DOI:10.1145/3380322

Article development led by acmqueue
queue.acm.org

The use of post-incident artifacts in high-performing organizations.

BY J. PAUL REED

Beyond the 'Fix-It' Treadmill

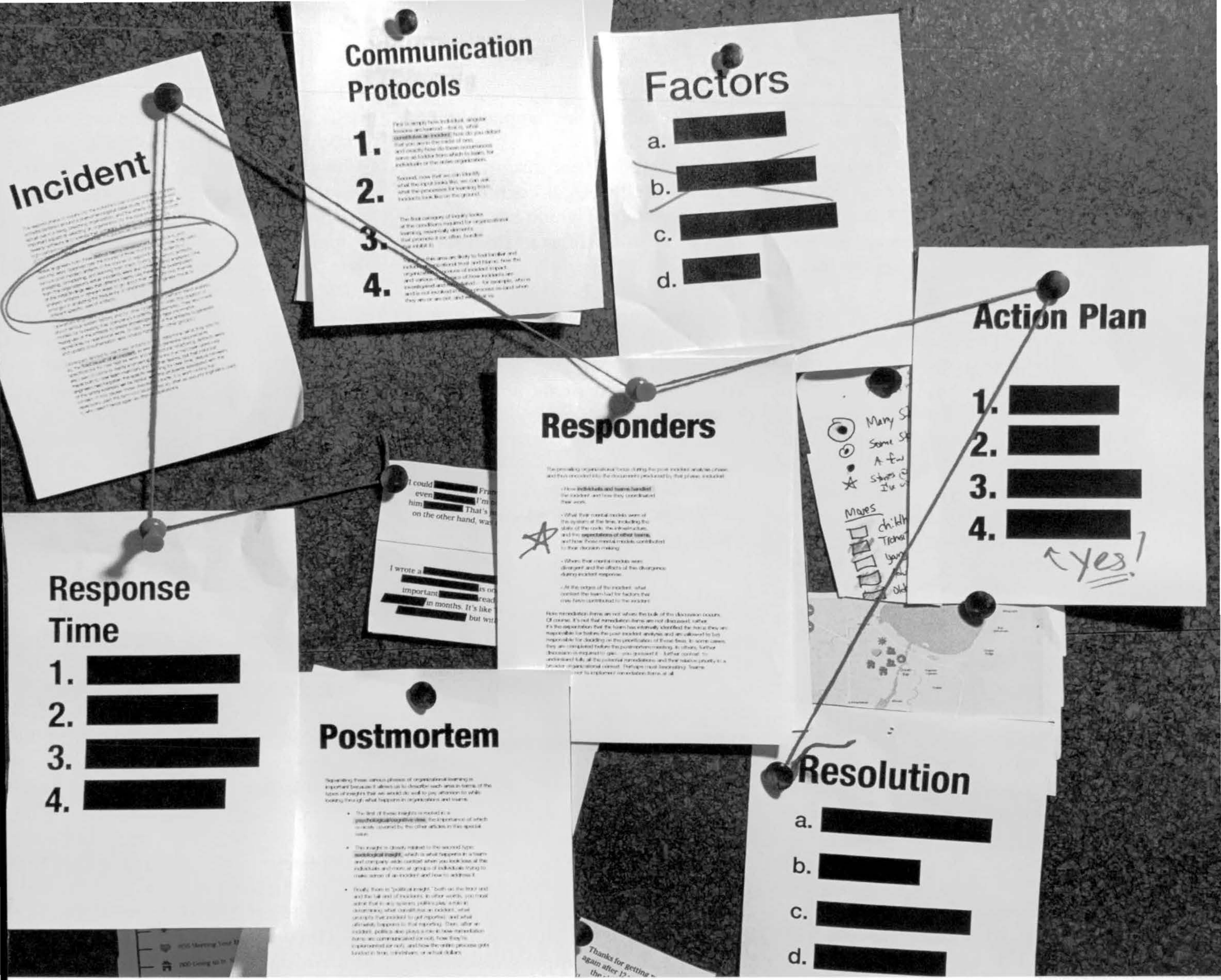
OF ALL THE traits the technology industry is known for, self-reflectivity and historical introspection do not rank high on the list. As industry legend Alan Kay once famously quipped, "The lack of interest, the disdain for history is what makes computing not-quite-a-field." It is therefore somewhat cognitively dissonant, if not fully ironic, that the past few years have seen renewed interest in the mechanics of retrospectives and how they fit into the daily practice of our craft.

Of course, retrospectives are not new, in software development at least. For more than 15 years capital-A Agile software development methods have been extolling the virtues of a scheduled, baked-in reflection period at the end of each development sprint. (Whether these actually occur in organizations practicing Agile remains an open question.) Those same 15 years have also seen a tectonic shift in the way software is delivered: the general industry trend has sharply moved from packaging up those bits and bytes into boxes to be shipped to users to "operate" themselves toward deploying it on massive server installations that *we* are responsible for maintaining, operating the software we have developed *for* users.

This shift has made the practice of software operations, and thus the study of how to do it and do it well, of interest to industry practitioners and spectators alike. As a part of the practice of software operations, there is renewed examination into the role played by operational retrospectives—more commonly referred to in an industrial context as *postmortems*. In short, looking back at the past to improve the future has become front-of-mind for many companies, precisely because the cost of not doing so in the development phase of software can be nebulous to measure, but the cost of not doing so in software operations is very apparent: Service-impacting incidents can be (and often are) easily translated to eye-popping dollars of lost revenue or service-level agreement penalties.

Think back to the last incident postmortem in which you participated (or if you have never had the opportunity to participate in one, take a moment and imagine what might occur there). It probably looks something like this: A few days after the incident, a group of people meet for an hour. (It's always an hour.) The size of the group (and how many managers are present) is directly proportional to how *important*—code for *visible* or *costly*—the incident was. The discussion kicks off by going through the details of the incident, often starting with the specifics of exactly how costly or how visible the outage was. Next up, what "actually happened" during the incident is discussed: how it started, who did (or didn't) do what, and perhaps how the teams interacted with each other to address the problem. Maybe this discussion is aided by a timeline compiled beforehand (or maybe this timeline is put together at the meeting); logs and other metrics might be presented.

Conversations might tend toward



tense, and depending on a number of organizational factors, blame might be flung around the room. Or maybe it's someone's job to remind everyone they are all blameless. Maybe they believe it. Maybe whether or not they believe it depends on who is in the room. At some point, either to defuse a tumultuous situation, because someone notices there are 10 minutes remaining in the hour, or just to change a topic that no one wants to dive too deeply into, the discussion shifts to remediation items. The question is asked, "What are we doing to 100% *make sure this never happens again*?" The group brainstorms a list of remediation items. They range from low-cost, high-value items—"We already implemented *those*," one engineer proudly reports—to high-cost, questionably valuable items, which would otherwise be laughed at but in this specific setting everyone quietly nods their head in agreement. Someone writes down those remediation items or takes a picture of the whiteboard where they are written. And the team disperses.

Maybe the suggested remediation items get entered into a ticket-tracking system. Maybe the company has a team whose sole purpose is to chase down these items and ensure each development and infrastructure team completes every item on that list in some (maybe discussed, maybe agreed upon, maybe neither) time frame. Maybe the team completes a large number of the items on the remediation list in the next two or three sprints; hopefully, the organization feels pretty good about that. Or maybe the importance of that work, once thought so critical, gets lost in the shuffle to meet the continuing onslaught of other goals, like a promised new feature or a big platform migration. Or maybe another critical incident—possibly related?—takes up all the mindshare available for "do something" about the earlier incident.

If this pattern feels familiar, it should. Most operational retrospective and incident-analysis processes in technology companies look more or less like this. Some organizations are more experienced at the practice than others, some foster a "healthier" environment for it than others, and some value it more in the calculus of how they deliver software to and operate it for their customers. But the model, and its expected outputs, are generally the same, which leads to an important question: Are we missing anything in this prevalent rinse-and-repeat cycle of how the industry generally addresses incidents that could be helpful?

Put another way: As we experience incidents, work through them, and deal with their aftermath, if we set aside incident-specific, and therefore fundamentally static, remediation items, both in technology and process, are we learning anything else that would be useful in addressing and responding to incidents? Can we describe that knowledge? And if so, how would we then make use of it to leverage past pain and improve future chances at success?

What Is Meant by "Learning"?

The topic of organizational learning has been of long-standing interest to the safety sciences, and researchers have been observing how it works in the context of industries from aviation to healthcare to maritime shipping for almost 90 years. Organizational learning has been deconstructed into three distinct categories of inquiry, following an evolution not dissimilar to the operation of Web-scale infrastructure and software:

- First is simply how individual, singular lessons are learned—that is, what constitutes an *incident*, how do you detect that you are in the midst of one, and exactly how do these occurrences serve as fodder from which to learn, for individuals or the entire organization.
- Second, now that we can identify what the input looks like, we can ask what the processes for learning from incidents look like on the ground. Much of the focus of organizational learning is on this specific facet, because it gets into the details of how real-world teams identify lessons to be learned and go about implementing them in their systems (or don't).
- The final category of inquiry looks at the conditions required for organizational learning, essentially elements that promote it (or, often, hurdles that inhibit it). Topics in this area are likely to feel familiar and include organizational trust and blame, how the organization conceives of incident impact, and various mechanics of how incidents are investigated and remediated—for example, who is and is not involved in these processes (and when they are or are not, and why that is).

Types of Insights

Separating these various phases of organizational learning is important because it allows us to describe each area in terms of the types of insights that we would do well to pay attention to while looking through what happens in organizations and teams.

- The first of these insights is rooted in a psychological/cognitive view, the importance of which has been covered in recent articles in the Practice section.
- This insight is closely related to the second type: sociological insight, which is what happens in a team- and company-wide context when you look less at the individuals and more at groups of individuals trying to make sense of an incident and how to address it.
- Finally, there is "political insight," both on the front end and the tail end of incidents. In other words, you must admit that in any system, politics play a role in determining what constitutes an incident, what prompts that incident to get reported, and what ultimately happens to that reporting. Then, after an incident, politics also plays a role in how remediation items are communicated (or not), how they're implemented (or not), and how the entire process gets funded in time, mindshare, or actual dollars. (Or not.)

These frameworks for investigating organizational learning have been applied to numerous industries. (A personal favorite delved into how Swedish rail workers learn from incidents, versus how the rail company *thinks* they learn, versus how the rail company *itself* "learns.") Only in the past five or so years, however, have software operations been brought under the same lens, which necessarily drags software *development* along with it under the microscope (in an interesting twist, this is missing from other industries the safety sciences *have* studied).

A focus of these inquiries in the technology industry has been impactful or visible site/service outages, precisely because there are a set of practices that engineers and companies engage in during and after an event, but they are highly variable and not well described in the literature. (I aimed to change that in research conducted in late 2017 and recently published as a master's thesis.)

Post-Incident Analysis Artifacts

Even the most nascent of incident postmortem processes produce *something* as an output. Common examples include a postmortem report, remediation item tickets (relating to the software, the infrastructure, or both), updated documentation or runbooks, or distilled communications for other groups such as customers or executives. My deep dive into organizational learning in software development and operations organizations focuses specifically on these outputs, beginning with the various forms they take. All of the other details about the incident—the incident itself, what happened during the retrospective, and even how those artifacts came to be created—were considered to be a black box.

The study of these artifacts began at a broader, industry-wide level, by soliciting retrospective and postmortem templates via survey. These templates were then analyzed for structural elements in order to find commonalities (examples include an incident summary, basic timeline, and action items were the top three structures observed in postmortem templates), as well as the more unique structures. (Among the least common elements: document version/last modified date, a reminder to template users that root cause does not exist, and broad organizational findings.)

Perhaps the most notable finding from analyzing these various postmortem templates was that different template archetypes are used within the industry, each with a different focus and serving a different purpose. Three were apparent from the industry samples:

- *The Record-keeper.* This is the most common industry template and what most practitioners think of when they think of a postmortem report: It serves to provid ditional prompts and "hints" to facilitate the running of post-incident analysis processes. These can include questions the organization wants asked during postmortem meetings or reminders to participants about the cultural ethos the organization values (blamelessness, for example) or otherwise wants highlighted to participants or facilitators during these processes.
- *The Facilitator.* While similar in structure to the record-keeper, the facilitator includes additional prompts and "hints" to facilitate the running of post-incident analysis processes. These can include questions the or-

ganization wants asked during postmortem meetings or reminders to participants about the cultural ethos the organization values (blamelessness, for example) or otherwise wants highlighted to participants or facilitators during these processes.

▸ *The Signpost.* This template archetype can be aptly described as a pointer: It can provide either a reporting function, to be distributed to the larger organization for training or information purposes, or serve as a shorthand "itemized receipt," pointing to additional data sources, usually various organizational systems of record, regarding the incident. In either case, it is marked by a lightweight treatment of the incident and the analysis outcomes and, as such, is typically used as a means of broad organizational communication regarding (especially impactful) incidents.

These three template archetypes do not preclude the existence of others; if more industry templates were collected and analyzed, other commonalities with enough uniquely identifiable elemental structures could define additional archetypes. In fact, as the practice of incident analysis evolves within the industry, so too should these archetypes.

Artifact Usage in the Production Environment

The second phase of inquiry into the industry's use of post-incident analysis artifacts centered around a phenomenological case study of their observed actual use in a living, breathing organization, and the effects of that usage. An important aspect of selecting an organization for the case study was it both develop software *and* operate that software. It had to be considered a high-performing organization under the guidelines described in the 2016 and 2017 State of DevOps reports. Twelve engineers from three distinct teams (development, operations, and security) were observed over the course of three months to see how they used various post-incident artifacts in the course of responding to incidents—analyzing, remediating, and learning from them. During this period, artifacts from the organization's actual incidents were also collected and analyzed.

Looking back at the past to improve the future has become front-of-mind for many companies, precisely because the cost of not doing so in the development phase of software can be nebulous to measure, but the cost of not doing so in software operations is very apparent.

One of the initial findings was that different teams use these same post-incident analysis artifacts in different ways to go about their work. Various themes emerged in analyzing the frequency of references each engineer made to different specific uses of artifacts. Operations engineers, for example, used the artifacts to perform trend analysis about various system factors and for other longer-term uses (the creation of models for bucketing their company's incidents, for example). They also made heavy use of the artifacts to create knowledge base-type information repositories for operational work. (In fact, their use of the artifacts to generate and update documentation was notably higher than other groups.)

Developers tended to use these artifacts to help determine (what they refer to as) the "root cause" of an incident, as well as to generate requirements specifications for new feature work and architectural refactoring. Artifacts were also used to justify or clarify engineering decisions that had been previously made both to new team members and to other teams, but that individual engineers had forgotten the specific reasoning for over time. (Astute followers of the safety sciences will be familiar with the problems associated with the concept of root cause; those discussions aside, it is worth noting that developers used the term *root cause* twice as often as security engineers used it, who used it twice again as often as operations engineers, who seldom used it at all.)

Finally, security engineers used the artifacts more than other teams as one of the primary tools to drive their work. In the context of responding to security incidents, this makes intuitive sense: Security engineers need to respond to real-world threats they are seeing against production systems, so they use past incidents as a way of getting stronger signals indicating where they should plan their efforts and focus for the future. This includes guiding the generation and distribution of security-related documentation and driving internal security product roadmaps.

Taken together, these various uses add up to more than the sum of their parts. In today's modern distributed systems, it is neither novel nor controversial to point out that engineers work in complex systems. In the safety sciences, the term *complex socio-technical system* is usually used to point out that systems are an amalgam of not only code, compute, network, and storage, but also of people and teams. These people naturally have competing priorities, preferences, incentives, and goals, and they are often confronted with situations where they have to make critical decisions under extreme time and stress pressures, where all these factors consciously (and subconsciously) weigh into their decisions and actions.

One of the most important findings about the uses of these post-incident artifacts is that actors use them to help create and update mental maps of the emergent, complex socio-technical systems that they are responsible for engaging with. Because these Web-scale complex software and infrastructure systems constantly evolve, both in terms of technology and the teams behind that technology, individuals', teams', and even the organization's mental maps of how systems work can degrade over time. Anyone who has been frustrated at finding four architectural diagrams on the internal wiki, none of which is current, has experienced this. Incident artifacts provide, in effect, "patches" to these maps, allowing engineers and teams to update their above-the-line representations of the system and to discuss with each other where their cross-boundary (team or system) mental models were mismatched, inaccurate, or otherwise hampered their work.

This updating of the map of the organization's complex socio-technical systems was observed in a couple of ways. First, the artifacts provided evidence of a linkage between seemingly disparate, unconnected components of the wider system. There were many technical examples of this ("This microservice, in a particular failure mode, will call this other microservice that it used to rely on, but that dependency was thought to be removed; however, the dependency actually still exists, but only in this specific error condition"). But this effect also identified unknown and missing linkages *between people and teams* in the system. The most prominent example was a team that turned out to be fielding a large number of security issues. They were located in a different state and focused on customer support, so they had no way to contact security engineers who could help them; because of this, a security incident occurred, and one of the updates to the *socio* part of the socio-technical system map was, "These people need to be introduced to those people, and an ongoing channel of communication needs to be established between them." Part of this included a need for training, which was eventually rolled out to a series of teams.

The second way this artifact usage was observed was as a way to identify hot spots within the socio-technical system. The old adage, "Where there's smoke, there's fire," is apt here, and post-incident analysis artifacts give engineers a sense of whether the smoke is from a small grease fire that set off the kitchen smoke detector for a few seconds, or if the smoke is visible from four blocks away and potentially more attention should be paid. Again, this provides input into mapping the *terrain* of the complex socio-technical system on which not only operations engineers are operating, but also developers are updating and changing, and security engineers are defending from external attack. This "smoke" can be indicative of (again, both technical and social) areas the organization has neglected and needs to invest more in, but it can also highlight entirely emergent areas that need to be addressed merely because the complex system has evolved in some unconceived way.

As an example of this effect, a security engineer disabled a particular set of options available to engineers via the use of a company-wide networking library; this improved the company's security posture. Some days later, a team went to deploy a new version of their microservice, and the deployment prompted an outage. After the issue was detected and remediated, one of the "smoky" issues the incident analysis raised, via distribution of the post-incident artifacts, was that the security team did not have any data on which versions of their library were in use across the company.

This was not neglect in terms of the organization focusing on other priorities; rather, it was the system had evolved in terms of microservice- and software-dependency complexity to such a point that such data was now worth collecting and could highlight other potential problems, where a factor is teams using older versions that had been assumed to have been deprecated. This resulted in both a technical solution (starting to track library version use) *and* a social solution (that team now regularly engages other teams which the data shows are continuing to use old versions of the library to see why they have not migrated, if they can help them migrate, and if they need any new features before they do so).

A Move Toward Dynamic Remediation Items

Industry survey data indicates that 91% of respondents consider collection and recording of remediation items to be the core purpose of their post-incident analysis meetings and the artifacts produced from those meetings. Spending three months watching how a high-performing organization used their artifacts differently, however, sheds light on another approach: a focus on collecting, understanding, and sharing deeper, richer context about the technical state of a subsystem and the priorities, preferences, incentives, and constraints of the team responsible for operating and maintaining it. In this organization's environment, static lists of remediation items took a back seat to the search for and promulgation of this rich context.

The prevailing organizational focus during the post-incident analysis phase, and thus encoded into the documents produced by that phase, included:

- How individuals and teams handled the incident and how they coordinated their work.
- What their mental models were of the system at the time, including the state of the code, the infrastructure, and the expectations of other teams, and how those mental models contributed to their decision making.
- Where their mental models were divergent and the effects of this divergence during incident response.
- At the edges of the incident, what context the team had for factors that

may have contributed to the incident (that is, what other pressures, incentives, or circumstances the team faced with that may have made their local environment more prone to promoting factors identified as related to the incident).

Rote remediation items are not where the bulk of the discussion occurs. Of course, it's not that remediation items are not discussed; rather, it's the expectation that the team has internally identified the items they are responsible for *before* the post-incident analysis and are (allowed to be) responsible for deciding on the prioritization of those fixes. In some cases, they are completed before the postmortem meeting. In others, further discussion is required to gain—you guessed it—further context, to understand fully all the potential remediations and their relative priority in a broader organizational context.

Perhaps most fascinating: Teams can decide *not* to implement remediation items at all. They may determine that taking a series of small outages that they believe can be remediated quickly enough is the right decision, given the other priorities the organization has tasked them with. This works in their organization because it is recognized that the development, operations, and security teams are closest to the systems they operate, and therefore are trusted to make the right decisions, given their local rationality and the context they have gathered from the other teams and systems around them. If that decision results in further outages that impact the rest of the organization or customers, then the exchange of context flows the other way between the involved teams—not a lis t of remediation items for a specific incident—and drives a more resilient, flexible resolution. One engineer aptly describes this model as "strategic accountability more than tactical accountability."

This sharing of context has another benefit: It promotes the concept of blamelessness. The idea of the blameless postmortem has been bandied about in the industry for quite a while and has been met with some skepticism. With outages that have the potential to cost millions (or even pose an existential threat to the company—just ask Knight Capital), it is entirely understandable to wonder if blamelessness can ever exist when the tempo is high and the consequences are very real. But because this search for and exchange of the context of the various subcomponents of the socio-technical system are valued higher than remediation items alone, in the aftermath of incidents the first step to understanding what happened is "share the context for why whatever happened, happened." This is a marked departure from an approach that begins with the question, "What did you do?" and then seeks to hold a group referendum on whether or not that was the "correct" action to have taken.

Early Times, Exciting Times

The technology industry loves to hold aviation as the gold standard in incident and accident investigation, but it was not always that way. One of the biggest contributions to improved aviation safety was the introduction of crew resource management (CRM) in the 1980s. The insight that brought CRM to the fore of the aviation industry was not based on a set of remediation items from a specific accident, but rather from a holistic view of a series of accidents and looking for commonalities across companies, situations, equipment, and people. It was born not of a focus on piecemeal fixes but on a realization that improving how people go about doing their work, interacting with each other and their equipment, and effectively communicating about and responding to changes in their complex socio-technical environment is a place where some of the biggest discoveries of "hot spots" can be and where the biggest safety wins can emerge.

Given that humanity's study of the sociological factors in safety is almost a century old, the technology industry's post-incident analysis practices and how we create and use the artifacts those practices produce are all still in their infancy. So don't be surprised that many of these practices are so similar, that the cognitive and social models used to parse apart and understand incidents and outages are few and cemented in the operational ethos, and that the byproducts sought from post-incident analyses are far-and-away focused on remediation items and prevention (often with varying degrees of blame sprinkled in, whether we want to admit it or not).

But it doesn't have to stay this way. The industry is prime for a renaissance, but we must get past the notion the only value of post-incident analysis is in the list of static remediation items that so many of those processes are modeled, even optimized, to produce. Disavowing this notion requires becoming comfortable with moving away from the (admittedly comforting) assumption that if all the items on that list are implemented—we "100% remediate the incident!"—then it won't happen again.

Getting past that (admittedly tall) hurdle can create the cognitive and social space needed to explore all the various lessons an impactful, even painful, incident is trying to impart. Organizations can begin to approach solutions not from a list of tasks and bug fixes that try to address a situation that may never happen again, but instead from a place of moving toward broader solutions that address factors which tend to create situations where such incidents can occur. And this, ultimately, will push incident-analysis processes to evolve from such a laser-focus on the specific event that resulted in our Bad Day, toward what that Bad Day reveals about the true nature of our practices, processes, incentives, local contexts, the complex systems we operate every day, and perhaps most valuably: each other.

Related articles on queue.acm.org

Postmortem Debugging in Dynamic Environments
David Pacheco
https://queue.acm.org/detail.cfm?id=2039361

The Network is Reliable
Peter Bailis, Kyle Kingsbury
https://queue.acm.org/detail.cfm?id=2655736

Why SRE Documents Matter
Shylaja Nukala and Vivek Rau
https://queue.acm.org/detail.cfm?id=3283589

J. Paul Reed is a a senior applied resilience engineer on Netflix's CORE team in San Francisco, CA, where he focuses on incident analysis, systemic risk identification and mitigation, resilience engineering, and the human factors expressed in company's various socio-technical systems.