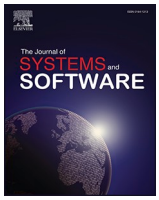
The Journal of Systems and Software xxx (xxxx) xxx

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



Correspondence

Dear researchers: help me deal with incidents!

A plea for help

Dear researchers

Allow me to briefly introduce myself. I am a software-engineering-researcher-turned-software-practitioner. My current job title is "*Staff Software Engineering, Reliability*". However, it would be more accurate to describe my role as a *Site Reliability Engineer*, or SRE. SRE is a sort of development-operations hybrid role, where someone with a software engineering background applies their skills to improve software operations. In my particular case, a major aspect of my SRE work is in *incident response*. What I mean by an incident is a major disruption to the correct functioning of the software systems that we are responsible for keeping up and running. Often you'll hear these referred to as *outages*. And incident response is where I need your help.

Incident response is different than debugging

At first glance, you might think that incident response is a type of debugging. The software is behaving incorrectly, and you're trying to figure out what's wrong and fix it. While incident response and debugging are related, they are not the same: the goals of the two activities are different. The goal of debugging is to figure out what the source of the problem is, and to fix it permanently. The goal of incident response is to get the system back into a healthy state as quickly as possible.

In incident response, we proactively undo changes to the production environment that correlate in time with impact (e.g. , rolling back a recent code deploy, flipping a feature flag) based on much weaker evidence than we would during debugging activity. We also perform therapeutic interventions during incidents, such as using Kubernetes to rapidly add more compute capacity, or quickly setting up new policies in a Content Delivery Network (CDN) such as Cloudflare, Akamai, or Fastly, to block a particular type of traffic that is causing problems. These interventions look very different from traditional debugging activities.

A notable example is a recent outage experienced by the social media company Bluesky. This outage involved a cascading failure where a new internal service was making remote procedure calls (RPC) with large payloads (a list of fifteen to twenty thousand URIs passed in a single message). Each entry triggered a new TCP connection, and the connections were being made concurrently. This led to a cascading failure where the number of ephemeral ports was exhausted, leading to a very large increase in error log lines being written synchronously to disk, which in turn led to a large number of operating system level threads being spawn, which created memory pressure which led to frequent *stop-the-world* garbage collection pauses, and the service being terminated by the operating system's out of memory (OOM) killer. You can find a public write-up of this outage by the Bluesky engineer Jim Calabro at his April 2026 Outage Post-Mortem and my own commentary on this write-up at Thoughts on the Bluesky public incident write-up – Surfing Complexity.

Note how the cascading error makes it difficult to identify what the underlying problem actually is. From the writeup:

> *It was all buried in there, but it was hard to know where to look when so much was falling over all at once. You need to have the mental discipline and high granularity in your metrics to be able to cut through the noise to find the real root cause. It's hard work!*

As for the mitigation, once again I'll quote Calabro::

> *The band-aid fix was insane but did the job. This is how we actually fixed the outage on Monday, before we found the true root cause:*
> *[The fix randomly selected a local IP address in the range 127.1.0.1 – 127.254.255.254 when making a connection, instead of relying on the default behavior of always using 127.0.0.1].*
> *That got us out of the death loop because it expands the client ip+port space. Crazy, but effective! We removed this once we fixed the true root [cause].*

Note his use of terms such as *insane but did the job* and *crazy, but effective* to describe the mitigation, emphasizing how implementations that would never be accepted into a production codebase are nevertheless employed during an emergency to get the system back to healthy.

What makes incident response hard

There are a number of elements that make incident response particularly difficult.

Time pressure. Every minute that the system is down costs the organization money, so the pressure is to get the system up as quickly as possible.

Dynamic problem. Systems are almost never "fully down"**.** Instead, they degrade, and the level of degradation can change very quickly over time. How bad things currently are shapes the type of interventions we're willing to take, but that can change quickly.

Nobody understands the whole system. Modern software systems are composed of multiple microservices owned by different teams, which reduces the scope of what individual software engineers need to understand. But incidents involve pathological behavior that spans multiple services. This means that incidents require coordination across engineers from multiple teams to resolve.

Integrating many ambiguous signals. During an incident, we need to synthesize operational data coming from the system (metrics, logs, error reports), with our own attempts to reproduce issues, reports from users, and so on. During a cascading failure, such as the Bluesky outage, it can be difficult to tease out cause and effect. In addition, the base rate of errors during normal operations is never zero. This means that an error signal seen during an incident may be unrelated to the incident.

https://doi.org/10.1016/j.jss.2026.112975

Available online 9 June 2026

Quoting one more time from the Bluesky write-up:

Also, the status page said this was an issue with a 3rd party provider. It was clearly not, apologies for that miscommunication! At the time I posted that status page update, I was looking at some traceroutes that indicated some pretty substantial packet loss from a cloud provider to our data center, but those were not the root cause of the issue.

Sometimes signals are inconsistent. For example, a particular service may be exhibiting some symptoms of being overloaded (latency of responses has increased, returns HTTP status codes consistent with overload such as 429 or 503), but other metrics such as volume of requests and CPU utilization appear normal. This may occur because, for example, the individual requests have become more expensive, and because the CPU metric being plotted is aggregated in such a way that it is hiding the fact that a particular container is experiencing high CPU utilization.

Sometimes, we may not even know what the actual customer impact is during an incident. Even when incident responders can reproduce a problem locally, this may not necessarily mean that there is widespread impact. For example, internal employees may report problems when trying to use the service, but software systems often have special code that treats employees as a special case, in order to expose employees to new features before they are more widely released. This can make it difficult to determine "is it broken for everyone or is it just impacting employees?" The site downdetector.com is not only used by customers of online services, it is sometimes used by the providers to confirm whether many customers are indeed having issues!

AI tools do not magically solve the signal integration problem. Rather, they give us *yet another signal* with a potential diagnosis. SRE AI tools today are hit-and-miss in terms of their ability to properly diagnose what the problem is: sometimes they figure it out, other times they are widely off, and it is difficult to tell from the output how much to trust them.

There's good research, but not from the software engineering community

There's some really great research out there about the work of incident responses. But this research isn't coming from the software engineering community. Instead, it is coming from branches of the safety science research community, in particular the fields of resilience engineering (RE) and cognitive systems engineering (CSE). I personally maintain lists of notable papers from these communities:

- https://github.com/lorin/resilience-engineering
- https://github.com/lorin/cognitive-systems-engineering

This work is useful to me because it generalizes across multiple domains that involve incident response (e.g., emergency medicine, aviation, power generation). But I'd also like to see research that is tailored to the software use case.

Yes, there is the ITIL framework, which was originally developed by the UK Government, and is currently stewarded by PeopleCert, which is a private company. However, it's precisely the research findings from RE and CSE that demonstrate the flaws in the process-oriented approach of ITIL: to embrace RE and CSE is to reject ITIL as being the wrong model of understanding how successful software operations work is actually done. From an ITIL perspective, the most relevant details of the above incident are the cause of the incident (referred to as a *problem* in ITIL parlance), and the duration of the incident. This provides incident data that can be tallied and plotted, but it discards the qualitative details that yield all of the real insight.

In this space, I would love to see academic researchers study the kinds of problems that we face, by doing case studies of incidents from a diagnostic and mitigation perspective, and publishing the results. Without getting at the messy details of what really makes software incident response difficult, I worry that any proposed new tooling won't actually address our problem.

For example, one problem I face is that it is very difficult for me to navigate through multiple operational dashboards, as well as doing ad-hoc queries. Both vendors and open-source software tools in this space are focused on solving the general problem, but any solution would also have to be tailored to the internal architecture of the particular system, as well as understanding the type of work that we have to do integrating all of this messy data in real time.

Here I'll call out two notable classic cognitive systems engineering papers as examples of the type of research I would love to see more of: Mental procedures in real-life tasks: a case study of electronic trouble shooting by Rasmussen and Jensen introduces the concept of the *abstraction hierarchy,* how troubleshooters move up and down a hierarchy where the top is functional purpose and the bottom is physical form. I would love to see tooling for software operations that takes this hierarchy into account. How Not to Have to Navigate Through Too Many Displays deals with the problem we face of navigating through operational dashboards and introduces the concept of *visual momentum*, which I have yet to see applied to dashboard design in my world. I would love to see some innovation in the observability space from academic researchers that focuses on helping operators navigate this data more quickly and build confidence about what they are seeing.

On the AI front, SRE AI is an enormous buzzword right now, with more and more SRE AI startups popping up by the day. These tools have not yet seen the level of adoption of AI coding tools; it's still unclear how effective they are in practice, when they work well, and when they don't, and how we can actually use them effectively in the context of an incident. We aren't going to get this from the vendors, nor from the proof-of-concept projects inside companies.

We also need help with better tools tailored for people to coordinate with each other during incidents. Today, we repurpose existing collaboration tools like Slack, Zoom, and Google Docs to help coordinate, but these tools weren't really designed to support incident response: a newcomer to an incident can be overwhelmed by the large volume of slack messages, or may look at a Google Doc and not know what information is the latest and what's out of date.

Finally, I'd ask that researchers who are interested in understanding this corner of the software world engage with us practitioners in our forums. I'd encourage you to attend SREcon, which is a practitioner conference series put on by USENIX. The Resilience Engineering Association is a professional organization of resilience engineering researchers that could use more representation from software engineering researchers. And there is a new professional organization of academically-minded SRE practitioners called the Resilience in Software Foundation. These are practitioners who read academic papers(!!!). Come meet us on our turf and talk to us about our experiences in dealing with software incidents!

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Lorin Hochstein
Airbnb, San Francisco, CA, USA
E-mail address: Lorinh@gmail.com.