vmware®

GET STARTED

Establishing an SRE-Based Incident Lifecycle Program

A reference guide for learning from outages

Author: Gustavo Franco

Table of Contents

Preface

Organizations with an ill-defined process for continuously analyzing, detecting, preventing and managing internet-related system outages—or even no process at all—risk making their customers unhappy, burning out their employees and wasting time on unnecessary toil. With that in mind, this e-book is designed to be used as a guide that will walk you through the process of designing a streamlined incident lifecycle program.

As a site reliability engineer (SRE) who teaches customers how to adopt reliability engineering practices, "incident response" or "incident management"—in other words, how to manage a production outage once it occurs—is a common topic of discussion. But while managing a production outage once it occurs is important, it's actually one small part of a much broader problem space, which is why I like to discuss responding to or managing incidents in the context of an incident lifecycle program.

An incident lifecycle program is composed of the terminology and processes adopted by a company to describe its approach to incident prevention, detection, response and analysis.

While prevention, detection, response and analysis require no further definition, what constitutes an incident in the context of internet-related system outages has to be discussed. This is where we will get started.

The Incident Definition

For many companies, “incident” is a vaguely defined synonym for alerts, tickets or customer issues. I recommend establishing a more formal definition for production incidents that is associated with their severity. Here is an example:

An incident is in progress when the expected behavior of the system falls below an acceptable level in such a way that an immediate coordinated response is required.

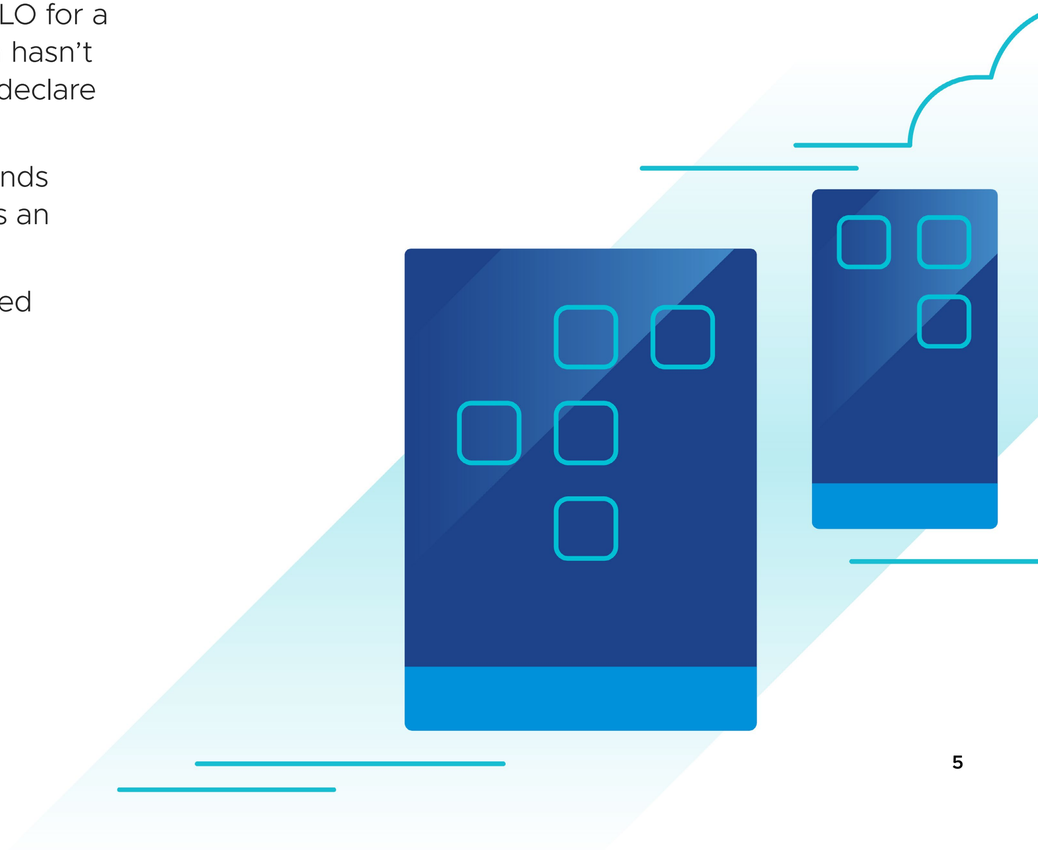
Note: Such an event might be termed an error budget violation. If you aren’t familiar with the concept of SLIs, SLOs and error budgets, here is a coffee shop-themed primer:

- **SLI** – You can think of an SLI, or service-level indicator, as an immediate or short-term indicator of service quality (e.g., the current temperature at your family-owned coffee shop)
- **SLO** – An SLO, or service-level objective, is the goal that once reached will make the majority of your customers happy (e.g., the temperature you’ve set on the thermostat in the coffee shop, plus or minus a couple of degrees)
- **Error budget** – How long you can fail to deliver on your optimal temperature promise, (i.e., your SLO before you will start offering your customers free coffee).

Read more about these terms in our blog post, *A Service-Level What?*

As part of your incident lifecycle program, we'd encourage you to define what an incident means to your team or company. But while a single definition across the entire organization is desirable, it might not always be feasible. So when defining what an incident means to you, be sure to consider the following:

- **Severity matters** – You should also consider defining your severity levels. For example, it's common to call a high-severity issue a "Sev1 incident." The actual details or threshold(s) that must be met for any given incident to be considered a high-severity issue—or not—should also be documented.
- **Not all SLO violations have to become incidents** – A system should be able to underperform its SLO for a certain period of time without requiring a coordinated response. We often determine that a system hasn't exhausted its error budget or hasn't done so quickly enough to require an immediate response or declare that there is an incident, for example.
- **Defining error budgets helps** – In the absence of a formally defined SLO, the incident definition tends to be conflated with and driven by alerts and pages. You can prevent the "every minor disruption is an all-hands-on-deck situation" approach by defining an error budget, too.
- **Necessary coordinated responses can get overlooked** – It's possible that an immediate coordinated response had been necessary at some point in the past but was completely missed. In such cases, we'd say that an incident has taken place and a follow-up is likely to be required.



The Life of a Production Incident

Let's consider a simplified timeline of production incident detection and response (this intentionally excludes prevention and analysis):

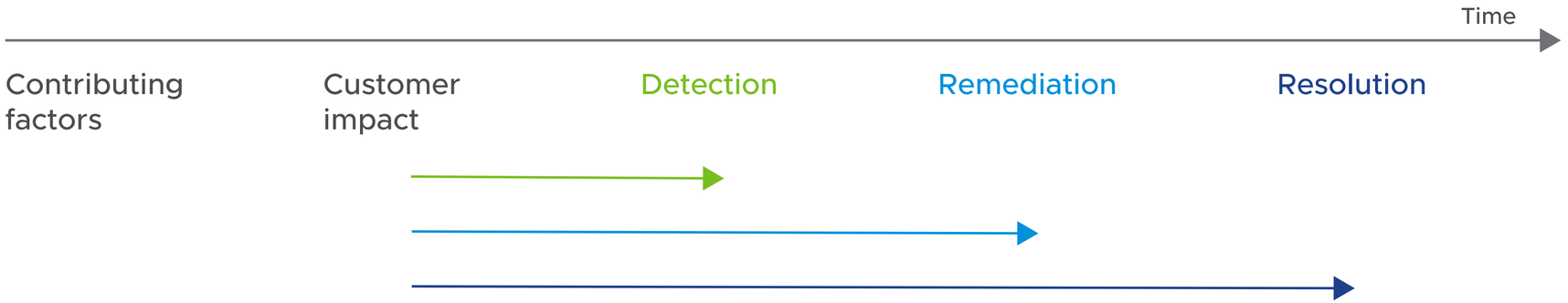


FIGURE 1: A simplified timeline of production incident detection and response.

Contributing factors – As with all things that contribute to a production incident, these might occur at any point in time, not just at the beginning of the timeline.

Customer impact time – In the absence of a formal SLO, this represents a shared understanding that significant customer impact (or customer unhappiness) is underway that will require a coordinated response later on (see "Detection" on the following page). The philosophy "a lack of SLO means your SLIs are de facto SLOs" applies here.

NOTE:

It's common to see a "root cause" or "root causes" being discussed as part of an incident timeline instead of "contributing factors." Here is an argument in defense of root cause analysis using a methodology known as five whys, and here's one contrary to five whys. My stance is that you should form your own opinion, documenting your approach to contributing factors and/or root cause(s) as part of your incident lifecycle program. For example, if you are part of a new company, it might be easier to train folks on "contributing factors" as a term and on an approach other than five whys. If you have a long history of using a structured "root cause analysis" based on five whys, it might be useful to know about different approaches in order to determine contributing factors to a system issue, but it also might not be feasible to ask your entire company to stop referring to them as "root causes."

It's worth noting that immediately alerting upon an SLO violation effectively means that either the error budget for your system is equal to any SLO violation of more than 0 seconds or your SLO burn rate is fairly high at a given moment—or that your SLO burn rate alert needs adjusting. If this isn't clear, you should read *Alerting on SLOs*. It's also worth keeping in mind that the incident detection, even if it is automated, doesn't necessarily occur at the same time as an SLO violation, and that's by design. To that end, it's important to become familiar with the following terms:

Detection – Also known as time to detect (TTD), it represents the time elapsed between the SLO violation and its detection, and can be automated or set manually, (e.g., via a customer report).

Remediation – Also known as time to remediate (TTRemediate), it's usually the time elapsed between an SLO violation and the time a given service is reinstated to either most users, most use cases or a previously established list of strategic customers. It does not necessarily mean that a full resolution has been achieved. And it is often referred to as "mitigation." But in our experience, remediation better communicates that we'd like to achieve more than just a temporary fix, even if it's not yet the full resolution.

Resolution – Also known as time to resolve (TTResolve), it's the time that has elapsed between an SLO violation and the given service being reinstated to all users without temporary solutions. In some cases, the time to resolve and time to restore will be the same.

It's extremely important to establish a common terminology. For example, TTR or MTTR can stand for time to or mean time to either repair or recovery, respectively, depending on the context, and so might cause confusion. Later, we'll cover why direct optimization for reducing time invested in these lifecycle phases needs to be done very carefully.

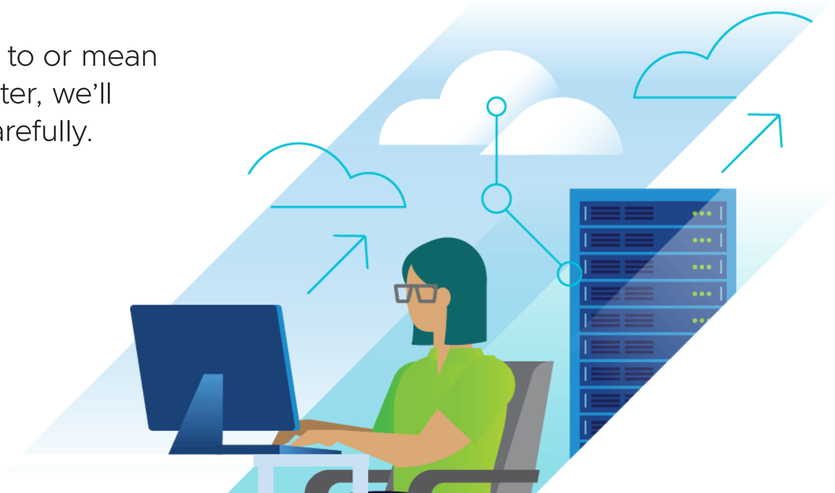


Figure 2 shows a more detailed, but not yet complete and arguably harder to read, clock-based timeline of production incidents, including prevention and analysis (items inside the circle can occur at any time and in parallel to other events).

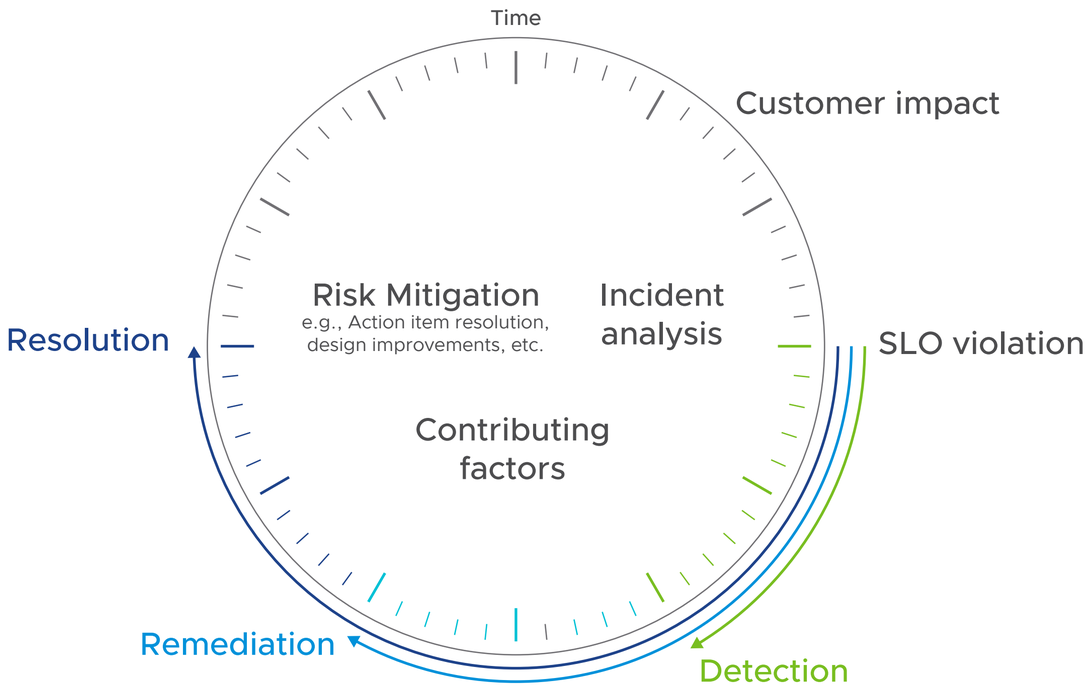


FIGURE 2: A clock-based timeline of production incidents, including prevention and analysis.

As you might have guessed, we'd recommend defining your own timeline to establish a common terminology. For example, your company or team might have a formally defined escalation process, from your technical support teams to your operators, and it might look quite different from the timeline presented above.

At this point, your incident lifecycle program will contain the following items:

- Production incident and severity definitions
- A production incident timeline definition

Defining Your Approach to Incident Prevention

Regardless of your incident definition, I believe that attempting to prevent all incidents, forever, is an extremely costly endeavour. I strongly encourage you to approach the prevention space from complementary points of view, acknowledging that:

- some types of incidents can be eliminated.
- some other types of incidents can have reduced impact.

To start, I recommend performing a reliability risk assessment, which can be done before establishing any SLIs, SLOs or error budgets.

A reliability risk assessment is a way to assess risks and their likely impact on the reliability of your systems, accompanied by a prioritized list of mitigations to be implemented. Our peers from Google CRE have published the following article and spreadsheet, both of which might be useful to you.

The order of priority will vary based on the assessment performed above. But while evaluating known risks, the following aspects are commonly considered (in no particular order):

- Configuration changes and binary deployments
- Traffic management
  - Cache
  - Retry logic
  - Graceful degradation
- Data management
- Dependency management
- Capacity / saturation / performance

Note: Security risks, as a threat to reliability, can be assessed and prioritized in a similar manner. In practice, they are generally performed concurrently by a team of security specialists.

Once a reliability risk assessment is complete, the next step is to prioritize and staff fix projects. This is where having an SLO and error budget will help, as they'll assist you with knowing when it's OK to focus on taking additional risks (e.g., increase feature velocity) as opposed to investing in reliability in an unbound manner.

It's commonly understood there will be unknown unmitigated risks to the reliability of your systems. I believe it is still worth improving upon the most impactful risks that are known.

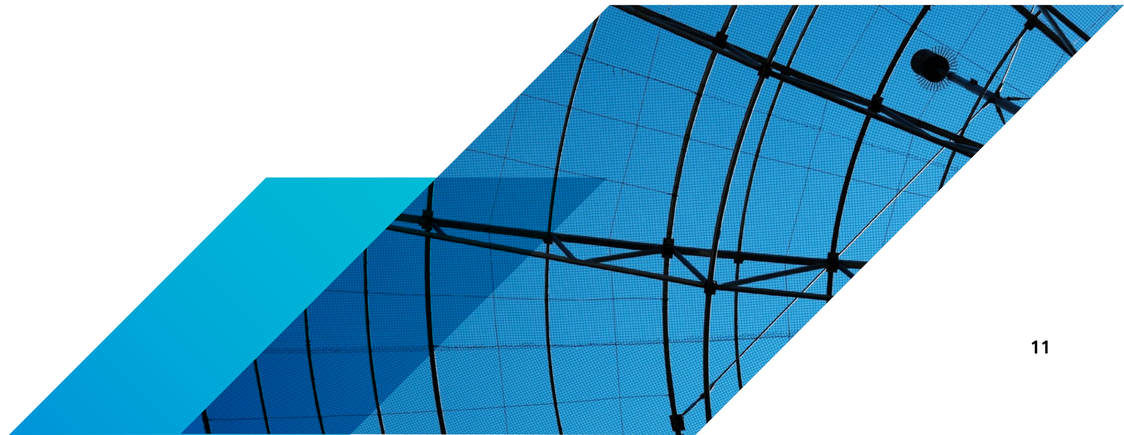
In order to avoid obvious regressions, the best tool to have in the incident prevention toolkit within your incident lifecycle program is chaos engineering. Despite what the name might imply, this is a practice to *prevent* chaos.

At this point, your incident lifecycle program will contain the following items:

- Production incident and severity definitions
- A production incident timeline definition
- A documented approach to incident prevention, including
  - a periodic reliability risk assessment
  - fix projects based on the output of the risk assessments, which are prioritized based on the SLOs
  - chaos engineering tests to catch reliability regressions

FURTHER READING:

- *Taming chaos: Preparing for your next incident*
- *Shrinking the impact of production incidents using SRE principles*
- *Podcast: Chaos Engineering, Explained*



A High-Level Overview of Incident Detection

The detection component of your incident lifecycle program should include both your approach to observability as well as any other way a production incident can be detected or declared.

In some cases, your monitoring might be telling you that your system works, but that some—or many—of your customers are unhappy. This is not uncommon. If your company or team has been handling incidents for a while, it's likely that some approach to customer escalations, including when to treat a customer escalation the same as an issue involving several or all of your customers, is already in place. If not, I recommend establishing that criteria sooner rather than later. The absence of such a process tends to cause confusion and frustration when your monitoring systems show that everything is fine but your customers and support teams are telling you things are in fact not functioning properly.

The threshold between a regular support issue and a production incident requiring a coordinated response will be arbitrary unless you define it. In the same vein, the most appropriate way to detect incidents will depend on how your organization defines them along with error budget(s), staffing level, etc.

It's fairly common for teams to try and detect a system failure by alerting the on-call person. But premature detection, which fails to take into account the degree to which customers are unhappy (or not) and any self-healing that can be done (e.g., by way of properly implemented retries or autoscaling), causes burnout. And unnecessary alerts pose a major risk to the teams that operate production, since it makes their job—to keep customers happy while also reducing operational load—significantly more difficult.

That said, I'm not recommending that you delete your alerts and start from scratch. I suggest you do the following instead:

- **Invest in aligning your alerts with signs of customer unhappiness** – In other words, define your error budgets and set up additional, burn rate-based alerting. And be prepared to detect bad events that will wipe out any error budget almost instantly (e.g., quick resource exhaustion).
- **Refrain from deleting existing dashboards and only remove old alerts once the new ones have been tweaked** – You'll know they are working well when both the number of false positives and false negatives are low.
- **Establish an escalation path from customers or technical support** – Only establish such a path if it's applicable, and if so, be sure to test it.
- **Dedicate engineering time for prevention** – Enable applications and platforms to both recover themselves from more failure modes and to degrade gracefully, as partial failures are better than complete failures.

FURTHER READING:

- *Observability 101: Terminology and Concepts*
- *Tune up your SLI metrics*
- *SLO Alerting with Wavefront*

At this point, your incident lifecycle program will contain the following items:

- Production incident and severity definitions
- A production incident timeline definition
- A documented approach to incident prevention, including
  - a periodic reliability risk assessment
  - fix projects based on the output of the risk assessments prioritized based on the SLOs
  - chaos engineering tests to catch reliability regressions
- A documented approach to incident detection, which
  - might consider monitoring misses (i.e., escalation from technical support or your customers)
  - has to consider what you currently have in terms of alerts while making sure it's aligned with your customer experience

How to Establish an Incident Response

The response component of your incident lifecycle program should include your approach for responding to an incident (including remediation and resolution), but also a continuous learning program so your team's experience doesn't get rusty.

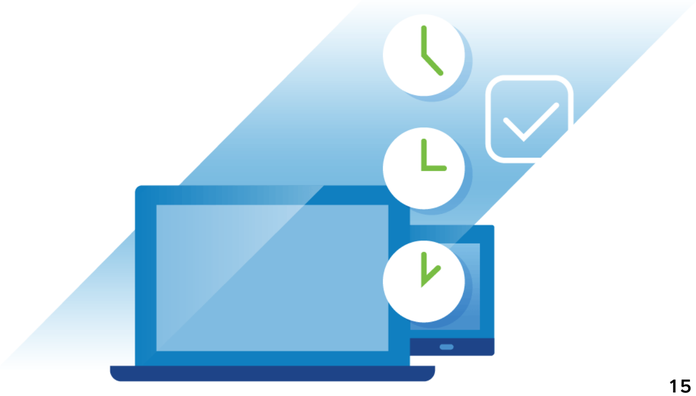
The most common approaches to incident response protocols I've seen are as follows:

- Solo responder
- ICS/FEMA- or IMAG(Google)-based

Unlike some other components of your incident lifecycle program, the pros and cons of these approaches are somewhat more clear.

Solo responder:

- Pros
  - Staff find a single point of contact straightforward. A single person is on call and nobody else will be expecting to be interrupted to assist with the response to an incident.
  - If the responder is familiar with the system, not a lot of additional training is required,
- Cons
  - The responder might struggle to operate the system while thinking about all the things that need to be done to remediate the issue.
  - Communication with stakeholders as the incident unfolds might become a problem. Customers or partners might not be clear as to whether or not anything is being done to correct the issue, including if there's been an estimate of the impact and/or necessary recovery time, or if there's anything they can do on their side to work around the problem.



ICS/FEMA/IMAG-based:

- Pros
  - There is a lot of bandwidth involved, which means incidents can be remediated quickly. While these usually start with a solo responder, more people can be summoned to perform different roles (e.g., incident lead, operations lead and communications lead).
  - Communication with stakeholders can be done in a timely manner.
- Cons
  - Regardless of a team' geographical distribution (i.e., in a single time zone or over multiple time zones), depending on your team/company size, staff might not find the process straightforward. If your solo responder approach used to rely on a single person on call, there will now effectively be multiple people on call (even if it's just informally).
  - To be done effectively, training is required so that folks can better understand the specialized roles they will be playing and how those roles might vary from one incident to another.

Regardless of your incident response protocol and staffing, I cannot overstate how important it is to continuously train folks in real-world situations.

A successful incident response program will include the following aspects:

- Healthy, rested and well-trained incident responders
- Proper tooling that enables incident responders to assess a system's current state and identify what's changed in order to remediate production incidents

Some readers might notice that I have left runbooks off the list. While the list is not exhaustive, I also believe that many teams place unnecessary emphasis on runbooks (or playbooks). If one or more runbook entries can be followed without much human judgment involved, please consider automating them instead.

While I don't want to discourage all uses of runbooks, I do want to encourage you to question whether some of your runbook entries are acting as a bridge until you have more time to prioritize automation.

At this point, your incident lifecycle program will contain

- Production incident and severity definitions
- A production incident timeline definition
- A documented approach to incident prevention, including
  - a periodic reliability risk assessment
  - fix projects based on the output of the risk assessments prioritized based on the SLOs
  - chaos engineering tests to catch reliability regressions
- A documented approach to incident detection, which
  - might consider monitoring misses (i.e., escalation from technical support or your customers)
  - has to consider what you currently have in terms of alerts while making sure it's aligned with your customer experience
- A documented approach to incident response that
  - answers the solo responder or ICS/FEMA/IMAG-based question
  - includes your training program
  - features a plan to keep incident responders healthy and rested before/during/after issues
  - ensures incident responders have the proper tooling to assess and mitigate outages

FURTHER READING:

- *Shrinking the time to mitigate production incidents*

Scaling Up Your Incident Analysis

The analysis component of your incident lifecycle program shouldn't be an afterthought. It also involves more than writing a postmortem or surveying your individual team members so as to document what happened and any lessons learned.

That said, I understand that several companies—and teams within companies—opt to track and optimize their incident lifecycles around the following aspects:

- Decreasing the volume of all incidents or those that are most severe
- Decreasing the mean or median time to detect, mitigate or resolve incidents
- Increasing the time between failures

While I recommend tracking all of these factors, I remain skeptical that direct optimization attempts will yield the best possible results. Since effectively tracking these numbers involves a wide range of unrelated incidents and a fair amount of human judgment, it is possible that the results will be too noisy and biased, especially if folks feel threatened by them.

I also understand that in certain regulated environments, tracking these factors and directly optimizing for them might be a requirement. With that in mind, when selecting the best approach for your team and your company, I recommend evaluating the following metrics:

- **Percentage of action items resolved** – Resolution here includes closing out any items that were brought up during discussions about an incident but weren't and won't be worked on.
- **Number and estimated impact of incidents prevented** – These are incidents that would have taken place if not for the broad fix projects implemented by your team or company. For example, if a canary analysis system was put in place, the number of times the canary caught a bad deployment to production might be a good metric to highlight. This concrete data motivates the people working on incident prevention by making clear their impact and, by extension, extends the time between failures.

Yet again, the best approach to take depends on your company's culture, and possibly the set of regulations to which it is subjected. One way or another, I recommend so-called blameless postmortems or retrospectives. The idea is to avoid assigning blame to humans and instead focus on systems and process improvements. Indeed, all systems—both human and non-human—will eventually fail.

There's also been a growing number of companies moving from the somewhat morbid terminology of "postmortems" to "retrospectives" instead. Such retrospectives can also be done jointly or separately (e.g., by way of individual surveys) to avoid biases.

I'm often asked to recommend the best postmortem template or what a great incident survey looks like. Unfortunately, there are no individual examples I can recommend for all use cases. However, the best ones I have seen meet the following criteria:

- They are both human writable/readable and machine readable
- The fields/questions are reviewed with a certain frequency and removals are as common, if not more common, than additions

FURTHER READING:

- *Improving Incident Management and Postmortem Analysis at Google*
- *Incident Metrics in SRE*

At this point, your incident lifecycle program will contain the following items:

- Production incident and severity definitions
- A production incident timeline definition
- A documented approach to incident prevention, including
  - a periodic reliability risk assessment
  - fix projects based on the output of the risk assessments prioritized based on the SLOs
  - chaos engineering tests to catch reliability regressions
- A documented approach to incident detection, which
  - might consider monitoring misses (i.e., escalation from technical support or your customers)
  - has to consider what you currently have in terms of alerts while making sure it's aligned with your customer experience
- A documented approach to incident response that
  - answers the solo responder or ICS/FEMA/IMAG-based question
  - includes your training program
  - features a plan to keep incident responders healthy and rested before/during/after issues
  - ensures incident responders have the proper tooling to assess and mitigate outages
- A documented approach to incident analysis that includes
  - established metrics and metadata that are worth collecting, a subset of which is worth optimizing for
  - retrospective/postmortems/incident surveys that are both human- and machine-readable
  - an established or planned feedback loop to help prioritize your prevention projects

Establishing Your Incident Lifecycle Program

Now that you've learned how to establish your incident lifecycle program, you might be wondering: What should I do first?

Make the process your own. Start small, survey your team and your company for the procedures, templates and software already in use for each of the phases I have covered here. You might end up defining more phases. For example, your company might consider incident debugging as a phase all its own.

You should then form an opinion as to the most important component(s) in need of change. A common choice is to either start by normalizing definitions (incident and timeline) to improve incident response or detection. This is the point when folks will often realize they don't have good data, so analysis becomes important.

I recommend following any definition normalization exercise with improvements to your incident analysis and letting your preferences and agreements in this realm drive change in other areas. A simple analysis might not be easy in the beginning because you won't have all the data you want in a format that can be analyzed. A common pitfall is to then optimize for the metrics you have (e.g., total number of incidents) as opposed to the ones that would work best for your company and your customers (e.g., customer satisfaction with your current volume and impact of incidents, number of incidents prevented, percentage of action items completed, etc.).

If you can agree on the metrics that would work best for your company and your customers but don't have them yet, it's worth investing in tracking them down before changing your approach elsewhere. In the meantime, I've also seen folks succeed by *investing in team well-being* and incident prevention first.

This book is dedicated to Elomar Fernandes Franco

Acknowledgments

Thanks to Alexandra McCoy, Corey Innis, Curt Micol, James Wynne, Jennifer Krazit and Kalai Wei for their contributions to this book.

Also thanks to Kripa Krishnan, Jelena Oertel, Nile Geisinger, Sue Lueder, Patrick Bernier, Rhan Singh, Sergio Salvi, Aaron Racine, Venkat Patnala, John Reese, Vivek Rau and all the SREs I had the opportunity to work with for the countless hours they spent practicing and discussing what's summarized in here.

And a special thanks to Andrew Hurst, Megan Bigelow and the VMware Tanzu team for supporting this work.

Start Learning More

If you have reached the point in your incident lifecycle program when it's time to consider improvements to incident detection, learn more about *VMware Tanzu Observability*, which supports SLO-based alerting.

Now that you understand a bit more about the life of an incident and why you should care, you might be wondering how to begin putting this knowledge to use. The VMware Customer Reliability Engineering team facilitates *workshops to introduce SRE concepts and practices* to our customers interested in learning about reliability engineering practices. We also offer consulting services through *VMware Tanzu Labs* to help our customers align their engineering practices with broader business goals and learn to apply reliability engineering and other modern software practices through collaborative, hands-on work.

Learn more about the VMware Tanzu portfolio of products and services at *tanzu.vmware.com* or reach out to your sales representative.

Join us online:

vmware®

VMware, Inc. 3401 Hillview Avenue Palo Alto CA 94304 USA Tel 877-486-9273 Fax 650-427-5001 vmware.com