Powers That Be must figure out who is to blame.

> "The operator shows up like the tip of the organizational iceberg because they're the ones operating at the many edges of the production system and the organization owning it. Ending the investigation at the constructed failures of operators prevents a deeper look at what brought them there." —Fred Hebert, Honeycomb SRE[15]

RCA is often a path that ends squarely at a person's feet. "Human error" is a quick and easy scapegoat for all kinds of incidents and accidents, and it's deceptive simplicity as a root cause is an inherent part of its larger-scale harmful effects. It's comforting to frame an incident as someone straying from well-established rules, policies, or guidelines; simply provide more training, and more guardrails and checklists in the future! But that so-called error transpired in a complex, socio-technical system, so stopping at faulty human behavior prevents organizational introspection and learning that could have far more impact on future outcomes.[16,17]

> "A 'human error' problem, after all, is an organizational problem. It is at least as complex as the organization that has helped to create it." —Sidney Dekker

Sometimes, it doesn't even take someone from management to lay down blame, engineers will also heap it on themselves. This was the case in this Instapaper outage which had many characteristics of a complex system failure: a filesystem-based limitation they weren't aware of and had no visibility into, which also impacted backups, leading to an intensive and risky course to remediation, which ultimately required the assistance from engineers at another organization. Despite acknowledging that "this issue was difficult to foresee and prevent," the developer involved (and the author of the report) still insisted that the full fault was his, including the lack of a concrete disaster recovery plan:

> "I take full responsibility for the incident and the downtime. While the information about the 2TB limitation wasn't directly available to me, it's my responsibility to understand the limitations of the technologies I'm using in my day-to-day operations, even if those technologies are hosted by another company. Additionally, I take responsibility for the lack of an appropriate disaster recovery plan and will be working closely with Pinterest's Site Reliability Engineering team to ensure we have a process in place to recover from this type of failure in the event it ever happens again."

The Language of Blame

Our focus on language in reports is informed by research into the relationship between language and how people perceive and/or assign blame.[18] Even absent concrete references to "human error" or "blame," the structure of how an event is described can influence how people perceive and recall those events, and whether they are more likely to ascribe blame and assess potential punishments or consequences.

Lera Boroditsky has conducted a number of studies on what is called "agentive language," which is whether or not a sentence contains an agent of an action or not. For example, "John broke the plate" is agentive, in that John is identified as the person involved in the action, whereas "The plate broke" is non-agentive language (often referred to as "passive voice"). She found that the use of agentive language in court cases was more likely to lead to assignment of blame (such as guilty verdicts) and increased punishments and/or fines.[20]

She also found, by comparing between inherently agentive and non-agentive languages (English and Spanish, respectively) or between situations described either agentively or non-agentively, that people's recollection of individuals involved in accidents and the details of those accidents differed depending on the type of language used.[21]

Boroditsky's research suggests that an analysis of the type of language in incident reports could yield interesting results about assignment of blame. While we plan to conduct more thorough research on language in incident reports in the future, we can say now that there's only a small amount of incidents (less than a percent) that directly call out human or operator error as a "root cause." This is encouraging and we look forward to tracking this over time as we add more incident reports.

Nearly No Near Misses

“Close calls...provide opportunities... to reflect on and identify the limits of current knowledge.”
—Carl McRae[22]

The final pattern we’ll cover is the lack of “near miss” incident reports. We classify a near miss as an incident that the organization noted had no noticeable external or customer impact, but still required intervention from the organization (often to prevent an actual public incident from eventually happening). As of today, there are only seven near-miss reports in the VOID—not even a half of a percent of the total entries. While this isn’t a surprising aspect of the reports in the VOID, we call it out here as an important opportunity to establish a potential baseline and track how it changes over time.

While we don’t yet objectively know this, we suspect there are a number of reasons why organizations don’t publish near miss reports:

Near Misses Can Be Harder to Detect
In the context of complex systems, a near miss can be quite difficult to discover, given that failures often emerge from a particular combination of contributing factors. The path to an incident is a slow drift towards failure, where discerning signal from noise is challenging, and engineers often don’t know when they are nearing a potential safety boundary. These systems also exist in *some* form of degradation and it’s potentially more work than a team can tackle to try to chase all the possible faint signals.

They're Not Considered a Formal Incident
If by nature a near miss doesn't have any customer impact, then it may not reach the level of "incident" internally, and hence might not receive as much attention, resources, and the kind of retrospective analysis that a customer-facing incident would. In this regard, they could be organizationally perceived as additional work that shouldn't be prioritized over more pressing requirements.

Public Perception and/or Competitive Factors
A near miss report also likely makes lawyers and PR teams nervous. Depending on the organization's attitude towards failure, transparency, and needing to "control the narrative," they may be concerned that sharing these types of events could reflect poorly on them or unnecessarily reveal competitive or other proprietary information.

Near Misses Provide Deeper Data

Near miss reports often include much richer information about sociotechnical systems, including breakdowns in communication, dissemination of expertise, and cultural or political forces on those systems. In his coverage of near miss studies ("close calls") in the aviation industry, Carl McCrae notes that near misses "provide the source materials to interrogate the unknown, test current assumptions, invalidate expectations, become aware of ignorance and undergo experience." They reveal gaps in current knowledge, mental models, and the range of assumptions that practitioners have about their systems, before those gaps lead to more significant incidents.

"Near misses are generally more worth our time, because they come without the pressure of dealing with post-incident fall-out and blame, and allow a better focus on what happened." —Fred Hebert, Honeycomb SRE

The lack of pressure inherent in non-customer-facing incidents leaves breathing room that encourages less-rushed thinking and exploration when revisiting the events, while also avoiding pitfalls around root cause and blame (as discussed in the previous section). A near miss is a form of success, after all, so as humans, we're less prone to hindsight bias—thinking something was predictable after the fact—and defensive attributions—the desire to defend ourselves from the concern that we will be seen as the cause, or the victim, of a mishap—than we would be in the case of a public-facing incident.

In their series of post-mortems about how the Gamestop short squeeze frenzy wreaked havoc on Reddit, engineers Courtney Wang, Fran Garcia, and Garrett Hoffman delved into unprecedented detail about the various ways their systems failed under the duress of a deluge of traffic, but also wrote up near-misses that helped them better understand the ways in which their systems were successful:

"The r/WallStreetBets events taught us a great deal about how our systems worked, but more importantly they taught us many things about how we as a company work. Even though the focus of these stories revolve heavily around technical triggers, they also highlight how every team at Reddit played an important role in containing and mitigating these incidents."

We plan to dig deeper and more systematically into the information we can gather from near miss reports as we find them. For now we'll highlight a few patterns and potential areas for future research.

"That's Weird…"
One pattern we're seeing from reviewing these near miss reports is that the predicating event is typically an engineer (or other practitioner) noticing that something seems "off," such as this GitLab report on The Consul Outage That Never Was:

"The issue came to our attention when a database engineer noticed that one of our database servers in the staging environment could not reconnect to the staging Consul server after the database node was restarted."

Or this report from PrometheusKube:

"It's Tuesday morning. We receive a message from a Developer that we are missing some of the logs in production. Strangely enough, it appears that logs only from a single node are missing."

And this UK governmental investigation into a series of flights that took off with calculated passenger weights that were dramatically different from the actual passenger weights:

"[The flight crew] noticed that there was a discrepancy, with the load sheet showing 1,606 kg less than the flight plan. They noted that the number of children shown on the load sheet was higher than expected, at 65, compared to the 29 which were expected on the flight plan. The commander recalled thinking that the number was high but plausible; he had experienced changing loads on the run-up to the temporary grounding as passengers cancelled and altered trips at short notice."

These cases all highlight the role that practitioner expertise plays in awareness of system safety boundaries and conditions that can lead to incidents. The subsequent investigations tend to reveal more information about where that expertise stemmed from, which often exposes the gaps in knowledge for other people/teams, along with potential ignorance or assumptions based on those gaps.

"Oops, I Did It Again"
Near-miss reports are also a unique opportunity to reframe the debate around human error. In a near-miss situation, someone committing a typo or "bad" configuration change that doesn't fully take down a service is a chance to understand how the system can be changed to avoid an actual catastrophe in the future.

In this Cloud Native Computing Foundation talk from 2019, a Spotify engineer details how, by clicking on the wrong browser tab, he managed to delete their entire US production Kubernetes cluster. In the end, the good news was that there was no end-user impact—despite the fact that the team's efforts to recover from the initial event led to deleting the Asian cluster, which led to a series of efforts that then also took out the U.S. cluster yet again! (This is another meta-incident pattern that we also hope to investigate in the future, whether interventions in a current incident unknowingly lead to future incidents.) As we find more near-miss reports, we plan to look into whether these analyses are less prone to finger pointing or agentive blame-focused language.

"Mind the Gap"

We also strongly suspect that near-miss investigations provide analyses that better uncover gaps in knowledge and misalignments in socio-technical systems. These can include things like:

- Expertise held by only a few individuals
- Assumptions about how systems work
- Pattern matching behaviors that lead to "garden path" thinking based on previous incidents or known degraded behaviors
- Lack of knowledge across teams about how systems work
- Production and business pressures (new features, constrained team members, insufficient funding for improvements)

The Consul Outage That Never Happened

Engineer noticed something was "off": "The issue came to our attention when a database engineer noticed that one of our database servers in the staging environment could not reconnect to the staging Consul server after the database node was restarted."

Gap in collective knowledge w/in the org: "After looking everywhere, and asking everyone on the team, we got the definitive answer that the CA key we created a year ago for this self-signed certificate had been lost."

Transitory knowledge, situation normal: "These test certificates were generated for the original proof-of-concept installation for this service and were never intended to be transitioned into production. However...the expired test certificate had not been calling attention to itself."

Production pressures, limited people: "...a year ago, our production team was in a very different place. We were small with just four engineers, and three new team members: A manager, director, and engineer, all of whom were still onboarding."

Surprises, gaps in knowledge: "We were unsure why the site was even still online because if the clients and services could not connect it was unclear why anything was still working. "

Precarious conditions, safety boundaries becoming clear(er): "Any interruption of these long-running connections would cause them to revalidate the new connections, resulting in them rejecting all new connections across the fleet."

Efforts to fix make it worse: "Every problem uncovered other problems and as we were troubleshooting one of our production Consul servers became unresponsive, disconnected all SSH sessions, and would not allow anyone to reconnect."

Further safety boundaries and degraded state uncovered: "Not having quorum in the cluster would have been dangerous when we went to restart all of the nodes, so we left it in that state for the moment."

Inherent risk in solving the problem: Multiple solutions are considered, all "involve the same risky restart of all services at once."

Socio-technical realities: "While there was some time pressure due to the risk of network connections being interrupted, we had to consider the reality of working across timezones as we planned our solution."

This complex, high-tempo coordinated work was only possible due to a depth of system expertise from the operators (they left the work with the team that had figured out the solution, letting them sleep and then get back to it the next day).

And yet, the proposed solution led to more knowledge gaps, more troubleshooting, and additional fixes that were required to proceed.

Even after the big main coordinated fix went out, there were still a few lingering issues.

Another near miss example comes from Mailchimp, who experienced a puzzling, multi-day internal incident that didn't affect customers but "prompted a lot of introspection." The conclusions from their analysis were almost entirely social, cultural, and historical:

MailChimp: Computers are the Easy Part

Production pressures: "many of our on-call engineers were already tired from dealing with other issues"

Not the usual suspect: "The responding engineers, based on our prior experience dealing with this type of failure, first suspected...However, the only change that had landed in production as the incident began was a small change to a logging statement, which couldn't possibly have caused this type of failure. "

Cost of coordination: "late in the evening, we still hadn't had any breakthroughs, so we put a temporary fix in place to get us through the night and waited for fresh eyes in the morning...By the second day, the duration of the incident had attracted a bunch of new responders who hoped to pitch in with resolution."

Mental model mismatch driven by cultural and historical factors: "we realized that this was a failure mode that didn't really line up with our mental model of how the job system breaks down. As an organization, we have a long memory of the way that the job runner can break, what causes it to break in those ways, and the best way to recover from such a failure. This institutional memory is a cultural and historical force, shaping the way we view problems and their solutions."

Hard-earned human expertise: "With this new instrumentation in place, we noticed something incredibly strange...a couple of engineers who had been observing realized that they'd seen this exact kind of issue some years before."

Difficulty adapting in the face of novelty: "We rely on heuristics and collections of mental models to work effectively with complex systems whose details simply can't be kept in our heads. On top of this, a software organization will tend to develop histories and lore—incidents and failures that have been seen in the past and could likely be the cause of what we're seeing now. This works perfectly fine as long as problems tend to match similar patterns, but for the small percentage of truly novel issues, an organization can find itself struggling to adapt."

Challenges with changing the frame: "Given the large amount of cultural knowledge about how our job runner works and how it fails, we'd been primed to assume that the issue was part of a set of known failure modes."

We rely on heuristics and collections of mental models to work effectively with complex systems whose details simply can't be kept in our heads.

These near miss reports tackle internal incidents in ways that few public reports do. Of note, they capture things like knowledge gaps, heuristics and mental models, transitory or siloed knowledge, operating pressures, cognitive processes and biases, and other largely social or organizational factors typically absent in traditional incident reports. We strongly suspect (and yes, plan to study) that organizations that engage in these efforts will have better adaptive capacity[26] to handle disruptions and incidents than their counterparts that don't.

There are these two young fish swimming along and they happen to meet an older fish swimming the other way, who nods at them and says “Morning, boys. How’s the water?” And the two young fish swim on for a bit, and then eventually one of them looks over at the other and goes “What the hell is water?”

–David Foster Wallace[27]

A New Approach Is Needed

We need a new mindset, toolset, and skillset for talking about, analyzing, learning from, and sharing incidents.

Software has changed dramatically in the past decade, but our mental models have not caught up. Based on our preliminary review of incident reporting from the VOID, we recommend four key areas that organizations can focus on to improve their approach to complex system failures.

Treat Incidents as Opportunities to Learn

Numerous pressures push organizations in the direction of wanting to determine what happened during an incident, and be able to ensure it won't happen again. The desire for certainty and closure inhibit learning, which is the key to wrangling complex software systems. Creating a learning culture is a paradigm shift that must be embraced by the entire organization in order to take root and thrive. Google has an excellent overview of what this kind of learning culture looks like, and how to foster it.[28]

Favor In-Depth Analysis Over Shallow Metrics Like Mttr, Frequency, or Time Between Incidents

Our operational dashboards capture and display an increasingly dense amount of data about our technical systems, but they tell us very little about our sociotechnical systems. As we've shown, typical MTTx metrics about incidents yield minimal actionable information, notably about the impact an incident has on the team(s) handling it. Tackling complex system issues requires an understanding of the safety boundaries of those systems—where are the cliffs and invisible minefields—something that can only come from investing in analyses that dig into the messy details of incidents.

Uncovering knowledge gaps, identifying diverging mental models, and revealing organizational pressures is a distinct skill set from building or operating software systems—they're not necessarily mutually exclusive, but the former requires time and resources that are rarely granted in a sustainable way. We are seeing an emerging trend of teams hiring dedicated Incident Analysts, and tooling like Jeli is emerging that tracks social or organizational incident factors along with technical contributors—we plan to pair broader survey methods with data from the incident reports in the VOID to track these developments over time.

View Humans as Solutions, Not Problems

Language shapes not only how we think about or remember a situation, it also influences whether or not we blame people for accidents (either explicitly or implicitly via nameless "human error"). Blame creates an artificial stopping point for inquiry, which hinders an organization's ability to learn from events and foster a safer working environment. It also fails to acknowledge that humans are what keep our systems running successfully most of the time, through expertise, adaptation, and even, at times, improvisation in uncertain circumstances. Guardrails and additional processes, while intended to reduce incidents, often serve to impair the necessary unpredictable human behavior that prevents them in the first place.

Study What Goes Right Along With What Goes Wrong

Near misses help teams better understand the aspects of their sociotechnical systems that support adaptive capacity so they can further invest in those. Near miss analysis often yields detailed insights into social and organizational degradations that are likely to contribute to complex system failures, by side-stepping the pressured process that is more likely to occur in public incident analysis.

Creating a learning culture is a paradigm shift that must be embraced by the entire organization in order to take root and thrive.

What’s Next?

A Rising Tide Lifts All Boats

Help us fill the VOID.

A rising tide lifts all boats. However, the VOID only has a tiny subset of all the boats out there. We want to conduct more thorough, quantitative and qualitative incident research, but we need to get as many of the available reports into the VOID as possible. Help us fill it!

Some sample research questions we'd like to pursue include:

- Do remediations or fixes from past incidents lead to future incidents?
- Does language about incidents differ significantly between primary and secondary sources?
- Does the depth of investigation vary based on the severity of the incident (as determined by the organization)?
- Are there differences in organization size, private vs public, and their likelihood to publish or the frequency with which they do so?
- Does the amount of time between the incident and a public account of it lead to different depths of detail?
- Can we map dependencies, like the impact of large cloud provider incidents on downstream customers or users?
- What role does new technology adoption play in the prevalence of incidents?
- Can we determine relationships between business or production pressures (e.g. new product or feature development) and incidents?

Acknowledgements

Lead Writer
Courtney Nash

Designer
Tiffany Knudtson

Partner
Indeed

Review and Feedback
John Allspaw
Štěpán Davidovič
Eric Dobbs
Fred Hebert
Tiffany Knudtson
Jason Koppe
Niall Murphy
Lex Neva
Laura Nolan
Casey Rosenthal
James Wickett

Special Thanks

The VOID and this report wouldn't be possible without ongoing support, encouragement, and mind-boggling insightfulness of the following people:

John Allspaw
Richard Cook
Nora Jones
Casey Rosenthal
David Woods
The Learning From Incidents (LFI) community

Our Partners

NS1.
SecurityScorecard
Adaptive Capacity Labs
VERICA

Our Members

Auxon
LFI

References

[1] Jones, Nora, “Incident Analysis – Your Organization’s Secret Weapon.” Talk delivered at DevOps Enterprise Summit Europe, 2021.

[2]Billings, Charles. “A Tale of Two Stories: Contrasting Views of Patient Safety” (Appendix B of Report from a Workshop on Assembling the Scientific Basis for Progress on Patient Safety), National Health Care Safety Council of the National Patient Safety Foundation at the AMA, 1998.

[3]Ibid

[4]McGuire, Laura. Managing the Hidden Costs of Coordination: Controlling coordination costs when multiple, distributed perspectives are essential, 2020. ACM Queue, Volume 17, issue 6.

[5]Heinrich, H.W. (1931). Industrial accident prevention; a scientific approach. McGraw-Hill.

[6]Dekker, Sidney. The Field Guide to Understanding ‘Human Error’, 2014 (3rd edition). Ashgate Publishing

[7]In doing this, we are not aiming to blame or shame. We also grapple with the desire to reduce complex situations to more simple narratives or reasons. This analysis aims to illustrate how complex system failures arise from a combination of factors that don’t have a direct cause-and-effect relationship.

[8]How Complex Systems Fail, 2002

[9]Bainbridge, Lissane. “The Ironies of Automation” Automatica, Vol. 19, No. 6. pp. 775-779, 1983.

[10]Leveson, Nancy. Engineering a Safer World: Systems Thinking Applied to Safety, 2016. MIT Press.

[11]Dekker, Sidney. Drift Into Failure, 2001, pg 129. Ashgate Press.

[12]Sagan, Scott D. The Limits of Safety, 1995. Princeton University Press.

[13]Ibid

[14]Cook, Richard and Woods, David, 1994. Operating at the Sharp End: The Complexity of Human Error. Human Error in Medicine, 255-310.

[15]https://www.thevoid.community/incident-detail?recordId=rec2Vy8zihRs5xdmF

[16]Cook, Richard and Nemeth, Christopher. Those found responsible have been sacked: some observations on the usefulness of error, 2010. Cognition, Technology & Work (Springer Verlag).

[17]By far some of the most thorough research on human error in the context of system safety comes from Erik Hollnagel and Jens Rasmussen. We include a number of pointers to their work in the References section at the end of this report.

[18]Vesel, Crista. Agentive Language in Accident Investigation: Why Language Matters in Learning from Events, 2020. ACS Chemical Health & Safety, 27 (1), 34-39.

[19]See her Long Now Foundation talk for an overview of her research: https://www.youtube.com/watch?v=I64RtGofPW8&t=3110s

[20]Fausey, C. M.; Boroditsky, L. Subtle linguistic cues influence perceived blame and financial liability, 2010. Psychonomic Bulletin & Review, 17 (5), 644–650.

[21]Fausey, C. M.; Boroditsky, L. English and Spanish Speakers Remember Causal Agents Differently. Proceedings of 30th Annual Meeting of the Cognitive Science Society, Washington, DC, July, 2008.

[22]McRae, Carl. Close Calls: Managing Risk and Resilience in Airline Flight Safety, 2014. Palgrave Macmillan.

[23]Ibid

[24]That's not saying that all public incident reports lack this level of detail seen in near-miss reports—there are some that also go into this level of depth around gaps in knowledge, assumptions and expectations, social/cultural factors, etc—however, they are in the minority.

[25]Vesel, Crista. Agentive Language in Accident Investigation: Why Language Matters in Learning from Events, 2020. ACS Chemical Health & Safety, 27 (1), 34-39.

[26]Branlat, Mattieu, & Woods, David, How do systems manage their adaptive capacity to successfully handle disruptions? A resilience engineering perspective, 2010. Association for the Advancement of Artificial Intelligence Fall Symposium - Technical Report, FS-10-03.

[27]David Foster Wallace, "This Is Water," 2005. Commencement speech delivered at Kenyon College.

[28]For a deeper look at how organisational culture bears a predictive relationship with safety, we highly recommend Ron Westrum's work, notably his 2005 paper: A Typology of Organisational Cultures. Quality & Safety in Health Care, 13 Suppl 2.

VOID