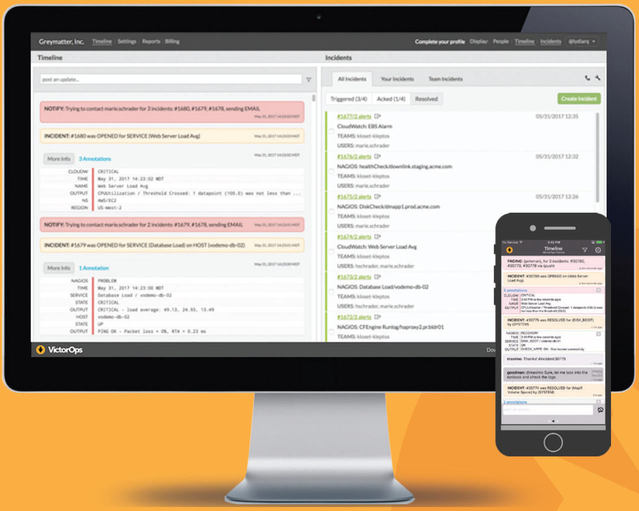
O'REILLY®

Compliments of VictorOps

Post-Incident Reviews

Learning from Failure for Improved Incident Response

Jason Hand



Post-Incident Reviews

Learning From Failure for Improved Incident Response

Jason Hand

Beijing · Boston · Farnham · Sebastopol · Tokyo O'REILLY®

Post-Incident Reviews

by Jason Hand

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles *http://oreilly.com/safari*). For more information, contact our corporate/institutional sales department: 800-998-9938 or *corporate@oreilly.com*.

Editors: Brian Anderson and Virginia Wilson
Production Editor: Kristen Brown
Copyeditor: Rachel Head
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

July 2017: First Edition

Revision History for the First Edition

2017-07-10: First Release

While the publisher and the author have used good faith efforts to ensure that the information and instructions contained in this work are accurate, the publisher and the author disclaim all responsibility for errors or omissions, including without limitation responsibility for damages resulting from the use of or reliance on this work. Use of the information and instructions contained in this work is at your own risk. If any code samples or other technology this work contains or describes is subject to open source licenses or the intellectual property rights of others, it is your responsibility to ensure that your use thereof complies with such licenses and/or rights.

978-1-491-98693-6

[LSI]

Table of Contents

Foreword. vii

Introduction. xi

1. Broken Incentives and Initiatives. 1

Control 2

A Systems Thinking Lens 3

2. Old-View Thinking. 7

What’s Broken? 8

The Way We’ve Always Done It 8

Change 11

3. Embracing the Human Elements. 13

Celebrate Discovery 13

Transparency 14

4. Understanding Cause and Effect. 17

Cynefin 17

From Sense-Making to Explanation 20

Evaluation Models 20

5. Continuous Improvement. 25

Creating Flow 26

Eliminating Waste 26

Feedback Loops 27

6. Outage: A Case Study Examining the Unique Phases of an Incident 35

Day One 35

Day Two 38

7. The Approach: Facilitating Improvements. 43

Discovering Areas of Improvement 43

Facilitating Improvements in Development and Operational Processes 44

8. Defining an Incident and Its Lifecycle. 49

Severity and Priority 49

Lifecycle of an Incident 51

9. Conducting a Post-Incident Review. 59

Who 60

What 61

When 62

Where 63

How 63

Internal and External Reports 71

10. Templates and Guides. 77

Sample Guide 77

11. Readiness. 89

Next Best Steps 91

Foreword

“I know we don’t have tests for that, but it’s a small change; it’s probably fine...”

“I ran the same commands I always do, but...something just doesn’t seem quite right.”

“That `rm -rf` sure is taking a long time!”

If you’ve worked in software operations, you’ve probably heard or uttered similar phrases. They mark the beginning of the best “Ops horror stories” the hallway tracks of Velocity and DevOps Days the world over have to offer. We hold onto and share these stories because, back at that moment in time, what happened next to us, our teams, and the companies we work for became a epic journey.

Incidents (and managing them, or...not, as the case may be) is far from a “new” field: indeed, as an industry, we’ve experienced incidents as long as we’ve had to operate software. But the last decade has seen a renewed interest in digging into how we react to, remediate, and reason after-the-fact about incidents.

This increased interest has been largely driven by two tectonic shifts playing out in our industry: the first began almost two decades ago and was a consequence of a change in the types of products we build. An era of shoveling bits onto metallic dust-coated plastic and laser-etched discs that we then shipped in cardboard boxes to users to install, manage, and “operate” themselves has given way to a cloud-connected, service-oriented world. Now we, *not* our users, are on the hook to keep that software running.

The second industry shift is more recent, but just as notable: the DevOps movement has convincingly made the argument that "if you build it, you should also be involved (at least in *some way*) in running it," a sentiment that has spurred many a lively conversation about who needs to be carrying pagers these days! This has resulted in *more* of us, from ops engineers to developers to security engineers, being involved in the process of operating software on a daily basis, often in the very midst of operational incidents.

I had the pleasure of meeting Jason at Velocity Santa Clara in 2014, after I'd presented "A Look at Looking in the Mirror," a talk on the very topic of operational retrospectives. Since then, we've had the opportunity to discuss, deconstruct, and debate (blamelessly, of course!) many of the ideas you're about to read. In the last three years, I've also had the honor of spending time with Jason, sharing our observations of and experiences gathered from real-world practitioners on where the industry is headed with post-incident reviews, incident management, and organizational learning.

But the report before you is more than just a collection of the "whos, whats, whens, wheres, and (five) whys" of approaches to post-incident reviews. Jason explains the underpinnings necessary to hold a productive post-incident review and to be able to consume those findings within your company. This is not just a "postmortem how-to" (though it has a number of examples!): this is a "postmortem why-to" that helps you to understand not only the true complexity of your technology, but also the *human side* that *together* make up the socio-technical systems that are the reality of the modern software we operate every day.

Through all of this, Jason illustrates the positive effect of taking a "New View" of incidents. If you're looking for ways to get better answers about the factors involved in your operational incidents, you'll learn myriad techniques that can help. But more importantly, Jason demonstrates that it's not just about getting better answers: it's about *asking better questions*.

No matter where you or your organization are in your journey of tangling with incidents, you have in hand the right guide to start improving your interactions with incidents.

And when you hear one of those hallowed phrases that you know will mark the start of a great hallway track tale, after reading this guide, you'll be confident that after you've all pulled together to fix the outage and once the dust has settled, you'll know exactly what you and your team need to do to turn that incident on its head and harness all the lessons it has to teach you.

— J. Paul Reed
DevOps consultant and retrospective researcher
San Francisco, CA
July 2017

Introduction

In the summer of 2011 I began my own personal journey into learning from failure. As the Director of Tech Support for a startup in Boulder, Colorado, I managed all inbound support requests regarding the service we were building: a platform to provision, configure, and manage cloud infrastructure and software.

One afternoon I received a support request to assist a customer who wished to move their instance of Open CRM from Amazon Web Services (AWS) to a newer cloud provider known as Green Cloud whose infrastructure as a service was powered by "green" technologies such as solar, wind, and hydro. At that time, running an instance similar in size was significantly more cost effective on Green Cloud as well.

Transferring applications and data between cloud providers was one of the core selling points of our service, with only a few clicks required to back up data and migrate to a different provider. However, occasionally we would receive support requests when customers didn't feel like they had the technical skills or confidence to make the move on their own. In this case, we established a date and time to execute the transition that would have the lowest impact on the customer's users. This turned out to be 10 p.m. for me.

Having performed this exact action many times over, I assured the customer that the process was very simple and that everything should be completed in under 30 minutes. I also let them know that I would verify that the admin login worked and that the MySQL databases were populated to confirm everything worked as expected.

Once the transfer was complete, I checked all of the relevant logs; I connected to the instance via SSH and stepped through my checklist of things to verify before contacting the customer and closing out the support ticket. Everything went exactly as expected. The admin login worked, data existed in the MySQL tables, and the URL was accessible.

When I reached out to the customer, I let them know everything had gone smoothly. In fact, the backup and restore took less time than I had expected. Recent changes to the process had shortened the average maintenance window considerably. I included my personal phone number in my outreach to the customer so that they could contact me if they encountered any problems, especially since they would be logging in to use the system several hours earlier than I'd be back online—they were located in Eastern Europe, so would likely be using it within in the next few hours.

Incident Detection

Within an hour my phone began blowing up. First it was an email notification (that I slept through). Then it was a series of push notifications tied to our ticketing system, followed almost immediately by an SMS from the customer. There was a problem.

After a few back-and-forth messages in the middle of the night from my mobile phone, I jumped out of bed to grab my laptop and begin investigating further. It turned out that while everything looked like it had worked as expected, the truth was that nearly a month's worth of data was missing. The customer could log in and there was data, but it wasn't up to date.

Incident Response

At this point I reached out to additional resources on my team to leverage someone with more experience and knowledge about the system. Customer data was missing, and we needed to recover and restore it as quickly as possible, if that was possible at all. All the ops engineers were paged, and we began sifting through logs and data looking for ways to restore the customer's data, as well as to begin to understand what had gone wrong.

Incident Remediation

Very quickly we made the horrifying discovery that the backup data that was used in the migration was out of date by several months. The migration process relies on backup files that are generated every 24 hours when left in the default setting (users, however, could make it much more frequent). We also found out that for some reason current data had not been backed up during those months. That helped to explain why the migration contained only old data. Ultimately, we were able to conclude that the current data was completely gone and impossible to retrieve.

Collecting my thoughts on how I was going to explain this to the customer was terrifying. Being directly responsible for losing months' worth of the data that a customer relies on for their own business is a tough pill to swallow. When you've been in IT long enough, you learn to accept failures, data loss, and unexplainable anomalies. The stakes are raised when it impacts someone else and their livelihood. We all know it could happen, but hope it won't.

We offered an explanation of everything we knew regarding what had happened, financial compensation, and the sincerest of apologies. Thankfully the customer understood that mistakes happen and that we had done the best we could to restore their data. With any new technology there is an inherent risk to being an early adopter, and this specific customer understood that. Accidents like this are part of the trade-off for relying on emerging technology and services like those our little tech startup had built.

After many hours of investigation, discussion, and back and forth with the customer, it was time to head in to the office. I hadn't slept for longer than an hour before everything transpired. The result of my actions was by all accounts a "worst-case scenario." I had only been with the company for a couple of months. The probability of being fired seemed high.

Incident Analysis

Once all of the engineers, the Ops team, the Product team, and our VP of Customer Development had arrived, our CEO came to me and said, "Let's talk about last night." Anxiously, I joined he and the others in the middle of our office, huddled together in a circle. I was then prompted with, *"Tell us what happened."*

I began describing everything that had taken place, including when the customer requested the migration, what time I began the process, when I was done, when I reached out to them, and when they let me know about the data loss. We started painting a picture of what had happened throughout the night in a mental timeline.

To cover my butt as much as possible, I was sure to include extra assurances that I had reviewed all logs after the process and verified that every step on my migration checklist was followed, and that there were never any indications of a problem. In fact, I was surprised by how quickly the whole process went.

Having performed migrations many times before, I had a pretty good idea of how long something like this should take given the size of the MySQL data. In my head, it should have taken about 30 minutes to complete. It actually only took about 10 minutes. I mentioned that I was surprised by that but knew that we had recently rolled out a few changes to the backup and restore process, so I attributed the speediness of the migration to this new feature.

I continued to let them know what time I reached out to the Ops team. Although time wasn't necessarily a huge pressure, finding the current data and getting it restored was starting to stretch my knowledge of the system. Not only was I relatively new to the team, but much about the system—how it works, where to find data, and more—wasn't generally shared outside the Engineering team.

Most of the system was architected by only a couple of people. They didn't intentionally hoard information, but they certainly didn't have time to document or explain every detail of the system, including where to look for problems and how to access all of it.

As I continued describing what had happened, my teammates started speaking up and adding more to the story. By this point in our mental timeline we each were digging around in separate areas of the system, searching for answers to support our theories regarding what had happened and how the system behaved. We had begun to divide and conquer with frequent check-ins over G-chat to gain a larger understanding about the situation from each other.

I was asked how the conversation went when I reached out to the customer. We discussed how many additional customers might be affected by this, and how to reach out to them to inform them of a possible bug in the migration process.

Several suggestions were thrown out to the Operations team about detecting something like this sooner. The engineers discussed adding new logging or monitoring mechanisms. The Product team suggested pausing the current sprint release so that we could prioritize this new work right away. Everyone, including the CEO, saw this as a learning opportunity, and we all walked away knowing more about:

- How the system actually worked
- What problems existed that we were previously unaware of
- What work needed to be prioritized

In fact, we all learned quite a bit about what was really going on in our system. We also gained a much clearer picture of how we would respond to something like this. Being a small team, contacting each other, and collaborating on the problem was just like any other day at the office. We each knew one another's cell phones, emails, and G-chat handles. Still, we discovered that in situations like this someone from the Ops team should be pulled in right away, until access can be provided to more of the team and accurate documentation is made available to everyone. We were lucky that we could coordinate and reach each other quickly to get to the bottom of the problem.

As we concluded discussing what we had learned and what action items we had as takeaways, everyone turned and headed back to their desks. It wasn't until that moment that *I realized I had never once been accused of anything*. No one seemed agitated with me for the decisions I'd made and the actions I took. There was no blaming, shaming, or general animosity toward me. In fact, I felt an immense amount of empathy and care from my teammates. It was as though everyone recognized that they likely would have done the exact same thing I had.

Incident Readiness

The system was flawed, and now we knew what needed to be improved. Until we did so, the exact same thing was at risk of happening again. There wasn't just one thing that needed to be fixed. There were many things we learned and began to immediately improve. I became a much better troubleshooter and gained access to parts of the system where I can make a significant positive impact in the recovery efforts moving forward.

For modern IT organizations, maintaining that line of reasoning and focus on improving the system as a whole is the difference between being a high-performing organization or a low-performing one. Those with a consistent effort toward continuous improvement along many vectors come out on top. Looking for ways to improve our understanding of our systems as well as the way in which teams respond to inevitable failure means becoming extremely responsive and adaptable. Knowing about and remediating a problem faster moves us closer to a real understanding of the state and behavior of our systems.

What would have happened if this latent failure of the automated backup process in the system had lain dormant for longer than just a few months? What if this had gone on for a year? What if it was happening to more than just Open CRM instances on AWS? What if we had lost data that could have taken down an entire company?

In order to answer those questions better, we will leverage the use of a *post-incident review*. A type of analytic exercise, post-incident reviews will be explored in depth in Chapter 8; you'll see how we know what an incident is as well as when it is appropriate to perform an analysis.

As we'll learn in the coming chapters, old-view approaches to retrospective analysis of incidents have many flaws that inherently prevent us from learning more about our systems and how we can continuously improve them.

By following a new approach to post-incident reviews, we can make our systems much more stable and highly available to the growing number of people that have come to rely on the service 24 hours a day, every day of every year.

What's Next?

This short book sets out to explore why post-incident reviews are important and how you and your team can best execute them to continuously improve many aspects of both building resilient systems and responding to failure sooner.

Chapters 1 and 2 examine the current state of addressing failure in IT organizations and how old-school approaches have done little to help provide the right scenario for building highly available and reliable IT systems.

Chapter 3 points out the roles humans play in managing IT and our shift in thinking about their accountability and responsibility with regard to failure.

In Chapters 4 and 5 we will begin to set the context of what we mean by an incident and develop a deeper understanding of cause and effect in complex systems.

Chapter 6 begins to get us thinking about why these types of exercises are important and the value they provide as we head into a case study illustrating a brief service disruption and what a simple post-incident review might look like.

The remainder of the book (Chapters 7 through 10) discusses exactly how we can approach and execute a successful post-incident review, including resources that may help you begin preparing for your next IT problem. A case study helps to frame the value of these exercises from a management or leadership point of view.

We'll conclude in Chapter 11 by revisiting a few things and leaving you with advice as you begin your own journey toward learning from failure.

Acknowledgments

I'd like to give an extra special "thank you" to the many folks involved in the creation of this report.

The guidance and flexibility of my editors Brian Anderson, Virginia Wilson, Susan Conant, Kristen Brown, and Rachel Head was greatly appreciated and invaluable. Thank you to Matthew Boeckman, Aaron Aldrich, and Davis Godbout for early reviews, as well as Mark Imbriaco, Courtney Kissler, Andi Mann, John Allspaw, and Dave Zwieback for their amazing and valuable feedback during the technical review process. Thanks to Erica Morrison and John Paris for your wonderful firsthand stories to share with our readers.

Thank you to J. Paul Reed, who was the first presenter I saw at Velocity Santa Clara in 2014. His presentation "A Look At Looking In the Mirror," was my first personal exposure to many of the concepts I've grown passionate about and have shared in this report.

Special thanks to my coworkers at VictorOps and Standing Cloud for the experiences and lessons learned while being part of teams tasked with maintaining high availability and reliability. To those

before me who have explored and shared many of these concepts, such as Sidney Dekker, Dr. Richard Cook, Mark Burgess, Samuel Arbesman, Dave Snowden, and L. David Marquet…your work and knowledge helped shape this report in more ways than I can express. Thank you so much for opening our eyes to a new and better way of operating and improving IT services.

I'd also like to thank John Willis for encouraging me to continue spreading the message of learning from failure in the ways I've outlined in this report. Changing the hearts and minds of those set in their old way of thinking and working was a challenge I wasn't sure I wanted to continue in late 2016. This report is a direct result of your pep talk in Nashville.

Last but not least, thank you to my family, friends, and especially my partner Stephanie for enduring the many late nights and weekends spent in isolation while I juggled a busy travel schedule and deadlines for this report. I'm so grateful for your patience and understanding. Thank you for everything.

CHAPTER 1
Broken Incentives and Initiatives

All companies have and rely on technology in numerous and growing ways. Increasingly, the systems that make up these technologies are expected to be "highly available," particularly when the accessibility and reliability of the system tie directly to the organization's bottom line. For a growing number of companies, the modern world expects 100% uptime and around-the-clock support of their services.

Advancements in the way we develop, deliver, and support IT services within an organization are a primary concern of today's technology leaders. Delivering innovative and helpful service to internal and external stakeholders at a high level of quality, reliability, and availability is what DevOps has set out to achieve. But with these advancements come new challenges. How can we ensure that our systems not only provide the level of service we have promised our customers, but also continue a trajectory of improvement in terms of people, process, and technology so that these key elements may interact to help detect and avoid problems, as well as solve them faster when they happen?

Anyone who has worked with technology can attest that eventually something will go wrong. It's not a matter of if, but when. And there is a direct relationship between the size and complexity of our systems and the variety of factors that contribute to both working and non-working IT services.

In most companies, success and leadership are about maintaining control—control of processes, technologies, and even people. Most of what has been taught or seen in practice follows a command and control pattern. Sometimes referred to as the leader–follower structure, this is a remnant of a successful labor model when mankind's primary work was physical.[1]

In IT, as well as many other roles and industries, the primary output and therefore most important work we do is cognitive. It's no wonder the leader–follower structure L. David Marquet describes in *Turn the Ship Around!* has failed when imposed in modern IT organizations. It isn't optimal for our type of work. It limits decision-making authority and provides no incentive for individuals to do their best work and excel in their roles and responsibilities. Initiative is effectively removed as everyone is stripped of the opportunity to utilize their imagination and skills. In other words, nobody ever performs at their full potential.[2]

This old-view approach to management, rooted in physical labor models, doesn't work for IT systems, and neither does the concept of "control," despite our holding on to the notion that it is possible.

Control

Varying degrees of control depend on information and the scale or resolution at which we are able to perceive it. Predictability and interaction are the key necessary components of control.

> We sometimes think we are in control because we either don't have or choose not to see the full picture.
>
> —Mark Burgess, *In Search of Certainty*

Unfortunately, we never have absolute certainty at all scales of resolution. Information we perceive is limited by our ability to probe systems as we interact with them. We can't possibly know all there is to know about a system at every level. There is no chance of control in the IT systems we work on. We are forced to make the most of unavoidable uncertainty.[3]

[1] L. David Marquet, *Turn the Ship Around!* (Portfolio/Penguin), xxi.

[2] Ibid.

[3] Mark Burgess. *In Search of Certainty: The Science of Our Information Infrastructure* (O'Reilly, Kindle Edition), 23.

To manage the uncertain and make IT systems more flexible and reliable, we employ new models to post-incident analysis. This approach means we gather and combine quantitative and qualitative data, analyze and discuss the findings, and allow theories from unbiased perspectives regarding "normal" (but always changing) behaviors and states of our systems to form, spreading knowledge about how the system works.

This growth mindset approach encourages teams to carve out time for retrospection in order to analyze what has taken place in as much detail as possible. Along the way, specific details are surfaced about how things transpired and what exactly took place during recovery efforts. Those may include:

- A detailed timeline of events describing what took place during remediation
- Key findings that led to a deeper and unified understanding and theory of state and behavior with regard to people, process, and technology
- An understanding of how these elements relate to the evolving systems we build, deliver, operate, and support

These details are analyzed more deeply to discover ways to know about and recover from problems sooner. A larger understanding of the system as well as everything that contributed to a specific incident being discussed emerges as the timeline comes together. Thus, details for actionable improvements in many areas are provided along an incident's timeline.

A Systems Thinking Lens

Nearly all practitioners of DevOps accept that complicated, complex, and chaotic behaviors and states within systems are "normal." As a result, there are aspects to the state and behavior of code and infrastructure that can only be understood in retrospect. To add to the already non-simple nature of the technology and process concerns of our systems, we must also consider the most complex element of all of the system: people, specifically those responding to service disruptions.

We now know that judgments and decisions from the perspective of the humans involved in responses to managing technology must be included as relevant data.

"Why did it make sense to take that action?" is a common inquiry during a post-incident review. Be cautious when inquiring about "why," however, as it often may subtly suggest that there's going to be a judgment attached to the response, rather than it simply being the beginning of a conversation. An alternative to that question is, *"What was your thought process when you took that action?"* Often arriving at the same result, this line of inquiry helps to avoid judgment or blame.

This slight difference in language completely changes the tone of the discussion. It opens teams up to productive conversations.

NOTE

One approach that can help reframe the discussion entirely and achieve great results is to begin the dialogue by asking *"What does it look like when this goes well?"*

Consider the company culture you operate in, and construct your own way to get the conversation started. Framing the question in a way that encourages open discussion helps teams explore alternatives to their current methods. With a growth mindset, we can explore both the negative and the positive aspects of what transpired.

We must consider as much data representing both positive and negative impacts to recovery efforts and from as many diverse and informative perspectives as possible. Excellent data builds a clearer picture of the specifics of what happened and drives better theories and predictions about the systems moving forward, and how we can consistently improve them.

Observations that focus less on identifying a cause and fix of a problem and more on improving our understanding of state and behavior as it relates to all three key elements (people, process, and technology) lead to improved theory models regarding the system. This results in enhanced understanding and continuous improvement of those elements across all phases of an incident: *detection*, *response*, *remediation*, *analysis*, and *readiness*.

Surfacing anomalies or phenomena that existing behavior and state theories cannot explain allows us to seek out and account for exceptions to the working theory. Digging into these outliers, we can then improve the theories regarding aspects that govern our working knowledge of the systems.

During analysis of an incident, such as the data loss example described in the Introduction, we categorize observations (missing data) and correlations with the outcomes (unsuccessful data migration) of interest to us. We then attempt to understand the causal and contributing factors (data had not been backed up in months) that led to those outcomes. My presumptuous correlation between the speedy migration and a recent feature release led me to dismiss it as an area to investigate sooner. That assumption directly impacted the time it took to understand what was happening. These findings and deeper understanding then turn into action items (countermeasures and enhancements to the system).

Why is this important? Within complex systems, circumstances (events, factors, incidents, status, current state) are constantly in motion, always changing. This is why understanding contributing factors is just as (or possibly more) important as understanding anything resembling cause.

Such a "systems thinking" approach, in which we examine the linkages and interactions between the elements that make up the entire system, informs our understanding of it. This allows for a broader understanding of what took place, and under what specific unique and interesting circumstances (that may, in fact, never take place again).

What is the value in isolating the "cause" or even a "fix" associated with a problem with such emergent, unique, and rare characteristics?

To many, the answer is simple: "none." It's a needle-in-a-haystack exercise to uncover the "one thing" that caused or kicked off the change in desired state or behavior. And long-standing retrospective techniques have demonstrated predictable flaws in improving the overall resiliency and availability of a service.

This old-school approach of identifying cause and corrective action does little to further learning, improvements, and innovation. Furthermore, such intense focus on identifying a direct cause indicates an implicit assumption that “systems fail (or succeed) in a linear way, which is not the case for any sufficiently complex system.”[4]

The current hypothesis with regard to post-incident analysis is that there is little to no real value in isolating a single cause for an event. The aim isn’t only to avoid problems but rather to be well prepared, informed, and rehearsed to deal with the ever-changing nature of systems, and to allow for safe development and operational improvements along the way.

> While some problems are obvious and the steps required to prevent them from occurring are tractable, we cannot allow that to be our only focus. We must also strive to design our human and technical systems to minimize the impact of inevitable problems that will occur, despite our best efforts, by being well-prepared.
>
> —Mark Imbriaco, Director of Platform Architecture, Pivotal

Failure can never be engineered out of a system. With each new bit that is added or removed, the system is being changed. Those changes are happening in a number of ways due to the vast interconnectedness and dependencies. No two systems are the same. In fact, the properties and “state” of a single system now are quite different from those of the same system even moments ago. It’s in constant motion.

Working in IT today means being skilled at detecting problems, solving them, and multiplying the effects by making the solutions available throughout the organization. This creates a dynamic system of learning that allows us to understand mistakes and translate that understanding into actions that prevent those mistakes from recurring in the future (or at least having less of an impact—i.e., graceful degradation).

By learning as much as possible about the system and how it behaves, IT organizations can build out theories on their “normal.” Teams will be better prepared and rehearsed to deal with each new problem that occurs, and the technology, process, and people aspects will be continuously improved upon.

[4] Jennifer Davis and Katherine Daniels, *Effective DevOps: Building a Culture of Collaboration, Affinity, and Tooling at Scale* (O’Reilly, Kindle Edition), location 1379 of 9606.

CHAPTER 2
Old-View Thinking

We've all seen the same problems repeat themselves. Recurring small incidents, severe outages, and even data losses are stories many in IT can commiserate over in the halls of tech conferences and forums of Reddit. It's the nature of building complex systems. Failure happens. However, in many cases we fall into the common trap of repeating the same process over and over again, expecting the results to be different. We investigate and analyze problems using techniques that have been well established as "best practices." We always feel like we can do it better. We think we are smarter than others—or maybe our previous selves—yet the same problems seem to continue to occur, and as systems grow, the frequency of the problems grows as well. Attempts at preventing problems always seem to be an exercise in futility. Teams become used to chaotic and reactionary responses to IT problems, unaware that the way it was done in the past may no longer apply to modern systems.

We have to change the way we approach the work. Sadly, in many cases, we don't have the authority to bring about change in the way we do our jobs. Tools and process decisions are made from the top. Directed by senior leaders, we fall victim to routine and the fact that no one has ever stopped to ask if what we are doing is actually helping.

Traditional techniques of post-incident analysis have had minimal success in providing greater availability and reliability of IT services.

In Chapter 6, we will explore a fictional case study of an incident to illustrate an example service disruption and post-incident review—

one that more closely represents a systematic approach to learning from failure in order to influence the future.

What's Broken?

When previous attempts to improve our systems through retrospective analysis have not yielded IT services with higher uptime, it can be tempting to give up on the practice altogether. In fact, for many these exercises turn out to be nothing more than boxes for someone higher up the management ladder to check. Corrective actions are identified, tickets are filed, "fixes" are put in place…. Yet problems continue to present themselves in unique but common ways, and the disruptions or outages continue to happen more frequently and with larger impact as our systems continue to grow.

This chapter will explore one "old-school" approach—root cause analysis—and show why it is not the best choice for post-incident analysis.

The Way We've Always Done It

Root cause analysis (RCA) is the most common form of post-incident analysis. It's a technique that has been handed down through the IT industry and communities for many years. Adopted in large part from industries such as manufacturing, RCAs lay out a clear path of asking a series of questions to understand the true cause of a problem. In the lore of many companies the story goes, "Someone long ago started performing them. They then became seen as a useful tool for managing failure." Rarely has anyone stopped to ask if they are actually helping our efforts to improve uptime.

I remember learning about and practicing RCAs in my first job out of college in 1999, as the clock ticked closer and closer to Y2K. It was widely accepted that "this is how you do it." To make matters worse, I worked for a growing manufacturing and assembly company. This approach to problem solving of "downtime" was ingrained into the industry and the company ethos. When something broke, you tracked down the actions that preceded the failure until you found the one thing that needed to be "fixed." You then detailed a corrective action and a plan to follow up and review every so often to verify everything was operating as expected again.

We employed methods such as the “5 Whys” to step through what had happened in a linear fashion to get to the bottom of the problem.

Sample RCA (Using the “5 Whys” Format)

1. Q: Why did the mainframe lose connection to user terminals?

   A: Because an uplink cable was disconnected.
2. Q: Why was a cable disconnected?

   A: Because someone was upgrading a network switch.
3. Q: Why was that person upgrading the switch?

   A: Because a higher-bandwidth device was required.
4. Q: Why was more bandwidth required?

   A: Because the office has expanded and new users need access to the system.
5. Q: Why do users need access to the system?

   A: To do their work and make the business money.

If we were to stop asking questions here (after 5 Whys), the root cause for this problem (as a result of this line of questioning) would be identified as: *the business wants to make money!*

Obviously this is a poor series of questions to ask, and therefore a terrible example of 5 Whys analysis. If you’ve performed them in the past, you are likely thinking that no one in their right mind would ask these questions when trying to understand what went wrong.

Herein lies the first problem with this approach: objective reasoning about the series of events will vary depending on the perspectives of those asking and answering the questions.

Had you performed your own 5 Whys analysis on this problem, you would have likely asked a completely different set of questions and concluded that the root of the problem was something quite different. For example, perhaps you would have determined that the technician who replaced the switch needs further training on how (or more likely when) to do this work “better.”

This may be a more accurate conclusion, but does this identification of a single cause move us any closer to a better system?

TIP

As availability expectations grow closer to 365×24×7, we should be designing systems without the expectation of having "dedicated maintenance windows." Try asking the collective group, "What is making it unsafe to do this work whenever we want?" This is a much better approach then asking, "Why were we doing this during business hours?"

This brings us to the second problem with this approach: our line of questioning led us to a human. By asking "why" long enough, we eventually concluded that the cause of the problem was human error and that more training or formal processes are necessary to prevent this problem from occurring again in the future.

This is a common flaw of RCAs. As operators of complex systems, it is easy for us to eventually pin failures on people. Going back to the lost data example in the Introduction, I was the person who pushed the buttons, ran the commands, and solely operated the migration for our customer. I was new to the job and my Linux skills weren't as sharp as they could have been. Obviously there were things that I needed to be trained on, and I could have been blamed for the data loss. But if we switch our perspective on the situation, I in fact discovered a major flaw in the system, preventing future similar incidents. There is always a bright side, and it's ripe with learning opportunities.

The emotional pain that came from that event was something I'll never forget. But you've likely been in this situation before as well. It is just part of the natural order of IT. We've been dealing with problems our entire professional careers. Memes circulate on the web joking about the inevitable "Have you rebooted?" response trotted out by nearly every company's help desk team. We've always accepted that random problems occur and that sometimes we think we've identified the cause, only to discover that either the fix we put in place caused trouble somewhere else or a new and more interesting problem has surfaced, rendering all the time and energy we put into our previous fix nearly useless. It's a constant cat-and-mouse game. Always reactionary. Always on-call. Always waiting for things to break, only for us to slip in a quick fix to buy us time to address our technical debt. It's a bad cycle we've gotten ourselves into.

Why should we continue to perform an action that provides little to no positive measurable results when availability and reliability of a service is ultimately our primary objective?

We shouldn't. In fact, we must seek out a new way: one that aligns with the complex conditions and properties of the environments we work in, where the members of our team or organization strive to be proactive movers rather than passive reactors to problems. Why continue to let things happen to us rather than actively making things happen?

No matter how well we engineer our systems, there will always be problems. It's possible to reduce their severity and number, but never to zero. Systems are destined to fail. In most cases, we compound the issue by viewing success as a negative, an absence of failure, avoidance of criticism or incident.

NOTE

When the goal is prevention, there is no incentive to strive for better! We're striving for absence over substance.

We've been looking at post-incident analysis the wrong way for quite some time. Focusing on avoiding problems distracts us from seeking to improve the system as a whole.

Change

What makes a system a system? Change. This concept is central to everything in this book. Our systems are constantly changing, and we have to be adaptable and responsive to that change rather than rigid. We have to alter the way we look at all of this.

If you're like me, it's very easy to read a case study or watch a presentation from companies such as Amazon, Netflix, or Etsy and be suspicious of the tactics they are suggesting. It's one thing to learn of an inspirational new approach to solving common IT problems and accept that it's how we should model our efforts. It's something quite different to actually implement such an approach in our own companies.

But we also recognize that if we don't change something we will be forever caught in a repeating cycle of reactionary efforts to deal with

IT problems when they happen. And we all know they *will* happen. It's just a matter of time. So, it's fair for us to suspect that things will likely continue to get worse if some sort of change isn't effected.

You're not alone in thinking that a better way exists. While stories from "unicorn" companies like those mentioned above may seem unbelievably simple or too unique to their own company culture, the core of their message applies to all organizations and industries.

For those of us in IT, switching up our approach and building the muscle memory to learn from failure is our way to a better world—and much of it lies in a well-executed post-incident analysis.

In the next four chapters, we'll explore some key factors that you need to consider in a post-incident analysis to make it a successful "systems thinking" approach.

CHAPTER 3

Embracing the Human Elements

As we all know, humans will need to make judgment calls during response and recovery of IT problems. They make decisions regarding what tasks to execute during remediation efforts based on what they know at that time. It can be tempting to judge these decisions in hindsight as being good or bad, but this should be avoided.

Celebrate Discovery

When accidents and failures occur, instead of looking for human error we should look for how we can redesign the system to prevent these incidents from happening again.[1]

A company that validates and embraces the human elements and considerations when incidents and accidents occur learns more from a post-incident review than those who are punished for actions, omissions, or decisions taken. Celebrating transparency and learning opportunities shifts the culture toward learning from the human elements. With that said, gross negligence and harmful acts must not be ignored or tolerated.

[1] Gene Kim, Jez Humble, Patrick Debois, and John Willis, *The DevOps Handbook* (IT Revolution), 40.

NOTE

Human error should never be a “cause.”

How management chooses to react to failure and accidents has measurable effects. A culture of fear is established when teams are incentivized to keep relevant information to themselves for fear of reprimand. Celebrating discovery of flaws in the system recognizes that actively sharing information helps to enable the business to better serve its purpose or mission. Failure results in intelligent discussion, genuine inquiry, and honest reflection on what exactly can be learned from problems.

Blaming individuals for their role in either the remediation or the incident itself minimizes the opportunity to learn. In fact, identifying humans as the cause of a problem typically adds to the process, approvals, and friction in the system. Blaming others for their involvement does only harm.

The alternative to blaming is praising. Encourage behavior that reinforces our belief that information should be shared more widely, particularly when exposing problems in the way work gets done and the systems involved. Celebrate the discovery of important information about the system. Don’t create a scenario where individuals are more inclined to keep information from surfacing.

Transparency

Nurturing discovery through praise will encourage transparency. Benefits begin to emerge as a result of showing the work that is being done. A stronger sense of accountability and responsibility starts to form.

Make Work (and Analysis) Visible

Many principles behind DevOps have been adopted from lean manufacturing practices. Making work visible is an example of this. It reinforces the idea that sharing more information about the work we do and how we accomplish it is important, so we can analyze and reflect on it with a genuine and objective lens.

Real-time conversations often take place in group chat tools during the lifecycle of an incident. If those conversations are not captured, it's easy to lose track of the what, who, and when data points. These conversations are relevant to the work and should be made visible for all to see. This is a big reason for the adoption of practices like ChatOps, which we'll learn more about in Chapter 9. Teams should be able to respond, collaborate, and resolve issues in a shared and transparent space. When work and conversations are visible, it becomes easier to spot areas for improvement, particularly as third-party observers and stakeholders observe remediation efforts unfolding.

Think back to the data loss incident described in the Introduction. My testimony the following morning on exactly what took place, when, and how helped make the work visible. Others who had no previous exposure to this work now had questions and suggestions on how to improve the process and the customer experience. It was an eye-opening exercise for many to discover a truer sense of how our systems were designed, including those responsible for responding to failures in IT. Everyone felt included, engaged, and empowered.

Making work visible during incident response helps to paint a very clear picture of what has occurred, what can be learned from the unplanned disruption of service, and what can be improved. Another important benefit of making work more transparent is that a greater sense of accountability exists.

It is very common for people to mistake responsibility for accountability. We often hear people proclaim, "Someone must be held accountable for this outage!"

To that, I'd say they are correct. We must hold someone accountable, but let's be clear on what we mean by that.

Holding someone *accountable* means that we expect them to provide an accurate "account" of what took place. We do expect as much information and knowledge as can be gleaned from responders. However, this is rarely what people mean when they demand accountability. What they actually mean is that someone must be held *responsible*.

NOTE

While blaming individuals is clearly counterproductive, it is important to seek out and identify knowledge or skill gaps that may have contributed to the undesirable outcome so that they can be considered broadly within the organization.

There's a small but significant difference between responsibility and accountability. Everyone is responsible for the reliability and availability of a service. Likewise, everyone should be held accountable for what they see and do as it relates to the development and operation of systems.

We often see a desire to hold someone responsible for outcomes, despite the fact that the person in question may not have had the authority to provide the expected level of responsibility, much less the perfect knowledge that would be required.

We need to evolve and create an environment where reality is honored and we extract it in a way that creates a culture of learning and improvement. Making problems visible provides an opportunity to learn, not an opportunity to blame.

> I know that for my team, it has been a really good learning process to not focus on "who." Now, team members come by and tell me things that they never would have told their previous leaders. Even examples of identifying something proactively and preventing an incident. They are proud of it. We celebrate those findings, publicly.
>
> —Courtney Kissler, Vice President, Retail Technology,
> Starbucks

Those holding leadership positions demanding to know who pushed the button that ultimately deleted a month's worth of customer data during a routine migration, for example, is *not* the way to encourage discovery in how to make things better. It is an expectation of someone to justify their actions or accept blame for the negative outcome. Unfortunately, the ultimate result of blaming, as pointed out previously, is that it leads to individuals keeping information to themselves rather than sharing.

When engineers feel safe to openly discuss details regarding incidents and the remediation efforts, knowledge and experience are surfaced, allowing the entire organization to learn something that will help avoid a similar issue in the future.

CHAPTER 4

Understanding Cause and Effect

When problems occur we often assume that all we need to do is reason about the options, select one, and then execute it. This assumes that causality is determinable and therefore that we have a valid means of eliminating options. We believe that if we take a certain action we can predict the resulting effect, or that given an effect we can determine the cause.

However, in IT systems this is not always the case, and as practitioners we must acknowledge that there are in fact systems in which we can determine cause and effect and those in which we cannot.

In Chapter 8 we will see an example that illustrates the negligible value identifying cause provides in contrast to the myriad learnings and action items that surface as the result of moving our focus away from cause and deeper into the phases of the incident lifecycle.

Cynefin

The Cynefin (pronounced kun-EV-in) complexity framework is one way to describe the true nature of a system, as well as appropriate approaches to managing systems. This framework first differentiates between ordered and unordered systems. If knowledge exists from previous experience and can be leveraged, we categorize the system as "ordered." If the problem has not been experienced before, we treat it as an "unordered system."

Cynefin, a Welsh word for habitat, has been popularized within the DevOps community as a vehicle for helping us to analyze behavior and decide how to act or make sense of the nature of complex systems. Broad system categories and examples include:

Ordered

Complicated systems, such as a vehicle. Complicated systems can be broken down and understood given enough time and effort. The system is "knowable."

Unordered

Complex systems, such as traffic on a busy highway. Complex systems are unpredictable, emergent, and only understood in retrospect. The system is "unknowable."

As Figure 4-1 shows, we can then go one step further and break down the categorization of systems into five distinct domains that provide a bit more description and insight into the appropriate behaviors and methods of interaction.

We can conceptualize the domains as follows:

- Simple—known knowns
- Complicated—known unknowns
- Complex—unknown unknowns
- Chaotic—unknowable unknowns
- Disorder—yet to be determined

Ordered systems are highly constrained; their behavior is predictable, and the relationship between cause and effect is either obvious from experience or can be determined by analysis. If the cause is obvious then we have a *simple* system, and if it is not obvious but can be determined through analysis we say it is a *complicated* system, as cause and effect (or determination of the cause) are separated by time.[1]

[1] Greg Brougham, *The Cynefin Mini-Book: An Introduction to Complexity and the Cynefin Framework* (C4Media/InfoQ), 7.

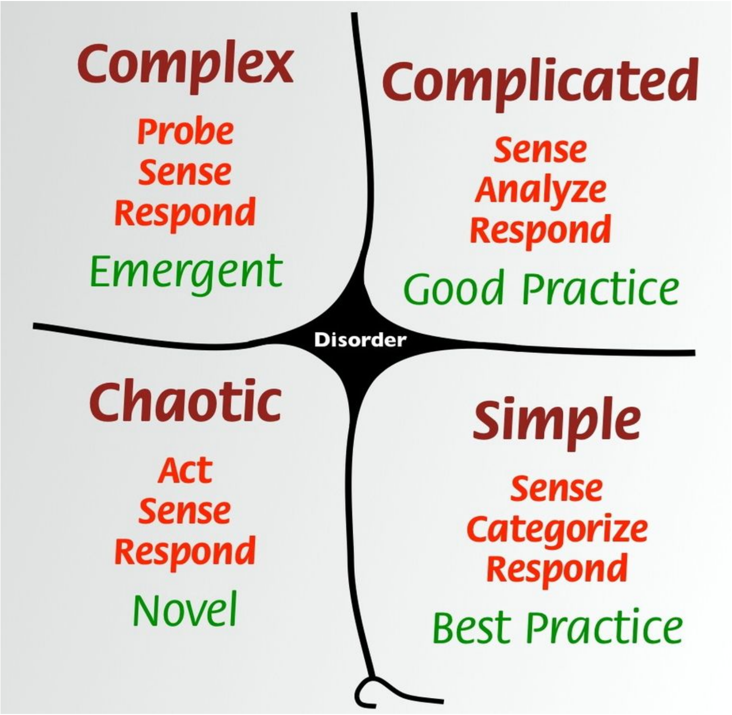


Figure 4-1. Cynefin offers five contexts or "domains" of decisionmaking: simple, complicated, complex, chaotic, and a center of disorder. (Source: Snowden, https://commons.wikimedia.org/w/index.php?curid=53504988)

TIP

IT systems and the work required to operate and support them fall into the complicated, complex, and at times chaotic domains, but rarely into the simple one.

Within the realm of unordered systems, we find that some of these systems are in fact stable and that their constraints and behavior evolve along with system changes. Causality of problems can only be determined in hindsight, and analysis will provide no path toward helping us to predict (and prevent) the state or behavior of the system. This is known as a "complex" system and is a subcategory of unordered systems.

Other aspects of the Cynefin complexity framework help us to see the emergent behavior of complex systems, how known "best practices" apply only to simple systems, and that when dealing with a chaotic domain, your best first step is to "act," then to "sense" and finally "probe" for the correct path out of the chaotic realm.

From Sense-Making to Explanation

Humans, being an inquisitive species, seek explanations for the events that unfold in our lives. It's unnerving not to know what made a system fail. Many will want to begin investing time and energy into implementing countermeasures and enhancements, or perhaps behavioral adjustments will be deemed necessary to avoid the same kind of trouble. Just don't fall victim to your own or others' tendencies toward bias or to seeking out punishment, shaming, retribution, or blaming.

> Cause is something we construct, not find.
>
> —Sidney Dekker, *The Field Guide to Understanding Human Error*

By acknowledging the complexity of systems and their state we accept that it can be difficult, if not impossible, to distinguish between mechanical and human contributions. And if we look hard enough, we will construct a cause for the problem. How we construct that cause depends on the accident model we apply.[2]

Evaluation Models

Choosing a model helps us to determine what to look for as we seek to understand cause and effect and, as part of our systems thinking approach, suggests ways to explain the relationships between the many factors contributing to the problem. Three kinds of model are regularly applied to post-incident reviews:[3]

Sequence of events model

Suggests that one event causes another, which causes another, and so on, much like a set of dominoes where a single event kicks off a series of events leading to the failure.

[2] Sidney Dekker, *The Field Guide to Understanding Human Error*, (CRC Press), 73.

[3] Ibid., 81.

Epidemiological model
: Sees problems in the system as latent. Hardware and software as well as managerial and procedural issues hide throughout.

Systemic model
: Takes the stance that problems within systems come from the "normal" behavior and that the state of the system itself is a "systemic by-product of people and organizations trying to pursue success with imperfect knowledge and under the pressure of other resource constraints (scarcity, competition, time limits)."[4]

The latter is represented by today's post-incident review process and is what most modern IT organizations apply in their efforts to "learn from failure."

In order to understand systems in their normal state, we examine details by constructing a timeline of events. This positions us to ask questions about how things happened, which in turn helps to reduce the likelihood of poor decisions or an individual's explicit actions being presumed to have caused the problems. Asking and understanding how someone came to a decision helps to avoid hindsight bias or a skewed perspective on whether responders are competent to remediate issues.

In modern IT systems, failures will occur. However, when we take a systemic approach to analyzing what took place, we can accept that in reality, the system performed as expected. Nothing truly went wrong, in the sense that what happened was part of the "normal" operation of the system. This method allows us to discard linear cause/effect relationships and remain "closer to the real complexity behind system success and failure."[5]

[4] Ibid.

[5] Ibid., 92.

NOTE

You've likely picked up on a common thread regarding "sequence of events" modeling (a.k.a. root cause analysis) in this book. These exercises and subsequent reports are still prevalent throughout the IT world. In many cases, organizations do submit that there was more than one cause to a system failure and set out to identify the multitude of factors that played a role. Out of habit, teams often falsely identify these as root causes of the incident, correctly pointing out that many things conspired at once to "cause" the problem, but unintentionally sending the misleading signal that one single element was the sole true reason for the failure. Most importantly, by following this path we miss an opportunity to learn.

Within modern IT systems, the accumulation and growth of interconnected components, each with their own ways of interacting with other parts of the system, means that distilling problems down to a single entity is theoretically impossible.

> Post-accident attribution [of the] accident to a 'root cause' is fundamentally wrong. Because overt failure requires multiple faults, there is no isolated "cause" of an accident. There are multiple contributors to accidents. Each of these is [necessarily] insufficient in itself to create an accident. Only jointly are these causes sufficient to create an accident.
>
> —Richard I. Cook, MD, *How Complex Systems Fail*

Instead of obsessing about prediction and prevention of system failures, we can begin to create a future state that helps us to be ready and prepared for the reality of working with "unknown unknowns."

This will begin to make sense in Chapter 8, when we see how improvements to the *detection*, *response*, and *remediation* phases of an incident are examined more deeply during the post-incident review process, further improving our readiness by enabling us to be prepared, informed, and rehearsed to quickly respond to failures as they occur.

To be clear, monitoring and trend analysis are important. Monitoring and logging of various metrics is critical to maintaining visibility into a system's health and operability. Without discovery and addressing of problems early, teams may not have the capacity and data to handle problems at the rate at which they could happen. Balancing efforts of proactive monitoring and readiness for inevitable failure provides the best holistic approach.

The case study of CSG International at the end of Chapter 9 serves as an illustration of this point. They weren't thinking holistically; a post-incident review provided them with a clearer vision of the work actually performed, leading to the opening up of much-needed additional resources.

This is not to say RCAs can't provide value. They can play a role in improving the overall health of the system.

Corrective actions can effectively implement fixes to the system and restore service. However, when we identify something within our systems that has broken and we instinctually act to identify and rectify the cause, we miss out on the real value of analyzing incidents retrospectively. We miss the opportunity to learn when we focus solely on returning our systems to their state prior to failure rather than spending efforts to make them even better than before.

We waste a chance to create adaptable, flexible, or even perhaps self-healing systems. We miss the opportunity to spot ways in which the system can start to evolve with the service we are providing. As demand for our services and resources changes in unexpected ways, we must design our systems to be able to respond in unison.

Think back to the huddled group discussing the events surrounding the botched migration I was involved in. Had we performed an analysis focused solely on identifying cause—who was to blame and what needed to be fixed—we would have left a lot of amazing information on the table.

Instead, we discovered a huge amount about the system that can now be categorized as "known" by a greater number of people in the organization, nudging our sense of certainty about the "normal" behavior and state of our system further in our favor.

CHAPTER 5
Continuous Improvement

Successful organizations understand that continued success depends on a core value, lived daily, of continuous improvement. Entering a new market or defending an existing position requires a constant evaluation of the current state.

Retrospective analysis has become a commonplace exercise in many organizations, particularly as IT orgs have begun to fully adopt Agile software development and DevOps principles. Shortened software development and delivery cycles means reduced feedback loops to understand what went well, what didn't, and how we can do better the next time. In fact, shortening the available time to identify problems, investigate options, and take action is a key reason why post-incident reviews have become so important and prevalent in modern IT organizations.

Continuously improving the development practices, delivery pipelines, and operational techniques provides the best environment for business success. Long cycle times to produce new features and lengthy service disruptions are a common property observed in organizations that have not begun their journey into modern IT incident management and retrospective analysis. Making the idea of continuous improvement a central part of company culture provides extraordinary value across the entire organization.

Creating Flow

Identifying what to improve can often feel like an exercise in who has the most authority. With so many areas of effort and concern in an IT organization, the decision of where to begin may simply come from those in senior positions rather than from the individuals closest to the work. This isn't wholly unjustified: practitioners of the work itself may struggle to see the forest for the trees. Without the presence of generalists or consultants trained in seeing the big picture, identifying the work that brings the most value can be challenging.

Understanding the entire process, from idea to customer usage, requires a close examination of every step in the lifecycle. This can enable inefficiency and friction to be identified and examined further.

Creating a value stream map, a method for visualizing and analyzing the current state of how value moves through the system, is a common practice to assist in understanding this flow. Mapping the entire process and the time between handoffs not only helps all stakeholders have a much larger and clearer picture, but begins to surface areas of possible improvement. Finding ways to trim the time it takes to move through the value stream paves the path toward continuous delivery and highly available systems.

Eliminating Waste

Post-incident reviews work in much the same way. When we are responsible for maintaining the availability and reliability of a service, anything that prevents us from knowing about a problem is friction in the system. Likewise, delays in efforts to remediate and resolve service disruptions are considered waste.

Identifying impediments, such as delays in contacting the right individual during the detection phase of the incident timeline, sets the stage for a clear direction on not only what needs to be improved but where to begin. Regardless of where friction exists in the overall continuous delivery lifecycle, everything that makes it into a production environment and is relied on by others has started in the mind of an individual or collective team. Focusing on where the most value can be gained earlier and often in the entire process shows you your path of continuous improvement.

We will explore flow and waste value streams again later when we break down the components of a post-incident review. These elements will emerge as we plot tasks along the timeline, their impact on service recovery, and how much time elapsed. Exposing and addressing waste in the phases of detection, response, and remediation leads to higher availability.

Feedback Loops

The key to successful continuous improvement is reducing the time it takes to learn. Delaying the time it takes to observe the outcomes of efforts—or worse, not making an effort to receive feedback about the current conditions—means we don't know what's working and what's not. We don't know what direction our systems are moving in. As a result, we lose touch with the current state of our three key elements: people, process, and technology.

Think about how easy it is for assumptions to exist about things as simple as monitoring and logging, on-call rotations, or even the configuration of paging and escalation policies. The case study in Chapter 8 will illustrate a way in which we can stay in touch with those elements and more easily avoid blind spots in the bigger picture.

Post-incident reviews help to eliminate assumptions and increase our confidence, all while lessening the time and effort involved in obtaining feedback and accelerating the improvement efforts.

NOTE

Feedback loops exist in many forms. Monitoring of systems, customer sentiment, and improvements to incident response are only a few forms of feedback loops many companies rely on.

Retrospectives

Retrospective analysis is a common practice in many IT organizations. Agile best practices suggest performing retros after each development sprint to understand in detail what worked, what didn't, and what should be changed. Conditions and priorities shift; teams (and the entire organization) aren't locked into one specific way of working. Likewise, they avoid committing to projects that may turn out to be no longer useful or on the path of providing value to the business.

It's important to note that these retrospectives include the things that worked, as well as what didn't quite hit the mark.

NOTE

A common pitfall of post-incident review efforts is focusing solely on the things that went poorly. There is much to learn from what went well when it comes to identifying and recovering from a problem.

Learning Reviews

In an attempt to place emphasis on the importance of understanding the good and bad in a post-incident analysis, many teams refer to the exercise as simply a *learning review*. With the agenda spelled out in the name, it is clear right from the beginning that the purpose of taking the time to retrospectively analyze information regarding a service disruption is to *learn*. Understanding as much about the system as possible is the best approach to building and operating an IT system.

Regardless of what you call these reviews, it is important to clarify their intention and purpose. A lot of value comes from a well-executed post-incident review. Not only are areas of improvement identified and prioritized internally, but respect from customers and the industry as a whole emerges due to the transparency and altruism of sharing useful information publicly.

NOTE

Other names for learning reviews include:

- Incident Debriefing
- After Action Report
- Rapid Improvement Event

Many call these reviews simply *postmortems*. However, this type of language insinuates an investigation into the "cause of death" (i.e., failure), which is counter to the true purpose of the analysis.

The idea of death sets the wrong tone and allows fear to bias or persuade individuals toward self-preservation and justification rather than encouraging genuine transparency and learning. Thus, in this book I have chosen to discuss a type of learning review known as a *post-incident review*.

We had no label for it in my opening story. All I could have told you at the time was that I wasn't scared to tell everyone what had happened, I was never blamed, and when it was all over and done, we went about our day like normal, with a broader understanding of the system and holding a list of action items to make it even better.

NOTE

What you choose to call the analysis is up to you. Understanding the value the process provides, regardless of naming convention, is the true takeaway from this book. Don't get hung up on the name, so long as you follow the principles laid out.

Objectives

It is important to clarify the objective of the analysis as an exercise in learning. It is common for many to approach these exercises much like troubleshooting failures in simple systems, such as manufacturing and assembly systems, and focus their efforts on identifying the cause.

In nonlinear, simple systems, cause and effect are obvious and the relationships between components of the system are clear. Failures can be linked directly back to a single point of failure. As made clear by the Cynefin complexity framework, this in no way describes the systems we build and operate in IT. Complexity forces us to take a new approach to operating and supporting systems—one that looks more toward what can be learned rather than what broke, and how it can it be prevented from happening ever again.

Depending on the scope and scale of issues that may contribute to IT problems, information can be gained from a variety of areas in not only the IT org, but the entire company. Learning becomes the differentiating factor between a high-performing team and a low-performing team.

Shortening the feedback loops to learn new and important things about the nature of the system and how it behaves under certain circumstances is the greatest method of improving the availability of the service.

To learn what will contribute the most to improvement efforts, we begin by asking two questions—questions designed to get right to the heart of shortening critical feedback loops. Answering the first two questions leads to the third and most important of the group:

1. How do we know sooner?
2. How do we recover sooner?
3. How (specifically) will we improve?

How do we know sooner?

The longer it takes to know about a problem, the lengthier the delay in restoration of services will be. Because of this, one of the main takeaways of any post-incident review should be an answer to the question, “How do we know sooner?”

Here are just a few areas within the detection phase where improvement opportunities for technology, process, and people may exist:

- Monitoring and logging
- On-call rotations
- Humane paging policies
- Documentation and runbooks
- Communication and collaboration methods
- Escalation paths
- Automation

TIP

In Chapter 6, we’ll dive into a fictional story illustrating a service disruption. As you follow along with the recovery efforts and analysis on day two, notice how a list of improvements within the detection phase of the incident is produced that includes several items mentioned in the preceding list. As with most post-incident reviews, action items from that case study included work to shorten the time it takes to know about a problem. The sooner we know, the sooner we can do something about it.

With the always-changing nature of IT systems, it is extremely important to have a finger on the pulse of the health and state of a system. Shortening the time it takes to detect and alert first responders to a problem, be it potential or active, means that teams may in fact be able to reduce the impact of the problem or prevent it from affecting end users altogether.

A seasoned engineering team will always make understanding this question the highest priority.

How do we recover sooner?

Once we are aware of problems, such as missing backup data required for a successful cloud migration, the recovery process begins focusing on elements of response and remediation.

Within each of these efforts lies data that, when analyzed, will point to areas of improvement as well. Understanding how teams form, communicate, and dig into the related observable data always uncovers opportunities for improvement. It's not good enough to only be paged about a problem. Detecting and alerting is an important first step of restoring service, but a very small piece of the larger availability picture.

A majority of remediation efforts are spent investigating and identifying the problem. Once the data has been explored, allowing theories and hypotheses to be built, efforts shift to actually repairing and restoring the services.

TIP

> Take note in the next chapter's case study of how the time required to restore services is extended due to poor availability of documentation. Efforts to restore service often can't begin until information on access and triaging procedures has been obtained. Not every problem has been seen before, either. Documentation may not exist. If so, how can that be confirmed early on to prevent engineers from wasting time searching for something that does not exist?

Again, this phase of recovery typically holds many areas of potential improvement if we just take the right approach to discovering them. Identifying weaknesses, such as lack of monitoring or not having complete and accurate information and access to parts of the system, should be a priority.

Multiple areas of the incident lifecycle will prove to be excellent places to focus continuous improvement efforts. The return on investment for IT orgs is the ability to recover from outages much sooner. The 2017 State of DevOps Report reported that high-performing organizations had a mean time to recover (MTTR) from downtime 96 times faster than that of low performers.[1] Scrutinizing the details related to recovery helps to expose methods that can help achieve that improvement.

How (specifically) will we improve?

By focusing on the two previous questions, several areas of improvement should emerge within the detection, response, and remediation phases of the incident lifecycle. The next thing to do is to establish how we are going to go about making these improvements. What actionable takeaways have come from our discussions surrounding those questions?

Action is required in order to fully begin the journey of learning and improvement. Thus, the post-incident reviews mustn't be reduced to something that more closely resembles a debate or blame game and has little to no focus on actively prioritizing improvements to the system. Nor is this simply an exercise in "finding the cause and fixing it."

The aim is to identify actionable improvements: enhancements or countermeasures to minimize or offset the impact of a future similar occurrence. Regardless of the approach, the goal is to make improvements to the system that will increase its availability and reliability.

NOTE Traditional approaches to post-incident analysis are often unpleasant exercises. It's common to see an opportunity to learn more about the system wasted because excessive time was spent debating the cause and who should be "held accountable." In so doing, not only do we narrow our opportunity for learning, but we create an environment in which it's easy to point the finger of blame toward one another, or even ourselves.

[1] Puppet + DORA, "2017 State of DevOps Report," 25.

The systems thinking approach to IT-related service disruptions and the retrospective analysis is best understood with an example. We turn to that next.

CHAPTER 6

Outage: A Case Study Examining the Unique Phases of an Incident

Day One

Detection

Around 4 p.m. Gary, a member of the support team for a growing company, begins receiving notifications from Twitter that the company is being mentioned more than usual. After wrapping up responding to a few support cases, Gary logs into Twitter and sees that several users are complaining that they are not able to access the service's login page.

Gary then reaches out to Cathy, who happens to be the first engineer he sees online and logged into the company chat tool. She says she'll take a look and reach out to others on the team if she can't figure out what's going on and fix it. Gary then files a ticket in the customer support system for follow-up and reporting.

Note that Gary was the first to know of the problem internally, but that external users or customers first detected the disruption and a sense of urgency did not set in until Twitter notifications became alarmingly frequent. Still, responding to support cases was of higher priority to Gary at that moment, extending the elapsed time of the detection phase of the incident.

How many ideas for improvement can you spot in this scenario?

Response

Cathy attempts to verify the complaint by accessing the login page herself. Sure enough, it's throwing an error. She then proceeds to figure out which systems are affected and how to get access to them. After several minutes of searching her inbox she locates a Google Document explaining methods to connect to the server hosting the site, and is then able to make progress.

As Cathy attempts to investigate what's happening, she is met with delay while trying to access the system. The documentation appears to be correct, but isn't easily accessible. Too much time is wasted searching for critical information. How effective is documentation if it is difficult to find or out of date?

What small improvements can be made to shorten the time it takes to investigate, identify, and triage the problem? The response phase offers many opportunities to learn and improve.

Remediation

Upon logging in to the server, Cathy's first action is to view all running processes on the host. From her terminal, she types:

```
cathy#: top
```

to display the running processes and how many resources are being used. Right away she spots that there is a service running she isn't familiar with and it's taking 92% of the CPU. Unfamiliar with this process, she's hesitant to terminate it.

Cathy decides to reach out to Greg, who may be able to shed light on what that process is and what the next best steps are. He deals with this stuff a lot more often. She pulls up her chat only to see Greg isn't available on Slack; he must be out of the office. Cathy then starts looking through her phone for a number to call. After a few moments searching her contacts, she realizes she doesn't have his mobile number. So she pulls up her email and searches for Greg's name. At last, she finds his number in a shared Google Doc someone had sent out nearly a year ago.

Greg joins the investigation process and asks Cathy to update the status page. Not sure how to accomplish this, she nevertheless replies "Will do," not wanting to burden Greg with explaining how this is done while he digs into higher-priority issues. Cathy is able to get in touch with Gary from support, and he walks her through updating the status page. Gary mentions that he's now received 10 support requests and seen a few additional tweets regarding the site being down.

Cathy reaches out to Greg now that he's signed on to chat to let him know the status page has been updated and to ask if there's anything else she can do. Greg responds that he's figured out the problem and everything seems to be clearing up now. Cathy confirms from her own web browser that she is able to get to the login page now, and Gary chimes in to say he'll take care of updating the status page again.

When Cathy asks Greg in chat what he found and what he did, he says that he found an unknown service running on the host and he killed it. Following that, he gracefully stopped and started the web services and archived a few logs so he can take a closer look at them later. He and his wife are celebrating their daughter's birthday, and guests are beginning to arrive. Once service was restored, he needed to get back to what he was doing. Cathy offers to send out a calendar invite for them to discuss what went wrong and report the findings to management.

In total the service was offline for approximately 20 minutes, but Gary informs the others that he has now received 50 support requests and people are still talking about the company on Twitter.

What would you have done had you discovered an unknown service running on the host? Would you kill it immediately or hesitate like Cathy?

Thankfully Greg was able to help restore service, but at what expense? A more severe problem might have forced Greg to miss his daughter's birthday party entirely. How humane are your on-call and response expectations?

How mindful were the participants of the external users or customers throughout this phase? Often, customers are constantly refreshing status pages and Twitter feeds for an update. Were transparent and frequent updates to the end users made a priority?

This is a great opportunity for a discussion around the question "What does it look like when this goes well?" as first suggested in Chapter 1.

Day Two

Analysis

Cathy, Greg, Gary, and several additional members from the engineering and support teams huddle around a conference table at 10 a.m., with a number of managers hovering near the door on their way to the next meeting.

Greg begins by asking Cathy to describe what happened. Stepping the group through exactly what transpired from her own perspective, Cathy mentions how she was first alerted to the problem by Gary in support. She then goes on to explain how it took a while for her to figure out how to access the right server. Her first step after accessing the system was to check the running processes. Upon doing so she discovered an unknown service running, but was afraid to kill it as a remediation step. She wanted a second opinion, but explains that again it took her some time to track down the phone number she needed to get in touch with Greg.

Several engineers chime in with their opinions on what the service was and whether it was safe to stop or not. Greg then adds that those were his exact first steps as well and that he didn't hesitate to kill a process he wasn't familiar with. Cathy asks, "Did you run *top* to see which process to kill?" Greg responds, "I like *htop* a little better. It's easier to read." "Hm. I haven't heard of that tool, I'll install it for the future," Cathy says.

Gary adds that he took over updating customer stakeholders through the status page and support Twitter account, allowing Greg and Cathy to focus solely on restoring service. He says he also reached out to Debby, the social media manager, to ask her to help share the updates from their support account in hopes of easing the escalating chatter on Twitter.

> Notice how Cathy describes the timeline in a very "matter of fact" manner, rather than attempting to defend any part of it. Along the way, she points out friction in the process. Hard-to-find documentation and contact information slowed down the recovery of service.
>
> Cathy was transparent about what she saw and did and, maybe most importantly, felt. Not killing the unknown service was a judgment call. It's important to understand how she came to that decision.

While the timeline is being thoroughly discussed, a list is generated on a shared Google Doc displayed in the flat-screen TV at one end of the conference room. It includes summaries of communication interactions as well as anything the participants have learned about the system. The list of findings or learnings looks like this:

Learnings

1. We didn't detect this on our own. Customers detected the outage.
2. We don't have a clear path to responding to incidents. Support contacted Cathy as a result of chance, not process.
3. It's not common knowledge how to connect to critical systems regarding the service we provide.
4. Access to systems for the first responder was clumsy and confusing.
5. We aren't sure who is responsible for updating stakeholders and/or the status page.

6. A yet-to-be-identified process was found running on a critical server.
7. Pulling in other team members was difficult without instant access to their contact information.
8. We don't have a dedicated area for the conversations that are related to the remediation efforts. Some conversations were held over the phone and some took place in Slack.
9. Someone other than Greg should have been next on the escalation path so he could enjoy time with his family.

Armed with an extensive list of things learned about the system, the team then begins to discuss actionable tasks and next steps.

Several suggestions are made and captured in the Google Doc:

Action Items

1. Add additional monitoring of the host to detect potential or imminent problems.
2. Set up an on-call rotation so everyone knows who to contact if something like this happens again.
3. Build and make widely available documentation on how to get access to systems to begin investigating.
4. Ensure that all responders have the necessary access and privileges make an impact during remediation.
5. Establish responsibility and process surrounding who is to maintain the status page.
6. Define escalation policies and alerting methods for engineers.
7. Build and make widely available contact information for engineers who may be called in to assist during remediation efforts.
8. Establish a specific communication client and channel for all conversations related to remediation efforts and try to be explicit and verbose about what you are seeing and doing. Attempt to "think out loud."
9. Come up with a way for engineers to communicate to their team availability to assist in remediation efforts.

As each action item is documented, someone in attendance takes ownership of it and offers to file a ticket so that time to work on it can be prioritized.

Bill, a member of the product team, admits that these new action items are a high priority and suggests they be scheduled immediately before something like this happens again. “We have a few SLAs in place and I’m worried we are already causing concern for some important customers,” he says. Then he adds, “Wait! No one ever said anything about the root cause of this. Was it that unknown service that was running?”

Greg responds with, “Yes and no. That service shouldn’t have been pegged like that, but there’s definitely more to the story. I Googled it last night after my daughter’s party and it has something to do with a backend caching service. I’m manually checking it every couple of minutes for now and I’m going to do some more research on it. The caching service touches a lot of things, including many services I’m not that familiar with, and until I can get a better understanding of how all of those services are interacting with each other, we need to be able to spot a similar problem quicker. Besides, we have scheduled work to replace that caching service pretty soon.” Bill mentions he thinks that work should be prioritized if it’s going to help. “That would be awesome,” Greg says. “I’ll file a ticket and prioritize this first while the others tackle the other tasks.”

Cathy offers to summarize the timeline of events, findings, and action items from the meeting and share the Google Doc with the entire company. She’s already had several people come by her desk asking about what happened last night, including some from the C-level. “I’ll also create something we can post to our status page to let our customers know what happened and what we are doing to make our service more reliable.” “Good call,” Gary says. “I can help you with that if you want.”

The group decide that further investigation is needed to understand how the caching service played a role in the problem. Until then, attention will be focused on replacing that service and implementing countermeasures and enhancements that make the system as a whole much more available.

Recap

This example illustrated a fairly brief Sev2 incident in which the time to resolve was relatively low. Lengthier outages may result in exercises that go on longer. The total time of this exercise was just under 30 minutes, and it resulted in 9 extremely beneficial tasks that will make a huge impact on the uptime of the site and the overall reliability of the service the business is providing.

CHAPTER 7

The Approach: Facilitating Improvements

There are two main philosophical approaches to both what the analysis is and the value it sets out to provide organizations. For many, its purpose is to document in great detail what took place during the response to an IT problem (self-diagnosis). For others, it is a means to understand the cause of a problem so that fixes can be applied to various aspects of process, technology, and people (self-improvement).

Regardless of the approach you take, the reason we perform these exercises is to learn as much about our systems as possible and uncover areas of improvement in a variety of places. Identifying a "root cause" is a common reason most claim as to why analysis is important and helpful. However, this approach is shortsighted.

NOTE

The primary purpose of a post-incident review is to learn.

Discovering Areas of Improvement

Problems in IT systems arise in many different forms. In my opening example in the Introduction, our system did not experience an outage, but rather an unexpected (and bad) outcome as a result of normal operation. The backup process was not performing as it was expected to, but the system as a whole did not suffer a disruption of service. All signs pointed to a healthy working system. However, as we learned from the remediation efforts and post-incident review, there were latent problems in the migration process that caused a customer to lose data. Despite not directly disturbing the availability of our system, it certainly impacted an aspect of reliability. The system wasn't reliable if it couldn't perform the tasks related to its advertised functionality, namely seamless migration of applications between cloud providers.

Another situation might be the failure of a database in a staging environment failed. Despite not harming the production environment or real customers, whatever problem exists in staging will likely make its way to production if not caught and addressed. Discovering and analyzing such "non-production" disruptions is important as well. Often an incident in a development or staging environment is a precursor to that same condition occurring in the production environment.

TIP

Many organizations build infrastructure and test continuously. Disruptions in a non-production environment will have an impact on cycle time and ultimately the ability to deliver value.

Incidents show themselves in many ways. Learning as much as possible from them dramatically increases the reliability of a service. Analyzing many aspects of the data that is available before and during an incident is the path to that learning.

Think about the lists of learnings and action items from the case study in Chapter 6. We discovered nine interesting findings. Those converted perfectly into nine specific action items, each of which was prioritized and assigned an owner. Opportunities to learn and improve are there if we just know where to look.

Facilitating Improvements in Development and Operational Processes

In many cases, teams focus too much on the one thing that broke and what needs to be done to fix it. All too often we don't take the time to examine existing processes tied to the development and operations of software and infrastructure that make up the service.

In the case study in Chapter 6, the review process allowed Cathy to inform a larger audience, including key decision makers, how work was performed. It allowed everyone to informally examine the "way we do things" and spot inefficiencies that had emerged along the lifespan of the system as a whole. Efforts that made sense previously may no longer be optimal today.

Because post-incident analysis includes discussions around process, it helps subject matter experts and management form a much clearer picture of the system as a whole, including how components go from ideas to functional features that impact the bottom line. These analyses allow senior developers to spot improvement opportunities in the delivery pipeline. They allow management to see where breakdowns in communication and collaboration are delaying restoration of service. The entire organization begins to see the big picture and where their individual efforts play a role in delivering and operating IT services.

Continuous improvement means that we are constantly evaluating many aspects of IT, and a well-executed post-incident review is a great way to easily identify new and better ways to improve areas of development, operations, security, and more.

Identifying Trade-offs and Shortcomings in IT

One of the best things to come out of a post-incident review is that teams begin to see where current trade-offs exist, as well as identifying inherent problems or technical debt that has crept into the system.

The post-incident review in the case study in Chapter 6 exposed problems with detection methods, escalation policies, communication channels, access to systems, available documentation, and much more. Often a result of technical debt, these problems are easily addressed yet rarely prioritized.

These shortcomings are important to identify and address early on so that a clear picture of the system exists for everyone, as well as pointing out obvious areas of improvement. A good example of this is the difference between software or operations work as it is designed and how it is actually being performed.

Work as designed versus work as performed

It's easy to set in motion process improvements or implement a new tool to attempt to provide more reliable and available IT services, but all too often the work that is actually performed looks quite a bit different than how it was scoped out and established previously.

Etsy's "Debriefing Facilitation Guide" suggests approaching exercises like the post-incident review with a true "beginner's mind," enabling all involved to frame the exercise as an opportunity to learn.

To better understand work as designed versus work as performed, the guide recommends asking the following questions:

- How much of this is new information for people in the room?
- How many of you in the room were aware of all the moving pieces here?[1]

Workarounds, changes to teams or tools, and general laziness can all play a part in creating a gap between work as imagined and agreed upon and what actually happens. When we discuss what happened and exactly how during a post-incident review, these differences are surfaced so that the teams can evaluate discrepancies and how they can or should be corrected.

Efficiency versus thoroughness

In many cases, drift in work as performed compared to how it was drawn up initially comes down to a constant struggle to manage the trade-off between being highly efficient or extremely thorough. This is very prevalent when it comes to remediation efforts. When a service disruption occurs, first responders are working as quickly as possible to triage and understand what's going on. Engineers with various areas of expertise work in parallel to investigate what's going on, forming theories on how to remediate the problem and taking action to restore services efficiently.

This aim for speed directly impacts the ability of responders to be thorough in their investigation and actions to recover. We are forced to find a balance between being extremely efficient or extremely thorough; we can't be on the extreme side of both at the same time.

[1] Etsy, "Debriefing Facilitation Guide", 26.

The efficiency–thoroughness trade-off,[2] and what that balance looks like, becomes much clearer during post-incident review. By asking questions that inquire objectively about how work is accomplished and the decision-making process involved, we can point out patterns of extreme swings in either direction between efficiency and thoroughness.

Hastily performing actions during remediation efforts may help to restore service quickly, but it's common for instinctual actions taken by responders to have little or no impact—or worse, cause more harm than good.

Recognizing the trade-off will go a long way in improving the behavior of first responders and teams that are pulled in to assist during remediation efforts.

We've laid the groundwork regarding what makes up a post-incident review, but there are still a number of questions to consider when planning to conduct the exercise:

- What warrants the exercise?
- When should we perform it?
- Who should be there?
- How long will it take?
- What documents should come from the exercise?
- Who should have access to the artifacts created?

The following chapters will address these questions.

[2] Erik Hollnagel, *The ETTO Principle: Efficiency-Thoroughness Trade-Off* (CRC Press)

CHAPTER 8

Defining an Incident and Its Lifecycle

How do we know what an incident is or when it is appropriate to perform a post-incident review?

An incident is any unplanned event or condition that places the system in a negative or undesired state. The most extreme example of this is a complete outage or disruption in service. Whether it is an ecommerce website, a banking app, or a subcomponent of a larger system, if something has happened causing the operability, availability, or reliability to decrease, this is considered an incident.

In short, an incident can be defined as an unplanned interruption in service or a reduction in the quality of a service.

Severity and Priority

A standard classification of incidents helps teams and organizations share the same language and expectations regarding incident response. Priority levels as well as an estimation of severity help teams understand what they are dealing with and the appropriate next steps.

Priority

To categorize types of incidents, their impact on the system or service, and how they should be actively addressed, a priority level is assigned. One common categorization of those levels is:

- Information
- Warning
- Critical

Severe incidents are assigned the critical priority, while minor problems or failures that have well-established redundancy are typically identified with a warning priority. Incidents that are unactionable or false alarms have the lowest priority and should be identified simply as information to be included in the post-incident review.

NOTE

First responders should never be alerted to unactionable conditions. For the health and sanity of our teams, priority and severity levels help everyone set expectations.

Severity

Categorization of severity levels often varies depending on industry and company culture. One example of labeling severity levels is as follows:

- Sev1 (Critical)—Complete outage
- Sev2 (Critical)—Major functionality broken and revenue affected
- Sev3 (Warning)—Minor problem
- Sev4 (Warning)—Redundant component failure
- Sev5 (Info)—False alarm or unactionable alert

NOTE

Noise and unactionable alerts account for approximately half of the "top-reported problems" of life on-call.[1]

[1] VictorOps, "State of On-Call Report 2016-2017," 6.

Lifecycle of an Incident

One way to assess the health and maturity of an organization with respect to incident management, post-incident analysis, and the phases outlined in this section is to dissect the flow and properties of incidents. One method to do this is J. Paul Reed and Kevina Finn-Braun's Extended Dreyfus Model for Incident Lifecycles, which describes the typical language and behaviors used by various people and teams in an organization, from a novice practitioner to an advanced one. (It should be noted that organizations and teams can fall into different categories for each of the stages of the incident lifecycle.)

Five phases make up the entire lifecycle of an incident (see Figure 8-1). Each phase may vary in degree of maturity with respect to the tools, processes, and human elements in place. Some teams may be far advanced in detection but weak and therefore beginners in response and remediation. Typically, these teams have established well-defined processes to detect failures but need to focus improvement efforts on responding to and recovering from problems.

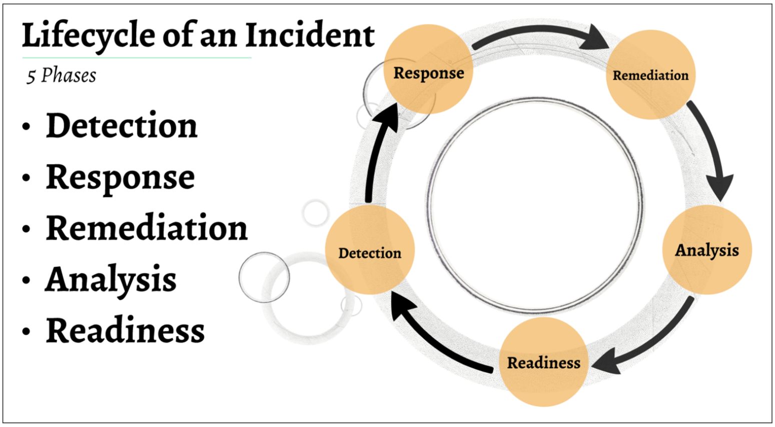


Figure 8-1. The five phases of an incident

Detection

Knowing about a problem is the initial step of the incident lifecycle. When it comes to maintaining service availability, being aware of problems quickly is essential. Monitoring and anomaly detection tools are the means by which service owners or IT professionals keep track of a system's health and availability. Regularly adjusting monitoring thresholds and objectives ensures that the "time to know" phase of an incident is continuously improved upon, decreasing the overall impact of a problem. A common approach to monitoring is to examine conditions and preferred states or values. When thresholds are exceeded, email alerts are triggered. However, a detection and alerting process that requires an individual to read and interpret an email to determine if action needs to be taken is not only difficult to manage, it's impossible to scale. Humans should be notified effectively, but only when they need to take action.

Response

Once a problem is known, the next critical step of an incident's lifecycle is the response phase. This phase typically accounts for nearly three-quarters of the entire lifecycle.[2] Establishing the severity and priority of the problem, followed by investigation and identification, helps teams formulate the appropriate response and remediation steps. Consistent, well-defined response tactics go a long way toward reducing the impact of a service disruption.

Responding to problems of varying degree is not random or rare work. It is something that is done all the time. It is this responsiveness that is the source of reliability.

TIP

Examine the "response" rather than focusing solely on the outcome (i.e., cause or corrective action).

In organizations that implement post-incident reviews, tribal knowledge, or critical information that often only resides in the heads of a few, is spread beyond subject matter experts. First responders have relevant context to the problem as well as documentation and access to systems to begin investigation. As new information is identified and shared with the greater group to create a shared contextual awareness of the situation, roles such as Incident Commanders often prove to be helpful in establishing effective and clear communication (especially between large teams or in the case of cascading failures). As organizations mature in their responses, teams begin to

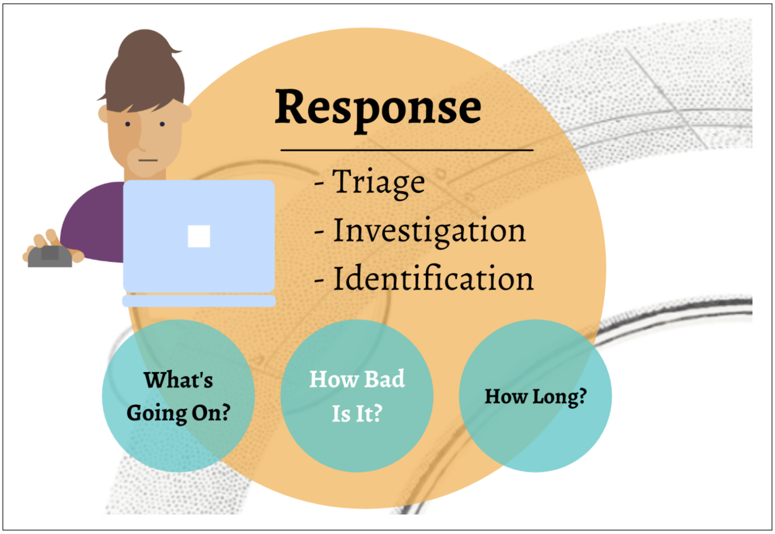
[2] VictorOps, "State of On-Call Report 2015," 13.

track metrics regarding the elapsed time of the incident. In an effort to improve the average duration it takes to recover from a service disruption (i.e., mean time to recover), teams find improved ways to collaborate and understand how best to respond to a problem.

If there's a phase in which to identify areas of improvement, it's this one, in terms of how teams form and begin to manage the incident. A well-crafted post-incident review will uncover the improvements that can be made in team formation and response. Focusing less on the cause of problems and more on how well teams formed, areas of improvement are continuously identified, and the focus on improving response to incidents indicates a solid understanding of "systems thinking." A proactive response plan provides the most effective method of decreasing downtime.

The response phase can be unpacked even further into the following stages (Figure 8-2):

- Triage
- Investigation
- Identification



Response

- Triage
- Investigation
- Identification

What's Going On?

How Bad Is It?

How Long?

Figure 8-2. Unpacking the response

Remediation

We don't know what we don't know. If detection and response efforts are chaotic and undefined, the resulting remediation phase will be just as unsuccessful. In many situations, remediation may start with filing a ticket as part of established procedures. Without a sense of urgency, recovering from a service disruption as quickly as possible is not made a priority. Knowledge is not shared, improvements to the system as a whole are not made, and teams find themselves in a break/fix cycle. This chaotic firefighting is no way to approach remediation.

> **NOTE**
>
> In the following chapter, we'll read about how CSG International approached their own problem with a lack of knowledge sharing. They were able to improve on this shortcoming as well as identifying bad practices that they currently had in place. Along with increasing the sharing of knowledge, these improvements helped to shape their company culture.

Through post-incident reviews, teams look to understand more regarding contributing factors of incidents during remediation. A small change or "fix" in one area may have larger implications elsewhere. Reliance on ticketing systems to actively "manage" incidents begins to fade as teams begin to realize that not only do incidents need to be managed in real time, but a deeper discussion is required to fully address a problem. A stronger sense of ownership for the availability of a service leads to a proper sense of urgency and a system of prioritization, allowing teams to remediate effectively and continuously improve their methods. A belief that strength is greatest in a collective team and a desire for open sharing of information and knowledge are part of the evolving culture.

Focusing on response efforts feeds the evolution as more aspects of the incident lifecycle are improved. Detection and response methods evolve, leading to cognitive load and response times improving. Methods of automating much of the remediation phase become central to improvements, focusing more on how humans can be involved only when required. Service resiliency and "uptime" become byproducts of continuously improving aspects of the detection, response, and remediation phases, rather than attempting to protect systems.

Analysis

The analysis phase of an incident is often missing entirely in organizations with low maturity in preceding phases of the lifecycle. Frequently considered a low-priority exercise, "day job" responsibilities commonly prevent analysis from taking place. Additionally, a fear of reprimand for involvement in an outage prevents many from sharing critical details that are relevant to not only the problem itself, but the response. Teams feel that their jobs are on the line when incidents happen as it is common for blame to be assigned to individuals, incentivizing them to keep useful and sometimes critical information to themselves. As teams mature, a tendency to ask more about "how" something happened than "why" helps avoid such negative incentive structures.

Root cause analysis becomes less common once an understanding of complex and simple systems has been realized. High availability of the complex requires a different approach.

Clarification that learning from an incident is the top priority helps to establish expectations for any analysis. Gaining a diverse and multifaceted understanding of the incident as it cycles through the phases helps to uncover new and interesting ways to answer the questions "How do we know about the problem faster?" and "How do we recover from the problem faster?"

Embracing human error is part of a healthy and mature understanding of complex IT environments. Successful analysis requires accurate and complete data to thoroughly discuss and scrutinize. This allows for a better understanding of incident management throughout the lifecycle. A clear path and a focus on continuous improvement of managing the entire incident lifecycle helps build resilient systems.

Readiness

Organizations have gone to great lengths to try to predict and prevent disruptions to service. In simple systems where causation and correlation are obvious, outages may be reduced by looking for known patterns that have previously led to failure. However, in complex systems such obvious relationships and patterns are only understood in hindsight. As a result, mature organizations understand that prediction and prevention of service disruptions is practically impossible. This is not to say that efforts should not be made to identify patterns and act accordingly. However, relying heavily on spotting and avoiding problems before they happen is not a viable long-term or scalable solution.

Instead, teams begin to take more of a “readiness” approach rather than concentrating on “prevention.” Efforts become focused more on reviewing documentation, response processes, and metrics in a day-to-day context.

Signs that an organizational culture of continuous improvement is gaining traction include updating out-of-date documentation or resources as they are found. Providing the team and individuals with the space and resources to improve methods of collaborating over important data and conversations brings the greatest value in terms of readiness and, as a result, leads to highly resilient systems with greater availability.

As teams and organizations mature, crew formation and dissolution are considered the top priority in responding to, addressing, and resolving incidents. Frequent rehearsal of team formation means teams are well practiced when problems inevitably emerge. Intentionally creating service disruptions in a safe environment assists in building muscle memory around how to respond to problems. This provides great metrics to review and analyze for further improvements to the various phases of an incident’s lifecycle. Creating failover systems, tuning caching services, rewriting small parts of code…all are informed by analysis and set the team up in bigger ways. Service disruptions cannot be prevented, but the only logical path toward providing highly available systems is for teams to collaborate and continuously learn and improve the way they respond to incidents.

NOTE

We use the idea of “readiness” rather than “prevention” for this final phase because our focus is on learning and improvement rather than prediction and prevention. This is a distinction from the Dreyfus model mentioned earlier.

These five distinct phases make up the incident lifecycle in its entirety. Post-incident reviews are reflected specifically in the “analysis” phase of an incident, but the learnings that come from them are fed back in to improve each phase of the lifecycle. This is why we value learning over identifying the root cause of an incident: by focusing solely on determining the cause of problems large and small, we miss the opportunity to learn more about possible improvements across all phases of an incident and the system as a whole.

It is within these phases and subphases that we'll focus our efforts on what to learn and how to improve. By establishing the goals of what can we learn to help us "know sooner" and "recover sooner," we will uncover action items to shorten the impact of a service disruption.

In the following chapter, I'll show you how to conduct your own post-incident review.

CHAPTER 9
Conducting a Post-Incident Review

Now that we've established the "what" and "why" of a post-incident review, we can begin to take a closer look at the "how."

There are various approaches to conducting a successful analysis. Some follow strict formats, while others tend to be much more informal. This book attempts to provide some structure or guidance for conducting your next analysis. If you are worried about it being too restrictive or not providing what management is asking for, don't panic. If management needs additional information, it is reasonable to adjust the suggested guidelines provided in Chapter 10 to meet your needs.

My advice when asked to stray too much from the guide is to keep in mind the basic principles of Chapter 5, where we discussed that the point of this exercise is to learn so that we may improve our methods of knowing about a problem sooner (*detection*) as well as how we can recover (*response* and *remediation*) sooner. Cause, severity, impact, and additional topics of interest may be included in your own analysis, but don't let them distract you from a learning opportunity.

A well executed post-incident review has a clearly stated purpose and repeatable framework. Let's take a moment to address some of the key components.

Who

Having many diverse perspectives on what took place during response and remediation efforts helps to bring high-value improvements to the surface. Rather than focusing simply on identifying what went wrong and targeting that as what should be fixed, we now understand that there are many contributing factors to an incident, and avoiding opportunities to discuss and improve them all but guarantees a scenario where engineers are in a constant break/fix situation, chaotically reacting to service disruptions. Focusing the perspectives and efforts toward making slight but constant improvements to the entire incident lifecycle provides the greatest gains.

Essential participants in the post-incident review include all of the people involved in decisions that may have contributed to the problem or recovery efforts:

- (Primary) First responders
- (Secondary) Escalation responders (additional subject matter experts who joined in)
- Incident Commander (where necessary)
- Management and stakeholders in other areas

We want to involve all the people who identified, diagnosed, or were affected by the problem. Others who are curious about the discussion and process or have something to contribute should be included as well. Anyone who has ideas or something to share that may help us explore value-adding improvements should be invited to join in.

NOTE

The role of Incident Commander can be performed by engineers or leadership, depending on the circumstances, structure, and culture of the organization. Differences exist in the enterprise versus smaller companies. Leadership may assume or be assigned the role, but that is not always the case.

The Facilitator

In many cases, leveraging an objective third-party facilitator can provide several key benefits. Including someone who wasn't directly involved in the incident removes opportunities for human bias to taint the process of learning. When possible, get someone who wasn't involved to facilitate.

There are predictable flaws that arise when bias begins to creep into conversations regarding remediation efforts. One example of this is *hindsight bias*, or the inclination, after an incident, to see it as having been predictable—despite there having been little or no objective basis for predicting the incident, you truly believe you could have foreseen it somehow. Failing to spot bias allows us to go down a path of discussion and reasoning that provides no real value. It's very easy to begin using counterfactual statements such as "I should have known X" or "If I had just done Y." The result of this is simply discussing alternative versions of events that have already occurred.

Another common bias that manifests during the post-incident review is *confirmation bias*, or the tendency to interpret new evidence as supporting or confirming what we believe has occurred in the system. Theories should be formed, but thoroughly tested to reduce confirmation bias. We should rely only on hypotheses that have been verified.

All members participating in a exercise should be vigilant to spot bias as it occurs in discussions, and encouraged to speak up.

What

Now that we have gathered all parties to the exercise, the tone and mission should be established. First and foremost, we are here for one reason only: to *learn*.

Yes, understanding the causes of problems in our systems is important. However, focusing solely on the cause of a problem misses a large opportunity to explore ways in which systems can be designed to be more adaptable. More important than identifying what *may* have caused a problem is learning as much as possible about our systems and how they behave under certain conditions. In addition to that, we want to scrutinize the way in which teams form during incident response. Identifying and analyzing data points regarding these areas won't necessarily bring you to a root cause of the problem, but it will make your system much more "known"—and a greater and more in-depth understanding of the system as a whole is far more valuable to the business than identifying the root cause of any one problem.

In addition to establishing a space to learn, post-incident reviews should be considered an environment in which all information should be made available. No engineers should be held responsible for their role in any phase of the incident lifecycle. In fact, we should reward those who surface relevant information and flaws within our systems, incentivizing our engineers to provide as much detail as possible. When engineers make mistakes but feel safe to give exhaustive details about what took place, they prove to be an invaluable treasure chest of knowledge. Collectively, our engineers know quite a lot about many aspects of the system. When they feel safe to share that information, a deeper understanding of the system as a whole is transferred to the entire team or organization. Blaming, shaming, or demoting anyone involved in an incident is the surest way for that deeper understanding to *not* take place. Encourage engineers to become experts in areas where they have made mistakes previously. Educating the rest of the team or organization on how not to make similar mistakes in the future is great for team culture as well as knowledge transfer.

Leadership Disappearing Act

One important behavior to watch out for is when leadership "disappears," expecting teams to do all of the analysis and improvement coordination themselves.

It's important that leadership attend and demonstrate that they value the learning process, as well as what they themselves have learned. CxOs and managers can then communicate with others, including broader leadership within the organization, further sharing what was learned to cross-pollinate with the various teams.

Transparency is an opportunity in many organizations, especially enterprises. Be transparent and share the findings as broadly as possible.

When

Analyses conducted too long after a service disruption are of little value to the overall mission of a post-incident review. Details of what took place, memories of actions taken, and critical elements of the conversations will be lost forever. Performing the review exercise as quickly as possible ensures that the maximum amount of relevant information and context about the incident can be accurately captured. The longer we wait to discuss the events, the less we'll be able to recall, and the likelihood of establishing useful action items is diminished. Try to conduct the review exercise the following business day, or at your first opportunity. Scheduling the analysis and inviting key stakeholders as soon as recovery efforts have concluded helps get the appointment on the calendar before something else comes up.

Performing a post-incident review on all significant incidents will uncover areas of improvement. However, performing one for each incident that comes up may prove to be too frequent. Inquiries to learn should, at the very least, be conducted for all Sev1 and Sev2 incidents.

TIP

Set aside post-incident review exercises as something separate from "regular work" and establish it as a safe space for inquiry and learning to occur. Make it a ritual.

Where

Team members don't necessarily work in the same office, or even the same time zones. When possible, physical meetings should be taken advantage of, but these are not essential. In fact, virtual conference calls and meetings can allow more diverse perspectives to be remotely pulled into the discussion. This in turn can help avoid groupthink pitfalls and provide more opportunities for genuine analysis of not only what went wrong, but how well teams responded.

How

Once we have gathered the people involved and established our intent, it's time to begin discussing what took place.

> Words are how we think; stories are how we link.
>
> —Christina Baldwin

Establish a Timeline

The facilitator may begin by first asking, “When did we know about this problem?” This helps us to begin constructing a timeline. Establishing when we first knew about the problem provides a starting point. From there, we can begin describing what we know.

NOTE

A mindful communication approach—*describing* the events of the timeline rather than *explaining* them—helps to ward off cognitive bias and natural tendencies toward blame.

The timeline should include as many perspectives as possible. The more data points collected, the clearer the overall picture of exactly what took place during the entire incident. Collect information on what the engineers actually did, and how. What contributed to response and remediation efforts, and what affected the time to recover?

NOTE

In Chapter 10, I’ve provided a guide to help document and plot the data you collect during your post-incident review. This may be helpful when you try to understand better what engineers were doing, how that work was accomplished, and whether it helped or hurt your recovery efforts.

As engineers continue to describe their experiences, the facilitator may want to probe deeper by asking questions like “How do we currently perform X?” An open-ended inquiry such as this may spark a deeper discussion around alternate, better methods, or at the very least point out action items to tackle inefficiencies in remediation efforts.

While discussing what took place, naturally we will point out things that did not work out so well. Obvious opportunities for improvement will present themselves. After an incident, those are typically the things we tend to focus on. We often overlook the things that went well. However, these too should be collected, analyzed, and shared as learnings back into the system and the organization. Just as there are many factors related to failures within IT systems, so too are there multiple factors involved in things that go well.

NOTE

While the events of the timeline are being described, listen for and help participants gain awareness of bias, blaming, and counterfactual statements.

Human Interactions

As engineers step through the process of investigating, identifying, and then working to solve the problem, conversations are happening among all who are involved. Engineers sharing in great detail exactly what they were thinking and doing and the results of those actions helps others involved in the firefight build a shared context and awareness about the incident. Capturing the dialogue will help spot where some engineers may need to improve their communication or team skills.

NOTE

Having an objective moderator run the exercise can help prevent one person (or small group) from steamrolling the conversation and avoids "groupthink." Having another person asking questions from a genuine place of curiosity forces thoughtful discussion of actions rather than relying on assumptions and existing beliefs around state and behavior.

Remediation Tasks

Included in those conversations should be fairly detailed documentation describing exactly what happened throughout the timeline. This may include specific commands that were run (copied and pasted into chat), or it could be as simple as engineers describing which commands they were using and on which systems. These descriptions teach others how to perform the same type of investigation—and not only will others learn something from discussing these tasks, but the engineers can analyze if they are taking the most efficient route during their investigation efforts. Perhaps there are better ways of querying systems. You may never know if you don't openly discuss how the work is accomplished.

TIP

Take note of all tasks completed during the timeline. By indicating if a task provided a positive or negative impact, we can learn and identify areas of improvement.

ChatOps

One method teams have employed to facilitate not only the conversation and remediation efforts but the forthcoming post-incident review is the use of ChatOps.[1]

NOTE

ChatOps is the practice of leveraging group collaboration tools to dually serve as the interface for development and operations efforts, as well as the conversations related to those interactions.

This new method of interacting, combining tools and conversations into a single interface, provides a number of benefits. In the context of a post-incident reviews, a system of record is automatically created that serves as a detailed recollection of exactly what took place during the response to the incident. When engineers combine conversations with the methods in which they investigated and took action to restore the service, a very clear timeline begins to emerge. Details on which engineers ran what specific commands at what time and what the results were provide a clear and accurate account of what took place. Much of the timeline is already constructed when ChatOps is part of the detection, response, and remediation efforts.

Metrics

Not all tricks and commands used during a firefight are suitable for chat or available via chatbots and scripts, but describing what was done from the engineer's local terminal at the very least helps to share more with the team about how to diagnose and recover from problems. Incidents become teaching opportunities as engineers step through the incident lifecycle.

One example of this is the querying and retrieval of relevant metrics during the response to an incident. First responders typically begin their triaging process by exploring a number of key metrics such as time-series data and dashboards. The collection of time-series data is extremely easy and cost-effective. You may not need to watch a

[1] For more on this subject, see my report "ChatOps: Managing Operations in Group Chat" (O'Reilly).

dashboard all day in an attempt to predict problems, but having those data points available during the investigation process helps engineers understand the current state of systems and reason about how they got there.

Any metrics that were leveraged during response and remediation efforts should be included in the review discussion. Not only does it help to tell the story of how teams responded to problems, but it also shares helpful information with a larger audience. Some engineers may not have been aware of such metrics or how to access them. Studying the ways that engineers used (or didn't use) metrics during a firefight will present areas for improvement.

Time to Acknowledge (TTA) and Time to Recover (TTR)

Two important metrics in a post-incident review are the time it took to acknowledge and recover from the incident. A simple way to measure improvement in responding to failure is the average time it takes to acknowledge a triggered incident. How are teams organized? How are first responders contacted? How long does it take to actually begin investigating the problems? Improvements in each of these areas (and more) will reduce the time it takes teams to recover from problems.

Likewise, measuring the average time it takes to recover from disruptions presents a metric to focus improvements. Once teams are formed, how do they work together in unison to investigate and eventually recover from the problem? Where is there friction in this process? Do engineers have the necessary tools or access to systems? Where can we make things better? Consistently reducing the average time it takes to recover from a problem reduces the cost of downtime.

Real Numbers

Cost of downtime = Deployment frequency × Change failure rate × **Mean time to recover** × Hourly cost of outage[2]

[2] Puppet + DORA, "2016 State of DevOps" 47.

In other words, improving the time to recover lowers the real dollar cost of failure. What accounts for the time between "time to acknowledge" and "time to recover" is examined during a well-executed post-incident review. Endless ways in which teams can improve can be discovered.

> Mean time to recover (MTTR) is an arithmetic average. Averages are not always a good metric because they don't account for outliers and therefore can skew the reality of what's actually happening. Still, when coupled with other metrics, it helps to tell a story in which you can discover where to focus improvement efforts.

Status Pages

As mentioned previously, transparency is extremely important, and not only within teams or the organization as a whole. Customers and users of our services expect transparent and timely communication about problems and remediation efforts. To provide this feedback, many employ status pages as a quick and easy way of issuing real-time updates regarding the state of a service.

Examples include:

- *https://status.newrelic.com/*
- *https://status.twilio.com/*
- *https://status.aws.amazon.com/*

Information regarding when status pages were updated and what they were to say should also be examined during a post-incident review. Including this in the discussion can help reduce the time it takes to inform our customers and users that we know about a problem and are investigating it, shortening that feedback loop.

Severity and Impact

For many stakeholders, the first question they have is "How bad is it?" Those in management positions may need just a high-level view of what's going on, freeing up engineers to continue remediation efforts without too many interruptions. Previously, we defined an incident. One component of that incident that many will ask about and which can help to create context during a post-incident review is the level of severity. Including this metric in analysis discussions means that it is documented.

Tracking the overall improvements in reduction of Sev1 or Sev2 outages will provide evidence of any increases in reliability and availability and decreases in the cost of downtime.

SLA impact

Along with the severity of the incident, many in management roles may be very concerned about the possible impact on any service level agreements (SLAs) that are in place. In some cases, harsh penalties are incurred when SLAs are broken. The sad truth about SLAs is that while they are put in place to establish a promised level of service, such as 99.999% (five nines) uptime, they incentivize engineers to avoid innovation and change to the systems. Attempts to protect the systems hinder opportunities to explore their limits and continuously improve them. Nevertheless, discussing the potential impact to SLAs that are in place helps everyone understand the severity of the failure and any delays in restoring service. If engineers are to make the case that they should be allowed to innovate on technology and process, they will have to ensure that they can work within the constraints of the SLA or allowable downtime.

Customer impact

Someone in attendance to the exercise should be able to speak to questions regarding customer impact. Having a member of support, sales, or elsewhere providing that kind of feedback to the engineers helps to build empathy and a better understanding of the experience and its impact on the end users. Shortening the feedback loop from user to engineer helps put things into perspective.

TIP

Throughout the discussion of the timeline, many ideas and suggestions will emerge regarding possible improvements in different areas.

It’s important to document these suggestions as this sets us up for concrete action items to assign and prioritize.

Contributing Factors

When discussing an incident, evidence will likely emerge that no single piece or component of the system can be pointed to as the clear cause of the problem. Systems are constantly changing, and their interconnectivity means that several distinct problems typically contribute, in various ways and to varying degrees, to failures. Simple, linear systems have a direct and obvious relationship between cause and effect. This is rarely the case with complex systems. Many still feel that by scrutinizing systems, breaking them down, and figuring out how they work we should be able to understand and control them. Unfortunately, this is often impossible.

> We are in an era, one in which we are building systems that can't be grasped in their totality or held in the mind of a single person; they are simply too complex.
>
> —Samuel Arbesman, *Overcomplicated*

Causation and correlation are difficult to triangulate, but discussion surrounding what factors contributed to a problem, both before and during remediation efforts, does have great value. This is where suggestions can be made with regard to countermeasures against factors that impacted the system. Perhaps additional monitoring would help engineers notice an early signal of similar problems in the future. Identifying and discussing as many factors as possible will set the foundation for what is to come next: action items.

NOTE

In a well-executed post-incident review, proximate or distal causation may be relevant to management who only want a high-level understanding of what went wrong. However, in complex systems there is never a single root cause.

Action Items

Once we have had a chance to discuss the events that took place, conversations that were had, and how efforts helped (or didn't) to restore service, we need to make sure our learnings are applied to improving the system as quickly as possible. A list of action items should be captured when suggestions are made on how to better detect and recover from this type of problem. An example of an action item may be to begin monitoring time-series data for a database and establish thresholds to be alerted on. Shortening the time it takes to detect this type of problem means teams can form sooner and begin remediation efforts faster.

Action items should be assigned to owners responsible for seeing them through. Ticketing systems are often used for accountability and to ensure that the work is prioritized appropriately and not simply added to the bottom of the backlog, never to actually be completed. It is recommended that action items that come from the exercise are made a top priority. Pausing existing development until the action items are complete indicates that teams respect the post-incident analysis process and understand the importance of balancing service reliability and availability with new features. Collaboration of between ops and product teams to ensure time is spent on these concerns along with new features allows for agile software development on infrastructure the entire team has confidence in.

A great new feature provides no value to the business if it is operated on an unreliable and unstable system.

NOTE

Don't allow for extended debate on action items. Place ideas into a "parking lot" for later action if need be, but come up with at least one action item to be implemented immediately.

Internal and External Reports

Internal process reports and external summary reports are the primary artifacts of a post-incident review. Internal reports serve to share the learnings about what happened with the larger team or organization and feed work back into improving the system. Transparency and sharing emphasize an importance placed on learning and improving. External reports provide the same benefit to stakeholders outside of the company and typically lack sensitive information or specific system details.

During remediation of service disruptions, a lot of information is retrieved and discussed openly among the responders to the event. It's important to capture this context for a deeper understanding of where improvements can be made. However, some information may be unnecessary or too sensitive to share with the general public. As a result, an internal investigation will typically serve as the foundation for a summarized version to be made available to the public as quickly as possible.

TIP

Save and store your analysis report artifacts in a location where they are available to everyone internally for review or to assist during future similar incidents.

Like we saw in Chapter 6's case study, users of systems and services are often aware of issues before the provider, particularly if that service is part of their own workflow. When someone relies on the uptime of a service or component, such as an API endpoint, its availability may directly impact their service or business. In the modern age of connectivity, we generally have a low tolerance for services that are unreliable. More and more, end users want (or need) to be able to access IT services nearly 24×7×365.

Following an incident, it's important to share relevant information with the general public. Often shorter and less detailed than the internal documentation created, an external analysis report serves as a summary of the high-level details of what happened, what was done to restore service, and what new commitments are being made to improve the availability of the service moving forward.

For most, the cause of a disruption in service isn't nearly as important as knowing that everything was done to recover as quickly as possible. Users want to know that something has been learned from this experience that will reduce the likelihood of a repeat occurrence of the problem, as well as being kept apprised of any improvements being made.

In Chapter 10, we will take a look at a guide that can serve as the starting point for both the process and summary reports.

Post-Incident Reviews at CSG International

Backstory

Like in many large organizations prior to DevOps transformation, the Ops organization at CSG International frequently dealt with production issues and fixed them with little help or involvement from the Development org. With little insight into the frequency or nature of these issues from the developers' perspective, Operations staff held post-incident reviews at the leadership level, but Development wasn't involved.

Development was concerned primarily with releases. Any problems that occurred during a release were followed by a post-incident review. Problems with operability in the production environment were not part of their concern.

This left a large gap in involving the right people in the right discussions. We weren't looking at things holistically.

After we joined our Development and Operations organizations together in 2016, Development began getting an up-front seat to the outages and Operations finally got some needed relief from a staffing standpoint. We also started holding joint analysis exercises (called After Action Summaries) at the leadership level.

Room for Improvement

Ultimately, CSG wanted three things from their post-incident reviews:

1. *Identify opportunities for improvement.*
2. *Share knowledge.* We were single-threaded in numerous areas, and this was a good opportunity to share knowledge. What did the troubleshooting look like? What steps were taken? What was the result of those steps? What went well that should be repeated next time?
3. *Shape company culture.* Simplifying the act of having these discussions and looking for growth opportunities reinforces a culture of continuous improvement. This helps overcome the culture of "fix it and move on."

The Problem

Lack of involvement from Development meant Operations often band-aided problems. Automation and DevOps best practices were not typically considered key components in improvements to be made to the system. Often problems were attacked with more process and local optimization, without addressing proximate cause. In a lot of cases, they moved on without any attacking of the problem.

The Shift in Thinking

The first large post-incident review I participated in was related to a product I inherited as part of our DevOps reorganization. I was still fairly new to working with the product and we had a major failed upgrade, causing our customers numerous issues. It was very eye-opening for me to simply walk through the timeline of change preparation, change implementation, and change validation.

Coming from the development world, I was very familiar with retrospectives. However, I'd never applied the concept to production outages before.

We invited members of multiple teams and just had a conversation. By simply talking through things, a number of things came to light:

- There were pockets of knowledge that needed to be shared.
- We had a number of bad practices that needed to be addressed. So many assumptions had been made as well. While we were working to remove silos within a team of Dev and Ops, we still had not fully facilitated the cross-team communication.
- This was a great way to share knowledge across the teams and also a great vehicle to drive culture change. As the team brainstormed about the issues, many great points were brought up that would not have been otherwise.

It quickly became clear this should be done for many more incidents as well.

The Plan

After we saw the real value in the exercise, we set out to hold a team-wide post-incident review on every production outage.

We didn't have a lot of formality around this—we just wanted to ensure we were *constantly learning and growing each time we had an issue.*

Additionally, we beefed up the process around our leadership After Action Summaries (AASs). We moved the tracking of these from a SharePoint Word doc to JIRA. This allowed us to link to action items and track them to closure, as well as providing a common location where everyone can see the documents.

Challenges

One of the biggest challenges has been analysis overload. Similar to agile team retros, you have to select a few key items to focus on and not try to boil the ocean. This takes discipline and senior leadership buy-in to limit work in process and focus on the biggest bang for the buck items.

The End Result

Teams hold post-incident review for outages where one or more of the following criteria are met: 1) major incident, 2) estimated cause or path forward, 3) good opportunity for knowledge sharing.

These continue to be more informal in nature, which encourages openness, sharing, and brainstorming of future solutions. All members of a team are invited. If it's a cross-team issue, members of other teams are invited as well. Meeting minutes are captured on a Confluence wiki page. We always ensure we discuss the timeline of the event, what went wrong, and things we can do to improve for next time. Where possible, we aim to solve things with automation over process. If the opportunity arises, we also discuss helpful troubleshooting steps and best practices. The focus is on *things* we can do to improve. If a person failed, we inspect ways the system they operated within failed to guard against human error. Action items are part of the minutes and are also recorded as JIRA tasks so they can be tracked.

While the process around team-level analysis is still fairly informal, our AAS process is much more structured. All major incidents and associated action items are entered into our ticketing system to be tracked.

The Transformation

Part of the transformation we've experienced is the attitude that we must get better and we are empowered to do so. Failure should not be accepted, but we learn from it and improve our systems as a result. When we have an outage now, we fix the immediate issue and then immediately flip the narrative to "What did we learn to help us avoid this happening again?" Another key component was the switch to tracking follow-ups in JIRA. This ensures follow-up and accountability so things aren't lost.

Comments

While post-incident reviews provide a number of benefits, I find the culture shift they unconsciously drive has been the most valuable thing about them for us. "What is within my control that I can change to make this system better?"

—Erica Morrison
Director of Software Development,
CSG International

CHAPTER 10

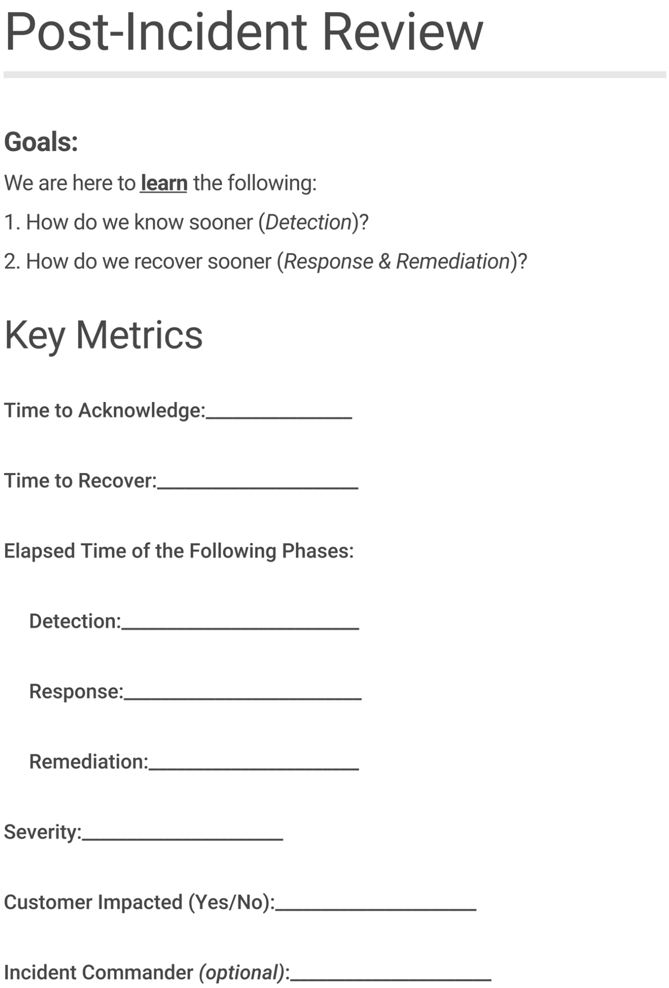
Templates and Guides

The structure of a post-incident review was described in Chapter 9, but here we'll look more closely at the output the exercise provides, as well as a procedural guide on how to conduct your own.

Sample Guide

This chapter presents a guide to help get you started. A downloadable version is available at *http://postincidentreviews.com*.

Begin by reflecting on your goals and noting the key metrics of the incident (time to acknowledge, time to recover, severity level, etc.) and the total time of each individual phase (detection, response, remediation).



Post-Incident Review

Goals:

We are here to **learn** the following:

1. How do we know sooner (*Detection*)?

2. How do we recover sooner (*Response & Remediation*)?

Key Metrics

Time to Acknowledge:______________

Time to Recover:___________________

Elapsed Time of the Following Phases:

Detection:______________________

Response:______________________

Remediation:____________________

Severity:___________________

Customer Impacted (Yes/No):___________________

Incident Commander *(optional)*:___________________

Establish and Document the Timeline

Document the details of the following in chronological order, noting their impact on restoring service:

- Date and time of detection
- Date and time of service restoration
- Incident number (optional)
- Who was alerted first?
- When was the incident acknowledged?
- Who else was brought in to help, and at what time?
- Who was the acting Incident Commander? (optional)
- What tasks were performed, and at what time?
- Which tasks made a positive impact to restoring service?
- Which tasks made a negative impact to restoring service?
- Which tasks made no impact to restoring service?
- Who executed specific tasks?
- What conversations were had?
- What information was shared?

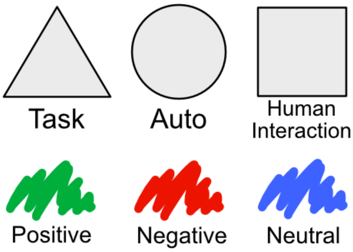
Plot Tasks and Impacts

Identifying the relationships between tasks, automation, and human interactions and their overall impact on restoring service helps to expose the three phases of the incident lifecycle that we are analyzing:

- Detection
- Response
- Remediation

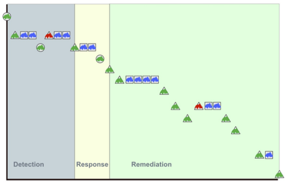
Areas of improvement identified in the detection phase will help us answer the question "How do we know sooner?" Likewise, improvements in the response and remediation phases will help us with "How do we recover sooner?"

Indicate the type and impact of
each entry on the timeline



Tasks and their impact on restoring service

Plot the tasks, human interactions, and automation along a timeline marking the impact on restoring service (good, bad, neutral). Noting the time between the phases of detection, response, and remediation, new target conditions can be established *(e.g. acknowledge incidents 50% faster)*.



By plotting the tasks unfolding during the lifecycle of the incident, as in Figure 10-1, we can visualize and measure the actual work accomplished against the time it took to recover. Because we have identified which tasks made a positive, negative, or neutral impact on the restoration of service, we can visualize the lifecycle from detection to resolution. This exposes interesting observations, particularly around the length of each phase, which tasks actually made a positive impact, and where time was either wasted or used inefficiently. The graph highlights areas we can explore further in our efforts to improve uptime.

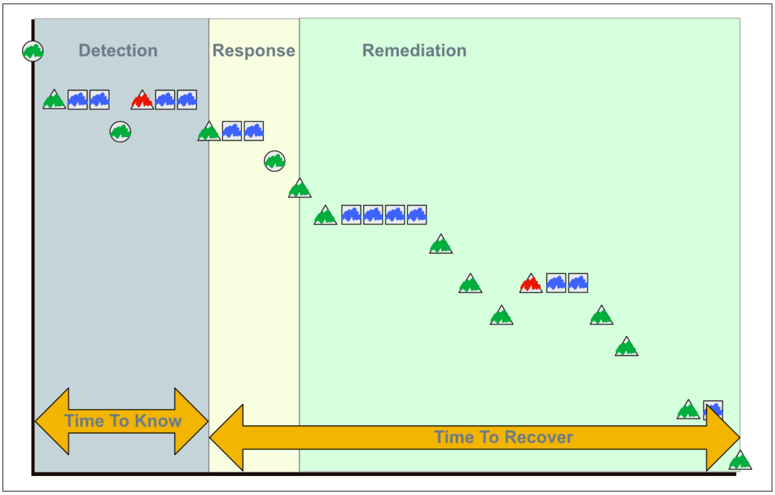


Figure 10-1. Relationship of tasks to processes and their impact on time to acknowledge and time to recover. The x-axis represents time. The y-axis indicates the evolving state of recovery efforts, an abstract construct of recovering from a disruption. Values are implied and not tied to any previously mentioned severity or incident category. Positive tasks drive the recovery path down and to the right.

Understand How Judgments and Decisions Are Made

Throughout the discussion, it's important to probe deeply into how engineers are making decisions. Genuine inquiry allows engineers to reflect on whether this is the best approach in each specific phase of the incident. Perhaps another engineer has a suggestion of an alternative quicker or safer method. Best of all, everyone in the company learns about it.

In Chapter 6, Gary exposed Cathy to a new tool as a result of discussing the timeline in detail. Those types of discovery may seem small and insignificant, but collectively they contribute to the organization's tribal knowledge and ensuring the improvement compass is pointed in the right direction.

Engineers will forever debate and defend their toolchain decisions, but exposing alternative approaches to tooling, processes, and people management encourages scrutiny of their role in the organization's ongoing continuous improvement efforts.

Learnings

The most important part of the report is contained here.

Genuine inquiry within an environment that welcomes transparency and knowledge sharing not only helps us detect and recover from incidents sooner, but builds a broader understanding about the system among a larger group.

Be sure to document as many findings as possible. If any member participating in the post-incident review learns something about the true nature of the system, that should be documented. If something wasn't known by one member of the team involved in recovery efforts, it is a fair assumption that others may be unaware of it as well.

The central goal is to help everyone understand more about what really goes on in our systems and how teams form to address problems. Observations around "work as designed" vs. "work as performed," as mentioned in Chapter 7, emerge as these findings are documented.

What have we learned by discussing the timeline? What new information was gained?

Learnings

____

____

____

____

____

____

______________________________ .. continue in additional space

Contributing Factors

Many factors that may have contributed to the problem and remediation efforts will begin to emerge during discussions of the timeline. As we just covered, it's good to document and share that information with the larger teams and the organization as a whole. The better understanding everyone has regarding the system, the better teams can maintain reliability.

What significant components of the system or tasks during remediation were identified as helpful or harmful to the disruption and recovery of services?

NOTE

Factors relating to remediation efforts can be identified by distinguishing each task as discussed in the timeline with a value of positive, negative, or neutral.

Contributing Factors

____

____

____

____

____

____

___________________________ .. continue in additional space

As responders describe their efforts, explore whether each task performed moved the system closer to recovery, or further away. These are factors to evaluate more deeply for improvement opportunities.

How Is This Different Than Cause?

While not quite the same as establishing cause, factors that may have contributed to the problem in the first place should be captured as they are discovered. This helps to uncover and promote discussion of information that may be new to others in the group (i.e., provides an opportunity to learn).

In the case study example in Chapter 6, the unknown service that Cathy found on the host could have easily been identified as the root cause. However, our approach allowed us to shed the responsibility of finding cause and continue to explore more about the system. The runaway process that was discovered *seemed like* the obvious problem, and killing it *seemed to have* fixed the problem (for now). But as Greg pointed out, there are other services in the system that interact with the caching component. What if it actually had something to do with one of those mystery services?

In reality, we may never have a perfectly clear picture of all contributing factors. There are simply too many possibilities to explore. Further, the state of the system when any given problem occurred will be different from its current state or the one moments from now, and so on. Still, document what you discover. Let that open up further dialogue.

Perhaps with infinite time, resources, and a system that existed in a vacuum, we could definitively identify the root cause.

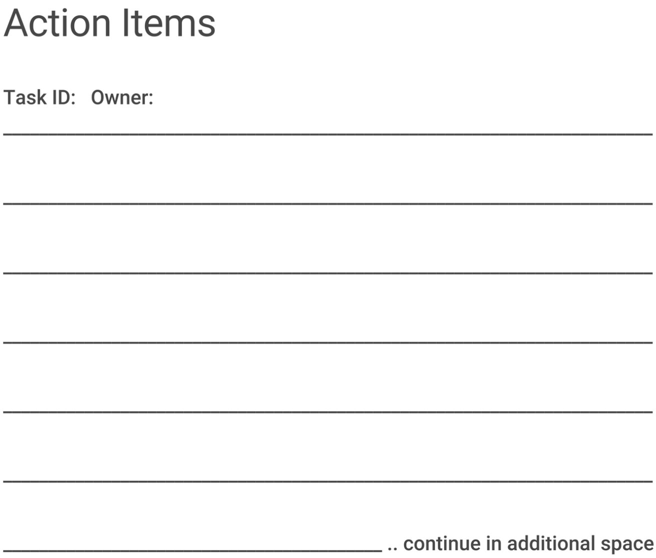
However, would the system be better as a result?

Action Items

Finally, action items will have surfaced throughout the discussion. Specific tasks should be identified, assigned an owner, and prioritized. Tasks without ownership and priority sit at the bottom of the backlog, providing no value to either the analysis process or system health. Countermeasures and enhancements to the system should be prioritized above all new work. Until this work is completed, we know less about our system's state and are more susceptible to repeated service disruptions. Tracking action item tasks in a ticketing system helps to ensure accountability and responsibility for work.

NOTE

Post-incident reviews may include many related incidents due to multiple monitoring services triggering alarms. Rather than performing individual analyses for each isolated incident number, a time frame can be used to establish the timeline.



Action Items

Task ID: Owner:

____

____

____

____

____

____

_________________________________ .. continue in additional space

Summaries and Public Reports

These exercises provide a great deal of value to the team or organization. However, there are likely others who would like to be informed about the incident, especially if it impacted customers. A high-level summary should be made available, typically consisting of several or all of the following sections:

- Summary
- Services Impacted
- Duration
- Severity
- Customer Impact
- Proximate Cause
- Resolution
- Countermeasures or Action Items

Amplify the Learnings

John Paris (Service Manager, Skyscanner) and his team decided they needed to create a platform for collective learning.

Weekly meetings were established with an open invite to anybody in engineering to attend. In the meetings, service owners at various levels have the opportunity to present their recent challenges, proposed solutions, and key learnings from all recent post-incident reviews.

“There are many benefits from an open approach to post-incident reviews,” says Paris. “But the opportunities for sharing and discussing outcomes are limited and as the company grows it becomes harder to share learnings throughout the organization. This barrier, if not addressed, would have become a significant drag on both throughput and availability as the same mistakes were repeated squad by squad, team by team.”

Exposing learnings to a broader audience can create a self-imposed pressure to raise the quality of post-incident analysis findings. This was especially true for the owners of Skyscanner’s data platform.

CHAPTER 11
Readiness

> You can't step in the same river twice.
>
> —Heraclitus (Greek Philosopher)

We never had a name for that huddle and discussion after I'd lost months' worth of customer data. It was just, "Let's talk about last night." That was the first time I'd ever been a part of that kind of investigation into an IT-related problem.

At my previous company, we would perform RCAs following incidents like this. I didn't know there was another way to go about it. We were able to determine a proximate cause to be a bug in a backup script unique to Open CRM installations on AWS. However, we all walked away with much more knowledge about how the system worked, armed with new action items to help us detect and recover from future problems like this much faster. As with the list of action items in Chapter 6, we set in motion many ways to improve the system as a whole rather than focusing solely on one distinct part of the system that failed under very unique circumstances.

It wasn't until over two years later, after completely immersing myself in the DevOps community, that I realized the exercise we had performed (intentionally or not) was my very first post-incident review. I had already read blog posts and absorbed presentation after presentation about the absence of root cause in complex systems. But it wasn't until I made the connection back to that first post-incident review that I realized it's not about the report or discovering the root cause—it's about learning more about the system and opening new opportunities for improvement, gaining a deeper understanding of the system as a whole and accepting that failure is a natural part of the process. Through that awareness, I finally saw the value in analyzing the unique phases of an incident's lifecycle. By setting targets for small improvements throughout detection, response, and remediation, I could make dealing with and learning from failure a natural part of the work done.

Thinking back on that day now gives me a new appreciation of what we were doing at that small startup and how advanced it was in a number of ways. I also feel fortunate that I can share that story and the stories of others, and what I've learned along the way, to help reshape your view of post-incident analysis and how you can continuously improve the reliability and availability of a service.

Post-incident reviews are so much more than discussing and documenting what happened in a report. They are often seen as only a tool to explain what happened and identify a cause, severity, and corrective action. In reality, they are a process intended to improve the system as a whole. By reframing the goal of these exercises as an opportunity to learn, a wealth of areas to improve become clear.

As we saw in the case of CSG International, the value of a post-incident review goes well beyond the artifact produced as a summary. They were able to convert local discoveries into improvements in areas outside of their own.

They've created an environment for constant experimentation, learning, and making systems safer, all while making them highly resilient and available.

Teams and individuals are able to achieve goals much more easily with ever-growing collective knowledge regarding how systems work. The results include better team morale and an organizational culture that favors continuous improvement.

The key takeaway: focus less on the end result (the cause and fix report) and more on the exercise that reveals many areas of improvement.

When challenged to review failure in this way, we find ingenious ways to trim seconds or minutes from each phase of the incident lifecycle, making for much more effective incident detection, response, and remediation efforts.

Post-incident reviews act as a source of information and solutions. This can create an atmosphere of curiosity and learning rather than defensiveness and isolationism. Everyone becomes hungry to learn.

Those that stick to the old-school way of thinking will likely continue to experience frustration. Those with a growth mindset will look for new ways of approaching their work and consistently seek out opportunities to improve.

> The only thing we can directly and reliably control in the complex socio-technical systems in which we operate is our reactions. That's what makes post-incident analysis—and getting really practiced, both as individuals and as teams at it—so important. At the end of the day, taking time to understand how and for what reasons we reacted, how we can react better today, and ponder and practice how we collectively might react tomorrow, is an investment that pays consistent, reliable dividends.
>
> Technology is always going to change and progress; we have little control over that; the real danger is that we, as the "socio" half of our socio-technical systems, remain static in how we interact in that system.
>
> —J. Paul Reed, DevOps consultant and retrospective researcher

Next Best Steps

Which of the following is your next best step?

1. Do nothing and keep things the same way they are.
2. Establish a framework and routine to discover improvements and understand more about the system as a whole as it evolves.

If you can answer that question honestly to yourself, and you are satisfied with your next step, my job here is done.

Wherever these suggestions and stories take you, I wish you good luck on your journey toward learning from failure and continuous improvement.

About the Author

Serving as a DevOps Champion and advisor to VictorOps, **Jason Hand** writes, presents, and coaches on the principles and nuances of DevOps, modern incident management practices, and learning from failure. Named "DevOps Evangelist of the Year" by DevOps.com in 2016, Jason has authored two books on the subject of ChatOps, as well as regular contributions of articles to Wired.com, TechBeacon.com, and many other online publications. Cohost of "The Community Pulse", a podcast on building community within tech, Jason is dedicated to the latest trends in technology, sharing the lessons learned, and helping people continuously improve.