O'REILLY®

What Is SRE?

An Introduction to Site Reliability Engineering

Kurt Andersen & Craig Sebenik

REPORT

What Is SRE?

An Introduction to Site Reliability Engineering

Kurt Andersen and Craig Sebenik

Beijing • Boston • Farnham • Sebastopol • Tokyo

What Is SRE?
by Kurt Andersen and Craig Sebenik

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

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

Editors: Nikki McDonald and Eleanor Bru
Production Editor: Kristen Brown
Copyeditor: Rachel Head
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

May 2019: First Edition

Revision History for the First Edition

2019-05-15: First Release
2019-06-25: Second Release

978-1-492-05441-2

[LSI]

Table of Contents

CHAPTER 1
Defining "SRE"

Site Reliability Engineering.

Even when the acronym is spelled out, confusion often remains. The "E" can stand for the practice ("Engineering") or the people ("Engineers")—we'll use it to mean both. The "R" generally stands for "Reliability," but we've heard people use "Resilience" instead. And the original interpretation of the "S" ("Site," as in "website") has expanded over time to include "System," "Service," "Software," and even more widely "online Stuff."

In general, SREs work across the realm of *"Anything" as a Service*, whether that is Infrastructure (IaaS), Networking (NaaS), Software (SaaS), or Platforms (PaaS)—anywhere the fundamental customer expectation is that the online service can and must be reliable.

> SRE is an *organizational model* for running online services more reliably by teams that are chartered to do *reliability-focused engineering work.*[1]

1 Hat tip to Laura Nolan for this wording. Also note that the skills and capabilities to troubleshoot production problems and feed that learning back into making things better *can* and do exist in teams where reliability may be a shared mandate. The relative balance of concerns between reliability and "other things" will affect the effectiveness of the execution.

The use of service level indicators (SLIs) and service level objectives (SLOs) as meaningful indicia of service health is one of the distinguishing characteristics of SRE practice. It is important to recognize that SLOs are symptoms of a healthy relationship between the reliability (SRE) team and the feature team, not a compliance exercise dictated by management. In the pursuit of greater reliability, SREs will focus on bringing as many components of the greater system space as possible into a resilient, predictable, consistent, repeatable, and measured state. Major areas of expertise can include:

- Release engineering
- Change management
- Monitoring and observability
- Managing and learning from incidents
- Self-service automation
- Troubleshooting
- Performance
- The use of deliberate adversity (chaos engineering)

> As a discipline, SRE works to help an organization *sustainably* achieve the *appropriate level of reliability* for its services by implementing and continually improving *data-informed production feedback loops* to balance availability, performance, and agility.[2]

As Stephen Thorne puts it:

> [SREs] … have the skills and the mandate to apply engineering to the problem space. [A] well functioning SRE team must do […] operations mindfully and with respect to their actual goal, [helping] the entire organisation take appropriate risks.

SREs (engineers) can be deployed to focus on infrastructure components, as short-term consultants for feature-oriented teams, or as long-term "embedded" teams working with their feature-oriented counterparts.

2 Hat tip to David Blank-Edelman and the Azure SRE leadership team for this wording.

Depending on the size and organizational structures present within a company's engineering organization, SRE may be visibly manifested in distinct roles and teams with distinct management, or SRE principles and approaches may be evangelized through portions of the engineering team(s) by motivated individuals without explicit role recognition. SRE will look different when instantiated in organizations of 50, 500, or 5,000 engineers. This context is important, but often missing when writers or speakers are discussing how their companies implement SRE.

Digging into the Terms in These Definitions

While it can be helpful to have pithy definitions to refer to, it is important to understand and share an understanding of the key terms within those definitions. Let's explore them in a bit more detail.

Production Feedback Loops

Everyone knows and loves feedback loops—at least in theory. Often, feedback processes and systems don't get the care, feeding, and attention that they need to be effective. Feedback loops are, at their core, about communication within a sociotechnical system: communication on a technical level between threads, processes, servers, and services; and communication on a social level between individuals, teams, companies, regions, or any other level of distinction.

Inadequate feedback and communication channels lead to scenarios such as the classical divide between (feature) developers and operations. Jennifer Davis and Ryn Daniels explain in *Effective DevOps* (O'Reilly) that people naturally shift to focus more and more narrowly on the areas that they are interested in and/or are rewarded and evaluated on. Feature developers are evaluated on their success at creating and delivering "features." In the classical dev/ops split, operators or SysAdmins are evaluated on their success at keeping systems running and stable. Because of these different incentives, the teams are pushed into conflict as each contends for the primacy of "its" goal.

SREs have an intermediary role, and part of their effectiveness comes from having a dedicated purpose that includes establishing and maintaining feedback loops from operations to the feature developers. If services are not working well and the developers don't know about it, then either the right feedback mechanisms have not been built or the mechanisms have been built but inadequately socialized with or adopted by the dev teams.

Data-Informed

It is critical that these feedback loops be automated in order to scale. Scale is further enabled by relying on data rather than opinion. Measurements are inevitably artifacts of their time and environment, constrained by the technologies that are used to obtain them. Changes in the environment or better understandings of the dynamics of a system can lead to valid technical arguments about whether a measurement is accurate or effective in a particular context. Continually improving the measurements to adequately inform product decisions is one of the benefits of having a standing SRE team. As noted by Lord Kelvin:

> When you can measure what you are speaking about, and express it in numbers, you know something about it; but when you cannot measure it, when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind; it may be the beginning of knowledge, but you have scarcely, in your thoughts, advanced to the stage of science, whatever the matter may be.

Appropriate Level (of Reliability)

A simple assumption is that a service should "always" be available. In the Western world and throughout many of the major cities around the globe, consumers are accustomed to a continuous supply of electricity, water, and "the internet." The suppliers of those services put a significant amount of work into making them "always" available, but if you look closely at the long-term availability there are frequently outages. Often the outages are unnoticed by the end consumers, but when they are prolonged—caused, for example, by major natural disasters such as hurricanes—the loss of usual services becomes a headline issue.

In the mid-third century B.C. philosophers in China captured the paradox of trying to make a service never have an outage. The Chinese phrasing of the issue is "a one-foot stick, every day take away half of it, in a myriad ages it will not be exhausted"[3]—and this applies directly to reducing outages.

If a service has 500 units of outage in a given measurement period, it will take progressively greater efforts to maintain that same cumulative outage count as longer and longer measurement periods are considered.

Increasing the reliability requirement will also always require the reliability increase to be supported by all of the dependencies.

Determining the appropriate level of availability for a service based on the nature of the service, the users, and the costs involved is a business decision, not a technical one. The SRE team(s) provide mechanisms to track and manage the outage "budget." The usual industry terms in this realm are:

Service level indicator (SLI)
: *What* you measure and *where* the measurement is taken

Service level objective (SLO)
: The *goal* or *threshold* of acceptable values for the SLI *within a given time period*

Identifying good SLIs and establishing meaningful SLOs can be difficult and nuanced. It is a perennial topic of discussion on Twitter and in various conferences. For now, we'll refer you to the much longer expositions of this topic in Chapter 4 of *Site Reliability Engineering* and Chapters 2 and 3 of *The Site Reliability Workbook*, edited by Betsy Beyer et al. (O'Reilly).

When failures do happen, SREs are frequently at the forefront to remediate them because of the system-wide perspectives that they are able to bring to the incident response team. SREs often play a role in incident response processes because of their commitment to learning from failures and reducing outage durations. They have an explicit concern with making incident response and learning as efficient as possible.

3 Interestingly, at around the same time the Greek philosopher Zeno posed his "Achilles and the tortoise" paradox, which is an alternate formulation of the same puzzle.

Sustainable

A site needs to have an appropriate target of availability, based on an analysis of the business costs and benefits. Part of those costs are the human aspects of stress and disruption involved in developing and maintaining the desired level of availability. Traditional "ops" roles have often romanticized the heroic on-call first responder who single-handedly gets the systems back on line by carrying bits from one rack to another at speeds typically only achieved by The Flash.

Generally, the SRE community deplores the need for heroic measures and strives instead for response patterns and system capabilities that do not require extraordinary efforts. This leads to valuing low-noise, actionable alerting, teams that are sized for reasonable on-call rotations, automated response and remediation, and self-service platforms for feature teams to be able to perform their appropriate work without interrupting the SREs' development work.

"Sustainable" also drives the emphasis on blameless postmortems to learn from the failures that do happen so that the systemic defects that led to a failure can be addressed in both current and future services.

Reliability-Focused Engineering Work

To be considered an SRE team, the team needs to be working on projects that will "make tomorrow better than today". It needs to be fixing reliability problems in the product codebase as well as building tools and systems that will contribute to the reliability of the systems that it supports.

In some cases this may involve building out continuous integration/continuous delivery (CI/CD) pipelines for the organization, but in many cases SREs take that level of automation for granted and are able to focus elsewhere: on fixing design and code choices that degrade reliability or working on monitoring/alerting/observability or capacity modeling/forecasting or load balancing or chaos engineering or dozens of other areas appropriate to a given organization's needs.

Continuous Improvement

Especially in the consumer-facing online internet service world, nothing stands still for long. Services add new features daily, if not multiple times per day. User expectations rise inexorably, and they demand more, better, faster. Ongoing investment is required to meet these expectations.

Organizational Model

Effective and successful teams don't happen by accident. The discussions and agreements around SLOs take time and conscious effort to negotiate and track. Keeping teams from being consumed by the ever-increasing demands of users, developers, and services so that they can do the necessary design and development to engineer solutions also requires a nontrivial organizational commitment to reliability and SRE.

Companies in which SRE teams are successful are ones which have made reliability a priority. They staff their SRE team(s) appropriately to have sustainable on-call responsibilities *and* long-term engineering output. They support the engineering project work balance of SRE teams by pushing back on the forces of entropy (interruption) that would erode the team's ability to have productive product output.

Where Did SRE Come From?

Site Reliability Engineering is, first and foremost, an outgrowth of the "always-on" world of online services. When customers or users are able to immediately detect service-impacting events, and when delivering either a fix or a new experience has blurred from a discrete "new version" delivery to a continuous process, critical measures for the services become the time to detect (TTD) problems and time to respond (TTR) to or mitigate (TTM) such events. SRE can trace some of its thought pattern lineage back to various historical precursors. However, the term was first applied to a designated role at Google in around 2003, when Ben Treynor Sloss and his team recognized that traditional approaches could not effectively scale to handle the massive growth of pervasive online services and began to apply software engineering approaches to the previously heavily manual processes of system operations. Besides being manual, these processes were frequently "bespoke," with custom work being done for each system, rather than following a more "factory" model of churning out hundreds or thousands of identical, largely interchangeable commodity systems (and systems to manage those systems).

Case Study: SRE at Google

According to Treynor Sloss in his keynote talk at SREcon14, the origin of SRE as a role at Google dates back to his assignment to run the "production" team in 2003. At the time, that team consisted of seven people. He was no more interested in the flawed approach of using ever more human labor (toil) to prop up badly functioning systems than any other developer, so he undertook the task of avoiding the historical divide between dev and ops by designing aligned incentives for both groups through objective data (SLOs). With a commitment to reliability that was backed organizationally from the top, Google set in place incentive frameworks that supported a balanced approach to reliability and new, shiny features: release control by SLO,[4] hiring and workload management practices that kept SREs from succumbing to operational overload,[5] and the outage-related goals of minimizing impact and learning from each event to prevent repeat occurrences.[6]

At the time that Treynor Sloss was creating the SRE team, Google also had a team that was known as "cluster ops" to tend to problems in the cluster systems that provided the foundational environment for all of the other Google services to function. Over time, the cluster ops team was upleveled to the SRE functions and eventually merged into the SRE team.

Google's SRE team has grown along with the size and scope of Google engineering: by late 2018 it included over 2,500 people. The SRE team at Google is also supported by technical writing experts and technical program managers, who have the same engagement/disengagement prerogatives that SRE teams have with feature teams.

4 As observed by William Gibson, the implementation of these principles was unevenly distributed.

5 The principle was "Drop urgent, nonimportant tasks if you can't make time for important, nonurgent tasks."

6 Private communications indicate that this "fully formed" picture of SRE did take some time to evolve and become generally instantiated throughout the organization. The fits and starts, the blind alleys, have been somewhat lost in the mists of time.

These types of specialized roles become important scale enablers as the teams grow in size.

What's the Relationship Between SRE and DevOps?

There have been lots of opinions expressed across various online and print media comparing and contrasting SRE and DevOps. Donovan Brown distilled one of the most widely accepted definitions of DevOps as "the union of people, process, and products to enable continuous delivery of value to our end users." Somewhat more expansively, Ryn Daniels and Jennifer Davis wrote:

> Devops is a cultural movement that changes how individuals think about their work, values the diversity of work done, supports intentional processes that accelerate the rate by which businesses realize value, and measures the effect of social and technical change. It is a way of thinking and a way of working that enables individuals and organizations to develop and maintain sustainable work practices. It is a cultural framework for sharing stories and developing empathy, enabling people and teams to practice their crafts in effective and lasting ways.

Drawing out the distinction between the most common DevOps practices and SRE, Jayne Groll wrote:

> DevOps focuses on engineering continuous delivery to the point of deployment; SRE focuses on engineering continuous operations at the point of customer consumption.

The formal definition of DevOps as reflected in Brown's description and in his more expansive blog posts exploring the topic differs from general industry practice, which limits the focus of "DevOps engineers" to the "continuous delivery" part of the software lifecycle (as noted in the preceding quotation). Having feature developers on call for incident response for their services in production use may be a part of the "we do DevOps" picture, or it may not.

The priority for SRE teams is on the "delivery of value to end users" portion of Brown's definition. For an online service, value can't and won't be delivered if end users can't rely upon accessing it—hence the importance of identifying and tracking service reliability. By focusing on value delivery, SRE provides a complement to teams that focus on developer productivity and CI/CD pipelines.

How Do I Get My Company to “Do SRE”?

You can’t buy SRE in a box or off the shelf.[7] One of the areas that companies struggle with is the paradox of the underlying simplicity of the principles and the difficulty of applying them. It’s like the tagline for the game Othello says: “A minute to learn…a lifetime to master.”

7 Much as “people sell devops but you can’t buy it”.

CHAPTER 2

Understanding the SRE Role

Culture/Capabilities/Configuration

In *Innovation Prowess* (Wharton Digital Press), George S. Day, a business professor at Wharton, identified a framework for the underlying components in highly innovative companies. He classified them into the "three C's":

- **Culture**. An organisation's shared values and beliefs, defining appropriate and inappropriate behaviours. It is often summed up simply as "the way we do things around here."
- **Capabilities**. The combination of skills, technology, and knowledge that allows the firm to execute specific activities and innovation processes.
- **Configuration**. The structure of the organisation, including how resources are allocated, who bears responsibility for achieving targets, and how success is measured.

Day explores the ways in which these components support each other to enable ongoing innovation practices that persist through both success and failure. While Day was focused on the aspects that lead to innovation, these same dynamics apply to reliability. Adapting his points:

> It takes sustained leadership and the long-run commitment of finance and human resources to build [reliability] prowess. Success begets success: Prowess improves the more it is applied....

> [The] elements are mutually reinforcing. They don't simply add together; instead they are multiplicative, as a weakness in one afflicts the others….
>
> Culture underlies and infuses everything an organization does….Culture and capabilities have a symbiotic relationship—one can't function without the other. They also have to be closely aligned to get superior results.

Culture

As covered earlier, clear and unambiguous support for site reliability is an absolute necessity for a successful SRE implementation. High (enough) executive-level support can be shown through organizational structures, but a *culture* that recognizes and rewards work that enhances reliability provides the environment in which SRE can be viable. When things are going badly, the organization must demonstrate its commitment to reliability by reallocating engineering resources across the board to address the deficiencies. Other important cultural components include:

- Fostering a learning mindset[1]
- Always looking for continuous improvement[2]
- Establishing psychological safety[3] to enable truth telling

Capabilities

For SREs, the ideal team member has a broad understanding of computer system dynamics—especially distributed systems. Effective SREs are able to zoom out to deal with system interrelationships and to zoom in, as needed, to debug the bit-level intricacies of networking or memory usage patterns.

1 See "Carol Dweck: A Summary of the Two Mindsets and the Power of Believing That You Can Improve" and her book *Mindset: The New Psychology of Success* (Ballantine Books). Also see Peter Senge's *The Fifth Discipline: The Art and Practice of The Learning Organization* (Doubleday).

2 See Chapter 11 of *Accelerate*, by Nicole Forsgren, Gene Kim, and Jez Humble (IT Revolution Press).

3 See Project Aristotle and Chapter 27 of *Seeking SRE* by David Blank-Edelman (O'Reilly).

They are adept at applying the leverage of coding and automation for scale. SREs need to have the ability to marshal data and present it to their partner teams in ways that are understandable within the context of the work priorities of the feature development teams.

The skills of empathy and compassion referred to in *Effective DevOps* are also important skills for an SRE because of the highly collaborative nature of the role. In order to minimize the impact of outages, SREs should be able to function effectively under the pressure of failing sociotechnical systems and both identify and implement methods to improve future responses.

While many individuals may exhibit some or many of these capabilities, the supportive underlying culture reinforces a learning mindset and supports actively practicing the skills. Active skill practice develops organizational "muscle memory," leading to greater capacity.

Configuration

Finally, in the *configuration* space, a strong SRE practice requires reporting structures that allow SREs to be evaluated and rewarded according to the distinct measures of performance that matter the most to them (not just how quickly features get shipped). Note that these reporting structures may be local distinctions, matrix-based, or a fully independent organization within engineering and still provide effective evaluation and recognition incentives. SRE success can be tracked across five areas that contribute to reliability:

- Providing useful *monitoring* frameworks that empower system understanding
- Characterizing, measuring, and improving *availability and performance*
- Accurately *forecasting capacity* requirements and improving efficiencies without undue impact on feature deployment velocity

- Improving *velocity* through reduction of toil and manual exception handling[4]
- Effectively *handling and learning from incidents*

Distinguishing SRE from Other Operational Models

SRE is the latest in a historical progress of operational models, so let's look at how it differs from previous approaches.[5] Just as earlier approaches were products of their time and context, so is SRE.

SRE Versus "Classical" SysAdmin

The system administrator (SysAdmin) role initially developed within the context of academic and research computing. In that context, SysAdmins benefitted from the deep systems knowledge around the role as well as the need to figure things out on their own when something went wrong.

Many SREs come from prior experience as SysAdmins. The troubleshooting skills and systems knowledge that they obtained from that background are highly valuable contributions to their SRE teams, but the focus of an SRE is more narrowly scoped than that of a SysAdmin.

SREs mainly focus on the operational characteristics of the applications that they participate in designing and supporting. Deep-level systems knowledge may be called upon to achieve the goal of service reliability or in troubleshooting aberrant application behavior.

SRE Versus "Classical" Ops

As computing was adopted into enterprise, bureaucratic contexts that retained components of Taylorist management constructs,

4 The antipattern version of this is referred to as "feeding the machine with the effort and toil of humans." Working in that way is not only inhumane but does not effectively scale, because you simply can't hire enough people to keep up with the demand—and it would be prohibitively expensive.

5 These characterizations are necessarily simplified in the interest of being succinct. There are many variations and a range of overlapping practice for all of the described roles which can make it difficult to distinguish one from another.

SysAdmins morphed into "ops" people. Ops is charged with keeping things running (stable) but not with doing the important "engineering" work.

Classical ops is divided by a mostly impermeable wall from the feature development teams. Depending on the age and size of the organization the barrier between the teams may be larger or smaller, but it tends to increase with the growth of specialization in the company's engineering organization. Usually, this divide between the two groups is accompanied by some degree of dismissiveness toward the "other" group. When this dismissiveness is carried up the management chain, the numerically smaller group (usually ops) ends up as the loser.

The SRE model restores the importance of engineering work that was lost with the ops phase and relies on close engagement with the feature development teams. Reliability is enabled by involvement throughout the full life cycle of a service—from conception through full production and on to retirement. Effective engagement with the dev teams is fostered by ensuring that incentives are aligned. All parties need to be constantly aware of the performance and reliability of the services. Any measures that insulate developers from the full costs (especially noneconomic costs) of keeping the service running in production end up building defensive and anti-productive walls between teams.

SRE for Internal Services

While most people may initially think of a company's products or services as mainly external-facing, SRE practices can be equally beneficial when applied to internal-facing platforms.

SRE for Backend or Platform Services

SRE is possibly even more relevant in a situation where everything becomes a platform than for standalone functionality (see Dave Rensin's 2017 SREcon talk for a discussion of why all online services are evolving toward being platforms for other services to make use of for their own needs and service levels). This "platformization" is more likely to be explicitly designed in the case of many internal services, and reliability aspects such as including SLIs and SLOs provide a common language to set expectations between teams.

Without well-established service expectations and management of outages, a microservice architecture or a system built on a large number of outsourced subservices turns into a fragile house of cards. User-facing services can't truly provide higher reliability than that provided by their critical backend dependencies, and those dependencies—particularly ones that may be several layers removed—may not be directly visible to or accessible by developers. Adequate feedback loops help to avoid cascading problems and the so-called "dark" debt within distributed systems.

SRE for Databases (DBRE)

In *Database Reliability Engineering* (O'Reilly), Laine Campbell and Charity Majors make the case that the collaborative principles of reliability for online services also apply to databases. In the case of a database, the prime directive is slightly adjusted from "site up" to "protect the data." See Table 2-1 for a comparison of the typical database administrator (DBA) approach with that of database reliability engineering (DBRE).

Table 2-1. Comparing classic DBA with DBRE

Classic DBA	DBRE
Strict separation of duties	Collaborative work, sharing the data protection responsibilities
Expensive and specialized software and storage hardware	Open source software, commodity hardware with durability requirements guiding the selection of each
Extensive change control processes	Automated procedures with rollback and impact mitigation components

SRE for Security

Just as an online service is pointless if it is not indeed "online" and accessible for its users, it is possibly even worse (harmful) if the service is not properly secured to protect itself and its users from attacks. For example, at LinkedIn the goal of what SREs do is "site up *and* secure."

There are a lot of people on the internet with nothing better to do than break into services or cause havoc, and as increasingly valuable information becomes accessible online, theft and destruction have become economically attractive activities. Because of this, reliability for security measures is even more important than for basic use features. Authentication and authorization are fundamental tasks within the realm of security, but they have often been approached on a more or less manual basis.[6] Developing automation, metrics, and monitoring in order to support "continuous security" can fall into the domain of SREs.

SRE for Internal IT?

Lampooned, underresourced, overcommitted, the plight of the typical corporate IT team is illustrated by Scott Adams's character Mordac, the Preventer of Information Services.

The importance of information technology to the daily work of practically everyone in a company is huge: "When the email [or substitute any other ubiquitous internal service] stops, everyone goes home."

Corporate IT has been populated by lots of specially built, one-off systems for unique purposes (the dreaded "snowflake" systems). Applying SRE principles *can* help to improve the delivery of value for internal business services, just as it helps with user-facing services. The SRE approach would bring such a rabble of differently configured systems into the regularity of an automated systems configuration. This regularity can help overcome the lack of investment with the efficiency gains of automation and duplication.

6 As a recent example, during the January–February 2019 shutdown of the US government, many government websites had TLS certificates that expired because no one was working. This caused a denial of service to anyone unwilling or unable to override the expired certificate warnings in their browsers. SREs would automate the certificate handling so that no manual work would be involved to keep the certificates current.

CHAPTER 3

Implementing SRE

When it comes time to implement a new SRE team, the main factor that contributes to the plan is whether you are starting "fresh"—a "greenfield" project—or taking a "brownfield" approach and migrating an existing team. In either scenario, the amount of cultural change needed can be daunting.

Even before a team is formed, one must prioritize the work to be done. A guide for figuring out where to start is the *Hierarchy of Reliability*. Since this hierarchy is going to guide the future changes, let's start by explaining what it is.

Hierarchy of Reliability

In late 2013, an SRE from Google, Mikey Dickerson, was asked to help the struggling HealthCare.gov. (To demonstrate the previous terms, he stepped into a situation that had a number of pieces in place, but the site as a whole was not functioning as desired. This is a great example of a "brownfield" scenario.) There was some intense time pressure to get things working quickly. He needed a way to explain "reliability" in a simple and straightforward way, so he borrowed from a theory in psychology, Maslow's Hierarchy of Needs. The Hierarchy of Reliability that Dickerson used to help HealthCare.gov is shown in Figure 3-1.

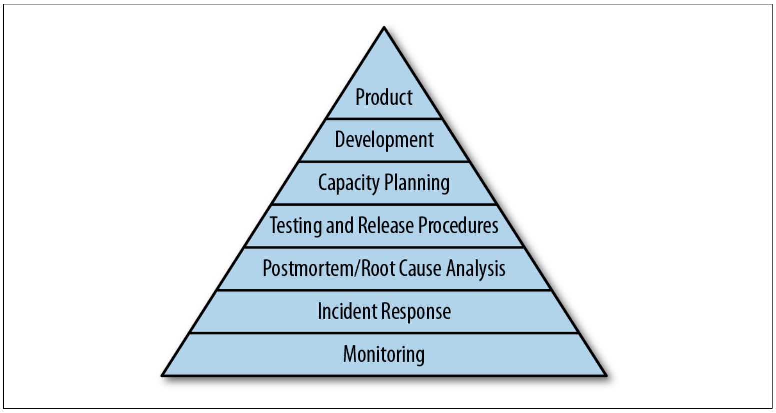

Figure 3-1. Hierarchy of Reliability

The idea is that topics at the bottom are more "basic," and they gradually get more advanced as you progress up the pyramid. But each topic (or "level," as we will refer to them) is not exclusively dependent on the levels below it. Rather, they build on one another. When each level is done well, then the other levels naturally benefit.[1]

As an extreme example, let's look at the very bottom ("monitoring") and the very top ("product"). Obviously, your company could have a product without monitoring. But nobody would know if, say, half of your customers only saw error pages, or if they saw the product (site) that you had designed.

While the levels used in Figure 3-1 are a proven set that a team can use to prioritize work, there are two things that we want to add.

In Chapter 1, we mentioned how being "data-informed" was critical to having valuable feedback loops. One of the key ways to gather data is through the various metrics that your software produces. While it is implied that (good) metrics are necessary for monitoring, we do want to call this out explicitly to enforce its importance.

Figure 3-2 adds that "metrics" level.

1 For a more complete discussion of the pyramid, see Part III of the book *Site Reliability Engineering*.

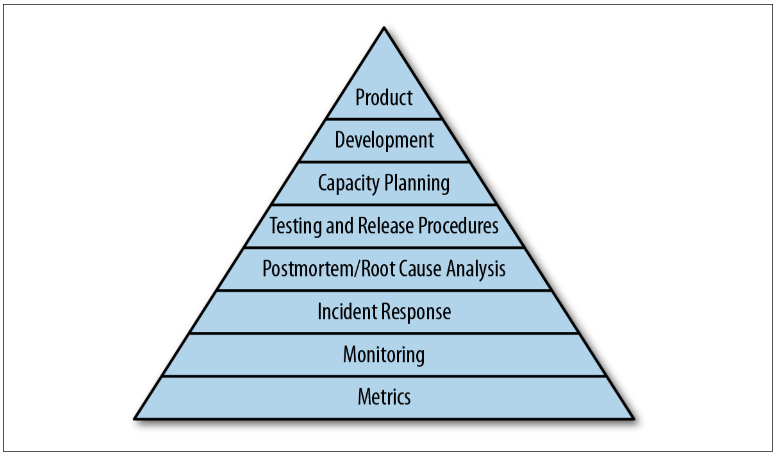

Figure 3-2. Hierarchy of Reliability with metrics

The other level to add has to do with the people that make up the team. Being on call can be very stressful. Even if there are no issues or outages, the people on call have to be available, which can impact the quality time they spend with their families and friends. And when an outage occurs, there is often immense pressure to get things working as quickly as possible. This can lead to long hours that drift into the early morning. Be aware of the impact this has on the people that work in the team. We add that "softer" level of "life" (food, sleep, family, etc.) in Figure 3-3.

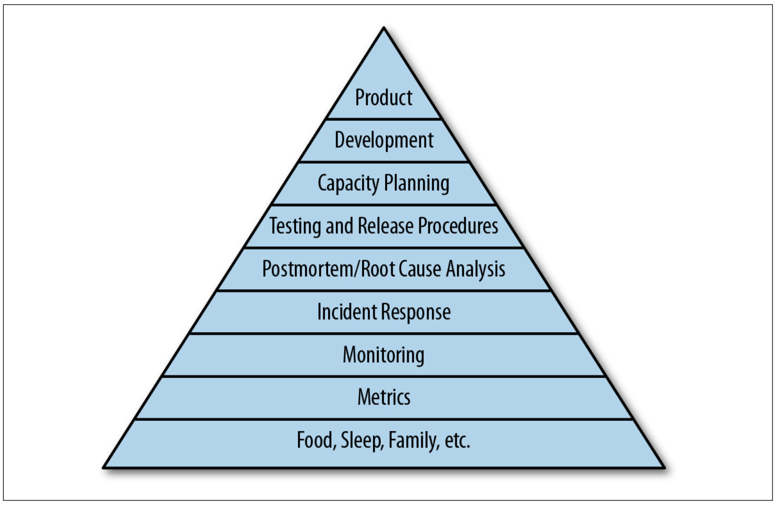
Product
Development
Capacity Planning
Testing and Release Procedures
Postmortem/Root Cause Analysis
Incident Response
Monitoring
Metrics
Food, Sleep, Family, etc.

Figure 3-3. Hierarchy of Reliability with metrics and life

The resulting pyramid presents a solid guide for the work that needs to get done to make a site (or system) reliable.

Starting a New Organization with SRE

We have already mentioned how critical it is to get buy-in from management. How far up the management chain one needs to go really depends on the size of the organization and the potential impact of the new team. As an example, if the team is going be driving a project that is critical to the long-term prospects of the entire company and the progress of that project (and, thus, the team) is discussed company-wide, then it becomes important to have the most senior portions of management sold on the concept of SRE. However, if the new project is implementing a small feature, then you may only need to convince the immediate manager to use the SRE model.

Before we continue, it is important to note that the hierarchy shown in Figure 3-3 presents a sorted list of tasks that need to be managed well in order to have a reliable site. If you are solid in every level, then one could say that you have implemented the SRE model. You don't have to hire people that specifically have the title "SRE" if the team is properly covering all of these bases. However, it is far too often the case that the engineers that are focused on feature development either have less interest or less knowledge when it comes to the details of the SRE role. Management needs to determine if they can deliver a reliable product with existing staff or need to hire specialists. In either case, the goal is to make sure the hierarchy is as solid as possible at every level.

In Chapter 2, we mentioned how "success begets success." Once you have demonstrated the value of the SRE model, either with existing staff or new hires, it becomes much easier to add SREs as the team grows. This success will be a result of building up those solid layers.

It's highly likely that at some point you will want to hire specialists (i.e., SREs). But hiring is always difficult. Arguably, this is even more true for SREs. A good SRE has the skills you would find in someone filling a "classic ops" position as well as the solid programming skills you would find in a software engineer from a product-focused team. Finding a good technical fit is only part of the challenge, though. A good cultural fit is just as important. This is especially true in a small, growing team because a single hire can have a drastic impact

—good or bad—on everyone else. However, in a larger company, the existing team members are likely to have established a culture that anyone new can fall into. The impact of every hire is just as important when one is trying to build a new team.

Once you've hired an SRE, it is important to enable that individual (and the entire team) to be successful. Make sure everyone understands the role of the SRE. The SRE is *not* the "ops person" for the team. It is easy for everyone to just hand off deployments, configuration management, etc. to this one individual. But if that happens you have implemented "classic ops," just at a smaller scale. The SRE is there to enable and empower every other engineer on the team. Each engineer is responsible for deploying their own code and managing their own configurations. The same goes for metrics, monitoring, etc. The SRE is the expert in these various aspects of delivering high-quality software. They are there to help other engineers with the details, but not to implement everything for them. Also, since the SRE is an engineer in their own right, they will be writing code to make these various processes simpler.

It is important to be aware of the type of work that the SREs on the team do. They are there to make the jobs of every other engineer easier and faster, but they are *not* just another software developer working on features. They need to remain focused on the overall reliability of the site. To assist in this, some companies use a "double reporting model." Essentially, the SRE works (and sits) with the development team, but they report to a different organization whose mandate is not product features. That other organization may or may not report to the same VP. Regardless of what common senior management exists between the development organization and the SRE one, it is important that SREs continue to focus on reliability and leave the product features up to other teams.

Case Study: SRE at Slack

Slack did not start, out of the gate, with an SRE model, but it has built one in the throes of hypergrowth in order to scale with the demand for its services.

According to Holly Allen, Slack had grown from just 100 AWS instances at the time of its initial public unveiling in 2014 to over 15,000 instances by late 2018 (just over four years later). In the same period, the company itself grew from less than 50 people to over 1,600.

As the company started specialization, its first phase was to have a centralized ops team that focused on provisioning cloud instances and building the Chef and Terraform tooling for automation. This team also served as first-tier response for all alerts and incidents. As Slack expanded, the ops/infrastructure team focused on infrastructure-related breakages and had to route any app-level incidents to the appropriate product teams.

The next phase of evolution was a reorganization of ops to "service engineering," with the inclusion of the internal developer tools team. The new combined team focused on figuring out how to push operational ownership of services back onto the dev teams so that the teams that could make the real code fixes received the incident alerts. Some feature teams had the skills to handle the on-call demands, but other teams needed training and hands-on help on how to handle the operational load. Slack created an SRE team to uplevel the operational capabilities of the dev groups.

Allen's talk illustrates one of the problems with excessive toil—in this case driven largely by low-quality, noisy alerting. SRE teams were so consumed by interrupt-driven toil from the noisy alerts that they were barely able to make any significant progress on improving the working conditions.

In September 2018, Slack explicitly committed to the importance of reliability over feature velocity and implemented a cathartic purge of all the historical, host-based alerting that was causing such a problem. Since then the SRE teams have been able to focus on making "tomorrow better than today" across the teams in which they are embedded.

Introducing SRE into an Existing Organization

Introducing any kind of cultural change into an existing organization is always difficult. As we mentioned earlier, existing teams often have a culture all their own. Altering that culture can take a lot of work. Regardless of the day-to-day challenges the team faces and how much desire there may be for something new, there is always comfort in "the devil you know." As a result, the people leading the change have to demonstrate that the place they are trying to get to is significantly better than where they are.

In a large organization with many teams, one way to accomplish this is to find a development team that is motivated to change and implement a small SRE team (or individual) there. Over time, you can use that success as a positive example to other teams. The idea is to focus all of your energy into that one product team that is leading the cultural change, and make sure it's successful with its transition. The members of that product team can then be advocates to their peers on other teams. Those advocates can explain what pains they went through and how much better things are with SREs.

This grassroots approach can be very powerful, as other engineering teams may be more pessimistic about an edict coming from upper management. Then those engineers can communicate their desire to try the new model with their managers. Hopefully, this will start a snowball of change throughout the organization.

However, it is rarely that simple. Even in the best of cases, there are likely to be some teams that simply do not want to change. At this point, management will need to step in, but they can still use the successful SRE implementations as the rationale behind any coming changes.

In the worst of cases, all progress is halted early on. At this point, it is up to senior management to move things forward.

Overlap Between Greenfield and Brownfield

The previous sections outlined some approaches for implementing SRE in both "greenfield" and "brownfield" situations. But it is rare that teams or organizations are that clear-cut. In fact, you can probably see some overlap in the previous discussion. The hope is that you can take inspiration from this discussion and come up with a strategy that works for your unique situation.

Case Study: LinkedIn

The SRE team at LinkedIn was created around 2010, when the company was about seven years old and staggering under an unsupportable weight of brittle, barely maintainable systems. Interestingly, the group of people who became the SRE team was about the same size as Google's initial formation team. As the number of users began to grow significantly, the site was experiencing daily outages during the morning usage peaks. New versions of the site required grueling merge gauntlets and inevitably broke when deployed into production. The “site ops” team of about 10 people was unable to keep up.

The SRE team was created and organized around three cardinal principles:

- *Site up and secure* is the prime directive.
- Everyone in the engineering organization should be able to safely deploy code.
- Operations is an engineering problem.

At around the same time, another team (now known as the *foundation* team) was chartered to develop consistent tooling and processes for the engineering org to use as they developed the site’s codebase. The foundation team focuses on engineering productivity, building and supporting the development environment tooling from base libraries, IDEs, and version control through the CI/CD pipelines into production.

As the engineering organization has grown, the SRE and foundation teams have also grown, with each now accounting for about 10% of the total engineering headcount. As the scale of the problems increased, so have the services and systems that are developed and maintained by the SRE organization in order to keep up with the demands of the site.

CHAPTER 4

Economic Trends Relating to the SRE Profession

For a profession that has only been a named role for about 15 years, SRE has grown into a significant force. Two SREs have even ended up on the cover of *Time* magazine.[1] Looking across a number of job posting sites, there are thousands of open positions around the world. Table 4-1 gives an idea of the numbers on some popular sites as of January 2019.

Table 4-1. SRE job listings, January 2019

Site	Number of listings
Indeed	5,985
Glassdoor	11,097
LinkedIn	2,032
Stack Overflow	1,384
Monster	2,289

Of course, some of the same job listings probably show up on more than one of those boards, but SRE is listed as one of the top 20 "Most Promising Jobs" in LinkedIn's annual reports for 2017, 2018, and 2019. The role has seen significant increases in the number of

1 Mikey Dickerson in 2014, and Susan Fowler in 2017.

job openings as well as median salary across the span of those three reports (base salary increased from $140k in 2017 to $200k in 2019).

Another perspective on the growth of the profession can be seen in Figure 4-1, which shows the attendance numbers for the USENIX SREcon conference series that began in 2014.

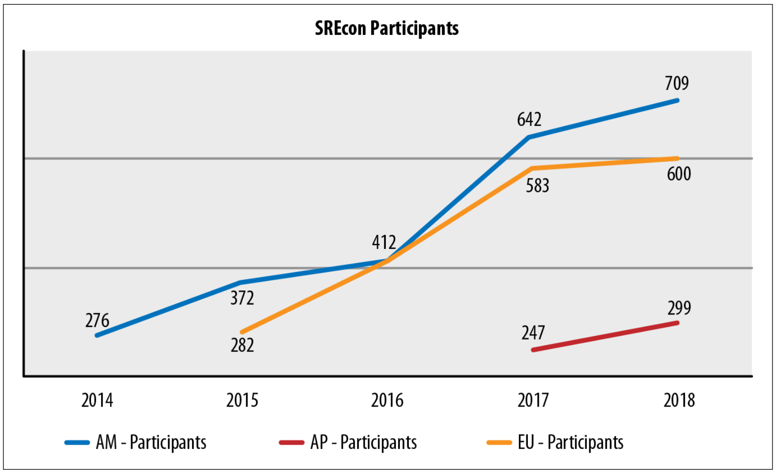

Figure 4-1. Attendance at SREcon conferences

Lex Neva's newsletter *SRE Weekly*, which covers blog posts and other online articles of interest to the profession, has seen similar growth from its beginning in 2016 (Figure 4-2).

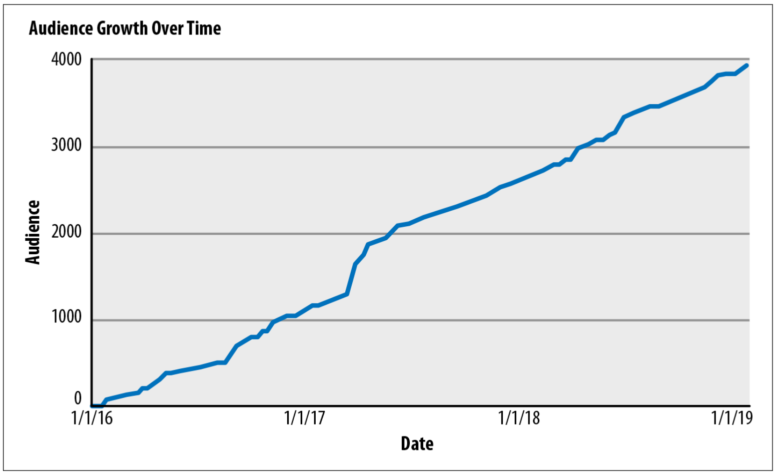

Figure 4-2. SRE Weekly subscriber growth

While SRE can help for every online service, the growing adoption of cloud-based “always on” technologies and teams distributed around the globe serve to highlight the need for reliability even for traditionally hard-to-use internal IT tools. Engineers who can build for and support reliability will be in ever-increasing demand.

CHAPTER 5

Patterns and Antipatterns of SRE

This IS NOT SRE

There are many ways that an attempt to implement SRE practices and teams can go wrong. You can find more on Twitter and in Chapter 23 of *Seeking SRE*, but here are some key problems to avoid:

- Changing the name of any existing team (usually "ops") to "SRE" without making the organizational adjustments required to enable them to do meaningful development work
- Using the SRE team to shield devs from the pain of how their services really function in production
- Failing to contain interrupts
- Attempting to do SRE project work without the same support (such as project managers, technical writers, etc.) that any other dev team would have (because SREs only spend 50% of their time on project work, we contend that support structures are even more important for SRE teams to make effective use of their development time)
- Valuing (perhaps simply through call-out recognition) incident response heroics over prudent design and preventative planning
- Implementing processes or systems that slow down the delivery of value to customers without incontrovertible benefit
- Building a "gatekeeper" team that functions as a chokepoint
- Static or ill-considered SLOs

- Thinking that SRE is a point solution to a particular problem rather than a fundamental cultural shift

This IS SRE

Hearkening back to the beginning:

> SRE is an *organizational model* for running reliable online services by teams that are chartered to do *reliability-focused engineering work*.
>
> As a discipline, SREs are devoted to helping an organization *sustainably* achieve the *appropriate level of reliability* for its services by implementing and continually improving *data-informed production feedback loops* to balance availability, performance, and agility.

Does it make sense for your company to commit heavily to reliability and pursue the implementation of SRE in your organization? Only you and the other leaders in your company can answer that question. Some companies will be at a size where having a distinct organizational component or team just does not fit, but the principles can be put in place to provide a foundation for the future.

Just like with any new methodology or cultural shift, when implementing SRE it will take time, grit, and humility to adjust to the changing circumstances—but the payoff will be an institutionalized commitment to the importance of the user's interaction with your site, service, system, or other "online stuff." Over time, with the SRE team(s) consistently representing reliability and operability concerns as well as actively contributing to the product codebase to improve reliability, feature developers will learn to factor these pieces into their plans as they develop new features. At that point, SREs will be able to shift their impact to a deeper and wider level, making next month's problems different from today's.

Our hope is that this brief introduction to Site Reliability Engineering will have provided you with an effective understanding of the what and how of SRE. There are lots of resources available to dive into greater detail. We've listed some of the best starting points for further reading in Appendix A.

APPENDIX A
Further Reading

- *Site Reliability Engineering* (aka "the SRE book"), by Betsy Beyer et al. (eds.)
- *The Site Reliability Workbook*, by Betsy Beyer et al. (eds.)
- *Seeking SRE*, by David Blank-Edelman (O'Reilly)
- *Effective DevOps*, by Ryn Daniels and Jennifer Davis (O'Reilly)
- *DevOps Defined*, by Ryn Daniels and Jennifer Davis (O'Reilly)
- "How SRE Relates to DevOps", by Niall Richard Murphy et al.
- *Database Reliability Engineering*, by Laine Campbell and Charity Majors (O'Reilly)
- "Introducing Database Reliability Engineering", by Laine Campbell and Charity Majors (O'Reilly)
- *SRE Weekly* newsletter, by Lex Neva
- *Accelerate*, by Nicole Forsgren, Gene Kim, and Jez Humble (IT Revolution Press)
- "Interested in Becoming a Site Reliability Engineer?", by Tammy Butow (for an idea of the topical areas that an SRE would be expected to be familiar with)

About the Authors

Kurt Andersen is a part of the Product-SRE team at LinkedIn. He has been one the co-chairs for SREcon Americas and has been active in the anti-abuse community for over 15 years. He also works as one of the program committee chairs for the Messaging, Malware and Mobile Anti-Abuse Working Group (M3AAWG.org). Kurt has spoken around the world on various aspects of reliability, authentication, anti-abuse, and security. He also works on internet standards through the IETF and serves on the USENIX Board of Directors and as liaison to the SREcon conferences worldwide.

Craig Sebenik is currently an SRE at Split Software. He has worked at several startups over the years, and a few large, well-known companies (including LinkedIn and NetApp). He is the author of *Salt Essentials* (O'Reilly) and has spoken at LISA, SREcon, and SaltConf. Craig also has a passion for cooking. He earned Le Grand Diplôme from Le Cordon Bleu, a master's degree in Italian cuisine from Apicius (Florence, Italy), and a master's degree in gastronomy from the University of Reims (France).