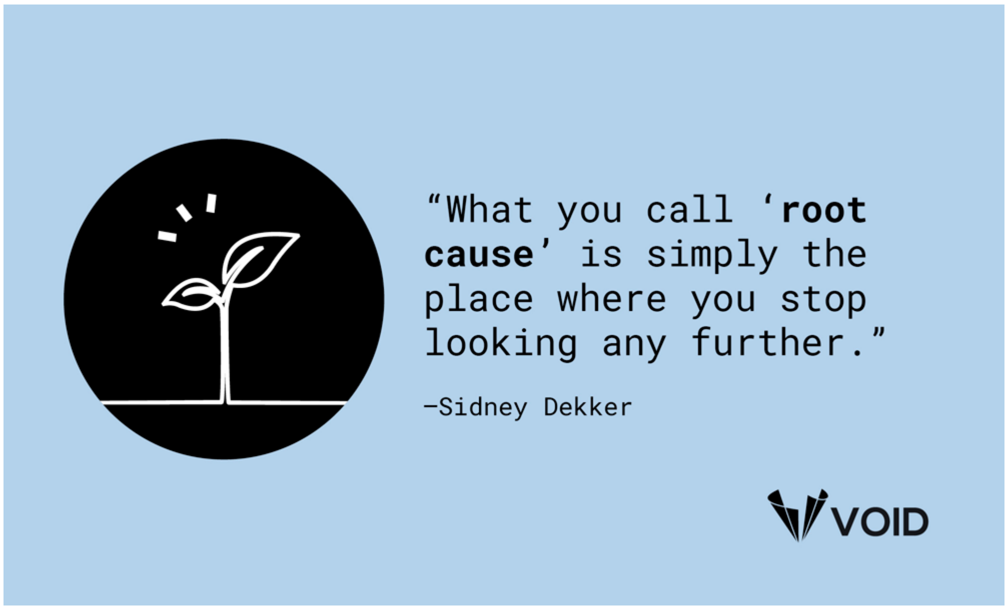
By Courtney Nash

Root Cause Is for Plants, Not Software

November 17, 2021
12 minute read


"What you call 'root cause' is simply the place where you stop looking any further."
–Sidney Dekker
VOID

Editor's note: This is the third part of a series that will focus on each of The VOID Report 2021 key findings. If you haven't had a chance to download the report, it's not too late. You can download the report at thevoid.community/report.

Roughly a quarter of the VOID 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 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 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."

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 u 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.

To illustrate this, let's look at a few examples of RCAs that are not so clear-cut. (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, linear cause-and-effect relationship.)

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, but 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 in how to follow them. In this case, the analysis revealed that someone didn't follow procedures, and the incident ensued as a result of their actions (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 behind that 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, HOW COMPLEX SYSTEMS FAIL

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, 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.

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.”

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 that Scott Sagan details in *The Limits of Safety*:

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 of 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.

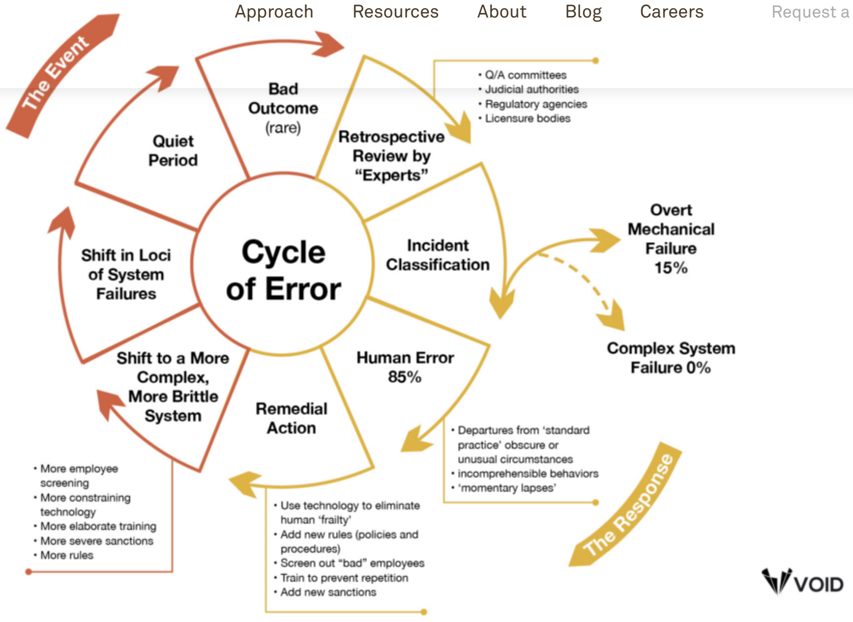
> "The complexity of these systems makes it impossible for them to run without multiple flaws being present. Because these are individually insufficient to cause failure they are regarded as minor factors during operations. Complex systems run as broken systems. The system continues to function because it contains so many redundancies and because people can make it function, despite the presence of many flaws."
>
> – RICHARD COOK, HOW COMPLEX SYSTEMS FAIL

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. Dr. Richard Cook refers to this as the "Cycle of Error."

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.


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

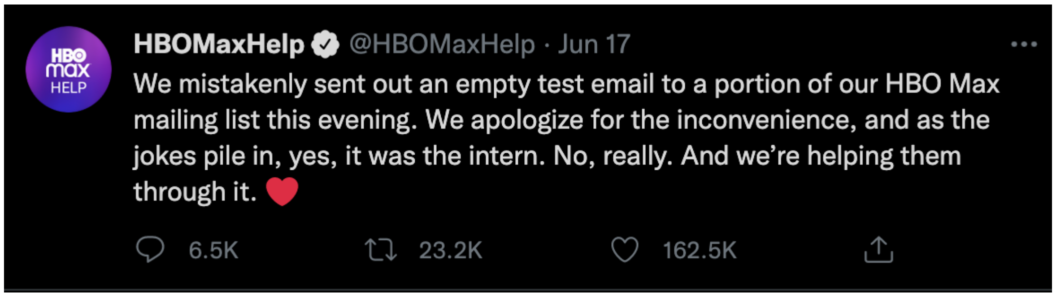
(1993, R.I. Cook, reprinted by permission)

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.


HBOMaxHelp @HBOMaxHelp · Jun 17

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. ❤️

6.5K 23.2K 162.5K

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

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; sim 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. (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 the full report.)

> "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 RDS 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, even 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. 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.

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.

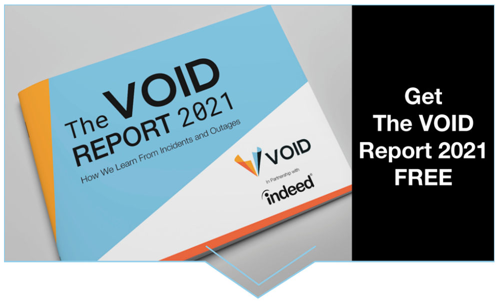
Boroditsky's research suggests that an analysis of the type of language in incident reports could yield interesting results about assignment of blame. While we plan to conduct more thorough research on language in incident reports in the future, we can say now that there's only a small amount of incidents (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.

If you've found this interesting or helpful, help us fill the VOID! Submit your incidents at thevoid.community/submit-incident-report. Read more about the VOID and our key findings in The VOID Report 2021.

Senior Research Analyst

Courtney Nash

Courtney Nash is a researcher focused on system safety and failures in complex sociotechnical systems. An erstwhile cognitive neuroscientist, she has always been fascinated by how people learn, and the ways memory influences how they solve problems. Over the past two decades, she's held a variety of editorial, program management, research, and management roles at Holloway, Fastly, O'Reilly Media, Microsoft, and Amazon. She lives in the mountains where she skis, rides bikes, and herds dogs and kids.


The VOID REPORT 2021
How We Learn From Incidents and Outages
VOID
In Partnership with
indeed
Get The VOID Report 2021 FREE

The VOID Report 2021

Discover key findings that could change how the industry thinks about MTTR, RCA and near misses.

Get the Report

Related Articles

NOVEMBER 4, 2021

MTTR is a Misleading Metric—Now What?

By Courtney

Software organizations tend to value measurement, iteration, and improvement based on data. These are great things for...

Read More

NOVEMBER 2, 2021

Answering the Unanswered: The VOID Podcast

By Courtney

Whenever we read a company's public incident report, there are so many questions that ultimately go unanswered....

So we thought: what if we could ask the people involved in those incidents to take us back to their experience, and answer...

Read More

Engineering with Confidence

Verica integrates with the most pervasive streaming and automation software platforms.

Verica gives you Kafka superpowers

Get control of Kubernetes chaos.

Request a demo of Verica

Email Address

REQUEST A DEMO