SRE Best Practices for Incident Management

Ways to leverage site reliability engineering (SRE) and modern incident management (IM) practices to keep your customers happy

Abstract:

In this white paper, you'll learn why traditional ways of responding to IT incidents is no longer good enough. You'll hear about the rise of a concept called site reliability engineering (SRE), and how the role of this type of incident management can not only coexist with, but also strengthen a DevOps approach to development. You'll learn the benefits of putting a modern incident management team in place. And you'll get a list of nine best practices for building an incident management program of your own.

In particular, you'll hear about an emerging practice called **chaos engineering** that can help your incident management team excel at keeping your systems and sites functioning at top performance and stability.

Introduction:

Why you need effective incident management

An incident is an unexpected disruption to a service. It disturbs the normal operation of your business and impacts your customers—whether indirectly or directly. Examples of incidents include application crashes, application locks, network service problems, and file-sharing difficulties—even Wi-Fi connectivity issues for employees.

The last item might sound trivial. Why would an internal Wi-Fi connectivity issue impact customers? Because the aftershocks of things not working properly for your employees cascade through your organization and eventually reach your customer. A report that is not ready, code that isn't tested properly, an email not received in a timely manner—all these things can reverberate until a customer—whether consumer or business—is made unhappy.

Incident management is an organized approach to addressing and managing the aftermath of a security breach, computer incident, or any other customer-impacting incident. The goal is to handle the situation in a way that limits damage and reduces recovery time and costs. There are many types of issues that may impact your incident management approach, each with a different level of urgency. (See Figure 1.)

Type of issue	Definition
Incident	Unexpected disruption to a service that impacts customers, whether directly or indirectly
Service request	A formal request from a user for a service or product
Problem	A series of incidents with no known root cause

Figure 1: Common terms in IM

Why is incident management important? Because every time there's a glitch in your IT operations, it costs you money. Gartner found the average cost of downtime to be $5,600 per minute. This is just the average. At the low end it can be $140,000 an hour; for average firms, $300,000 an hour; and more than half a million dollars ($540,000) per hour at the high end. Gremlin analyzed the downtime cost of the largest online brands, and found that Amazon loses more than $13 million per hour of downtime. Walmart.com loses $2.4 million per hour. And Homedepot.com loses $868,000 per hour.

The kind of incident management program we're talking about in this white paper is not the same as a security incident response program. Those types of programs specifically target security breaches such as distributed denial-of-service (DDOS) attacks, hackers, malware, or anything else that impacts your network due to the work of malicious intruders (whether internal or external).

Rather, we're focusing on those programs that investigate and remediate problems that occur within your IT infrastructure and application ecosystem, and for which you may not have an immediate answer.

Today, the need for modern incident management has grown because companies are more and more dependent on digital technology to function—technology that goes wrong can seriously impact a business' ability to function. Not only are we more dependent on the tech, but the tech has grown more complex and is effectively intractable with the rise of distributed architectures.

The digital transformation of the business universe is progressing at a dizzying pace. We are at the beginning what World Economic Forum calls the "Fourth Industrial Revolution," characterized by mass adoption of digital technologies across all industries—and a rapid disruption of the firms that don't pick up the digital transformation challenge. Specifically, we're seeing the adoption of "cloud-native" architectures that take full advantage of what the cloud has to offer in terms of scalability, flexibility, and low cost. This transformation requires a new take on incident management as well.

The origins of ITIL

The ITIL (IT infrastructure library) was first conceived of in the 1980s, when the British government decided it was receiving substandard—and too-expensive—service for IT problems. It charged the Central Computer and Telecommunications Agency (CCTA) with developing a framework to make it possible to utilize IT resources in more efficient and financially responsible ways.

In the 1990s, both large private companies and government agencies throughout Europe very rapidly adopted the framework. In 2000, the CCTA merged into the Office for Government Commerce (OGC) and in the same year, Microsoft used ITIL to develop its proprietary ITSM (IT service management) framework, the Microsoft Operations Framework (MOF).

Why you need a robust IM program

The reason is simple: always-on digital businesses can't afford outages. Ever. According to Fujitsu's 2018 global insights study:

- **42% of companies that operate online only have universally embraced digital transformation**
- **75% of companies with more traditional operating models have transformation initiatives planned or in process**

Chances are good that you and your company are more dependent on digital technology than ever. Does your company rely on an outside vendor for internal employee communication? What about for communication with customers? What if that vendor experienced an outage; how would your incident management program respond? What would you do if your web sales application suddenly stopped taking orders or allowing customers to check out on a typically busy sales day like Black Friday?

In 2001, Version 2 of the ITIL was released, and in 2007, Version 3 was published. Over the following decade, the ITIL became the most widely adopted ITSM framework in the world.

In its current form (ITIL 2011), the ITIL is five volumes of libraries, each of which covers a different ITSM lifecycle stage. ITIL describes processes, procedures, tasks, and checklists to be applied by organizations to ensure their IT departments deliver value and maintain a minimum level of competency.

Since July 2013, the ITIL has been owned by AXELOS, a joint venture between Capita and the U.K. Cabinet Office. AXELOS licenses organizations to use the ITIL intellectual property, accredits licensed examination institutes, and manages updates to the framework. Organizations that implement ITIL internally do not require licenses.

Here are the top reasons you need an IM program:

- **A demand for continuous delivery.** The days when you delivered an application and rarely revisited the code are long gone. Your engineers are constantly deploying new code to enhance both the features and performance of your internal applications and customer-facing apps. How well is this working in production for your firm? Do you find more software glitches that need to be responded to as incidents once the code is in production? Some 40% of applications get reworked because of the disconnect between developers and operations personnel. Deploying a DevOps model with a "you made it, you own it" philosophy helps to address this problem. But even with a well-oiled DevOps machine, you're still going to have problems in production. Unknown dependencies and integration glitches inevitably occur. You need a way to resolve them quickly, without resorting to assigning blame.
- **No tolerance for slow application response.** *Slow is the new down* is something you hear today in IT circles. A slow app or website means visitors are abandoning the site. You have to jump into action to fix any sluggish response before your users get vocal, because most won't (they will just leave and find another vendor).
- **Increasingly complex Infrastructure.** Microservices, containers, application programming interfaces (APIs), virtualization, and—especially—cloud deployments to someone else's infrastructure make it difficult to pinpoint the cause of many performance problems that impact customers. The benefits of using these architectures is that you increase the speed of releases, but their complexity increases both the likelihood of an incident occurring and the challenge of figuring out what went wrong. You need a trained incident management team to tackle these tougher problems.

A robust incident management program address the challenges of digital transformation. It adds a safety net to your continuous delivery pipeline of releases, and catches slow responses before your users complain or abandon you. It also provides a way to cope with the increasingly complex infrastructures of today's businesses.

Incident management programs also reduce costs. You save money by being able to minimize the need for customer-support operations because you are catching small problems early. You'll also improve customer trust and loyalty by offering services that are reliably available and you'll enhance competitiveness by being more agile in response to market demands and by freeing up your support engineers for more strategic work.

Traditional IT responses to operational "incidents"

In the past, most large enterprises conformed to ITSM (IT service management) processes, specifically using the ITIL (IT infrastructure library) framework to respond to IT problems. The ITIL is a set of best practices for an IT-centric way of responding to incidents that impact operations (see sidebar).

ISO 20000 defines the objective of ITIL as: To restore agreed service to the business as soon as possible or to respond to service requests.

Under traditional ITIL, any problems are first reported to the help desk. If they can't be resolved quickly, they are escalated to technical specialist support groups. If an incident requires programming or code changes, the appropriate resources are brought in to implement those changes.

The ITIL incident management process uses a workflow for efficient resolution. Incidents pass through these stages:

1. Incident identification
2. Logging
3. Categorization
4. Prioritization
5. Response
6. Diagnosis
7. Escalation
8. Resolution and recovery
9. Closure

Incidents are classed as hardware, software, or security, although a performance issue can often result from any combination of these areas. Software incidents typically include service availability problems or application bugs. Hardware incidents typically include downed or limited resources, network issues, or other system outages.

A critical aspect of the ITIL is strong **change management**. Any “non-standard” changes to applications or infrastructure have to go through a rigorous change-management process—in essence, a quality check to ensure a change doesn’t inadvertently adversely affect anything—before it can be released. The “change manager” in the ITIL process therefore acts as a lookout over any changes. Naturally, this can slow down the time it takes to fix a problem considerably. A rigid change management system—presided over by a human—can take up valuable time when a critical system is down. Some critics charge that this even encourages support specialists to apply short-term fixes. That way, they avoid arduous change-management processes rather than finding a permanent solution that solves a problem for good.

This approach is also obviously incompatible with DevOps and Agile-based continuous deployment software development methodologies. Traditional ITIL use simply doesn’t do it anymore for enterprises that have gone the DevOps and Agile route. Something had to give.

The goal of every enterprise is to fix things that aren’t working as quickly as possible. That is why continuous delivery / continuous integration is becoming so popular. By automating the testing of software to increase the velocity of code releases, a continuous delivery pipeline helps incident management fix things quickly, with as little friction as possible.That may involve writing new code. If so, a follow-on goal is to deploy the the fix without breaking anything else—but doing so at lightning speed. Jon Hall, a senior product manager in the enterprise service management space argued in a Medium article that this makes it difficult to assert that continuous delivery “can be aligned easily with existing ITSM norms [of change management].”

To understand why, you must understand that a “standard change” in IT infrastructure has a particular meaning in the ITIL, explained Hall. To be classed as “standard,” a change must be recurrent, well known, and “procedurallized to follow a pre-defined, relatively risk-free path.” It must also be the “accepted response to a specific requirement or set of circumstances.” If these conditions are met, ITIL allows the change to be approved without the change manager or the change advisory board stepping in. But how often are these conditions met in the complex real world when a critical website is down and customers are complaining?

The ITSM, defined this way, puts the change manager in a position of authority at times of crisis. Requiring a human to sign off on every change in every pipeline doesn’t just slow things down, it simply won’t work fast enough to resolve critical incidents at the speed modern businesses need.

By rebelling against the ITIL’s rigid model, businesses that are committed to continuous development, Agile, and DevOps are effectively saying, “Yes, we govern releases so they don’t break. But not your [ITIL’s] way,” wrote Hall.

The rise of site reliability engineering (SRE)

Needing to fix things rapidly in a DevOps environment ignited the birth of a new movement called **site reliability engineering (SRE)**. This new approach marries software engineering with IT operations. By applying software engineering practices and precepts to resolving IT incidents, issues frequently get resolved more swiftly—and more thoroughly—than when using the ITIL framework.

SRE is revolutionizing incident management. By moving to an SRE-based model, businesses can keep their DevOps frameworks and get code into production continuously and rapidly, while ensuring that critical software and systems remain available.

SRE does this by breaking down silos between the code being written, the infrastructure it lives on, and the deployment process. Rather than having those three activities managed by three separate teams, they are all owned by everyone. In short, each engineer owns the entire lifecycle of his or her code. This means that everyone has a more holistic view of the system. Rather than the "throwing problems over the wall" mentality of traditional development organizational structures, SRE aligns the incentives for resolving incidents with the people who wrote the code.

SRE was created at Google in 2003 by Ben Treynor, who was hired to lead a team of software engineers to run a production environment. SRE was the method that Treynor came up with to ensure that Google's sites were more reliably available, and that any issues were resolved swiftly. Originally starting with just seven engineers, the SRE team at Google is now comprised of more than 1,500 SRE engineers.

Moreover, Intuit, ServiceNow, Microsoft, Apple, Twitter, Facebook, Dropbox, Amazon, Target, Dell Technologies, IBM, Xero, Oracle, Zalando, Acquia, VMware, GitHub, Waze, and Ticketmaster have all put together SRE teams as well.

At its core, SRE attempts to improve the reliability of existing software while minimizing the work involved in keeping it maintained. Unlike traditional ITIL ITSM, SRE focuses on continuous improvement and—this is a very critical aspect of it—the automation of common IT support operations tasks. Through this automation, the SRE team is able to focus on more strategic, higher-level work.

The difference between DevOps and SRE

DevOps and SRE are not two competing methods for software development and operations, but rather allies in a joint objective to break down organizational barriers to deliver better software faster. In fact, you could say that SRE is simply a continuation or next iteration of DevOps.

The DevOps movement was inspired by the fact that developers would often write code without considering how it would work when in production. They would throw this code "over the wall" to the operations team, which was responsible for deploying applications and keeping them maintained. This could create situations where the operations team was bearing the burden of supporting code that wasn't ready for prime time. DevOps was the answer. DevOps is a merger of developers and operational workers. The basic culture of DevOps is, "you built it, you own it." That way, developers are very much incentivized to write code that will run in production smoothly.

SRE, which began at Google independently of the DevOps movement, is aligned with DevOps in spirit, but it is a much more directive approach to how to get it done. Whereas DevOps has "five pillars" to its philosophy, SRE actually prescribes how to implement the ideas behind these pillars.

It's possible that within your organization someone is already acting as a site reliability engineer without the title, according to an O'Reilly book on SRE authored by the Google SRE team:

> ***But for sizes between a startup and a multinational, there probably already is someone in your organization who is doing SRE work, without it necessarily being called that name, or recognized as such. Another way to get started on the path to improving reliability for your organization is to formally recognize that work, or to find these people and foster what they do—reward it.***

Site reliability engineers spend up to 50% of their time being "on call" to resolve issues and manually intervene in case of problems with whatever system, application, or site they are charged with overseeing. In that way, they are doing traditional ITIL ITSM tasks. But the other 50% of the time is where things are radically different: they spend those hours on development, enhancing features or adding new ones, scaling, or automating. If this sounds a lot like a DevOps philosophy, it should come as no surprise that SRE is considered (as cited in the Google SRE book) as a "specific implementation of DevOps with some idiosyncratic extensions."

The ultimate goal of SRE is to not just put systems in place and hope they'll continue to meet the needs of the organization—and stay up—without modification, but to assume that they will need adjusting as the business adapts to the market, and to ensure they remain robust in the midst of those changes. Part of an SRE's job responsibilities is to take a holistic view of a system to understand how all the pieces fit together.

The book's epigraph in fact, quotes a so-called "traditional SRE saying," that "Hope is not a strategy."

DevOps	**SRE**
Reduce organization silos	Shares ownership with developers by using the same tools and techniques across the stack
Accept failure as normal	Has a formula for balancing accidents and failures against new releases
Implement gradual change	Encourages moving quickly by reducing costs of failure
Leverage tooling and automation	Encourages “automating this year’s job away” and minimizing manual systems work to focus on efforts that bring long-term value to the system
Measure everything	Defines prescriptive ways for measuring availability, uptime, outages, and toil, among other things

Key aspects of a SRE program

An interview with Traynor, now a vice president of engineering at Google, highlighted some key aspects of the firm’s SRE program:

- Although in operations roles, SREs are developers also. Google hire people as SREs who “*are inherently both predisposed to, and have the ability to, substitute automation for human labor*.”
- SRE teams are responsible for “*availability, latency, performance, efficiency, change management, monitoring, emergency response, and capacity planning*.”
- Site reliability engineers act in a supporting role to software engineers—but also in an enforcing role. *Like the change managers in ITIL, they are charged with making sure software engineers don’t inadvertently break things*. To incentivize them, Google has created something called the “error budget,” or a certain percentage of downtime they are allowed for a particular project in a particular pipeline. If something breaks after a change is made in software engineering, that’s not necessarily held against them, *provided they stay within their error budget*.

Building a modern incident management program: best practices

Using SRE tenets as the basis for a modern incident management program will help you improve your ability to ensure availability, minimize latency, optimize performance and efficiency, and streamline emergency response, among other benefits.

Gremlin has come up with nine best practices for building an effective incident response program. Contributors to this list include Kolton Andrus, the co-founder and CEO of Gremlin who previously led SRE teams at Netflix and Amazon, and Tammy Butow, principal site reliability engineer at Gremlin and former SRE lead at Dropbox and Digital Ocean.

Best practice No. 1: Create 24/7 IM shifts

Although at first your business might respond to incidents on an as-needed basis, eventually you will want to schedule shifts so that designated engineers know when they might get paged to deal with an incident. You may want to start slow by having shifts cover business hours only, but eventually, especially if you run a business that is increasingly dependent on digital technologies, you will want to institute a 24 / 7 schedule for engineers to perform incident management. Just being scheduled doesn’t mean a page will necessarily come. But leading companies will be prepared.

Best practice No.2: Triage incidents into "severity" tiers

At most companies that use modern incident management practices, different services are divided into tiers based on criticality to the business. These incidents are typically called "sevs" for "severity." Sev 0 incidents are the most important. For Netflix, if it experienced a problem with its "identity service," which does the actual streaming of videos to subscribers, that would be classified as a Sev 0. At Amazon, anything that interferes with the service that processes customer orders is also obviously Sev 0. If anything goes wrong with a Sev 0 service, then the pagers light up. Here is excellent discussion of how to implement a high-severity incident management program.

Some examples (from the article) of what Sev levels might look like, and what the appropriate response would be.

SEV Level	**Description**	**Target resolution time**	**Who is notified**
SEV 0	Catastrophic Service Impact	Resolve within 15 min	Entire company
SEV 1	Critical Service Impact	Resolve within 8 hours	Teams working on SEV & CTO
SEV 2	High Service Impact	Resolve within 24 hours	Teams working on SEV

What does it mean to be a site reliability engineer?

Digital experience intelligence provider Catchpoint recently surveyed 416 site reliability engineers with the goal of understanding what it means to act in that role. Availability of applications and services is their main concern, with 84% of respondents listing availability of systems to users as one of the most important key performance indicators. During incident resolution activities, 94% of respondents reported reliance on instant messaging solutions over other methods such as war rooms, video conferencing, phone, and email. The top three tools the engineers reported being unable to live without were alerting, version controls, and chat tools.

Best practice No. 3: Designate an incident manager on call (IMOC) or “incident commander” for each shift

For each shift, you need to assign an IMOC or incident commander who will be in charge should an incident occur. This is a highly experienced person, preferably someone who has been an engineer himself or herself at some point, and has proved useful in the past at triaging incidents—particularly serious ones—managing teams, and directing people to find solutions quickly.

Ideally, you have a group of IMOCs on staff who rotate through the shifts, with one person on call at any point in time. It is critical to train your IMOCs thoroughly because they will be the point person in charge of coordination and communication if an issue occurs. It is best to compile an IMOC runbook to help get new IMOCs up to speed quickly.

Best practice No. 4: Train and rotate site reliability engineers throughout the company

Each important development team should have at least two engineers that have been trained in SRE techniques. Start small—choose the top three most critical systems in your business. Assign site reliability engineers to them, and to teach the rest of the team to get to be really good at incident management. Transfer the knowledge over, make sure they’re building good automation tools, reducing incidents, and focusing on getting to know what is going on in their particular system. In the ideal world, everyone would be trained to do SRE, and everyone would take their turn being on call to protect customers from being impacted by problems with critical systems.

Best practice No. 5: Perform blameless post mortems

The technology industry has evolved over the years to the point where the general philosophy is that no one should be fired for making a mistake. In the old world there was a tendency to find and terminate the “culprit” of a major outage, but today there is recognition that systems are complex, and although human error might contribute, it doesn’t do anyone any good to find a scapegoat. So one of the cultural advancements of the last five years has been blameless post-mortems. If a single human mistake was capable of bringing down a major system, then that’s a failure of the company, not that individual. One best practice is to use the “five whys” technique during the port mortems.

For example, at both Amazon and Netflix would have weekly meetings when they would review all incidents that occurred, and talk through all the contributing factors. Amazon in particular liked the five-whys method of searching for root causes—which could be cultural as well as technical. The technique started at Toyota, where it was a critical component of training within its manufacturing operations. The idea is to first ask a broad question, “*Why did the system fail?*” and then drill down with subsequent “*whys*” until the nature of the problem became clear. So you can move from “*Why did that code break?*” to “*Why wasn’t it tested properly?*” to “*Why wasn’t there enough time in the schedule?*” and “*Why were we under such pressure to deliver in an unreasonable time frame?*”

Best practice No. 6: Create runbooks and keep them up to date

The unspoken question in every incident review is: how do we prevent this from happening again? There are obviously many, many different ways that this question can be answered. But the runbook should be a mainstay of every team. The runbook is a per-service, per-team entity. But a runbook is only as good as people make it—so prioritizing that it is updated and kept absolutely current is essential. In an ideal world they're all up to date and people are looking at them regularly, they're practiced frequently, and the processes all become muscle memory to the team so the right things happen all the time.

Best practice No. 7: Continuously build (or buy) tools for automated incident management

After you have the why-did-this-happen discussion, you want to take your solution—the process you used to fix a problem—and whenever possible, automate it. For example, if a failure occurred because of the lack of a database backup, you would automate that process. Or if the system couldn't handle a certain kind of network failure, you would want to automate testing to verify that those kinds of failures were being tested for during build and deploy, and get caught before the service goes to production.

Best practice No. 8: Align incentives and pain

A very important action to take is to align incentives and pain. If pain is aligned with the person who is responsible, you'll naturally see the right behaviors emerge. When there's a disconnect, that's when you get dysfunctional behaviors. This is in fact what was behind the emergence of DevOps: the people responsible for the quality of the code (the developers) were not the ones putting it into production. With incident management, the same rules should apply: you build it, you own it. If you are on the site reliability engineering team and because you were in a hurry to fix something, you pushed some bad code that broke something else, you should be the one to get paged at 3am. Naturally that will lead to better software and better testing, because people don't enjoy that kind of disruption in their lives.

Best practice No. 9: Use chaos engineering as a "forcing function"

Chaos Engineering is a disciplined approach to identifying failures before they become outages. By proactively testing how a system responds under stress, you can identify and fix failures before they end up in the news. Using chaos engineering is the hallmark of a truly mature IM program.

Chaos Engineering lets you compare what you think will happen to what actually happens in your systems. You literally "break things on purpose" to learn how to build more resilient systems so availability and performance is not a problem for users or customers.

For more information on chaos engineering, read our white paper.

Conclusion: what's next?

To help you build your modern IM program, Gremlin put together the Gremlin Incident Management Maturity Model, which indicates what best practices have been implemented at each stage of maturity.

GREMLIN INCIDENT MANAGEMENT MATURITY MODEL

A: Doing chaos engineering exercises at least twice a week

B: Set up monitoring dashboards and alerts

C: Establish 24 /7 rotations

D: Establish business-hour rotations

E: Train engineers in SRE techniques and designate IMOCs to lead teams when incidents occur

You can start slow, with just one IMOC and two or three engineers trained in SRE. But the ultimate goal is to move from being reactive to being proactive: putting automated fixes in place and testing your systems all the time to the breaking point with chaos engineering.