VictorOps

BUILD THE RESILIENT FUTURE FASTER

Creating a culture of reliability

Site Reliability Engineering @ VictorOps

By Jason Hand

Build The Resilient Future Faster

Creating a culture of Reliability

Site Reliability Engineering @ VictorOps

By Jason Hand

Build The Resilient Future Faster
By Jason Hand

Printed in the United States of America

Published by VictorOps, Inc. 1401 Pearl St., #300 Boulder, CO 80302

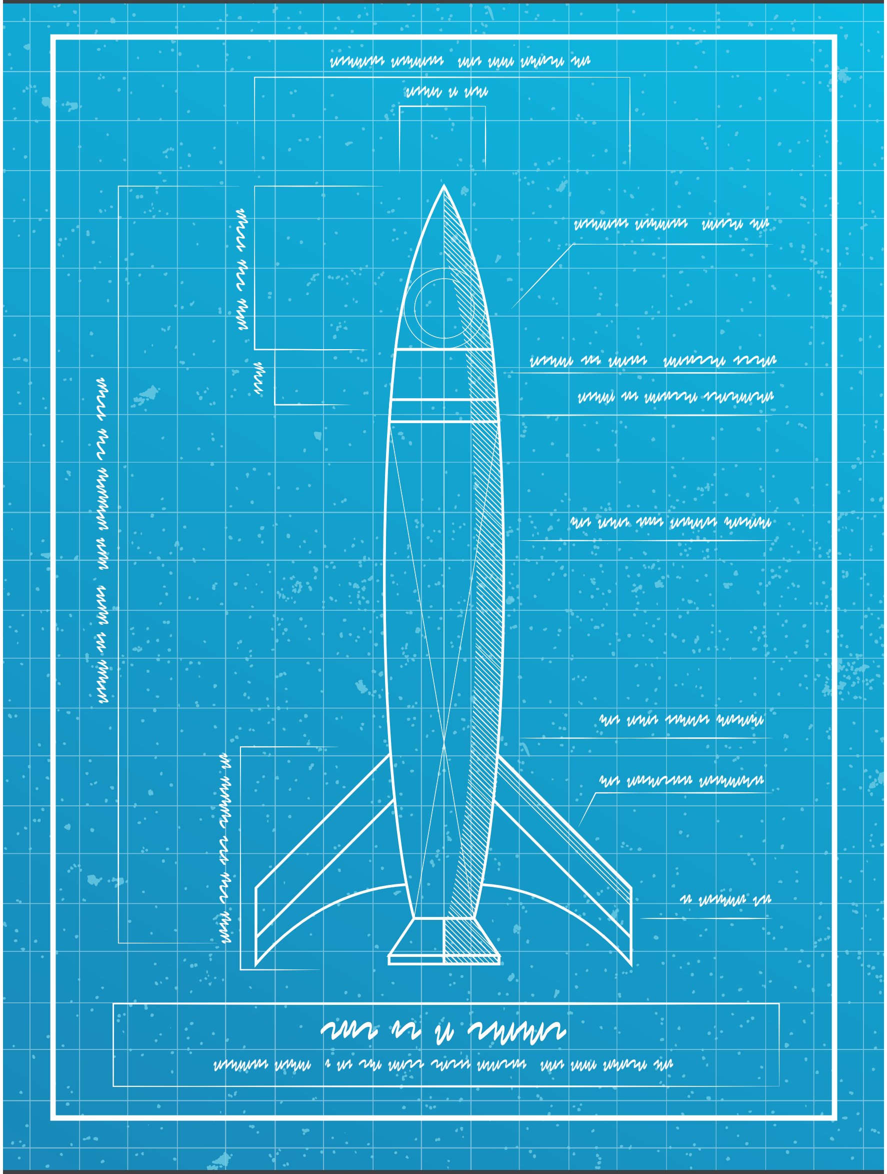
Online edition also available at
https://victorops.com/build-the-resilient-future-faster/

Editor: Kelsey Loughman
Copy Editor: Dan Holoran
Cover and Interior Designer: Rachel Stern

Special Thanks to:
Jonathan Schwietert
Brandon Barrett
Maggie Gourlay
Garrett Dawson
Eric Wagner
Andrew Fager
DeAndre Carroll
Sean Grady
Greg Frank
Alex Wellock
Sean McBride
Dan Jones
Dan Hopkins
Tom Hart
Jerry Pommer
Beau Christensen
Sean Grady
Mike Merideth
Benton Rochester
Jason Roth
Brandon Zarrella
John Egan
Summer Wollin
Courtney Kissler
John Allspaw
David Blank-Edelman
J. Paul Reed
Baron Schwartz
Charity Majors
Cindy Sridharan

Table of Contents

PART ONE



Expectations and Dependencies

Expectations and Dependencies of Modern Society

For most of us, our day begins with a few routine tasks. We turn on the lights, brew a pot of coffee, and warm the shower—a common series of events for anyone's morning.

As we move from task to task, there are expectations. We expect the lights to turn on when we flip a switch. We are confident turning the appropriate knob will cause warm water to come from the shower head. In fact, we expect the same experience each and every time. Consistently. Reliably.

Rarely do we stop to question or understand how these services are delivered to us on demand. Admittedly, there is very little reason to give much thought to the underlying complexity that delivers on-demand electricity, gas, water, or even the internet. These services "just work," and we rely on them to accomplish goals in other aspects of our lives. Simply put, these services enable us to achieve expected outcomes; we come to expect and depend on them.

But it's not just the electricity and hot water we have come to rely on. In fact, availability and accessibility are of equal importance, playing a significant role in assessing reliability and quality of service.

When we flip a switch, light doesn't just emerge from nowhere. It came from a provider and is delivered through a complex network of components, each with their own opportunity to fail. We rely on both the end-result service (i.e., electricity) as well as the way in which it is made available to us.

Digital services work in a similar way. Because the same provider builds and operates the application and infrastructure, functionality and reliability are part of the same value proposition. Energy companies can produce electricity, but if users can't access the service (electricity or gas), the value of the energy and business is diminished.

In most parts of the world, an immense and growing spectrum of digital services and technology influences nearly every aspect of our lives. To some degree, each of us rely on digital resources provided by businesses, governments, individual contributors, and more—and that reliance is steadily increasing.

Let's take travel as an example. For any given business trip, I personally rely on a variety of digital services. My airline alerts me to check in for my flight through the mobile app. I hail a ride-share to the airport using both my phone and another mobile application. The driver takes me to the airport in the most efficient way thanks to GPS and live-traffic updates. I scan my boarding pass at the TSA podium using my watch. While I wait to board my flight, I listen to a podcast streaming from my tablet. Suddenly, just as we are about to push back from the gate, I remember my mortgage payment is due this week, so I execute the transaction from my banking app.

It's part of the "Digital Transformation" changing the way we interact with the world around us. There are certain expectations the end user has regarding reliability. Accordingly, organizations are evolving in order to hold up their end of the agreement. The transformation is changing the way we set our expectations around quality and reliability of these digital services.

Access to this growing digital functionality and information is expected to be available and operating at all times. Much like the pipes that deliver water to our homes, the complex inner-workings of delivering a service to the end user is both critically important to the overall quality (read: reliability) of the service, yet intentionally abstracted away from the end user, and to some degree, the organization providing the service. We encounter layers upon layers of abstractions meant to simplify everything, including software, keeping it healthy, and getting it to the person who needs it.

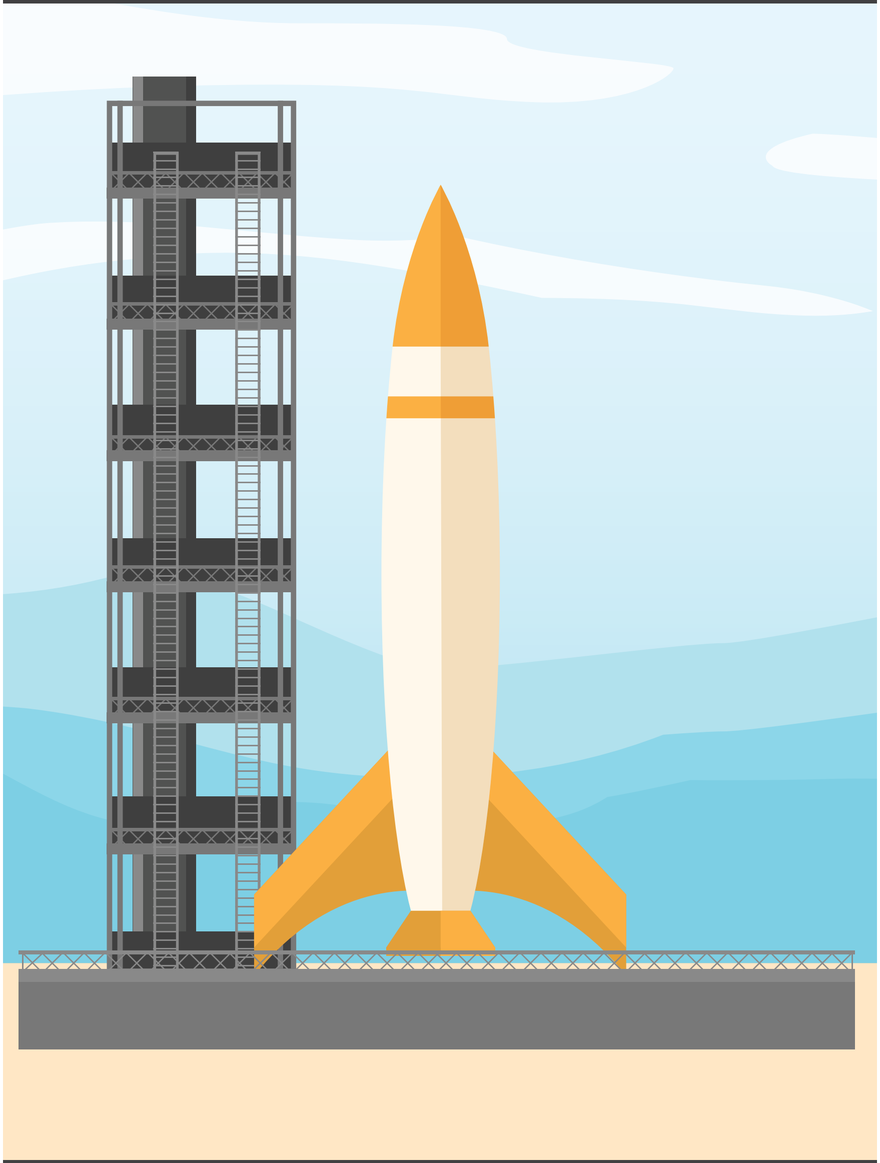
With the shift towards always-available digital services, the need to provide reliable and improved access to these services has increased. As a result, innovative engineering practices across nearly every industry have emerged to meet the modern world’s demand for access to digital services whenever, wherever, and however they want.

In fact, this “reliability” expectation has given birth to services and tools, VictorOps included, while also supporting entirely new approaches to building, operating, and iterating on software and infrastructure.

VictorOps, much like a utility company, provides services to enable its end users. Specifically, we empower the makers of the world to build resilient systems.

This is accomplished by designing, building, operating, and improving the VictorOps software as a service including the underlying physical and virtual infrastructure. Our constant pursuit is to explore new methods for delivering both quality software sooner and receiving faster feedback from real usage. Continuing to develop better methods of delivering software as a service to meet the changing needs of our user base is at the heart of our own journey into site reliability engineering, or “SRE.”

PART TWO



The VictorOps SRE Journey

Building A *Resilient* Future Faster

Trust is the foundation upon which we reach just a little higher and stretch a little further. Without trust, there are no risks taken, which means no exploration, experimentation, or advancement of the system (or society for that matter).

The advancement of the VictorOps service is largely based on trust. Trust and confidence in the process of building, deploying, and operating software and services. Trust in the development process. Trust the way in which software and services are deployed to customer-facing production environments. Trust that even when something goes wrong, we can recover extraordinarily quickly.

We must constantly explore new ways to maximize and meet expectations on reliability while simultaneously innovating and improving our service. We are a data-driven rocket ship, constantly swapping out components, experimenting with processes and tools. Iteratively learning and exploring more about the system's "knowable" properties—all while the ship is in flight.

How We've Been Doing It

The status quo for building and operating systems has long been for developers to hand off code to release engineers or operations teams to deploy and manage. Monitoring and alerting were afterthoughts, only bolted-on in the Production environment, if at all. Operations engineers and system administrators were paged for problems at any time, day or night. Taking a reactionary approach when it comes to reliability no longer met our needs.

There are more modern methods and approaches to increasing reliability available today that are better suited to how software and infrastructure are designed, built, and operated. Changing our posture from reactionary to proactive was the first thing we needed to change.

In April of 2017, VictorOps kicked off official SRE exploration and documentation of our internal efforts and discussions regarding both reliability and scalability. Our documentation of this process would serve both as historical records for VictorOps as well as a resource to customers, prospects, and the greater IT community. For VictorOps, SRE and the associated efforts are ongoing. This text includes four key assignments or exercises that helped VictorOps establish footing and move forward with confidence on our own journey towards building a highly available, resilient, and reliable system and service that is constantly improving and bringing more value to end users.

Assignments:

- *Identify "What Keeps You Up At Night?"*
- *Determine Value to Effort for Observing What Keeps You Up At Night*
- *Establish Blackbox Metrics and Service Level Expectations*
- *Make the Case For Chaos or a Game Day Exercies*

What is SRE?

In many organizations, Site Reliability Engineering (SRE) is the responsibility of very specific teams or individuals, typically those familiar with operations-like engineering efforts. They keep the critical infrastructure and applications up and running. Think of them as the keepers of "Production". System Administrators. The IT Operations, SRE Team, or individual engineers (i.e. SRE's) typically own this responsibility. In some cases, individual reliability engineers are embedded with development teams, while in other cases, there's a central SRE team.

However, an increasingly common approach to engineering where roles such as development, operations, quality, security, and others are combined into small, loosely coupled, yet highly collaborative teams have empowered organizations to respond to problems much faster when they inevitably arise. Perhaps more importantly, these collaborative teams are able to deliver value (in the form of digital services) to the end user much more quickly.

Terms such as DevOps have emerged to give a name to organizational efforts to bring disparate conversations around building, deploying, and operating applications and infrastructure into the same group. Previously siloed conversations about responsibilities slowed the process of delivering value as teams were essentially incentivized against each other. Developers were encouraged to pump out new functionality while the operations teams were incentivized on maximizing the availability of the resources (i.e., uptime). Without realizing it, competing efforts were in action to both introduce and limit the one common cause of IT failure: change. Conflicting incentive structures is a classic flaw in the makeup of many IT organizations.

As a company, VictorOps has an inherent passion for reliability. Founded by software builders and systems architects who deeply relate to those who are tasked with the pressures of maintaining uptime of systems, a culture of high availability has always been strong within the organization. It’s ingrained in the majority of our work and what we think about each day.

Engineering teams and IT professionals around the world rely on us to alert and assist in the mitigation of disruptions to services critical to the business. As our CTO puts it:

> **“Reliability is our most important feature”.**
> ***-- Dan Jones - CTO VictorOps***

If we are experiencing a problem impacting our service, the issue creates a ripple effect, impacting our customers, and our customers’ customers, and so on.

The value we, as a business, deliver is not only in the rapidly improving service itself (on-call and incident management) but also the ability to rely on services to work as expected when customers need it most—during their own high-stress service disruptions.

What Do We Mean by Reliability?

Protecting the VictorOps customer experience AND increasing our ability to deliver value more quickly is ultimately what we are attempting to tackle as a company-wide SRE effort. Still, associated responsibilities and expectations around our SRE efforts need to be specific about which problems we are trying to own and solve.

First, we began our efforts by defining and focusing on two primary areas tied to the customer experience aspect of reliability:

Correctness and Availability

Correctness:

- *Functions as expected*
- *Data is consistent*
- *Consistent, predictable performance*
- *Consistent innovation*

Availability:

- *Always on (24/7/365)*
- *Minimal downtime (planned or unplanned)*
- *Resilient to failure / fails gracefully*
- *Global accessibility*

The relationship between correctness and availability demands a balanced approach. Like efficiency and thoroughness, each can often carry incentive structures, which are often at constant odds with each other.

For most modern organizations, velocity is more than a “nice to have.” Halting functionality work in order to focus engineering resources towards improving only the reliability of a service doesn’t usually sit well with product owners and management. We need to achieve a balance between reliability and deployment speed.

Reliability from a Customer Perspective

VictorOps customers depend on us when there is an active problem within their own system. Their experience with the VictorOps service as they acknowledge, triage, collaborate, and resolve issues is far more important than whether or not the VictorOps core servers are experiencing high levels of CPU usage. Is VictorOps empowering them to do their best work?

Metrics such as CPU and memory usage are important to have observability around but do little to communicate the experience from the customer’s point of view. Users don’t give a damn if we have our own datacenter, a multi-cloud architecture, or a couple of hamsters on a wheel plugged into a Raspberry Pi. They do give a damn about fixing their own broken application or service. VictorOps enables them to resolve service disruptions as well as retrospectively analyze incident response efforts for deeper learning. They rely on us to enable them to solve their own problems. Plain and simple.

Here’s a real question…

> *“What is the user experience while interacting with VictorOps during an active incident?”*

This is an observability question. This is where we need the highest fidelity data if we want to accurately answer it.

More specifically, what happens (exactly) when:

- *Someone interacts with the software we've built,*
- *Running on the infrastructure we've architected, and*
- *Delivered through the pipelines we currently have in place,*
- *Using processes and tooling that have been established over the 6-year life of the service...*
- *During an active incident?*

Do we know? Is it possible to find out? Is it "knowable"?

Some engineers have intimate knowledge around parts of the system. Others haven't been with the company long enough to share the same mental representation of how the system actually works. What data needs to be collected in order to begin attempting to answer the questions above?

Scalability from a Customer Perspective

Consistent operability isn't quite enough to satisfy today's end users. The tech world moves fast. When was the last time you installed software from a disc and could operate it without a connection to the internet? That's rarely how things work today. We access services from our phones or devices whenever and wherever it's convenient for us. And because access to the best and most innovative software is only a "Sign up Now" button away, vendor lock-in isn't quite as prominent as it once was. That's great news for consumers and end users. It's a bit more worrisome for companies realizing that functionality and differentiating features quickly become commodities and the only real chance at differentiating yourself in the market is by outpacing the competition on feature releases and displaying dominance in reliable infrastructure.

Customers will educate themselves and choose the service that is, of course, reliable, but they will also pay close attention to the manner in which the service itself adapts to their own changing needs.

How innovative is the service? The answer reflects how empathetic the vendor is to the always changing landscape of IT.

Scalability is of great concern to end users whether they explicitly make the claim or not. It is directly related to the overall reliability of a service. You must demonstrate the ability to keep up with and support them. If you show the inability to enable them to succeed as things become more complex and mission-critical, the end user will begin the search to find a more suitable partner to explore the future of software.

We need to optimize for delivering improvements to our service safer and faster. Our users expect that the tools they leverage today will grow with them into the foreseeable future. They expect to influence and shape the roadmap of the service by providing feedback to welcome and eager product teams. We must be able to introduce changes to our systems based on feedback from the customer's experience. Finding ways to improve our ability to scale was important enough for us to call it out for the problem we are own and solve.

Our journey towards curating a specific culture of reliability is an ongoing one. But what we've learned and where we are headed all started with asking questions. Throughout this text, I'll share with you what those questions were, what kinds of conversations they generated, and what new questions and discussions that led to. The final sections of this text will conclude with the very first VictorOps Chaos Day orchestrated under SRE. We will use chaos engineering to learn how our system handles failure, then incorporate that information into future development.

Embracing risk is a big part of the cultural change we are trying to bring about not only in ourselves but in the rest of the tech community. It's one thing to say we embrace it, we need to mean it, as well as demonstrate the critical relationship between this new embracing of risk and its positive impact on the reliability and scalability of the VictorOps service.

There are no clear “best practices” to SRE. There is no official playbook. Like DevOps, there is no one-size fits all approach to Site Reliability Engineering. What works for a company like Google or Facebook doesn’t make sense for us. What works for VictorOps likely won’t plug and play into your organization without some adjustments.

Early Decisions

Very early, we evaluated two of the more popular approaches to SRE: embedded and dedicated. After many conversations internally as well as through interviews with reliability engineers from Twitter, Netflix, Github, and others, we made the decision to resist the tendency of hiring into the role of SRE. Likewise, we wanted to avoid unintentionally creating a new silo by forming an “SRE team.”

Worried that a specific team might induce assumptions about who owned our availability, we concluded that our approach to SRE was not limited to a distinctive team. From our perspective, the responsibility of building reliable systems is taken away from a majority of the engineering team almost entirely when following the dedicated model (i.e., a distinct site reliability engineer or team). We also weren’t in love with the embedded model as that carried the same problem. It might be a larger team with more context but we knew we wanted reliability and scalability to fall on the shoulders of everyone.

We are building a culture of reliability.

Much of what we wanted to accomplish was going to require a shift in the mindset of what we care about and how we accomplish goals associated with that care. We wanted to communicate explicitly that SRE was not a project. It’s not an initiative we will take on for a few months until we have achieved some empirically measured goal, such as 99.99% of availability. This initiative ought to align with a cultural change in not only our engineering team but also the entire company—a change to align the company with the objectives of the business and the needs of the customer.

A growth mindset with a hunger for continuous improvement is part of the company culture that is often hard to build and sustain. Something like this doesn't just emerge out of nowhere. It requires a change agent: a champion to challenge the status quo (i.e., how we do things around here).

Who Will Champion This?

We needed buy-in from management, from the Product team, as well as from all corners of the engineering team. We needed everyone to have a clear sense of responsibility and control over their role in our SRE efforts. We also knew that someone needed to champion this effort. Without a champion, it would be too easy for our SRE aspirations to get lost in the day to day business.

We chose to look internally for an individual to lead our efforts and create a company-wide focus. Someone who would serve as a coach to our entire engineering team, supporting and enabling them to embrace and own reliability in each of their own domains.

One platform engineer stepped forward and offered to assume this role. Much of their work on the "Platfrastructure" team (Platform & Infrastructure) was tied to these concerns already. Likewise, they were becoming increasingly more curious about the principles of DevOps and our own ability to get new functionality to users while also maintaining a hardened system. It was a natural fit.

For SRE to succeed, our engineers needed to see and feel that the value of their engagement was valuable. Above all, we wanted to know the truth about our systems, including the human components. As such, we valued transparency and feedback in pursuit of genuine inquiry and continued learning; we saw (and see) this as a hunger to expose more and develop a greater sense of the system (including the people).

This new hunger led teams across the entire organization to begin talking about a common challenge. A challenge of increasing velocity and maximizing uptime, which, when reframed from reactive to proactive, now seems a whole lot more interesting. A whole lot more like an engineering problem that, with the support from the rest of the company, we can prepare for trouble (i.e. unplanned work) by engineering ways to shorten feedback loops and expedite the remediation of service disruptions. That's something everyone from upper management to technical support can get behind. We'll all play a role in solving for it.

Start With Questions

> **"Monitoring tells you whether a system is working, observability lets you ask why it isn't working."**
> *- Baron Schwartz, CEO VividCortex*

Asking questions was the most important step early on for us, and in a really generic sense, observability is just that—asking and answering questions, any question. Filling in the blanks on what is known, or even knowable about our systems. If someone has a question about any aspect of our system, we want to be able to get an answer we feel confident about. Because we are going to make some really important decisions based off of those understandings of reality.

In order to reliably answer questions, you need to have access to information. Not only that, but you have to be able to make sense of it. No matter what question you have about your system, you should be able to answer it. It's about moving closer and closer to a clearer understanding of the reality of our systems. To us, this is what we mean when we speak to the topic of observability. We can't improve what we can't measure. We can't measure what we don't see. And, we'll never even know what to look for if we don't know what is important. What's important to VictorOps can help us shape what SRE is to VictorOps.

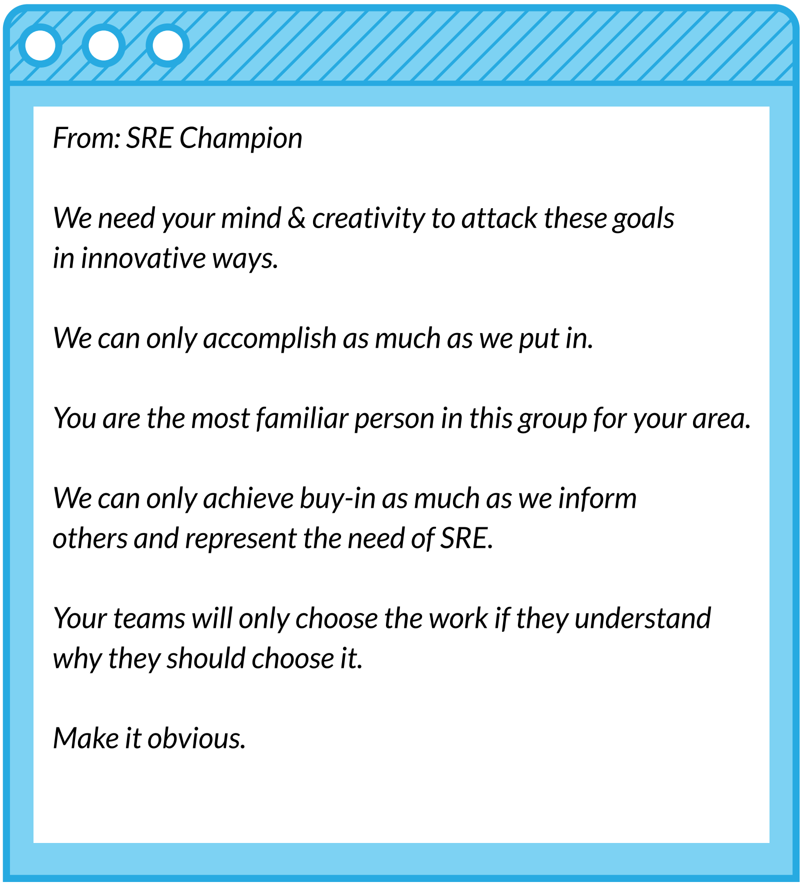
> *There are many great explanations on what observability is and is not. I suggest reading anything from Charity Majors, Baron Schwartz, Cindy Sridharan, or Jonathan Schwietert on the subject. Each has a deep understanding that goes beyond the scope of this book but is still super important. I definitely recommend giving their work a read.*

What is SRE to VictorOps?

For VictorOps, the SRE mentality would need to be central to the culture of our entire organization. The responsibility of owning the scalability and reliability of the product (VictorOps) from a customer experience point of view doesn't rest solely on an SRE team or individual engineer. Rather than assigning the SRE role and responsibility to a specific team or individual, we chose to assemble a cross-functional panel of engineers, support leads, and product representatives referred to as the SRE council.

The SRE Council

The council would be made up of at least one representative from each of the core teams with an immediate stake in reliability and scalability (i.e web client, mobile client, platform, QA, IT Operations, etc.). Our SRE Champion would facilitate discussions during scheduled meetings and serve as the main point of contact for SRE outside of council gatherings. Continuous improvements to reliability in the customer experience will continue to advance, as would scaling the speed and confidence of deployments. But, how can we sum this up into specifics? We were able to get buy-in from management on SRE efforts by communicating that we are most focused and empathetic toward the end user's experience. From the end users perspective, SRE would create a unified vision, mission, responsibility, and goal for the continued reliability of our product (i.e. VictorOps). The quality of our service when it comes to reliability must always be examined with true care towards the expectations of the end user. Empathy is necessary.


From: SRE Champion

We need your mind & creativity to attack these goals in innovative ways.

We can only accomplish as much as we put in.

You are the most familiar person in this group for your area.

We can only achieve buy-in as much as we inform others and represent the need of SRE.

Your teams will only choose the work if they understand why they should choose it.

Make it obvious.

Culture

You've no doubt heard many times that changes like these aren't accomplished solely with adjustments to tooling and process. And they definitely aren't accomplished by hiring an individual or even an entire team to "implement SRE" (or DevOps). There must be a cultural shift of some kind within the company.

In order to move quickly and in unison we must all maintain a sense of empowerment and freedom for engineers to explore and "own" their SRE efforts. The council will serve as the main point of contact for SRE but engineers are encouraged to take ownership and make proactive decisions based on data.

Facilitating the culture of SRE:

- *Empower each engineer's "reliability feels," so they can take ownership of improvements*
- *Proactively expose dependencies across systems starting with dialogue and data*
- *The council would serve as the point of contact for reliability conversations*

Taking ownership of something means empowering engineers to do what they think is right. We would encourage each engineer to engage their "reliability feels". In other words, if something feels like a concern, bring it up to the council and assume ownership for improving it.

Ordaining engineers with "you are empowered now" rarely works. In many cases, dependencies, as well as system and team dynamics, prohibit teams from actually being able to make much of a difference. Because of this, we made it clear that removing barriers and any resistance to the flow of value needs to be made an actual priority to accomplish early on.

When the council meets, each member was responsible for bringing concerns related to SRE from their respective domains. As a group, we would aggregate and vet concerns in order to begin adding work to our engineering backlog. Representatives would present before and after improvements once concerns had been addressed and supported by data. As a unit, they would provide input to the future SRE roadmap and efforts. We'd all decide together.

Along with a concise mission statement, we felt that formalizing the responsibility of the council as it relates to the mission statement made sense. Because we were attempting to formalize and legitimize SRE (at VictorOps), explicitly spelling these out felt appropriate, especially if we planned to share our journey outward.

Mission

Provide an avenue to direct VictorOps' hunger for reliability.

Vision

What's the big picture here? What are we trying to achieve? What will the SRE council own and solve? Formally, we established and communicated to the company that the official vision of SRE was:
SRE will maintain and continuously improve reliability and scalability in the customer experience.

Goals

Aiming for buy-in across our entire organization, there were a few conversations that surfaced early on around establishing some clear goals. We want to monitor and improve the customer experience in order to achieve an optimal balance between high reliability and scalability as it relates to deployment speed.

From a high level, this was broken down into:

- *Bring visibility to the system's reliability and scalability through instrumentation*
    - *R&D for unknown concerns*
        - *Load testing (API & benchmark)*
        - *Game day exercises to uncover unknown aspects of the system in a controlled environment*
- *Address and facilitate a resolution to current reliability and scalability concerns*
    - *Tackle existing known concerns*

- *Focus on proactive actions (demand forecasting/capacity planning)*
    - *Proactively pursue future concerns*
        - *Capacity & Saturation metrics*
        - *Anomaly detection*
        - *Product and Management team input to understand where we're going*
- *Operate with transparency and genuine inquiry*
    - *Open council meetings*
    - *Communicate vision and roadmap to VictorOps*

Process

Creating an efficient streamlined process to raise, discuss, and affect improvements to the reliability and scalability of VictorOps was the high priority early on and we wanted to build a first-class development workflow to address it. To achieve this, council members would regularly collect SRE concerns & improvements from their teams. These concerns would then be vetted together in front of the council in order to build an SRE backlog. This includes breaking work down into team-specific stories as well as epic level work. During subsequent program planning sessions, teams would then pull work into sprints. On a regular cadence, teams would present before and after improvements once a concern has been addressed. The combination of these efforts would help to shape and provide input to the SRE roadmap.

Formal Submission Process

We wanted to standardize what would be needed for all future submissions. This would allow us to evaluate and prioritize them accordingly. As a result, we established a formal process and outlined a few basic guidelines each concern would be evaluated against. Once a concern was identified, it would be raised in the following council session. The council has three initial criteria that each concern must address.

Required criteria to raise concerns to the council:

- *Why is this SRE?*
- *Why is this important?*
- *What is involved?*

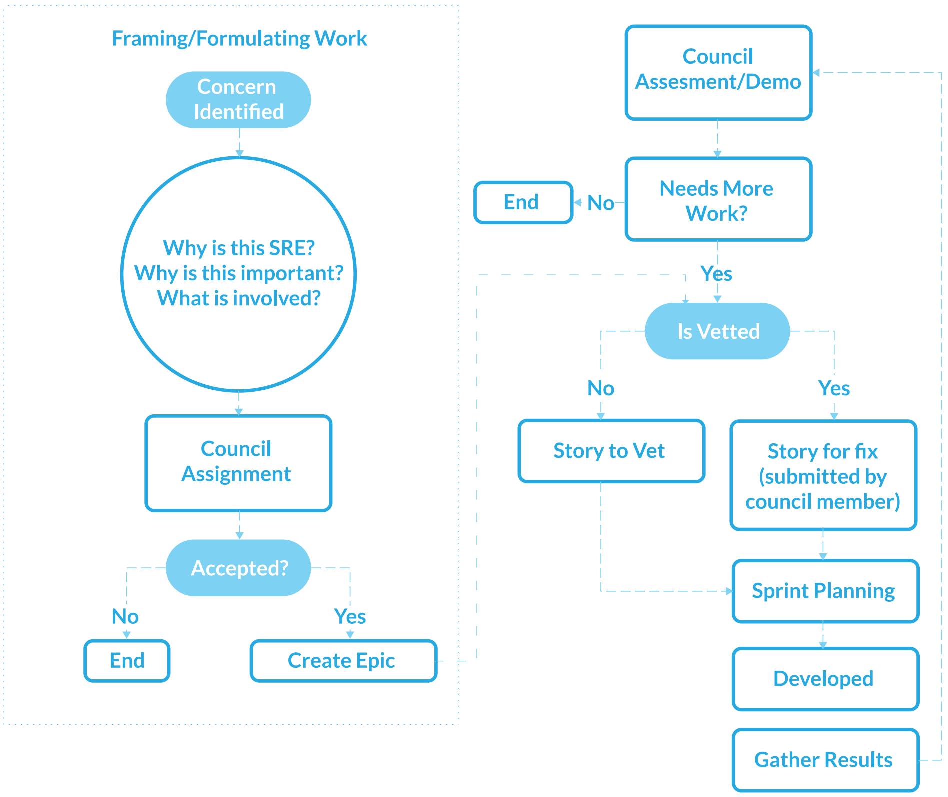
Collectively, the council would evaluate each and either accept or reject the concern. If a concern was accepted, we would create an "epic" together ensuring all relevant details are captured in our project planning tool.

If the council deems the epic to be properly vetted, a story would then be submitted by the council member who raised it. From here, it follows the path of any other engineering effort. Work is assigned during sprint planning, engineers follow their normal routine of building, testing, and deploying to the pre-production environment, at which point we begin gathering results from instrumentation that has just been added. This gives us more visibility into the health of a system.

> *Discussions of instrumenting applications earlier in the SDLC will begin taking place as a result of trying to understand the reality of the systems better. Once engineers realize that they will be the ones responding to problems in production environments, it begins to make a lot of sense to instrument earlier on. Engineers become familiar with the monitoring and alerting tools. They get to craft their own alerts, ensuring that when they are woken in the middle of the night for a problem, they know with greater certainty that this is indeed an actionable alert and, because they've seen this before in pre-production environments, they know exactly what kind of detail, context, and tools they will require in that sleepy-eyed moment. It's like helpful engineers from the past... traveling forward in time to help out during an outage!*

SRE Concern Submission Framework

1. Team representative submits a concern to the council
2. Council assesses concern using the following guidelines:
   a. *Why is this SRE?*
   b. *Why is this important?*
   c. *What is involved?*
3. Council determines if the concern is valid
4. If valid, an “epic” is created
5. If the epic is vetted, a story for the work is submitted by the council member
6. Sprint planning
7. Development
8. Gather results



Tools

Of course, in our efforts to create an observable system, conversations around tooling surfaced. The council would direct the team to useful and powerful tooling for instrumenting the system and serve as a unified resource for toolset decision making. Architecture reviews and decisions would be a social, group effort.

> *In the fall of 2017, VictorOps rolled out significant changes to several key components of our systems. We also launched a new website and two new mobile apps all in the same week. A lot of changes and improvements were being made to the system at the same time the council was forming. This provided a great opportunity for team representatives to collaboratively review architectural decisions including systems that served internal "customers" such as support, sales, marketing, and management.*

As we continue our journey we want to make sure that all new tooling MUST be data-driven. If any existing tooling is polluted or is inhibiting effective usage, let's correct that. When we don't have accurate data and telemetry on the flow of value through our system. (Reminder: The value is the service VictorOps offers AND the underlying infrastructure on which it is provided.) Then we have a very limited scope of reality.

A tangible result of our SRE efforts should be that we have empirically reduced the unknowns and increased what is now "knowable". You don't know what you don't know... And that's a problem when it comes to reliability. We need to operate with realities, not hunches. We need to be able to prove that work is important and the benefits should be measurable.

SRE aims to alleviate overhead in all other teams affected by this problem domain.

Meeting Frequency and Format

Sixty-minute meetings would be held every other week. All meetings were (and are) open to anyone in the company interested in either contributing to reliability conversations or learning more about how the system currently works and proposed improvements.

The Council was opt-in and until efforts were more formalized, SRE work was not to interfere with existing planned sprint work. There were no obligations to contribute to SRE conversations, yet everyone was encouraged to. The experience shall remain collaborative and engaging rather than a top-down "project". Let's create ownership that helps to move the needle on feeding our own culture of reliability.

Responsibilities

Most of the key responsibilities of the council became obvious very quickly. However, to formalize them, we established that the SRE council was at least initially responsible for the following:

Bring Concerns From Your Team

In order to encourage our entire engineering team to embrace and own reliability in their own domains, the council coaches and stimulates individuals to raise any concerns or ideas. Improvements would be continuously made to process and tooling to improve the system from a holistic point of view. By diversifying our council, we had subject matter experts from all corners of the business bringing ideas and concerns that others would not have had any knowledge about.

Vet Concerns in Order to Build SRE Backlog

Coming up with ideas is one thing, but if work is never performed to address the concerns, no improvements will be made. Functionality and features are perceived as better use of engineering resources unless we can make a bulletproof argument that our concerns and the associated work is actually tied to improving the system from the customer's experience. We knew we needed a process to convert these concerns into engineering work. We needed to provide a first-class workflow to address reliability and scalability into our backlog and prioritized as important engineering work. The council would help to break down high-level work into detailed story-level representations as well as be a representative during backlog refinement and sprint planning exercises.

Present Before and After Improvements Once a Concern is Addressed

To encourage accountability and acknowledgment for improving the system, we asked that representatives present before and after results towards improvements during the next program increment planning week.

We want to regularly demonstrate to the organization how we are continuously looking for methods to evaluate and improve the technology, process, and people as they relate to building, deploying, operating, and supporting the "value" of the VictorOps service—including minimizing the disruption of services from these efforts.

Provide Input to SRE Roadmap

Along our journey, the council would provide input to the overall SRE roadmap. By unifying an understanding of SRE and associated efforts across the council and organization, we will produce a comprehensive SRE roadmap with input from all teams and outline specifics on how we will get there. This would be an ongoing effort as the need and objectives of the business can and will change quickly and often dramatically. Bringing value to the end user is the ultimate goal. What that value looks like in the form of functionality may shift and change but reliability and scalability will remain a constant priority.

The SRE Council is responsible for:

- *Bring concerns from your team*
    - *Vet concerns in order to build SRE backlog*
    - *Form into epic level work - break into team-specific stories*
- *During program planning: Teams pull work into sprints*
- *Present before/after improvements once a concern is addressed*
- *Provide input to SRE roadmap*

The SRE Council is NOT responsible for:

- *Responding to immediate customer needs*
- *Discovering bugs in functionality and issues with user experience*
- *Exploring or defining creative user functionality*

To dive deeper into the responsibilities of SRE, there are a few more things our council chose to keep outside the scope.

According to the Support team, SRE was **NOT** responsible for responding to the immediate needs of customers.

While attending to and communicating trends indicating future reliability issues for customers is greatly appreciated, SRE was not part of an escalation path for customer issues received by the support team.

When we asked our QA team, they let us know that discovering bugs in functionality and issues with user experience was NOT the responsibility of SRE.

SRE would instead look for ways to support identifying reliability problems in the user experience through a number of approaches.

Not only did we solicit feedback from our different engineering teams, but we also wanted to hear from members of the Product team. Invoking input from many different perspectives should give us a more holistic approach to what SRE means to VictorOps and align our objectives and incentives.

To the Product team, SRE was **NOT** responsible for work related to exploring or defining creative user functionality. Ideas and feedback pertaining to product enhancements are always welcome, yet SRE would not own this as a core responsibility.

> *Assuring that new functionality is instrumented from a reliability perspective means bringing multiple areas of expertise together to inform improvements to the overall product faster and with fewer service disruptions. Involving Product Owners in these discussions surfaces effort that may be relevant to sprint planning and feature work. Don't forget to share findings that involve engineering resources that may not only be feature work.*

If you ask the front-end engineers where an SRE's role ends, they will make it clear that building out a new system and user functionality was their domain—and outside of the expectations for an SRE. If SRE could help ensure that new functionality is instrumented from a reliability perspective, the front-end engineers would own the rest.

Our IT Operations team informed us that building and supporting infrastructure that runs the product was **NOT** an expectation of SRE. However, any help with forecasting demand and proactively triggering automated scalability efforts would be greatly appreciated.

Last, we got together with our data team to gather their feedback on what SRE should NOT be for them. Their answer was simply... "Gathering business intelligence. That's our specialty." But they also added, "If you could help us understand what kind of data we could start collecting with regards to system reliability information, we'd love to dive in."

There were no real surprises with these conversations. Most teams are clear on their role and responsibility in delivering value to the end user. However, it did help surface talking points and suggestions around what efforts SRE might be able to bring to the table to increase our overall reliability, as well as increase our ability to deliver functionality (read: value) to the end user faster.

When examining these expectations, we realized that when we start to put ourselves in the perspective of the end user and empathetically understand what problems they are solving for, it was clear that the ideal customer profile, as they say, sounded a whole lot like ourselves.

VictorOps needs to be able to deliver value in the form of features that enable customers to do what they love (build systems that enable others) and we need to do it faster while still maintaining reliability.

This is a common challenge for many of our customers. While some are just looking for better ways to reduce downtime, others are experimenting with ways to introduce change (and therefore chances for failure) faster and faster into their systems; continuously improving the system with each release. Releases that used to go out to end users once every three weeks are now taking place at the very least once a week and, in some cases, even more often but with the intention of speeding up even more over time.

Involving the Product Team

At first glance, from a product owner's perspective, SRE might present what appears to be a "competing" value stream. For product owners, it's about getting functionality out the door as efficiently as possible.

> ***Example:***
> *As a user .. I want to …*

This is the language, and as a result, incentive structure product owners are working in… a "user story".

The user doesn't see the relationship between functionality and reliability. They do not necessarily know that they care about how the service is brought to them. They just want to perform their own task at hand.

Without an honest conversation with product owners about the relationship of feature velocity and system reliability, opposing incentives may cause dysfunction when prioritizing engineering resources for functionality, reliability, or scalability.

Thankfully, our product owners care a great deal about reliability from the customer's perspective. And not only do they understand that relationship, they and everyone on the engineering team can't wait to achieve greater confidence and speed in the delivery pipeline. As data-driven decision-makers themselves, they believe that the council's data-driven approach supports effective prioritization and the best approach to balancing reliability with scalability from the customer's perspective.

In order to achieve this balance, quantifiably measuring "reliability" using instrumentation of the running system in production became a top priority. Accordingly, we needed to find ways to examine and verify correctness and availability while also tracking release frequency. Measuring how often something goes wrong with releases is also related and important. How quickly our team was made aware of and were able to swarm to problems both right after changes to the system were made (deployment) and during unplanned service disruptions are metrics we watch closely. With an increase in deployment frequency, it becomes even more critical to have metrics available. These data points and observations would then inform a hypothesis for improvements—rather than opinions or hunches. Delivering the greatest value to our end user required us to challenge assumptions about how our system behaved.

Armed with this hypothesis, we could now take a data-driven approach to improving the underlying infrastructure of the system along with the application and experience of the customer. For any organization, knowing where to focus resources is essential. In our experience, when the data tells you where you have the biggest problems or where you'll get the largest return on engineering effort, resource allocation decisions become much easier. Access to high-fidelity data helps to create a well-informed and proactive engineering team.

What Do We Do First?

We want to see and analyze what's happening before much planning or effort is enacted. Shortened feedback loops are achieved simply by placing more emphasis on observability.

> **"[Accelerating the releasing of code] and the formation of the SRE council are big inflection points for when we started to really observe and measure things in our system. The visibility increase since we began has been like coming out of the dark ages."**
> *- Jason Roth, Senior Platform Engineer, VictorOps*

We want to turn high fidelity data into information that fuels changes to the system. Improvements to the reliability, enhancements to our delivery pipeline, shortened feedback loops to engineers, and faster deployments of features, as well as improvements in human performance, can all be driven by data.

In working closely with other teams and using this information to collaborate on solutions, we would need to begin not only collecting more data to enhance our observability into the system but we'd also need to establish baselines to establish our expectations of what a healthy system looks like (to us). What is "normal"?

Concluding each council meeting, action items were established for the group. The first four assignments kicked off some of the deepest discussions we've had around the reliability and resiliency of our service—not to mention what types of things we can't currently answer.

Assignment One: What Keeps You Up At Night?

We asked each member to go back to their teams and return with a list of the most obvious concerns you can think of when it comes to reliability of the VictorOps service. Something that has always bothered you and is easy to determine if you have the ability to confirm the concern or not using data. Is “that thing” that’s bothering you something we can actually see in the system using data?

> *These early conversations pointed out obvious blind spots in our own system. The truth is you don’t know what you don’t know about systems. When it comes to reliability, the last thing you want to do is make decisions based on emotions or anecdotes. All efforts should be aimed towards exposing the knowable and amplifying the known. The importance of observability (a superset to monitoring, logging, tracing, etc.) is increasing significantly as it allows you to learn and know more about your systems.*

What Do You Worry The Most About?

One by one, we went around the table and asked each representative on the council to share their list of concerns. What really keeps them up at night?

Our IT Operations representative pointed out some blind spots in monitoring that were recently uncovered. Scalability was a growing concern as our customer base has exploded recently as well.
For our Data team, not having enough good data in pre-production environments was proving to be problematic for testing effectively. Monitoring was often too noisy and, as a result, alerts weren’t always that meaningful or even actionable. Third-party tooling use was beginning to sprawl and we felt that we had poor visibility into the things that are touching our system.

When you talk to the representative from the Web client team, exception monitoring was at the top of their list. This held the largest area for opportunities for improvement. They also mentioned that there is no tie between exception tracking and deployments; another blindspot that was becoming more and more worrisome.

Scalability issues of the UI and UX were brought up as well. We need to get the design team involved sooner and get them better data to make informed decisions before our web client isn't able to meet user demand and expectations. They also felt that the deployment process could use some tweaks.

We asked the council members to provide a short list of top concerns. Dozens of ideas were presented. Once we had a list of solid concerns, the next meeting would be to discuss methods of observing data related to them. In order to build and test theories around how certain aspects of the system work under certain circumstances, we would need greater visibility.

What did we learn?

We needed more data. This would require engineering time. But, we are in a pretty good spot to make significant positive impacts in a very short period of time. Although the running system may not be well understood by all, engineering cares deeply about reliability. Especially when discussed in regards to VictorOps scaling to meet the needs of our customers who themselves are experiencing fast growth and demand. We have a lot of input on SRE concerns so far, but no way of prioritizing or assessing the risk of individual concerns.

Assignment Two: Value To Effort Analysis

Now that we have a list of concerns to address, we needed to begin breaking them down further so we could prioritize. We need to understand what is involved with making data related to these concerns obtainable. For each concern, we want to determine the value, effort, and blockers involved in adding instrumentation specifically addressing the concern. Additionally, if they could advise the council on the complexity, risk, and any supporting evidence as well. This should help us sort in a few ways.

Each council representative was then asked to begin researching the following information as they relate to each concern:

- *Value*
- *Effort*
- *Blockers*

If possible, provide the following as well:

- *Complexity*
- *Risk*
- *Evidence*

For the IT Operations representative who had previously mentioned monitoring coverage and scalability were at the top of their list of concerns, they informed the council of the following:

- Monitoring coverage:
    - *Value: High*
    - *Effort: Low - Medium*
    - *Blockers: More of a time commitment than it should be.*

- Scalability:
    - *Value: High*
    - *Effort: High*
    - *Blockers: Time is a large blocker on this one. Spinning up new servers takes a cross-department effort. IT needs to create and provision the server, dev needs to deploy to it.*

For our Data team that said not having enough good data in staging was proving to be problematic, as was noisy monitoring and alerting and Saas tooling sprawl, they came up with:

- *Monitoring of ETL processes:*
    - *Value: High, we will actually know if ETL is broken, on-fire, or just working*
    - *Effort: Moderate, we have some tooling in place with Sumo, but that is it*
    - *Risk: ETL breaks silently*
    - *Evidence: Count the SE's*
- *Tests (all levels):*
    - *Value: High, we really have no testing in ETL, making it fast-fail impossible, and all validation manual*
    - *Effort: High, nothing exists right now.*
    - *Blockers: Data Volume issues. ETL is heavily influenced by db size, production has had a number of issues that can't be seen in other testing environments.*
    - *Risk: Continued bugginess and unreliability of ETL/reporting, customer churn*
    - *Evidence: (See above)*

- *Monitoring of SaaS tools:*
    - *Value: Moderate, we have minimal monitoring of our third-party tools, causing a lack of visibility into current state, failures, and bugs*
    - *Effort: High, most of these tools provide minimal options for alerting/monitoring so in most cases the monitoring/alerting we have is very noisy*

What did we learn?

This exercise helped us to better understand the lift involved with efforts associated with these concerns. As a collaborative team, we all had a much clearer picture of the risk involved when contrasted with the reward it would provide. With this information, as a group, we could make decisions moving forward on how we prioritize SRE-related work.

Within just two 60-minute sessions (and some research outside of the council meetings) we had generated nearly 200 legitimate questions, hypothesized how we could collect data to answer them, and began analyzing them in order to prioritize them.

Monitoring & Alerting

> **"If you can't monitor a service, you don't know what's happening. You can't be reliable."** (*Site Reliability Engineering: How Google Runs Production Systems*)

A subset of observability, monitoring, plays a key role in engineers knowing when acceptable thresholds have been breached—provided established service levels and steady methods of system health data collection. Or, if something is moving the current state considerably away from the pre-established baselines. So long as monitoring is instrumented early on during the SDLC, engineers can decide very early what types of problems engineers should be alerted to.

We want to avoid unactionable alerts that lead to not only burnout but future inaction. When everything is urgent—nothing is urgent. It's too much noise and impossible to know what's actually important (from the customer's perspective). So many critical issues are often incorrectly ignored.

As we began establishing various metrics and going deeper into our discussions around monitoring and alerting, the council turned its attention to two distinct ways of looking at monitoring—Black Box and White Box.

Black Box & White Box Monitoring

When you hear the terms Black and White Box monitoring, there are a couple ways to speak to the ideas. One aspect is to think of Black Box as "pull-based" monitoring where White Box is "push-based". James Turnbull's book, "The Art of Monitoring," gives one of the better explanations of these two types of monitoring from this perspective.

However, another explanation of Black (and White) Box monitoring exists. If we abstract away all of the inner workings of VictorOps and purely look at the expectations of the service from a customer's perspective, this is Black Box monitoring. Is it working? What "it" is can vary but the idea is binary. It's either good or bad. It's either working or not. We are monitoring externally visible behavior as a user would see it. How does the "value" look?

Going deeper takes us to White Box monitoring in this model. This is monitoring based on metrics exposed by the new internals of the system, including logs, interfaces like the Java Virtual Machine Profiling Interface, or an HTTP handler that emits internal statistics. (O'Reilly).

Our SLIs will likely contain a mixture of both Black and White Box metrics. The next assignment for the council will be to discuss and record possible Black Box metrics. We established a vision and goal to keep in mind when suggesting metrics.

To keep in mind when selecting black-box metrics:

> Vision:
> *Catch issues before customers reach out and reduce time to notify for customer affecting issues.*
>
> Goal:
> *Page the most applicable team—not necessarily the team who's product caught the issue.*

Business Reliability Engineering

In engineering and ops, we can quickly forget about our secondary "customer"—our internal departments. At VictorOps, our sales, support, marketing, and success teams rely on access to our digital resources from our top-level domain. Outages to digital services often impact entire teams relying on those services to do their job.

More and more, systems are being plugged together through cooperative APIs and automation. Marketing and Sales teams rely on the flow of accurate data from customers and prospects as their sensitive (and often difficult to obtain) information is moved between various customer relationship and business analysis systems.

Visitors to the site are encouraged to download digital resources to further educate themselves on the most advanced methods of on-call and incident management. As I'm sure you are perfectly aware, in exchange for these digital resources, potential customers (hopefully) will share with us their information so that we can keep them updated on what's going on at VictorOps and the world of building scalable systems faster and safer. Love it or hate it, this is business and it's something nearly all organizations rely on to operate. VictorOps is no different. We need to make sure that aspects of the website that are marketing and "top-of-funnel" focused are also behaving as the customer (i.e. Marketing) expects.

For example, when a visitor to the site downloads our Post-Incident Report Book, their information is stored in a number of systems, pushed all around through automation—automation that we don't know much about. The process of obtaining this information safely and correctly relies on several steps and tools. Unless someone reaches out to us, how do we know if it's **NOT** working?

Measuring For Normal

We now have a list of top concerns as well as a better idea of how much effort and reward is involved with prioritizing related work. However, along with getting better visibility around potential worrisome areas, we also need to establish what a "healthy" VictorOps system looks like.

There were several suggestions regarding the specific scenarios we should be watching for and measuring. For instance, when a connection between the web client and back-end system are experiencing trouble, a bright gold bar displays at the top of the screen so users are aware that something is wrong.

What we sometimes forget is that WE also know about it, but are we watching for it? Thankfully the problem is rarely something on our end when the gold bar appears. There are many stops along the way in the complex system in which the alerts are delivered through. From shoddy WiFi to a DNS problem on the customers' end, numerous reasons exist as to why the web client and the back-end aren't talking to each other.

> *"A user on a 99.9% reliable smartphone cannot tell the difference between 99.99% and 99.999% service reliability. With this in mind, rather than simply maximizing uptime, SRE seeks to balance the risk of unavailability with the goals of rapid innovation and efficient service operations, so that users' overall happiness—with features, service, and performance—is optimized," (Site Reliability Engineering: How Google Runs Production Systems).*

At least for the moment, we don't care what specific possible disconnection scenarios exist and how we can protect ourselves from them. For now, we only need to determine what is acceptable. What is "normal"? We know disruptions happen. We can point fingers as to who is to blame or what is the cause, but the point is to discover what is the expected behavior? This will be our baseline.

Obviously, we know when the gold bar is displayed. Multiple logs can tell us that part of the story. But how often is the gold bar displayed? When it is displayed, on average how long does the user see it? Are many users experiencing it simultaneously or is it sporadic? What kind of information is in between the lines of the gold bar data that we have been (or can begin to) start collecting?

> *One tip we learned from our friends at Netflix was that they have a single metric that they watch very carefully. That being PPS (Plays Per Second). In other words, how many times does a customer press or click the "Play" button on Netflix? This single metric would act as both an overall health check but also a leading indicator of trouble on the system. Once a healthy "plays per second" baseline has been established, setting a reasonable threshold, in one way or the other, means that teams can be alerted to possible trouble early on.*

What might be our own version of the PPS metric? What about how long it takes for information to be displayed to the user? In what way could we more closely measure how long it takes for information to display in the incident timeline. Could this be our PPS or is there something else that might be even better to look at?

Ideally, the population of data into the timeline is so fast that humans would never know or have reason to think, "This seems to be taking longer than I expected." If it is slow, how slow exactly? And what threshold is deemed too slow? In order to really know how fast (or slow) it was, however, we would need instrumentation to measure it, a data store to collect it, a dashboard to visualize it in real-time, and, of course, alerts should established thresholds be breached. Find something. Try it. Adjust.

Another suggestion was to closely watch the response time for searching for users within the system. One of the most powerful features during a firefight is the ability to quickly pull in additional responders and collaborate in real-time.

Getting everyone up to speed on what is known about the incident speeds up the recovery efforts. To do this, much like social media, "mentioning" individuals and teams are achieved in the incident timeline by typing "@" in the chat field. This triggers a search module to display potential matches as the user begins typing. How fast (or slow) is this response time, and what value should trigger cause for alarm?

Clearly, we were going to need to not only roll out new instrumentation, we were going to need to collect data over time, establish thresholds, and build actionable alerts to contextually inform first responders what is broken and where to start looking. Not only do we want better observability on these things, we also want to learn more about how new code needs to be instrumented, as well as what makes for an actionable and helpful alert.

The engineers who wrote the code are likely the ones responding to it in our production environment. They should decide and establish the right types of metrics and alerts to provide in an incident related to their code and functionality. Engineers become much more familiar with the tooling earlier in the software development lifecycle.

Duplicating monitoring, alerting, and on-call rules in a development or staging environment mean the engineers responding to the problem in production are already well-versed in the tools used during real-life response and remediation efforts. Also, they are likely the ones that established the monitoring thresholds, on-call policies, and (hopefully) actionable alerts. If the information provided regarding a problem isn't helpful in reducing the time to detection or resolution, it's easier and less expensive to spot that in pre-production environments. Should the problem rear its ugly head in production, the proper metrics are in place to spot it, the right teams or individuals are contacted, and they are immediately familiar with the current status through helpful context appended to an alert that has already been confirmed as "actionable". What about the round-trip interactions when a user triggers a new incident? There are multiple things that take place during some of the most important pieces of functionality. For instance, when a user has been alerted to an incident, their first action is to acknowledge the incident. How long does it take for the user to press "ACK" on their mobile device, for VictorOps to receive and process this request, and for the user's confirmation to be displayed as an action? Specifically, how long does that take? While it should be extremely small, how small exactly?

Once our new instrumentation was capturing higher granular data and we started averaging out various metrics of the system, we were ready to start setting targets to both maintain or to achieve. It was time for us to establish Service Level Indicators, Service Level Objectives, and Service Level Agreements. This helps us determine what "normal" is. The baseline then serves as our early indicator that something is no longer "normal".

Measuring For Progress & Success

Measuring for outcomes is always at the top of our mind when approaching goals. While we do have specific targets we may be aiming for, circling back to confirm that the resulting outcome is in fact what you were after is extremely important. Small course corrections are required. Outcomes may be more general but often attract the attention and support of decision makers earlier.

Key measurements and thresholds to hold us accountable for our efforts as well as communicate expectations across the entire organization needed to be established.

Nearly every resource you find regarding site reliability engineering will talk about key metrics used to establish high-level objectives, indicators of the movement toward or away from those objectives, and ultimately what agreements are in place should objectives be unfulfilled.

SLIs will help us know how we are performing against our SLOs and our SLA will outline the consequences (good or bad) of meeting those objectives.

Once we have data to observe, we will begin orienting ourselves to it and establish what we believe our SLIs and SLOs to be.

Service Level Indicators (SLI)

Empathetic understanding of customer needs and expectations will help to inform indicators. And at first look, there are many possible indicators that could be measured. However, we found that landing only a handful of indicators that really matter was the right choice. Finding a good balance of indicators is important to help teams accurately examine and understand important aspects of the system.

Service Level Objectives (SLO)

Service Level Objectives (SLOs) are established measurements to inform Service Level Agreements (SLAs). This measurement establishes a target value (or range of values) for assessing the overall trajectory, and eventually, accuracy of your objectives.

Service Level Agreements (SLA)

What happens when SLOs are breached is what SLAs address. The council does not accept ownership of constructing SLAs because SLAs are closely tied to business and product decisions typically managed higher up the chain of command. The council will, however, be involved in helping to avoid the consequences of missed SLOs.

Assignment Three: Establish Blackbox Metrics and Service Level Expectations

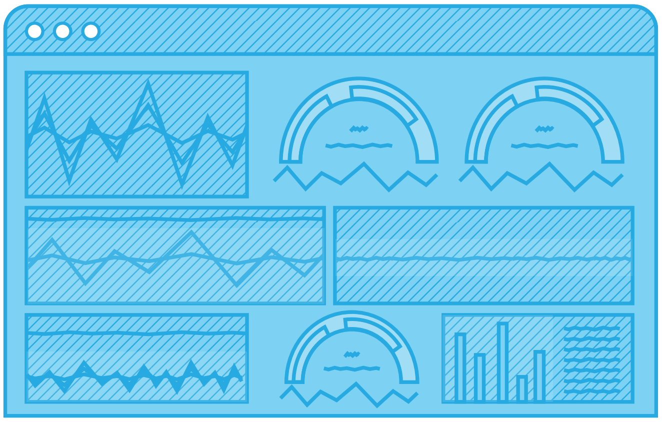
Our next assignment was to begin establishing our own Black Box metrics and expectations regarding service levels. This included coming to an agreement on thresholds, how we should address violations, what should we make visible right away, and what types of alerts should go to engineering?

As a group, we determined that we would collaboratively define SLIs and SLOs with interested parties (e.g. Engineering, Product). We would assure that all teams are in agreement on what constitutes violations. We determined that metrics we had surrounding service level expectations should be made visible through dashboards.

An assortment of various dashboards related to service expectations began circulating within our Slack groups. We also put in place alerts to reinforce the importance of our new metrics.As a group, we would address violations and aim to achieve SLO's with near perfection, allowing us to maximize change velocity without violating an SLO.

What Did We Learn?

Once we returned for the next council meeting, engineers had already begun plumbing in new instrumentation such as the Grafana "OpsDash", building dashboards, and cleaning up existing murky data collection. This lead to creating and socializing deployment dashboards (also in Grafana) for the company to have available any time they wanted.



We updated metrics to Jenkins jobs, as well as annotations for said jobs in Grafana, and we added a number of new metrics for deployment health and general health. This is all the result of simply establishing a few key metrics and processes that we felt were important to get visibility around.

Engineers added Prometheus to our bootstrap process, making metrics collection and experimentation accessible to any developer who is interested. We also implemented a new abstraction for collecting metrics in our platform. We determined healthy "golden signal" metrics around our socket traffic in the browser and established thresholds for all three visible panes (people, timeline, incident).

Initial golden signals:

- *Alerts Received / Processed*
- *Incidents Created / Resolved*
- *WebSocket Connections Connected / Disconnected*

These would serve as a starting point and we are already exploring more signals such as "Time to First Notification" (TTFN) as our best leading indicator. Could this possibly be an even better Netflix-like PPS metric?

Modernizing On-Call

One of the most important ideas we evangelize at VictorOps is that we, as software makers and digital service providers, must start thinking of our systems in a more holistic, "Systems Thinking" way. VictorOps, just like any other company is a socio-technical system. Aspects regarding both technology and humans must be considered part of the "system". Because of this, we need to find new methods of incorporating human considerations into our methods of building and operating services.

As good as we may be at software development and architecture, it still takes humans to respond to problems when they inevitably break. Some organizations are experimenting with self-healing systems but these are mostly focused on infrastructure. There are far fewer of these companies using self-healing systems than those of us out there who rely on our subject matter experts and developers to restore service whenever a disruption occurs.

Two sides of the same coin, on-call and reliability, are forever tied to each other. As new instrumentation exposes new areas of the system to pay closer attention to, we often forget the challenges of scaling and improving our on-call practices.

It has become more and more rare for us to encounter customers who are establishing Network Operations Centers (NOCs). The concepts outlined throughout this book should highlight the shift in the realization of urgency to service disruptions. Businesses can't wait until Monday morning when someone from IT gets into the office. Revenue, reputation, and more are tied directly to the site and accompanying digital services. If it's down for even a few moments, that's a HUGE problem.

This is an urgency that most companies can't outrun or ignore. Instilling a sense of the seriousness to restore services whenever a problem occurs is necessary for bettering the reliability of a service. Because of this, companies are cutting out as many "middlemen" as possible and getting the person or team who is ultimately most qualified in that moment to address and resolve an incident.

Bouncing support tickets through a tiered-support group while the clock ticks away is devastating to a company on its own digital transformation, particularly if they are early on in the journey. The stakes are often higher for a company with a big name but are still new to this way of appeasing customers. Tech startups and Silicon Valley garage projects can afford to experience some downtime. It's the price their users pay to ride the wave of innovation; as early adopters, they are willing to take that gamble just to be one of the first to use the service. But once a large, well-known organization has experienced a significant outage, customers will have difficulty reconciling the enormous company resources and any amount of downtime of the service on which they have come to "rely."

Modernizing the on-call practices should not involve the use of outsourced or tiered-support, relying on tickets to be created, assigned, or any other activity that further delays the restoration of service. When the system is no longer "normal", we have already deemed an associated alert or incident to be actionable and, therefore, the person or team who is most qualified to restore service (in that moment) should in all cases be the first responder—not someone who is going to escalate.

> *First responders should rotate often and the experience and systems knowledge will, of course, vary from engineer to engineer. While a more senior architect on the team may appear to be the most qualified, we never want to encourage a superhero mentality where only specific individuals contain the "know-how" to solve our most critical problems. Modernizing on-call rotations also includes bringing in more of the team while making more of the system available to all. With the right context alongside safe and accountable access to the same tools as any other engineer, even the most junior developer should be able to successfully respond to an incident first and begin making a positive impact to restoring service. The moment will vary, the "qualified engineer" should too. Transfer knowledge through your on-call practices. Have empathy towards not only the customer's perspective but to the first responders as well. What would be the most helpful when acknowledging an incident? How can I solve this problem the fastest? What can I give "future me" to restore service as quickly as possible?*

How would we know the answers to these questions? How could I possibly know what the future will need in a moment like this? As we have become more familiar with our systems and the tools earlier in the SDLC, engineers start to figure out what is going to be helpful. Being alerted during office hours to issues in pre-production environments based on new code means that problems can be caught much much sooner. Not only that, but we've verified that we can catch (or see) what we are looking for before it goes to production. Additionally, we have gained an intimate knowledge of the monitoring and alerting tools that are eventually used in production.

By the time this kind of problem has the opportunity to surface in production, we should have instrumented the service better and established only actionable alerts that contain whatever relevant context may be helpful to the responding engineer. It's all part of a bigger effort of continuous learning more about our systems

Improving Mental Models

"Learning from Failure" has long been a message we repeat. Strengthening and improving mental models of how systems actually work versus how we think they work. A large gap between individual realities of how systems behave almost always exsits; moreover, no single perception of the system is correct. The more we isolate our understandings of systems from each other and avoid exercises that help us to create more realistic understandings of them, the longer and more harmful a service disruption will be.

It is with this in mind that we determined our next assignment. A full-day event to proactively poke at our systems to improve our understanding of how they actually work, what we have visibility on, and ultimately what can be done in the near-term to improve things.

Assignment Four: Make The Case For Chaos

To encourage teams to begin thinking about how they can learn more about the system and be made aware of problems in the customer experience before the customer notices—the council decided that scheduling a Game Day exercise would help. With a date set on the calendar, council representatives understood when indicators (i.e. Black-box metrics) would need to be established and how they could be observed. The Game Day exercise would require that those SLIs be clearly defined and a method for triggering a breach of thresholds be established.

What is Chaos Engineering?

> **"Chaos Engineering is the discipline of experimenting on a distributed system in order to build confidence in the system's capability to withstand turbulent conditions in production."**
> *- Casey Rosenthal (CTO Backplane - Previously Eng. Mgr. Chaos Team / Netflix)*

To specifically address the uncertainty of distributed systems at scale, Chaos Engineering can be thought of as the facilitation of experiments to uncover systemic weaknesses.

These experiments follow four steps:

1. Start by defining 'steady state' as some measurable output of a system that indicates normal behavior.
2. Hypothesize that this steady state will continue in both the control group and the experimental group.
3. Introduce variables that reflect real-world events like servers that crash, hard drives that malfunction, network connections that are severed, etc.
4. Try to disprove the hypothesis by looking for a difference in steady state between the control group and the experimental group.

The harder it is to disrupt the steady state, the more confidence we have in the behavior of the system. If a weakness is uncovered, we now have a target for improvement before that behavior manifests in the system at large. (source: http://principlesofchaos.org/)

Why Chaos Engineering?

Contrary to what the name may indicate, chaos events are not performed in a chaotic fashion. Ultimately, they boil down to a specific set of well-planned scientific experiments. For VictorOps, SRE is a scientific practice which aims to make data-driven decisions to improve a system's reliability and scalability—as observed by the customer.

We are actively pursuing more knowable information about our systems in order to improve them while recognizing that this is a constant effort.

Several members of our engineering team have previously conducted Game Day exercises and so an internal presentation was given to the entire organization. This helped to set expectations and communicate to the broader company what would be taking place, how, and most importantly—why.

Before we could get too crazy with our Chaos Engineering aspirations, we needed to get buy-in from everyone on the council and those representatives needed buy-in from leadership. We needed a plan that would outline the process, why we are doing this, what our goals are, risk analysis, etc.

We knew that all simulated service disruptions were going to be taken in our pre-production environment in order to increase our confidence that it wouldn't impact users. There's always a small chance a customer could experience something. Remember, we don't know what we don't know. What if a service in our staging environment is actually talking to something in our production environment? We are still learning its reality. However, we need to "reduce the blast radius" as they say, so our initial exercises will take place in our pre-production "staging" environment.
This means that we need to ask questions about how we can make staging behave (as closely as possible) to the customer-facing environment. How do we want alerts to be delivered? It's probably not smart to co-mingle delivery methods in the event that a real incident is triggered for a separate problem while engineers are rehearsing failure in a simulated scenario. How long would it take us to figure out which alerts were "real" and which weren't?

As a group, the council discussed and came up with a formal test plan, setup considerations, and preparation checklist.

Test Plan:

- *Black-box Alerts*
- *Discuss test plans for each alert*
- *Meet with IT to evaluate RISK*

Staging VictorOps org Cleanup: *Discuss how close it should be to production's VictorOps org*

- *Special paging configurations because alerts will go off at any time a failure is detected*
- *Check:*
    - *Consider email-only for paging policies*
    - *No employee should be notified on their personal device*
    - *Teams/escalation policies/rotations?*
    - *What should these look like?*

Solid observability of the system is required before Game Day testing can be successful. This was why so much emphasis was put on determining our metrics in assignment three and then ensuring we had visibility of them. A lack of structure for the day would be detrimental to the Game Day efforts. Some sort of defined plan would need to be established. All tests would take place initially in our staging environment but future exercises would take place in the production environment. Teams testing at the same time can and will collide with each other. Be ready for this. Something else that was brought up due to previous experience was that we needed to not only minimize the blast radius of our efforts but we also need to limit the time the game day exercise was supposed to take place. A 14-hour day (typically on the weekend) was not the right approach. Engineers will lose focus and interest if game day exercises go on too long. We would not be "randomly unplugging shit". This is not the best place to start nor is it part of the principles of chaos.

Principles of Chaos:

- *Build a Hypothesis around Steady State Behavior*
- *Vary Real-world Events*
- *Run Experiments in Production*
- *Automate Experiments to Run Continuously*
- *Minimize Blast Radius*

Chaos Day

About a week prior to the event, our champion sent a company-wide email on behalf of the council. The message outlined the agenda for our Chaos Day event as well as offered a response to frequently asked questions.

What is a Chaos Day?

A dedicated time of performing experiments on a system.

Chaos Day Schedule

9:00 am: Kickoff with training for the day

10:00 am: Experimentation begins

3:30 pm: Retro and Lean-coffee

What is the goal of Chaos Day?

Using the principles of chaos engineering we will learn how our system handles failure, then incorporate that information into future development.

What is the goal of our experiments?

This time around, we're verifying our first round of Black-box Alerts in our staging environment.

What is the scope of a single experiment?

Some experiments will affect only one or two web browser clients where others could affect a major backend service, which would affect many other services and, potentially, clients as well.

Who is involved?

All of VictorOps Engineering and anyone else that has the capacity and would care to join and observe.

Will we be able to demo our product during this time?
Yes. These experiments are only in our staging environment.

Who will this effect?

The goal is to not affect production systems. However, it is possible we unintentionally affect production. So, if something does happen that is affecting or has affected production systems, we'll communicate that ASAP. We'll have a dedicated Chaos Incident Commander in communication with Ops Support.

How do we ensure it does not affect customers?

We're doing our best to assure this but cannot absolutely ensure there will be no effects. The test plan is being reviewed by the SRE Council along with IT Operations.

What happens if there is a Sev 1 SE on that day?

Chaos does not take priority over production emergencies. The appropriate people will be brought in and tasked per usual.

What happens if we actually break Staging in a way which takes longer than a day to fix?

We're aiming to avoid this with back-out criteria for experiments and reset criteria for bad/overloaded data. If, however, a long recovery time is needed, we'll communicate this and make arrangements with affected teams.

Roles

In our final council meeting before the Chaos Day, we discussed and established well-defined roles for everyone. This would ensure standardization to some degree on the make-up of teams during the exercise. Documenting as much as possible was the first role we wanted to assign—we needed a recorder.

Recorder

- *Assure hypothesis and risk assessment have been created*
- *Record how the experiment unfolds*
- *Collect data (graphs, alerts, times, etc) while experiment is performed*
- *Time to know (from moment we trigger → monitoring has identified)*
- *Time to detection (trigger → time VictorOps has notified us)*
- *Note whether or not the Black-box alerts were triggered*
- *Gather information from mini-retro after the test*

Someone should be responsible for driving the experiments as well.

Driver

- *Perform experiment*
- *Provide all history of actions performed (command line, Jenkins jobs, puppet modifications, etc)*
- *Verify alert was triggered*

A technical lead (typically the council representative) would assume the role of the incident commander to be the main point of contact and to maintain a high-level holistic awareness for the experiments.

Incident Commander (Tech Lead)

- *Assure back-out plan is defined*
- *Keep an eye on the back-out plan during tests*
- *In an incident, perform any communication with Chaos Incident*

Commander for the day

In addition to team incident commanders, one engineer played the role of the event I.C., communicating across all experiments throughout the day.

Chaos Incident Commander

- *Communicate with Ops Support & Incident Commander for the team under test*
- *Update internal Statuspage & Slack channel*

For each defined experiment, the following guidelines were provided. Again, this helps to standardize the exercise across all teams as well as serving as a checklist for engineers.

Step-by-step guide

- *Describe test to group*
- *Develop a hypothesis for what is expected to happen*
- *Take a poll to gauge the group's assessment of risk*
- *Execute test steps*
- *Once complete... Perform mini-retro on test*

What did we Learn?

Having a dedicated time set aside that was well communicated in advance helped on many fronts. Not only did it give us a kick in the butt to determine SLIs and SLOs, it helped formalize the event. Collaboration during the experiments was fantastic. In fact, digging into unexpected problems together was fun. As a group, we learned much about the process of Chaos Engineering. We learned a great deal about tools we had in place and how we need to improve monitoring in a few places.
In terms of opportunities for improvement, the most obvious regarding our Chaos Day event was how more people from different areas of, not only the engineering teams, but the rest of the company can be involved and participate. It became clear that some engineers felt left out even though it was an open exercise. People wanted to contribute but we had only defined a limited number of roles. In future exercises, more people throughout the company will play a specific role in the day.

Suggestions were made to move the event to a different part of our development cycles. Chaos Day might have been able to attract more attention if it took place at a slower point in time for development teams.

Inherently, the experiments individual teams chose affected only their own services. This is not the reality of complex systems. In future chaos events, we intend to group experiments by functional areas of the product in order to test cross-functionally.

In general, there were several suggestions on how we could prepare for the day a little better. Now that we've been through one, it has been determined that:

- *Test plans are much more important than we realized ahead of time*
- *Organization of the test plans would help divvy up work to more people*
- *Schedule focused time windows ahead of time for teams/groups of tests*

Last, we intentionally left the human response questions out of these experiments. For our next event, we'll want to observe and capture more around the elements associated with getting the right people on the (injected) problem as quickly as possible. Ideally, engineers will have established thresholds, paging policies, contextual alerts, runbooks, and anything else the first responder would need. Not preparing first responders with what they need in those moments is a weak spot for most organizations.At VictorOps, we've learned a few things about how to recover from failure quickly, but there's always room for improvement. So, we'll begin adding these ideas to our future test plans.

PART THREE

The Next Steps

Conclusion

Designing, building, operating, and improving the VictorOps Software as a service, including the underlying physical and virtual infrastructure is critical to the future of the business. Exploring new methods of both delivering quality software sooner and receiving feedback from real usage faster is our quest. Continuing to develop better methods of delivering software as a service to meet the changing needs of our user base will be a constant journey. We expect our own internal SRE efforts and philosophies to change dramatically, even by this time next year. The path we took provided results for us. They may not for others, especially large enterprise organizations distributed globally. The organizational structure and culture are strong contributing factors to the success and failure of these types of changes. As we grow, modifications to our processes, our technology, and even people will be constantly evolving. In fact, in just the last few months, changes to our SRE concern submission process has changed.

Changes & Improvements

The formal process of submitting SRE issues has already evolved. Now, engineers are identifying concerns on their own and taking the initiative to begin establishing observability around them, as well as implementing service level indicators and objectives. Conversations around concerns still take place in council meetings but the process of vetting and creating epics has shifted to the teams. They are empowered and responsible for identifying concerns, instrumenting visibility, and prioritizing the work on their own.

Company-wide minor improvements are taking place simply by asking questions and having more conversations about reliability. Dozens of large TVs have now popped up all over the office sharing anything from the current health of the system to what work is in flight or coming up related to SRE efforts and more. Links to various dashboards are passed around in our group chat rooms. Awareness is amplified and individuals are empowered to implement improvements, especially if they help to explore more “unknown unknowns”.

Just by having more conversations about reliability and creating more visibility, more confidence is generated on how things work, or at least how we think they work versus, often times surprising, reality. This, in turn, creates more questions. And so on. If you recall, our SRE journey, much like others before us, started by asking questions.

There's always more to discover. In complex systems, things are always changing. You can never know all of it.

Your Journey

We hope that following the VictorOps SRE journey will spark questions for you and your teams. Who is your customer? What are you doing well (or not well) at enabling the customer? What keeps you up at night about the availability of your systems? Can you quickly and safely introduce changes to your system? Can you answer these questions?

Good luck on your journey towards SRE and always keep reaching, stretching, and exploring. That's where the good stuff is!

Jason Hand

@jasonhand

Notes

Solutions for the Victors of Innovation

It’s time we do more than react to system outages and answer a page. VictorOps is incident management software purpose-built for DevOps. From fast forensics to rapid remediation, we empower engineering and operations to work together, solve problems faster, and continuously improve in high-velocity deployment environments.

Learn more about how VictorOps can empower your team to collaborate and solve problems.

Start Free Trial