O'REILLY®

Compliments of Google Cloud

Training Site Reliability Engineers

What Your Organization Needs to Create a Learning Program

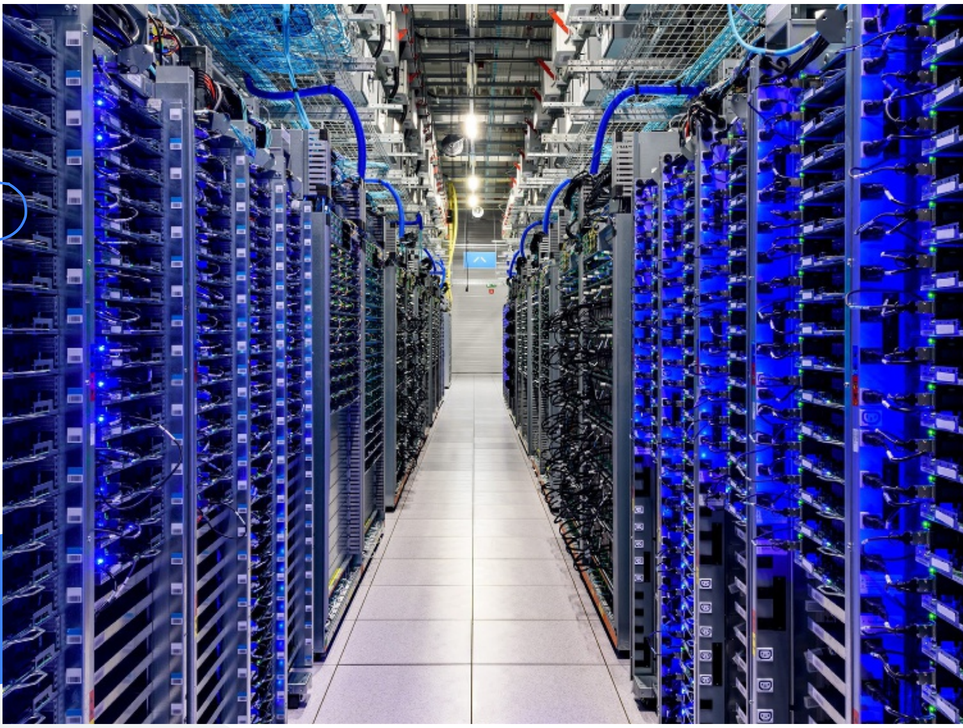
Jennifer Petoff, JC van Winkel & Preston Yoshioka
with Jessie Yang, Jesus Climent Collado & Myk Taylor

REPORT

Google Cloud

Want to know more about SRE?

To learn more, visit google.com/sre



Training Site Reliability Engineers

What Your Organization Needs to Create a Learning Program

Jennifer Petoff, JC van Winkel, and Preston Yoshioka, with Jessie Yang, Jesus Climent Collado, and Myk Taylor

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Training Site Reliability Engineers
by Jennifer Petoff, JC van Winkel, and Preston Yoshioka, with Jessie Yang, Jesus Climent Collado, and Myk Taylor

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

Acquistions Editor: John Devins
Development Editor: Virginia Wilson
Production Editor: Beth Kelly
Copyeditor: Octal Publishing, Inc.
Proofreader: Charles Roumeliotis
Interior Designer: David Futato
Cover Designer: Karen Montgomery
Illustrator: Rebecca Demarest

November 2019: First Edition

Revision History for the First Edition
2019-11-15: First Release

See *http://oreilly.com/catalog/errata.csp?isbn=9781492076001* for release details.

978-1-492-07598-1

[LSI]

Table of Contents

Preface

This report discusses how to train Site Reliability Engineers, or SREs. Before we go any further, we'd like to clarify the term "SRE." "SRE" means a variety of things:

- Site Reliability *Engineer* or a Site Reliability *Engineering team*, based on the context (singular, SRE, or plural, SREs)
- Site Reliability Engineering *concepts*, *discipline*, or *way of thinking* (SRE)
- Belonging to an SRE individual, team, or way of thinking (SRE's or SREs')

Ben Treynor Sloss, the founder of Site Reliability Engineering at Google, describes SRE, or the Site Reliability Engineering discipline, as what happens when "you ask a software engineer to design an operations function." The traditional systems administration model of software management in production requires an organization to scale the number of operators as the service increases in size and complexity.[1] SRE is able to scale humans sublinearly with the scale of the services they are supporting. This is done by applying proactive engineering solutions to eliminate repetitive, no-value-added tasks and toil.

We assume that you're familiar with the concepts in the *SRE Book*. As we were growing our SRE department, we noticed how difficult it was to get new hires up to speed. You might be surprised to find that technical skills are not necessarily the most important skills to

1 For more context, see *https://oreil.ly/53yK0*.

have. Without a doubt, troubleshooting is an important part of incident management, but we also show that growing the students' confidence, explaining the importance of good relations, and encouraging clear communications with other SREs and dev teams are essential to bringing your students up to speed.

In this report, we share our experience ramping up new SREs, but we also look at other use cases. For example, we have talked with several smaller organizations that are successful in ramping people up to do SRE (or SRE-like) functions.

Training should be purposefully designed and not just thrown together, without any thought. Therefore, we also discuss the theory behind the training design. You need to know what your training needs are and who your audience is. Set clear learning objectives and build your training content based on that. We've seen that making SRE training hands-on is extremely important for building the confidence of the students who ultimately go on-call[2] for a production service.

Finally, when teaching how to "SRE," we should implement the practices of SRE while administering the program: that is, "SRE" your SRE training program. In other words, monitor the results and be willing to adjust the training program if the monitoring shows it is necessary. We show that just like SRE has a hierarchy of needs, SRE training also has a hierarchy of needs, which follow SRE's needs.

While much of this report focuses on the specific experience of Google SRE, we aim to present best practices and lessons learned over the past several years, which can be applied to organizations that are at varying points along the spectrum in terms of size and maturity.

Acknowledgments

The authors would like to thank Google SRE EDU team members past and present for shaping the program and our ways of working including David Butts, Ben Weaver, Laura Baum, Brad Lipinski, Andrew Widdowson, Betsy Beyer, and Rob Shanley. We'd also like to acknowledge the small army of volunteers who teach for SRE EDU

2 On-call involves being available to address production issues during both working and nonworking hours: *https://oreil.ly/hQmCp*

and those who volunteered cycles to make our 'breakable' photos service a reality.

Jennifer Petoff: Thank you to Phil Beevers for timely review and feedback and to Nat Welch and Steve McGhee for giving their perspective on training practices that are important for organizations working to adopt the SRE model.

JC van Winkel: Many thanks to SRE leadership who gave the proposers of SRE EDU their trust, have supported the team through the years, and helped us build on our initial success.

Preston Yoshioka: Thank you to Evan Jernagan and Moira Gagen for input and mentorship.

CHAPTER 1

Identifying Your SRE Training Needs

Providing training and education for site reliability engineers is universally important to set them up for success in your organization. However, the specific training needs of each engineer varies depending on several factors:

- The maturity of your organization in adopting SRE principles, practices, and culture
- The knowledge those individuals have about your organization and infrastructure
- The experience of the individuals being trained, both in terms of technical skill and familiarity with the SRE model

These dimensions make up a matrix (see Figure 1-1) that describes different use cases for SRE education. The optimum training solution for your SREs varies, depending on the specific use case. In the sections that follow, we define each of the key dimensions.

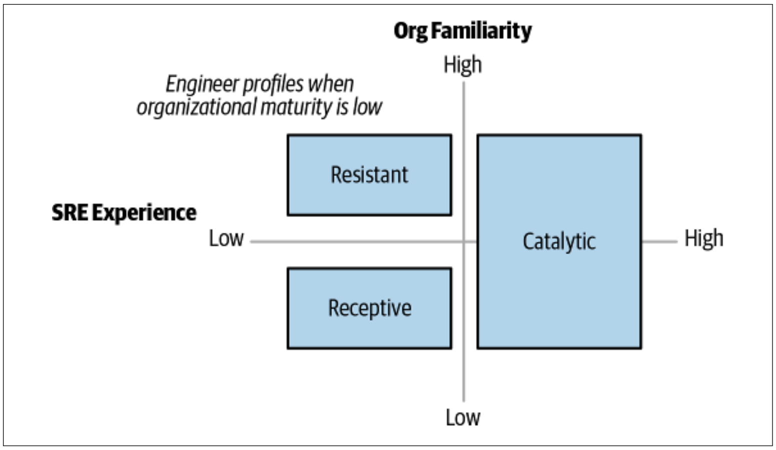


Figure 1-1. Matrix of SRE training use cases: low organizational maturity

Organizational Maturity

Organizational maturity is considered low if your organization has not yet adopted SRE principles, practices, and culture.[1] Organizational maturity is considered high if you have a well-established SRE team, or if SRE principles, practices, and culture are widely understood, implemented, and embraced. An organization with high SRE maturity is expected to have the following:

- Well-documented and user-centric service-level objectives (SLOs): a target level of reliability that should ideally be correlated with customer happiness.
- Error budgets: a budget for failure. The error budget is the difference between perfection and your SLO, allowing teams to move as fast as possible, as long as the budget is not exhausted, but with defined actions that will be taken to improve reliability if the production service falls short.
- A blameless postmortem culture: recognition that things will go wrong and human errors are really systems problems.

1 For purposes of this discussion, SRE principles, practices, and culture are taken as the key elements laid out in *Site Reliability Engineering: How Google Runs Production Systems*.

- A low tolerance for toil. According to the *SRE Book*, “Toil is the kind of work tied to running a production service that tends to be manual, repetitive, automatable, tactical, devoid of enduring value, and that scales linearly as a service grows.”

Organizational Familiarity

High organizational familiarity means that an engineer has worked for your company for a considerable length of time (at least a year or more). Low organizational familiarity means that an engineer is new to the company. Organizational familiarity determines how open or resistant an individual engineer might be to training content and what types of content are most important to consider (Figures 1-1 and 2-3).

SRE Experience

High SRE experience means that an engineer has worked as an SRE at your company, or elsewhere, for a number of years and understands the core SRE principles, practices, and culture outlined in the *SRE Book*. New university graduates are an example of engineers with generally low SRE experience because SRE concepts are not often taught in school. Experienced software engineers in product development, systems administration, and others making a career change into SRE are also considered to have low SRE experience, when evaluating SRE training needs.

Note that the training solution you choose also depends on the following:

- The size of your organization
- The speed at which your organization is growing
- The resources your team has to spend on training

Now that we’ve defined a framework that describes the types of students who you might encounter, let’s describe the skills these students might need or want to develop.

Types of Skills to Develop

Apart from “obvious” related subjects that SREs need to learn (such as the technical infrastructure and practical troubleshooting skills when on-call), there are other peripheral subjects related to SRE that help employees become better SREs. In the sections that follow, we use some Google-related examples of training that drive the development of specific skills.

Skills That Support a Career Shift Toward SRE

According to Ben Treynor Sloss, vice president of 24x7 at Google, “an SRE’s job is to apply software engineering skills to operations problems. This means that we expect SREs to spend a lot of time on software engineering. However, Google also hires highly qualified system administrators who’ve been in a more Ops-oriented function, with a bit of scripting experience, but no true software engineering experience. For these people, offer (software) engineering training, or pay these folks to take an external course at a university or online.

Sometimes, an organization makes a decision to change the use of a certain technology or move to a different technology, which then requires extra education. Here’s an example. At Google, as in the broader world of software engineering, we’re seeing more new projects being created in the Go programming language, particularly in SRE. Therefore, many SREs need to learn Go, and an education effort is needed to quickly ramp people up in it. In general, requirements change and SREs are expected to change with them, so it’s important to provide education and learning resources for SREs.

Finally, there’s a skill that’s difficult to acquire outside of companies with large complex systems: Non-Abstract Large System Design (NALSD). This a critical skill for SREs, as described in the SRE Workbook. In NALSD, we consider how to design large systems for reliability, resilience, and efficiency. NALSD is not only used when building a completely new system, but also when systems need to be changed due to changing requirements or growth. For example, a global service was not sharded initially, because when it was first designed, the number of users was small. However, as the user base grew, the new growth forced a redesign of the global service, such that it was sharded. It is important for SREs to demonstrate an appreciation and awareness of future scalability traps and why simplicity is critical for smooth operability and disaster recovery. Focus on building experience and judgment, not simply more algorithms.

The SRE approach and skillset is something that is useful for non-SRE developers in a company, as well. It's useful for the developer community to learn about SRE principles and practices, including large system design. This helps the developers build more resilient software. The training material for the developers is probably less extensive than a complete onboarding curriculum for SRE. The people involved with SRE education are the ones best suited to supply this material.

Troubleshooting Skills

It's important for an SRE to keep their troubleshooting skills sharp. Therefore, SREs should regularly be on-call for the service they support. Too much on-call, however, can burn them out. Too little, on the other hand, can cause them to lose familiarity with the service and troubleshooting processes. While on-call, they'll encounter outages that need to be resolved. SREs tend to like solving puzzles, have an inquisitive nature, wonder why things are the way they are, and follow an analytical approach to solving problems. During troubleshooting, it's important that SREs follow a scientific method: formulate one or more hypotheses and then rule some out. We at Google teach our SREs that troubleshooting is a series of failures, and it's OK to go through the process of *not* figuring out the problem, especially if it helps them rule things out. It's very important that SREs who are on-call know that they are not alone—they might be the first responder, but they are backed up by a large group of engineers whom they can ask for help.

Good SREs try to see the bigger picture—they try to find correlations with other outages. This (potentially) helps find the root causes of multiple incidents. This contrasts with solving only the immediate problem at hand. Running regular "Wheel of Misfortune" sessions also sharpens SREs' troubleshooting skills.

Training That Supports a Culture Shift and Promotes Trust

For SRE training at Google, we pay a lot of attention to the culture of trust between developers and SREs, and between different SRE teams. SREs and developers both share ownership of the service and user experience. Users receive the best service when we balance launching new features with reliability. To achieve this, we need a healthy relationship between SREs and the developers they work with. Through interviews with different teams, we've found that communication, agreement, and trust are paramount to healthy SRE–developer relationships, and the best functioning teams are those in which it's barely known who is an SRE and who is a developer—who you are is defined by what you do, not your job title.

Not only is a good relationship with a service's development team important, other teams are often relevant, as well: security, for example, because some major leak has been found and a change needs to be rolled out on short notice (in a reliability-safe way). Or when the privacy team has found out about a product for which Personally Identifiable Information (PII) is not erased or anonymized in a timely fashion and a Spanner database needs to be cleaned up. Here, too, it's important for students to learn how to work with other teams and respect that their requirements might sometimes be at odds with the goals of SRE.

Because SREs often must communicate with many different teams, it's important that SREs communicate effectively. When we train our SREs, we create cohorts in which the student encounters many other students who will work in other teams. This way, after training they will already know people in other offices and teams who can help them as the need arises. We encourage students to build a network and often see that the mailing lists we create for these cohorts are used by the students for a long time afterward. The students feel comfortable using these mailing lists to ask questions because they already know one another.

Incident Management Training and the Corresponding Soft Skills

In major incidents, the on-call person who was initially paged ropes in more people to help. This requires careful coordination and communication. At Google, we follow the *Incident Management at Google* (IMAG) protocol, which is a flexible framework based on the Incident Command System (ICS) used by firefighters and medics. IMAG defines how to organize an emergency response by establishing a hierarchical structure with clear roles, tasks, and communication channels. It establishes a standard, consistent way to handle emergencies, and organizes an effective response. Implementing incident management training is a good idea so that new SREs understand not just the technical troubleshooting elements of responding when something goes wrong but also the command and communication framework that is in use in the organization.

Soft skills are also important during an incident. Usually, when people go on-call for the first time, soft skills are less high on the agenda. Soft skills include things like explicit and clear communication, time and task management, and record keeping. In practice, mastering these skills is as important for timely incident resolution as mastering the technical knowledge. Therefore, consider developing training that teaches students how to spot hidden assumptions in incident communication that commonly cause misunderstandings with other people on-call; delegate tasks effectively with explicit communications; and think one step ahead by considering what would happen if they carried out a certain action.

Finally, no matter how skilled and knowledgeable the on-caller is, there comes a time when they feel overwhelmed by the problem and don't know what to do. It might seem unorthodox, but including training on human factors in incident management helps students understand how their body works when under stress—cold sweat, trembling, difficulty concentrating, loss of motivation, feeling tired, fatigued, exhausted, and ultimately perhaps, when stress levels get high enough, freezing and not being able to do anything. Such training helps students understand how to monitor themselves for these symptoms, recognize the danger of the last phase, escalate in time, and hand off the incident to someone else before the last phase actually occurs.

An Introduction to SRE Training Techniques

We've discussed a variety of topics that you might want to cover in your SRE training. We now discuss ways to deliver that training. There are many techniques for equipping SREs with critical skills, especially when they are new to an organization and ramping up to become productive in supporting specific systems. These techniques vary widely in sophistication and level of effort required on the part of those delivering the training. Figure 1-2 shows training techniques with regard to the level of effort to apply that technique.

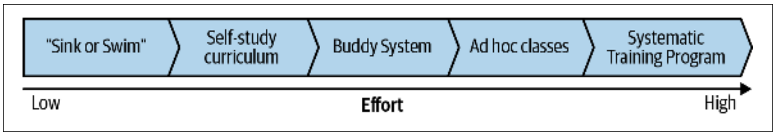


Figure 1-2. SRE training techniques plotted along a continuum from low to high effort

Sink or Swim

On the "low effort" end of the spectrum, there is the "sink or swim" model in which onboarding consists of telling a student to figure things out on their own. Throw your new person into the job on Day 1 with the expectation that they will learn by doing, without a specific framework for ramp-up. Because there are no guiding principles or guardrails showcasing what an SRE new to the team needs to know, "sink or swim" could also be described as "grokking SRE the hard way." Although "sink or swim" is a low investment approach, it's not a very inclusive approach, and it does not aim to set every new member of the organization up for success.

Why isn't "sink or swim" inclusive? As we discuss more in the section on theories of instructional design and adult learning, different people learn best using different learning modalities. Self-directed learning is just one modality. Others include lectures and hands-on exercises. "Sink or swim" leaves students guessing about what they should be focusing on, provides no guidance on what the learning objectives are, and generally leads to a higher level of stress and imposter syndrome.[2]

Self-Study

One step up from "sink or swim" on the spectrum of techniques for training SREs is to provide self-study materials. These materials can be documents, videos, or exercises. Typically, the SRE receives a checklist of things that are useful to know, with associated resources linked to the checklist. The latter items on the checklist might build on the knowledge learned from previous items. Even though self-study is better than "sink or swim" because SREs are at least given some guidance on materials and/or curriculum, there are some

2 Imposter syndrome is a psychological pattern in which an individual doubts their accomplishments and has a persistent fear of being exposed as a "fraud."

downsides to self-study material that can be frustrating or overwhelming. The SRE consuming the material may feel like they are on their own because they are left to learn on their own (albeit in a guided way), without an explicit channel for asking questions or getting support when they become stuck.

There is also a risk that an SRE encounters out-of-date or deprecated material. This is particularly problematic for students, and it can occur if no one is actively curating the self-study checklist. The student does not realize that some material is deprecated or out of date. We have seen examples where an experienced SRE walks by a student's desk, notices that they are watching a video recommended in the student checklist and says, "Oh, that thing has been deprecated for years. I wouldn't bother watching that." The student then feels like they have wasted their time, which leads to high degrees of frustration. It also contributes to a general lack of trust in the self-study materials.

Another downside of self-study training materials is that they can be more difficult to maintain, especially video formats. In this case, experience with video editing software is required or completely new recordings need to be made at some frequency to ensure that self-study training materials are kept fresh and up to date.

Buddy System

Training SREs, especially new SREs, can be enhanced by providing one-on-one mentoring and shadowing opportunities. Well-maintained self-study materials combined with a mentor who is an explicit point of contact for answering questions helps the new SRE have confidence in the training materials and not feel like they have no guidance and support. Shadowing an experienced team member and then having the experienced person reverse-shadow the student when the time approaches for the student to go on-call is a useful training technique and a variation of the *buddy system*. The buddy system also fosters experienced team members' confidence in the skills and abilities of the new person on the team.

Ad Hoc Classes

Ad hoc, in-person classes or whiteboard sessions are another approach to training SREs. Because this approach entails a live person giving a class, it requires more ongoing effort than self-study options. This can be particularly burdensome for small teams with few potential ad hoc instructors. However, this approach provides a useful structure for students, and an opportunity to have questions answered. Members of the team might maintain ad hoc slide decks on different aspects of the organization's infrastructure that they deliver as needed. Less formally, whiteboard sessions in which an experienced team member draws a system diagram that outlines key elements of the infrastructure and key dependencies and how they work requires less overhead.

As an added bonus, have someone new on the team teach back what they've learned about the system from their own exploration, combined with self-study and whiteboard sessions from experienced team members. The team as a whole often learns from this approach. Oftentimes, the new member of the team learns something about the system or some recent change that even experienced team members didn't know. The "teach back" approach ensures that the entire team has the most up-to-date mental map of how the systems they support work in practice. Teaching is the best way to learn (see "Teaching to Learn" on page 12) and is an important feedback mechanism to ensure that the student understood the material.

Systematic Training Program

If your organization is large enough or growing fast enough, it makes sense to invest in a systematic training program to ramp-up SREs on different topics. Creating an SRE training program ensures reliability and consistency in the ramp-up experience throughout your organization. Investing in a systematic training program that brings people together in person is also important for organizations driving a culture shift to SRE. Culture must be modeled in person—this is difficult to do with self-study formats. An organization trying to adopt SRE using a lower-touch training approach such as self-study might find this to be counterproductive. If possible, it's better that the training is done in person because that sends a signal that the organization really cares about the change and the development of its employees, leading to a higher probability of success.

For large organizations, program operations become more important. Program operations are the "how" of the training program. Let's draw an analogy to software development for which the "what" is the product features and the "how" is deploying to production in a reliable way to meet the needs of users. In the case of training, the "what" is the training content itself and the "how" is deploying it in a consistent and reliable way that meets the needs of students. Just like SRE focuses on the "how" of software development, we discuss how to apply SRE principles to training in Chapter 5.

A systematic training program allows an organization to build cohorts of new SREs. By putting people through the program together, people feel that "I'm not in this alone." This helps fight imposter syndrome and builds the confidence of new SREs.

A formal training program for SREs should be systematic, not just in operations but also in class materials. Consider building a centrally curated curriculum. We discuss more about how to build and curate an SRE training curriculum in Chapter 4 and Chapter 5.

SRE training can be in-person (at least to start with) and then move to video or video conference. Each approach involves trade-offs between effort and effectiveness. It's easier to obtain cycles for learning from engineers when they are new. The longer an SRE is with the organization, the more demands there are on their time, so prioritize in-person training as much as possible while people are new to the team. Impatient managers are also a concern. If you run an in-person training program, you might get push-back from managers who want their new team members to get started on the team as soon as possible. The risk of manager impatience and push-back increases with time, as evidenced by lower completion rates and higher cancellation rates the longer an engineer has been in the organization. For example, at Google, we achieve 99+% coverage of our target audience in an orientation program delivered in the second week on the job, whereas completion rates for classes related to incident management and getting ready to go on-call, which are delivered a few months after the new engineers start, drop to 50%.

With a formal training program, it's important to keep in mind inclusivity, especially if travel is involved. For example, limiting training to one week, with the option for people to travel on Monday and go home on Friday, shows consideration for SREs with family or other personal obligations. In fact, in some countries, business travel must be limited to working hours.

Although distributed training (e.g., by video conference) can be appealing because it requires less time from both the students and developers of the training, it's important to be aware that attendance and engagement decline for distributed training, compared to an in-person training model. Distributed training is not zero cost: there is the cost of logistics (meeting room bookings, getting the training on people's calendars, recruiting instructors). The main savings are on student travel time and in organizing travel, if that is centrally managed. However, doing training in a distributed way means, in our experience, that students are more likely to become distracted and not pay as close attention, or not show up at all.

Teaching to Learn

Teaching is, in fact, the best way to learn.[3,4] Take advantage of volunteer instructors and draw on former students to teach new students. This approach helps build a strong team and community and keeps people involved in education across the life cycle of an SRE.

It's very costly to hire full-time instructors, especially when the topics being taught are very technical and require in-depth knowledge. Hiring full-time instructors basically cannibalizes engineers who could be working to run your infrastructure. Instead, consider *crowd-sourcing* instructors. Volunteer instructors spend at most a few hours a week (in the case of an extremely large and rapidly growing organization) paying it forward to help others ramp-up on selected topics. For the volunteer instructor approach to work, incentives are important. For example, consider recognizing volunteers at a company or department all-hands meeting or distribute limited edition corporate swag.[5] Of course, there is also the innate incentive that if an experienced team member helps a new person ramp-up, that person would be ready to share the on-call and project load required to support their services faster.

Even better, teaching and knowledge sharing should be explicitly called out in the SRE role description. These community contributions should be taken into consideration when awarding raises and promotions. Being explicit about the importance of teaching shows

3 Koh, A. W. L., Lee, S. C., & Lim, S. W. H. (2018). The learning benefits of teaching: A retrieval practice hypothesis. Applied Cognitive Psychology, 32(3), 401–410, *https://oreil.ly/Qlb1h*.

4 Duran, D. (2017). Learning-by-teaching. Evidence and implications as a pedagogical mechanism. Innovations in Education & Teaching International, 54(5), 476–484, *https://oreil.ly/YO_W5*.

5 Swag is a common Silicon Valley term for promotional merchandise branded with a corporate or team logo.

that the company is serious about making the training program a success.

In a nutshell, if you are part of a small organization with limited resources and are growing slowly, focus on supported self-study techniques. If you are part of a larger organization, in-person classes using volunteer instructors are more effective. If you are large *and* growing rapidly, invest in a full-fledged training program with consideration for *how* the training is delivered in addition to *what* is taught. Sink or swim is never a good option and doesn't set new members of the team up for success.

Conclusion

In this chapter, we talked about identifying your SRE training needs. We introduced the Organizational Maturity Matrix and discussed what type of skills to develop. We also introduced some SRE training techniques and which approach might work best for your organization.

CHAPTER 2
Use Cases

Let's move on to explore different SRE training use cases, along with optimum approaches and trade-offs to consider. We consider these in terms of the effort/effectiveness continuum that we discussed in Chapter 1 (refer to Figure 1-2).

Organizations Adopting the SRE Model

We first discuss the use case of an organization adopting the SRE model.[1] Here, *organizational maturity* is low, as demonstrated in Figure 2-1. Organizations usually want to adopt SRE principles because they value the benefits of combining high reliability with high feature velocity while achieving lower organizational friction at the same time. Service-level objectives (SLOs), error budgets, and upholding blamelessness when things go wrong make this possible.

1 By the SRE model, we refer to the approach described in *Site Reliability Engineering: How Google Runs Production Systems*.

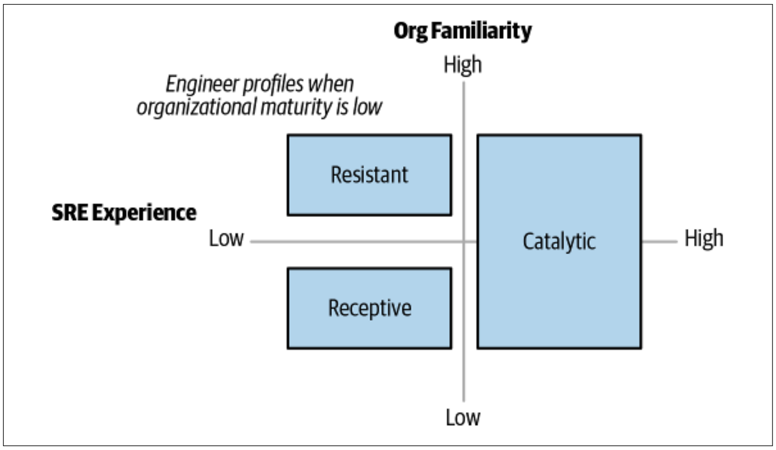


Figure 2-1. Matrix of SRE training use cases: low organizational maturity

Without a doubt, adopting an SRE practice involves an organizational culture shift. Some organizations have been tempted to change the name on the door from "Ops" to "SRE" and declare victory. However, undergoing a shift to SRE is not a branding exercise (i.e., changing the name on the door) but rather a *fundamental culture shift* that requires buy-in from stakeholders across the organization, all the way up to the very top, in order to be successful.

Building a Training Program to Drive Adoption of the SRE Model

Training plays an important role in setting the organizational transformation up for success after you have buy-in. Organizations that plan to adopt the SRE model must consider the following factors when building a training program:

- Gap between current skills and target skills in the existing workforce
- Receptiveness of the affected individuals to this change
- Building trust in the SRE approach

Here are some diagnostic questions to assess potential skill gaps and determine what training topics help the most for driving successful adoption of the SRE model:

- Does your workforce have a software engineering mindset?
  - Does your team use a defect tracking system (e.g., Jira, bugs)?
  - Does your team have a project planning model (e.g., Agile)?
  - Is work done in a version control system (e.g., Git)?
  - Are solutions clearly articulated and vetted by team members (e.g., with design docs) before starting implementation?
  - Does your team have a shared repository of commonly used tools and libraries?
- Do the people running your services today understand large systems design best practices?[2]

If your answer to the majority of the questions above is "no," it's important to start with foundational training to address any skill gaps[3] that might be a barrier to adopting the SRE model.

Encountering Resistance

Without actively investing and supporting current members of the team, you'll likely encounter resistance to change because people feel threatened that they'll simply be replaced. The reality is, even if their historic job function is replaced, there is *much* more work that still needs to be done in an SRE practice—it's just different. Point out examples of the work at a higher layer and train your staff to be ready for it early on (e.g., creating classes such as "Welcome to Terraform/Pulumi" or "Intro to Prometheus"). Pair training on these concepts with the logic that the new way of doing things exerts more leverage (i.e., you'll actually get a lot more done by doing this work, you'll be more fulfilled, and you'll have a bigger impact by working higher in the stack) is also a good idea. As you can see, upskilling existing talent is critical to successfully introducing the SRE model.

In one instance encountered by Google's Customer Reliability Engineering team, a company attempted to transition from a traditional Ops model to SRE. Their Ops division was content with 100% toil

2 See "Introducing Non-Abstract Large System Design" in the SRE Workbook.

3 There is a wealth of free content available on online learning platforms like Coursera to help build the necessary curriculum to bridge any skill gaps.

load and regarded that as job security. They then stated that their load was too high to spend any time on SRE training. Management had to figure out how to promote the SRE concept to this group of system operators who were resistant to change (i.e., recalcitrant Ops). Management took the following approach: it helped the team automate a small portion of the team's toil and then with the extra time saved by automation, tasked it with automating the rest of its toil on its own. Management put the operators in charge of their own tools so they never felt threatened. This gave the operators the opportunities to develop the software engineering skills needed for the SRE role.

Because SRE adoption requires a fundamental shift in thinking, training inherently focuses on SRE principles, practices, and culture[4] rather than the specifics of the production systems that SRE teams support. We assume that the team already has deep technical knowledge of the infrastructure—the biggest barrier to adoption is understanding what SRE involves, and convincing people to make the required shift in mindset.

Receptive, Resistant, and Catalytic Individuals

For an organization working to adopt SRE, you'll likely encounter *receptive*, *resistant*, and *catalytic* individuals (see Figure 2-1). The most receptive people in your organization are likely those who are less familiar with the SRE model and haven't been in your organization very long (low organizational familiarity and low SRE experience). For people who have been in the organization for a long time (high organizational familiarity), there can be a tendency to think "we've always done it this way," and be skeptical—or even resistant—to the change. Often times, the move to an SRE model follows on the heels of some other organizational transformation (e.g., Agile), which might have left some scars. You'll need to convince the resistant people in your organization that this time it's different, describe what's in it for them, and explain why the change is beneficial.

People in your organization who have experience with SRE at another company can be the catalysts you need for the change you're trying to drive. They presumably believe in the SRE approach and

4 SRE principles, practices, and culture are consistent with those defined in the *SRE Book*.

can speak on how SRE is an effective way to operate systems at scale. Take advantage of these catalysts to build trust in the SRE approach and showcase how to think like an SRE. Give these individuals the opportunity to tell their stories. Get them involved in teaching classes about SRE principles, practices, and culture. Your catalysts are important for establishing an SRE mindset and culture in the organization.

Benefits of Implementing SRE

Implementing SRE provides many benefits. These include pushing back on toil and resolving technical debt so that individuals and the team are less overloaded and less stressed, less boring/repetitive work, and more opportunities for career development. If a company implements management and structural changes in the organization to support transition to SRE (a key element that drives success), the team also has a "seat at the table" with the organization's development, product, and business teams, which is crucial given that SRE teams collaborate across teams by nature. Without such equal footing, an SRE team quickly finds itself beholden to development or product timelines and prioritization and cannot properly maintain reliability.

Convince Teams of the Benefits of the SRE Model

Training managers on specific techniques to convince teams of the benefits of the SRE model is helpful. For example, managers can introduce methods of tracking pager load, initial postmortem templates and review sessions, keeping on-call rotations sustainable, and collecting toil surveys early on. These show early commitment to making a positive change, before committing a team to a whole new path. Simply saying "Doing SRE will be great, trust us" doesn't work without tangible methods and early wins to point to.

Organization Size and Rate of Growth

For the organization moving toward the SRE model, the size of the organization and growth rate influence exactly how you bring the culture shift to SRE, as shown in Figure 2-2.

In large organizations, the most stress is on the culture shift. If the growth rate is low, focus the most effort on convincing members of the organization that the SRE approach is useful and has benefits for both individuals and the organization as a whole.

If the organization is large and growing rapidly, use this fact to catalyze change. Build a robust SRE orientation program for your new hires, focused on SRE principles, practices, and culture plus the specifics of your production systems. New people have fewer preconceived notions about how things should be and might ask naive questions about specific existing processes and approaches, which can drive change.

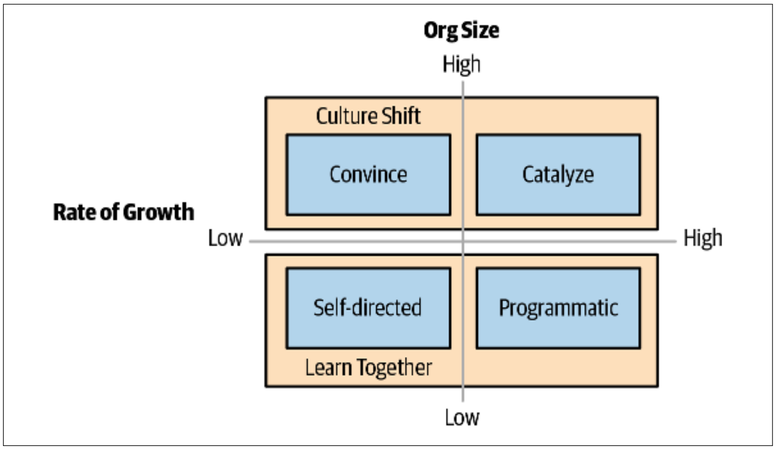


Figure 2-2. The influence of organization size and rate of growth on training best practices, when organizational maturity is low

If your organization size is small, an approach to learn together is most likely to be effective. In a low-growth situation, consider low effort approaches like starting an SRE book club, bringing in guest speakers, or hosting a party to watch a video, using recorded talks from recent SREcon conferences. Create a discussion group where the team meets and discusses how the principles, practices, and culture of SRE can be applied within your organization.

If your organization is small and growth rate is high, invest in a programmatic training program, with a focus on SRE principles, practices, and culture, as well as specifics of your production infrastructure. This ensures a consistent and reliable ramp-up experience for new members of the team, and helps deliver the desired culture shift to SRE.

Organizations with an Established SRE Team or Teams

There are different training considerations that arise when your organization has an established SRE team or teams, illustrated in Figure 2-3, which we explain in depth in upcoming sections. As before, we frame a few different use cases along the dimensions of organization familiarity and SRE experience. For organization familiarity, you might consider the following:

- How well does the individual know your organization?
- Are they new or have they worked at your company for a while?

For SRE experience, you might think about these questions:

- How well does the individual understand SRE principles, practices, and culture?
- Have they worked as an SRE at another organization before joining yours?

The exact approach taken to training these different use cases once again depends on the size and growth rate of your organization.

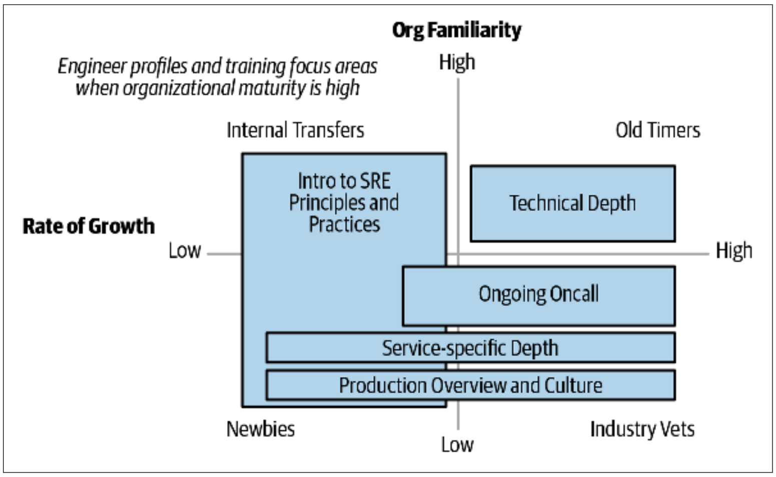

Org Familiarity
Engineer profiles and training focus areas when organizational maturity is high
High
Internal Transfers
Old Timers
Intro to SRE Principles and Practices
Technical Depth
Rate of Growth
Low
High
Ongoing Oncall
Service-specific Depth
Production Overview and Culture
Newbies
Low
Industry Vets

Figure 2-3. Matrix of training use cases: high organizational maturity

Newbies

The *newbies* use case is personified by those who are both new to your organization and new to the practice of SRE. A typical newbie might be a recent graduate who studied computer science or a related field. Newbies might also include someone new to your organization who is going through a career change.

In addition to specific training on the production infrastructure they'll be supporting, a solid foundation on SRE principles, practices, and culture is critical. For example, the Google SRE EDU team has created an orientation program that includes classes highlighting specific features of the SRE role, and a class on "Launches, Rollbacks, and Postmortems," in which we demonstrate the importance of a blameless culture. SRE principles such as setting user-centric SLOs and corresponding error budgets with consequences are also sprinkled throughout the more technical classes that we offer. We describe the SRE EDU Orientation program in more detail later in this report.

Old-Timers

Old-timers are people who've been at your company for a long time and are also long-time SREs. The biggest training need for this audience is technical depth. Offer elective classes and tech talks to pique their interest. Training for this audience is focused more on self-discovery, and should be less prescriptive or programmatic. Training on important, large-scale production infrastructure changes is also particularly important for this audience.

Internal Transfers

Members of your organization who've been at your company for a while but have limited SRE experience make up the *internal transfers* training use case. Again, the training focus should be on SRE principles, practices, and culture, and less on the specifics of how to deploy code to production. Refreshers on incident management best practices is also important, depending on whether the internal transfer has been on-call for a service prior to joining the team.

Industry Veterans

Engineers who are new to your organization but have SRE experience elsewhere make up the *industry veterans* training use case. It's important to show industry veterans how things are done within your company and how that might be different from what they've encountered elsewhere. If you are running a systematic training program that is instructor-led, you'll need to guard against derailing if industry veterans want to dispute the finer points of the tools used to do the job, for example, debating on the merits of Google Cloud Platform versus Amazon Web Services (AWS) or Microsoft Azure. Respect the knowledge of the industry veterans and the industry best practices that they share with the team, but give them a chance to experience how things are different at your organization.

Choosing Your Training Solution

For companies with an established SRE model, organization size and growth rate factor in to choosing the optimum training solution (see Figure 2-4). If growth rate is low, focus on a training plan with a lower time investment, such as shadowing and mentoring for new people. For organizations that are large but slow growing, ongoing education needs of the entire team should be the focus. If your growth rate is high but your baseline organization size is relatively low, focus on training new people on the team with a systematic SRE orientation. Establish what foundational training all of your SREs need across SRE principles, practices, and culture, as well as the fundamentals of your production systems. If your organization is both large and fast growing (as is the case for Google SRE), invest in a full life cycle training program with the following elements:

SRE orientation
: This covers SRE principles, practices, and culture, plus common technical elements across your production infrastructure. Consider this the foundation that teams build upon.

Service-specific training
: These are the elements that an SRE needs to understand to be able to effectively manage the specific production services their team is responsible for. This goes into a lot more depth, and it is usually the teams themselves who are best placed to define the curriculum.

Service-agnostic going on-call training
: Train new hires on incident management best practices and the specifics of how incident management works at your company.

Ongoing education
: This includes in-depth tech talks on a variety of topics, and a preview of upcoming, large-scale changes. These are important for keeping engineers' technical skills sharp, and potentially helping them in career mobility if they choose to change teams.

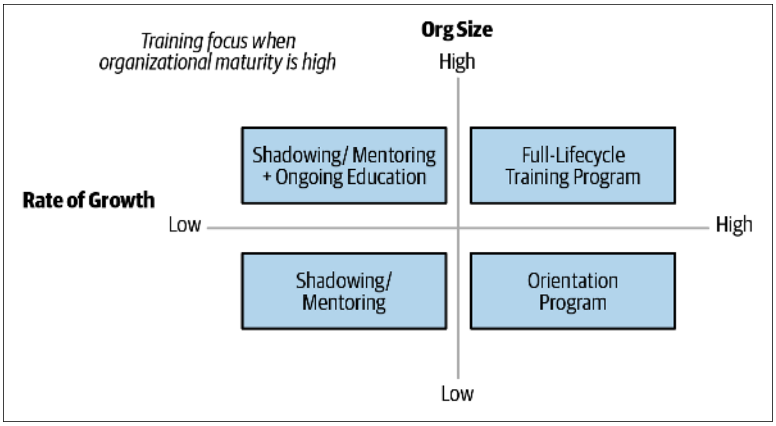

Training focus when organizational maturity is high
Org Size
High
Shadowing/ Mentoring + Ongoing Education
Full-Lifecycle Training Program
Rate of Growth
Low
High
Shadowing/ Mentoring
Orientation Program
Low

Figure 2-4. The influence of organization size and rate of growth on training best practices when organizational maturity is high

New Team Members on an Existing SRE Team

Onboarding new team members depends on the size of the organization. Large SRE organizations have more resources, and potentially a larger influx of new hires. Starting an SRE job, coming from either a software engineering or a system administration/Ops position, requires some adjustments. In this section, we look at some of the adjustments, and see where training helps align people with SRE.

Instill Key Elements of SRE Culture

A major benefit of SRE is that SRE enables faster feature development with an acceptable level of risk. Whenever a system is not undergoing change, the chances of it failing are minimized because most failures happen when the system is changed. However, without change, the service stagnates. SRE helps achieve the proper balance between reliability and feature velocity. However, this can be done only with the appropriate mindset.

Engineering mindset

In SRE, we don't want to just temporarily solve problems, outages, and incidents—we want to ensure that they don't happen again. The mindset of not just a quick hack and move on, but actually fixing the root cause of the problem, requires engineering effort. It might require designing and writing a tool or digging into the existing code and making changes to prevent the problems from happening again.

Embrace failure

One hundred percent reliable systems are impossible to build or prohibitively expensive. Perfect systems would also not be very interesting because in order to achieve perfection, changes to the system would need to cease. Given that we need to make changes to deliver functionality to our users, systems are bound to fail periodically. SREs expect systems to fail and see failures as an opportunity to improve on them in an economical way. They don't see a systems failure as their fault but as an inherent and inevitable part of running a complex distributed system.

Large-scale systems

If you run very large systems, something will always break. For example, suppose that the longevity of a disk, when using it 24x7, is three years. If you have 300,000 of these disks, on average, 100,000 of these disks will break per year. Because there are about 30 million seconds in a year, this means that every 300 seconds (or five minutes), a disk will break. This constant occurrence of hardware failure does not happen only with disks. If you have a large number of computers, the same thing happens for the computers and their power supplies.

To put this into perspective, according to Statista, in 2020 the total amount of disk space for all cloud providers will be 340 exabytes. Using 10 TB disks, that is 340 million disks. Therefore, in cloud storage worldwide, a few disks break every second.

Given such a large scale, it's not possible to manually repair every breaking part. For example, if a single machine is responsible for 14 disks, if one disk fails, you won't send a technician to that machine to replace the disk. Rather, you turn the disk off and mark it unavailable. When there are "enough" broken parts, you have a technician do a sweep and visit all the machines that have broken parts to replace them.

Likewise, Google's cluster management system automatically marks unresponsive machines as bad and avoids scheduling jobs there. There is no need to spend human effort keeping track of that. When there are enough machines marked as bad, a technician makes a sweep along the bad machines. The software gives a suggestion of the most probable resolution; this can be as simple as reseating a connector! This way, we make more efficient use of the technicians' time.

It's important to instill the notion in students that everything fails, including hardware and software, when used on a large time and physical scale. It forces them to think about *defense in depth*—one single reliable feature is never enough; rather, multiple layers of reliability measures must be taken.

Some breakage noise is good for testing and practicing the self-healing systems.
There is an upside to components breaking continuously: it keeps us sharp. Suppose that we had a system in which hardware failures happened only once every few years. We would not know whether our automation for handling broken hardware would actually work. It's good to stay practiced. This is also true for software systems. We aim for less than two incidents per on-call shift, but not close to zero—people who have no practice forget how to handle incidents.

Learning to first triage, then mitigate, and finally resolve

We want on-call engineers to follow a specific pattern, which we call TMR: Triage, Mitigate, Resolve. During training, we want students to practice TMR several times so that it becomes natural. In our environment, the tool that is used to handle incident and response management keeps track of what state the incident is in (prompted by the responder). The tool calls these stages Triage, Mitigate, and Resolve as well to help students remember them.

The first thing an on-call engineer should do (after acknowledging the pager) is triaging. The on-call should consider:

- What is the impact?
- Are end users affected?
- Is the problem global or very local?

Impact can also be estimated as severity multiplied by time. If the outage is minor and you can handle it by yourself, look at how to mitigate. Mitigation involves fast, nondestructive, and reversible actions such as getting temporary emergency resources, having the load balancer direct traffic away, and so on. Of course, the mitigation should be verified; for example, did we indeed receive emergency resource quota? The mitigation buys you time to investigate and gather data for a short-term fix and root-cause analysis.

A short-term fix might possibly be destructive, such as rolling back a change, restarting a job, or deleting poisonous data. For these cases, you should be able to undo the mitigation, like removing the traffic drain, and returning emergency quota. The short-term fix buys you days or weeks of time. Of course, verification is again necessary. This also buys you the time to start writing the (blameless) postmortem with the data that has been gathered.

Only the root-cause analysis, and following repair of the root cause—or, better yet, the class of causes this root cause was a member of—makes sure the incident does not happen again and that it has been resolved. Until the resolution has been implemented, the same trigger of the cause might happen and create a similar incident.

Blamelessness

Blameless postmortems help you to find root causes. If blaming is involved, people tend to sweep things under the rug, which hinders finding out the underlying causes, thereby indirectly allowing the same incident to happen again.

It can be difficult to explain to people without any SRE experience why postmortems should be blameless. People by nature tend to be worried about blame. One big help is making clear the distinction between root cause and trigger. Suppose that a person has put a flowerpot dangerously close to the edge of the balcony railing. This would be the root cause of the flowerpot falling down. The trigger might be a cat walking by or a little gust of wind. You cannot blame the cat or the wind for the pot falling.

Another way to think of blamelessness is cost. Think of production incidents as unplanned investments for which all the costs are paid up front. You might pay in lost revenue or lost productivity. You always pay in user goodwill. The returns on that investment are the lessons you learn about avoiding (or at least reducing the impact of) future production incidents. Blameless postmortems are a mechanism for extracting those learned lessons. For example, maybe build a ledge around the balcony railing so that flowerpots cannot be shoved over the edge.

Teaching people about blameless postmortems is done by having students read actual postmortems, see the impact the outage had, notice how the description in the postmortems did not allocate blame, and finally, observe how the root-cause analysis led to bugs/tickets being filed. Reading those tickets shows that something was actually being done about the root cause.

Celebrating failure (including the resolution) can illustrate how blamelessness works in practice. For example, we show a video of someone who gave a talk about how they caused a major outage, but by rolling back a change quickly, they limited the damage. This person also explains that they were actually rewarded for their actions. They were the trigger, not the root cause, and it was their swift action that was rewarded. People have called this video inspiring.

Beating down toil

Toil has many causes. For example, toil can come from manual actions that need to be done or recurring minor incidents. Having to handle a lot of toil is bad for motivation, work quality, and careers.

Fighting increasing toil by diluting work over multiple people does not work. Adding more people does not scale—it's much better to make the toil go away. Again, getting to the root cause helps, and that is something we need to actively teach our new SREs. To illustrate: suppose that a certain application crashes regularly; for example, when users give it corrupted data. When that happens, you go through a manual and tedious process of getting the crash dumps and investigating them. Suppose that once you've done that, it's relatively easy to do a quick fix to handle the problem. To reduce toil, a first step would be to spend less personnel on fetching the logs; this tedious process can be automated. After the humans have the logs, the process is relatively easy; for example, skip processing that specific bad data. A better solution would be to dig deeper and find out what the problem really is—that is, find the root cause and fix it (e.g., make the application more resilient to bad data in general). Of course, having the “get the log files” automation might come in handy later, but you must decide carefully whether it’s worth it.

Teaching people to take a critical view on the procedures they use, and fighting against the mentality, “but this is how we have always done it,” can turn out to be very useful in reducing toil.

“Making tomorrow better than today”

For situations in which finding the root cause is not applicable, automation is very useful to make work less tedious. Of course, automation is not a panacea—you should use it when it actually saves time and human toil.[5] Examples of areas where people should think about automation are Continuous Integration (CI) and Continuous Deployment (CD), testing, and anomaly detection.

Build confidence

The most important goal of SRE training should not be the technical tools and systems. Feeding people too much technical information leads to information overload. Much more important is building confidence in the new SREs. Give people the feeling that they can indeed “do this.”

Practice, practice, practice gives confidence. This is where practice plays an important part: teach about a certain tool, have the students use the tool and systems, and accomplish something they will do later in their job. Only teach as much about the tool as actually needed to finish the exercise—the students will get a feeling of accomplishment. This is also where the difficulties lie for the instructors. As described in Chapter 3, “Training of instructors” on page 41, instructors are worried that the information they teach is too shallow. However, that is because they are focused less on the goal of “building confidence,” and more on pumping as much information as possible into the students’ brains.

5 See “"Is It Worth the Time?”

Know how stress works and how to fight it. Preparing new SREs to "go on-call" not only involves providing technical knowledge, teaching procedures, and building confidence, it also involves educating SREs on how to deal with stress. During their career, SREs at some point will be involved in an incident in which the stakes are high (e.g., the main site is down and the company is losing money every second). This is when SREs need to "keep their cool," but the human body does not always cooperate. When stress levels rise too much, there comes a moment when the human brain "shuts down." We want to train SREs that this can happen so that they can recognize the symptoms and escalate to someone else before it's too late. In the same way, we want our SREs to watch over their colleagues so that when they see one of their colleagues display symptoms of being overstressed, they take action. At Google, the aviation industry inspired a training called "Human factors in incident management" (see Chapter 1, "Incident Management Training and the Corresponding Soft Skills " on page 6). This training is often seen as a wishy-washy, soft skills class at first, but after students attended the class, many said it changed their mind. The class explains some of the physiological mechanisms in the brain, which is (almost) out of your control. This explanation appeals to scientific and technically inclined students.

Fight imposter syndrome. Another thing we must prepare our new SREs for is feelings of impostor syndrome. Many new hires have accomplished much in previous endeavors, whether it be in university or in previous jobs. Now they come into an environment so big that it's impossible for them to be an expert in everything, and there are a lot of experts (in specific areas) around them. They might feel that everyone is better than them. We warn our new SREs that this is something that can happen to them, and as a matter of fact, a majority of SREs at Google encounter this at some point in their career at Google. Simply knowing that imposter syndrome exists is already a step to overcoming it and building confidence as an SRE.

The value of a cohort. At Google, we deliberately group new hires together into cohorts and set up an easy communications channel for them (a mailing list per cohort). Having people in the cohorts exchange experiences helps them to see that the impostor syndrome they feel is something that happens to many, many people and also shows them that they are not alone in this.

The cohorts also help new hires build a network within the company. Knowing not only the people on your team, and on the partner team in another office, but also other new SREs on other teams and offices, is useful when they need to reach out to other teams. When they have a question about some technology their team's service depends on, it's easier for them to reach out to a cohort member who happens to work on that team. They already know that person, and though this person might not be a Spanner expert already, for example, they most certainly know who in the Spanner team should be able to help.

Inclusiveness. We want all our students to be able to perform well in their job. Many factors contribute to performance, and some of these we select for during the recruiting process. After people have been hired, there are factors we can influence. Psychological safety, feeling "at home" in a team, and being respected and appreciated are all important contributors to performance. Therefore, we must create an inclusive environment in which everyone feels at home and knows that they can rely on their colleagues, despite all the differences there might be among people.

Different people have individual, preferred working modalities and we should aim to support them. Some people are "morning people," whereas others are "evening people." Having a diverse set of people on the team actually helps—the "morning people" might have an easier time communicating with people in more eastern time zones, whereas the "evening people" might find it easier to communicate in more western time zones.

Likewise, we must understand that different people learn in different ways (see "Learning Modalities" on page 61). Therefore, we must offer learning materials in different ways. We offer classes with plenty of hands-on learning, but all the materials can also be read. Some classes are videos (which can be played at 1.5 speed, if so desired) and of course, people can explore any documentation we have on their own. Offering training and training material in different formats is critical for building an inclusive organization.

Better collaboration among teams. People in teams know a lot about their team's "homemade" tools and procedures. This leads to the risk of all the teams "doing it their own way," causing more duplication of work, and siloing. An extreme example of this is that half a decade ago, more than a dozen different monitoring (and dashboarding) frameworks evolved within Google, built on top of Google's core monitoring systems. Some of these were created, knowing there were other efforts, but these were considered "not good enough" (also known as the *N*+1 problem,[6] in which by consolidating options, the end result is you've added one more option). Others were created out of local necessity and global ignorance.

To create more uniformity and collaboration within and across teams, there are several things you can do:

- Have people teach classes or give talks about how they do X. People who are not aware of other efforts can learn about them. The implementations that are obviously better are adopted by more people. Also, more teams begin using the solutions others made, preventing people from starting a new way to do X to begin with.
- Have a group of people discuss the current standards to do X, and merge the best parts. (This can lead to version *N*+1)
- Enforce mandates. People do not generally like this, but in combination with the previous suggestions, it can be effective. A mandate can also be of the kind: *we will not force you to migrate to the merged version, but all the others will no longer be supported.*

Another advantage of having these shared foundations is that it creates a common language. It's easier for people from different teams to collaborate, and it stimulates mobility. People moving from team to team need to learn less new localisms. It stimulates a common skill set that is beneficial for all teams, and it allows easier reuse of components. An extreme example is Google's Remote Procedure Call (RPC) infrastructure.[7] Because there is just one infrastructure, you can call any RPC service in the company to which you have access rights.

6 For an excellent illustration of this trap, see *https://xkcd.com/927/*.

7 See The Production Environment at Google, from the Viewpoint of on SRE in the *SRE Book*.

Experienced SREs Transferring to a New Team

Google encourages mobility between teams. Frequently, SRE team members move to another team after one and a half to three years on a team. These people do not need to be taught the general infrastructure, but they do need to learn about specific infrastructure that the team uses, which the previous team did not (e.g., a specific storage solution). Of course, the systems the new team is running should be covered extensively, as well.

This means that there needs to be learning material for the following:

- Infrastructure not covered by general onboarding orientation or for which orientation does not offer enough depth
- Team-specific material (as described in Chapter 3, "Team-specific training" on page 42)

These requirements are the same for inexperienced people onboarding a team (such as new hires). *Content*-wise, there is less difference between a new hire and an experienced person, but *confidence*-wise there is. If an experienced SRE moves to their fifth team, they need to learn about, say, Spanner, but they won't need to build up their confidence as much to get it done, whereas new hires' confidence needs to be built up. However, the Spanner material will be the same for the experienced SRE as for the new hires.

Training for infrastructure that is not used by the general engineering audience is usually produced by the SRE team responsible for that infrastructure (such as a Spanner 101 class produced by the Spanner SRE team) or by a team that has a special vested interest in that subject matter (such as the largest consumer of a specific database technology).

These classes should be available as broadly as possible and be available to any SRE as continuing education. This helps with the following:

- Mobility from team to team
- Dissemination of infrastructure changes (e.g., from version 1.8 to 2.0 of a product)
- Adoption of new facilities and platforms which stimulate uniformity in production

Mobility is important because it forces teams to have their documentation complete and up to date to set new team members up for success. Having new people join a team, with insights from their time on other teams brings new viewpoints into the team. New team members are more likely to object to a statement such as, "But that's the way we've always done it." Finally, mobility prevents single points of failure in a team: if a person has been on a team for eight years and has become the "local expert on everything," it becomes a disaster if that person leaves the team for whatever reason. It's better to spread the knowledge throughout the team and the broader organization.

After some time, a company might try to consolidate many different technologies that have independently been developed by different teams, for example, monitoring and alerting, CI and CD, automated testing, and so on. Having continuous education helps with the adoption of these "horizontal" efforts, creating more uniformity—and therefore mobility—between teams.

Experienced SREs at a New Company with an Existing SRE Culture and Practice

Experienced engineers face other challenges when they change companies to join an SRE team there. Culturally, there are differences. The new company might do things better, but the new SRE doesn't see that yet. It can also be the other way around. No matter what, the new company must have a way to imbue the new SRE into the SRE culture of the company. Especially if the engineer comes from a place where postmortems were not blameless (or postmortems were not written at all). There is a lot of trust to win with regard to the SRE way of doing things.

Learning happens best when doing lots of hands-on activities, even for experienced people. Therefore, people with industry experience (see "Industry Veterans," Figure 2-3) should go through the same orientation for SRE as the new university graduates (see "Newbies," Figure 2-3). When you set up groups of students, make sure that the groups all have a good mix of experienced people, new grads, and transfers from non-SRE teams in your organization (see Chapter 3, "Current state of SRE orientation at Google" on page 39). Then, let the new grads drive (hands on keyboard) during the hands-on exercises, with the industry veterans having a more guiding role. For us at Google, this has worked out very well. We expect more leadership skills from more experienced people compared to the new grads.

Sometimes, people with industry experience might ask disruptive questions because they have seen similar infrastructure and want to compare the infrastructure to the technology they are used to. It can be difficult for instructors to keep the class focused on the story that was laid out for that class. The instructor must then (respectfully) "dodge" the question, and during a break in the class, discuss the answer to the question separately. The instructor should also request that the person ask these types of questions during breaks, and not during class.

Conclusion

In this chapter, we talked about SRE training use cases, and what are the optimum approaches and trade-offs to consider for these cases. We explored organizations along the spectrum of organizational maturity and considered how to convince various types of teams to adopt the SRE model. We looked at different scenarios of what happens when SREs join a team, discussed instilling elements of the SRE culture within team members, and saw what that looks like in practice. In the next chapter, we get into details about how SRE training works in organizations of all sizes.

CHAPTER 3

Case Studies

Let's take a look now at how SRE training has been done in practice. We discuss training activities in place for organizations along a spectrum, from very large to very small. We use Google as an example of a large SRE organization; for the medium and smaller organizations, we look at other companies.

Training in a Large Organization

Google's SRE training program provides a case study of one possible way to implement such a program at a large organization.

When Google renamed its "production team" to Site Reliability Engineering in 2003, the team members were experienced software engineers tasked with "keeping Google running." These software engineers had deep knowledge of the systems Google was using. The number of different systems Google was running was limited; it was more or less possible to know most of the internals.

As Google grew, and systems grew increasingly specialized, we needed more Site Reliability Engineers (SREs). Instead of transferring experienced Google software engineers into SRE, Google began directly hiring SREs. Although Google had a handful of classes to train new software engineering hires, we didn't have any SRE-specific training. The newly hired engineers joining SRE had to "grok SRE the hard way."

In 2014, a couple of SREs began discussing the great difficulty of onboarding new SREs. Google SRE founded a team specifically geared toward education for SREs. Initially, this team concentrated mostly on new hires. Over time, the team also began organizing classes for experienced SREs who needed to learn a new technology.

Google has many different SRE teams dedicated to one or more services because there is a limited amount of state that any one engineer or team can retain. Different services can have completely different characteristics (for example a batch-oriented service compared to a streaming-oriented service). They also make use of many different supporting subsystems, such as widely differing database systems. There are many differences between the services, but there is also a lot of common ground, such as the Google infrastructure for networking, the Borg cluster management system, and so on (see Chapter 2 of the *SRE Book*). Therefore, it makes sense to split out these common subjects from the team-specific subjects. Google's SRE EDU team is responsible for acquainting students with SRE culture and the common infrastructure. Individual teams are responsible for the team-specific training (see the section on "Team-specific training" on page 42 in this chapter).

Large distributed systems have many moving parts, all with their peculiarities that only a few specialists know a lot about. It's impossible to know everything about all of that infrastructure. It's not surprising that knowledge about such large distributed systems will also be distributed. For new people on the job, it can be frustrating to find out that contrary to your previous job, where you knew about all the machinery, now you know only a small part. It's easy for new hires to fall victim to the imposter syndrome (see Chapter 2, "Fight imposter syndrome" on page 30).

Because the Google SRE team is both large and growing rapidly (see the upper-right quadrant of Figure 2-4), we designed and launched a full life cycle training program. This ensures that new people confidently ramp-up on the team while providing continuing education opportunities for experienced SREs to build new skills, or enable them to switch teams and support a different service.

Stages of Training

For Google SRE, onboarding begins the second week after hire, when all administrative details have been handled and new hires have been introduced to Google's general culture and procedures.

Orientation

Orientation is the program where we get new hires who have not had any Google-specific technical training yet. We teach them about SRE principles, practices and culture, and some technical aspects that are general to Google.

Legacy orientation. For a decade, Google had no structured education for new SRE hires. One could say we were following the "sink or swim" model discussed earlier. Apart from haphazard, team-specific materials, and random classes taught at irregular intervals, there was no SRE-specific training for new people.

In 2015, Google SRE formed an education team called SRE EDU. To ramp itself up as fast as possible, first, an inventory was made of all available classes (and all the different versions available for those classes). This led to a curriculum that put 11 classes, all one to two hours long, in a week. The students were happy, but there were frequent remarks that learning would be more effective if there were more hands-on-oriented classes. We quickly and effectively launched a minimum viable training curriculum and then used survey feedback (a form of monitoring that we discuss in Chapter 5) to find opportunities to innovate and improve.

Current state of SRE orientation at Google. After onboarding at the local office, we have students travel to one of our three hubs for orientation, where they receive a week-long, SRE-oriented training. The training concentrates on SRE culture, tools and systems, and applications.

For the culture aspect, we have a number of classes sandwiched between the more technical classes. Following are the important factors in these cultural classes:

- Failure always happens—it is a logical consequence of Google's size. Therefore, we must embrace failure and use redundancy and resilience to fight the effect of the failures that do occur.
- Toil is something we want to get rid of as much as possible. Our approaches are to automate as much as possible. Also, after an outage, investigating to find the root cause and fixing that lowers repeat outages and toil.
- Creating a good social fabric is important. This comprises having a good relationship with the developers, using blameless

postmortems after outages, and only having actionable alerts. Also, having diverse opinions matter. This helps fight tunnel vision, which can prolong the duration of outages.

For the more technical parts of the curriculum, we have changed our approach drastically. Adult learners acquire new knowledge most effectively by doing and applying what they've just seen. Therefore, we went from a model in which we had a number of slide deck–based classes, to a model with fewer (and less-detailed) classes with specific, hands-on practice. In particular, we created a photo upload service called the Breakage Service, discussed in more detail in Chapter 4.

Wherever relevant, the technical classes refer to our photo service as an example, so the students see right away how that works in real life. They immediately use the tools they heard about and investigate how the parts of the service hang together. For example, when we teach about Remote Procedure Calls (RPCs) in Google, the students also look at diagnostic consoles with regard to the RPCs sent from the frontend servers to the backends.

We've found that a day and a half after orientation begins, the students are able to correctly triage, mitigate, and find a resolution for the first outage. Once, after the first breakage exercise, a student correctly gave a four-sentence summary of what had just happened. When asked what they would have thought if they'd heard those four sentences only a day before, their eyes went big, their jaw dropped and they said, "I would not have understood a word of what I just said."

From surveys we held, we found that on a seven-point scale, from –3 (not very confident) to 0 (neutral) to +3 (very confident), the students, on average, rated their confidence nearly two points higher than they did before orientation. To further illustrate how SRE EDU orientation raises confidence, 89% of participants reported at least a one-point increase in confidence, with 29% of participants reporting at least a three-point increase in confidence. Figure 3-1 shows the shift in confidence in a histogram of survey responses.

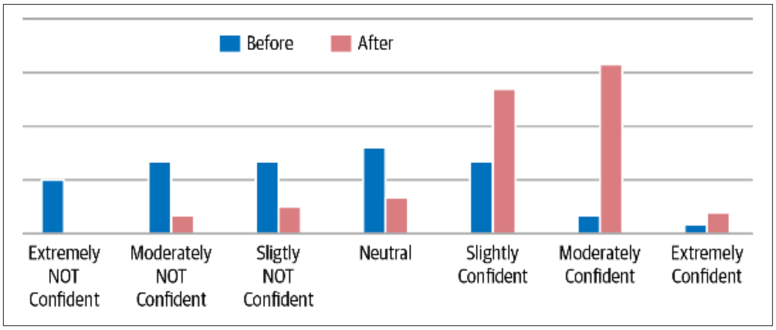


Figure 3-1. Histogram of survey responses of self-reported confidence.

Takeaways. We found it beneficial to move from a class-only model to a model centered around real-life and hands-on troubleshooting. We saw much better student participation, happier students, and above all, a rise in student's confidence that they'd be able to do their SRE jobs. We strongly suggest making your onboarding classes hands-on, with troubleshooting exercises that are as close to real world as possible. This might be more difficult to do when your organization is smaller (you would probably not build a complete application stack for your learning environment), but it's still worth it. We look at how this is done in "SRE Training in Smaller Organizations" on page 47, later in this chapter.

Training of instructors. A training model that centers more around the experience than on actual technical knowledge requires that the instructors know what the philosophy behind the program is.

We've noticed in the past that giving students the full depth of information about the systems we have leads to information overload, and the students don't remember the information afterward. However, to do the breakage exercises, only a minimum amount of detail is needed. Therefore, we have deliberately limited the depth of the class material to teach only what is necessary so that we can go through the breakage exercises. This shift in training method is sometimes difficult for instructors, who are often subject matter experts that volunteer to teach. The instructors often want to tell a lot of exciting details about their subject, and might even disagree with the amount of material that is left out, including specific details about the "missing" material. However, when the instructors see how students apply what they've been taught when doing the exercises, and resolving the deliberate breakages, they better understand why the curriculum is set up the way it is.

To help instructors see our reasoning behind the idea of less depth in the curriculum, we run regular "Train the Trainer" sessions. We have two different sessions: one for the class material, to give the background and reasoning behind the classes, and another for the story that ties them together.

The latter Train the Trainer class concentrates on how to be a facilitator (or teaching assistant) during the breakage exercises. We've found these sessions are valued and in high demand from our instructors. In the "breakage" part of Train the Trainer, we have the future facilitators be on the "receiving end" of the breakages, asking, before the exercise starts, to not just resolve the issue, but also observe how the Train the Trainer instructors behave. We have multiple breakage exercises, and in later exercises, the Train the Trainer instructor is more and more hands off. We discuss the tactics here and why it's used (as the new SRE students gain more confidence, we keep more distance). After a number of these exercises, we have the Train the Trainer students practice running a breakage exercise. This sometimes poses a challenge, as the students have to act like they don't know what is wrong, during the breakage exercise they actually solved earlier.

In the same way the entire curriculum is there to give the new SRE hires confidence, the Train the Trainer is also there to give facilitators and TAs the confidence they need to run a class. How we implemented the Train the Trainer program is described in detail in Chapter 4, *Instructional Design Principles*.

Having volunteers teach and facilitate is the only way we could scale our program to the size it is now, with a team of only seven people running 14 classes every month, in three different locations. To date, more than 3,000 students have gone through Google's SRE EDU orientation.

Team-specific training

After students have gone through foundational SRE training, they must learn the team specifics. The teams all have different ways of ramping up their new SREs for the service for which they are responsible. Having a company-wide checklist for SRE teams to fill is something that Google uses, though the extent to which each of the teams use a checklist differs. Several techniques are used by the teams which we discuss in the following sections.

Documentation. One source of information that new team members learn about the systems they will be responsible for comes from documentation. Documentation should describe the following:

- The systems the team is responsible for
- The procedures they follow; for example, for cluster turnups and turndowns
- A playbook/runbook that describes how to handle certain outages (see the section "Maintaining Playbooks" in Chapter 8 of the SRE Workbook).

This helps all new team members understand the systems the team is involved with. Some teams also have specific onboarding documents that are then referred to, from the onboarding checklist.

New team members should be encouraged to fix any inconsistencies they encounter in the material. If your documentation is checked into your versioning system after review, this is easy and safe to do. Having new team members verify the documentation helps keep the documentation up-to-date. New team members are in a unique position to conclude that the documentation no longer describes the setup of the system. Unfortunately, the state of documentation is often "slightly stale" more often than not.

In-person and recorded classes. Team members can teach classes about the systems, perhaps as impromptu one-offs or as deliberate ramp-up summits (if there are multiple new team members). If you record the classes delivered at the summits, they can also be used by new arrivals who join the team later. Even more than with documentation, the risk exists that the recordings are out of date because systems are continuously in flux. Although documentation can be updated, this is much more difficult to do for recorded videos.

Whiteboarding sessions. Having whiteboarding sessions is also useful. Here a team member who is the "owner" of a specific subsystem explains how that subsystem works, any weak spots it might have, and what the plans are for the future of the subsystem. Having these sessions not only helps other team members understand more about these subsystems, it also works as a forcing function to have people explain the subsystem in a clear and succinct way. Also, these sessions can be recorded for later consumption (with the same staleness risks).

Teach-back sessions. There is a special kind of whiteboarding session during which new team members are asked to prepare a session on a specific subsystem, and then use the experience from reading documentation, skimming through configuration, and looking at jobs running in production, to explain what they think the system does. At Google, we've found that usually, the new team members do a really good job, and any misunderstandings are quickly resolved. Many times, such teach-back sessions uncover new aspects of the subsystem that the non-experts of that subsystem did not know about yet. These sessions are of value not only to new team members, but to the team as a whole.

Mentoring. It's important that the team assigns the new hire a team member as a mentor, as a first point of contact for questions related to the team's systems and the company's infrastructure in general. We also advise new hires to have a mentor outside the team, for specific issues not related to the team—someone who can give a point of view without the consequences of day-to-day work. For example, "What can I expect from my manager?" or "How does *your* team use tool XYZ?"

Going on-call

A mid-term goal for team member ramp up is preparing the new team member to go on-call. Depending on the teams that the new SREs are in, this takes anywhere between three months to a year.

Classes. At Google, we offer newly hired SREs a set of classes after about six weeks, to prepare students for going on-call in the general sense. Of course, there are no team specifics in these classes. We have two classes that talk about the mechanics of being on-call (a little bit about the tools, and mostly about the procedures around incident management). As previously discussed, we also have two more "soft skills"–oriented classes: the first is about stress management during incidents. During incidents, the stakes can be very high, and keeping your cool in those situations is important. We want to teach this without scaring the students and having a negative influence on their confidence. This class was codesigned with an aviation consultant—stress management in the cockpit is vital!

The second soft-skills class is about proper handoffs and escalation—how to communicate when roping in help from other people. This teaches people the following:

- How to prevent misunderstanding by using very explicit communications
- How to behave, escalate, and ask questions during stressful situations, in such a way that the communication goes as smoothly as possible

Powerups. Some teams (most notably those with highly critical systems) use a mechanism called *powerups*. This works just like in video-games, in which a powerup gives a player certain abilities after having achieved an intermediate goal. In Google's case, after a new team member has demonstrated that they've acquired the knowledge and skills needed for the "next level," they are granted more permissions to manipulate the systems they are responsible for. There are usually multiple tiers. Having these powerups not only makes sure that people on the team with the permissions to administer the systems, have the skills for that, but also gives the new team members the confidence that they actually are "at the next level."

Shadowing. An important part of ramping people up to be on-call is shadowing an experienced team member who is on-call. The engagement here varies from "looking over the shoulder" of the on-caller, to an almost reverse situation in which the new team member is driving the investigation, with the experienced team member only supervising and encouraging. Of course, it depends on the urgency of the outage, and how involved the shadower is.

Ongoing education

In large environments like Google, systems are continuously in flux. Systems are frequently deprecated, and new systems emerge to replace them (for example, in 2011, Google replaced its internal cluster file system, GFS, with Colossus, a major undertaking with consequences for almost all teams within Google). Most of these changes led to extra work for service owners. Therefore, we encourage people to create classes about new emerging systems, and subject matter experts to teach them regularly.

Once a year, we organize an "Annual Week of Education" (AWE) during which people worldwide are invited to (create and) teach classes. Because we have offices around the world, this amounts to having classes being taught almost 24 hours around the clock, Monday through Friday, for that week. People can attend these classes locally, where they are being taught, but also remotely, from other offices, through video conferencing. Of course, these classes are recorded, so people who are in an inconvenient time zone can still attend. In the past few years, we had more than 70 classes during AWE, with 50% of the classes presented being newly developed. Following best practices described in section "Managing SRE Training Materials" on page 91, we have materials curated by subject matter experts. We monitor their age, and set dates by which a subject matter expert must verify that the material is still fresh enough (or the material is removed). We created a catalog of the material for discoverability, which seems to work well; though, as always, there is room for improvement.

Several of our SRE sites also organize regular, ongoing, education classes, taught by subject matter experts. This is either done in an on-demand fashion, or in a regular cadence; for example, 10 classes for a week, every quarter. Because the SRE EDU core team does not have the cycles to cover all of these sites, and the local SREs have knowledge of the training needs of the site, we have SRE EDU site contacts in all the SRE sites who are responsible for local, ongoing, education activities, with scaled support from the SRE EDU Core team.

Summary

In a large SRE organization, with a very wide scope of technologies in use, instilling confidence is the most important aspect of education. This confidence is best gained by having hands-on exercises that are very close to the real-life production systems. Generic ramp-up material is best created by a centralized group but team-specific information should be created and delivered by the teams themselves.

After initial ramp-up, continuous education is also important. We suggest that these training efforts be done in a distributed way, with subject matter experts creating trainings. If recordings of these trainings are made, they can easily be distributed in the organization. However, as always, care must be taken that the freshness of these materials is monitored and repaired and that classes be removed whenever they are stale beyond repair.

SRE Training in Smaller Organizations

In the previous sections, we've seen how Google delivers content in ways that are tailored for training diverse audiences (out-of-school new hires, senior engineer new hires, in-house transfers, etc.). We discussed how Google focuses on hands-on training to help establish a foundation of knowledge, muscle memory, and expert intuition, all of which are necessary for our SREs to succeed in their daily tasks. Although the discussions were focused on how large companies like Google can implement training, the approaches can be scaled down and modified for companies of any size, be they large or small.

Of course, for an organization with, say, a single SRE team, it might not be feasible to create a training program like the one described in "Training in a Large Organization" on page 37. For smaller organizations, we look at companies that apply or advocate SRE practices, not necessarily calling themselves SRE. This could include software engineers who care about reliability, DevOps practitioners, and so on. Many of the principles that Google uses can still be applied to these organizations. For example, it's important that students get a chance to practice what they just learned in a safe environment. With only a few students per year, it's difficult to create classes and have instructors teach them—that would not be very cost effective. For smaller organizations, more time is spent in self-study, mentoring, and shadowing.

Applying What They've Learned

The key point—people usually learn better when applying what they have just learned—still holds true for a smaller organization, but it's more challenging to set up an environment at a small scale. Even so, it's not impossible, as shown in the following examples.

The Swiss company, AdNovum, has consultants that install, configure, and troubleshoot their "NEVIS" security suite at customer sites. They train their new hires using reading materials and self-study exercises that are set up on virtual machines (VMs). For these exercises, the students install and configure the software and verify that it's correctly working. Then, they run a script that breaks a specific part of the system. That's when their troubleshooting exercise begins—the students must find the root cause of the problem. Because this is in a VM, in a learning environment, students don't need to be afraid of breaking something critical—it's easy to reset the situation by spinning up the VM from scratch, or by using the "UNDO" option provided by the script. Creating a safe environment helps the student gain confidence. Because the student has access to the breaking script, it might be a good idea to obfuscate it so that they don't get tempted to see what's broken by looking at what the script does.

Another example is from one of our authors who previously worked at a small teaching company, AT Computing, in the Netherlands. During network classes, AT Computing uses multiple VMs on a desktop computer to offer exercises on troubleshooting routing and firewall problems in the virtual network. The company makes the VMs as small as possible so that many VMs could be booted up from the desktop. This allows for elaborate virtual networks. This setup is used for a Linux network management class, a firewall class, and a DNS management class.

Finally, let's take a look at Yelp. In their SREcon Americas 2019 talk, Chie Shu, Dorothy Jung, and Wenting Wang describe how they surveyed their fellow engineers at Yelp about on-call readiness. Almost 65% of the respondents answered that they did not feel ready for going on-call. They then introduced a wargame that lets new SREs simulate an incident in a safe environment. Their work includes a template that is used to run the exercise with different participants. It instructs how to introduce the incident (in a nonproduction environment) so that others can play the wargame. Players take on various roles like investigator, communicator, and commander. Again, we see that creating an almost-real environment that is safe to "play" in and can be broken on command lets new SREs practice with the tools they need for troubleshooting.

Company X

Now let's look at an example of how another company (let's call it "Company X") has implemented its SRE training program. Company X developed its program independently from Google, and we discuss why its choices make sense for a company of its size and organizational maturity.

Company X employs nearly 1,000 developers, eight of which are SRE. They hire SREs instead of more software developers for a reason that is common in the industry: developers tend to treat available resources (network, computing power) as infinite and never-failing. SREs are comfortable with the uncertainty, ambiguity, and unpredictability of modern distributed systems and focus on reliability issues that are not generally top-of-mind for developers.

The onboarding process for all engineers, including SREs, consists of about 40 hours of classroom time spread out over their first few weeks. Each one of the training modules is one to two hours long and covers essential infrastructure: dev practices and tools, monitoring systems, service architectures, and so on. One of the most intensive modules (three hours) is "Incident Response and Postmortems" All engineers need this incident response training, not just SREs, because any engineer might be called upon to help mitigate an issue during an incident. A company of bigger size could afford having a dedicated team for incident response, but at Company X, all engineers are expected to play a part, in one way or another, of a coordinated response.

Each engineer is expected to walk through the entire process during the first three months of work. That gives them plenty of time to attend the courses they are requested to complete, work on the practical exercises they have been assigned to perform, and meet the people of the respective teams they'll be working with. The training process overlaps with regular work during that time because Company X does not have a dedicated training team to engage full-time with the new engineers during the first few weeks/months of their tenure.

With only eight members, the SRE team is small enough that each new individual receives a highly customized onboarding document of about 10 pages. The document contains personalized steps based on the new hire's expected role and past experience.

Most of the onboarding material is not SRE-specific, and only a couple of the self-study guides provided by the SRE team are targeted to develop SRE-specific practices. During the time the new hire is expected to work through these, lasting about two weeks, they are paired with an onboarding mentor who provides answers to any questions they might have.

The SREs go through the onboarding process, and get ready for going on-call by performing some drill exercises, using nonproduction environments that mimic the actual systems they will be dealing with, similar to the process described in the "Current state of SRE orientation at Google" on page 39 section of this chapter.

Company X also performs disaster recovery (DR) drills, once per year, that test services across the entire company. These tests are performed over the course of a single day, supervised by senior members of the engineering team.

In addition to DR drills, engineers participate in sessions called "Wheel of Misfortune," during which role-playing exercises are invented and solved together. These sessions tend to be arranged ad hoc, where engineers are presented with invented scenarios to resolve.

Readiness

Although there's no formal process to certify that a new hire is ready for on-call, other than the SRE feeling that they are ready (and their teammates agreeing), the new SREs generally shadow an on-call shift (a seasoned engineer is steering the wheel) before taking the lead role. Likewise, due to the limited numbers of SREs, it is unfeasible for Company X to have an established reverse-shadow process. Due to the small size of the team, and in order to provide an escalation path and production support, when new engineers are on-call, at least one other team member is expected to be easily reachable, 24/7.

Continuous Development

Finally, the training and development of the new hire does not stop after they are fully onboarded and assume the role and responsibilities of a regular SRE member. There is an annual, self-directed budget of $2,000 that each individual has at their disposal, for professional development, such as attending conferences.

Conclusion

In this chapter, we looked at specific case studies of training at large, medium, and smaller organizations. We discussed the stages of training, including orientation, team-specific training, and going on-call, and we considered examples of how training could be adapted for the size of the organization. In the next chapter, we examine how to implement such training.

CHAPTER 4

Instructional Design Principles

Creating training that sticks and is engaging is challenging in any business environment. That is because business trainings tend to suffer in three ways:

- Lack of immediate application
- Information overload, and
- No clear, actionable, learning objectives

The consequence is that employees feel that they have learned something, but can't quite identify what that was, nor could they show you anything substantial to prove that the training was worth their time. One way to combat a few of these challenges is to apply instructional design principles to the design of your training program. Here, we highlight approaches that you can use, no matter how large or small your organization is.

Instructional design is the practice of creating consistent and reliable materials, experiences, and activities that result in the acquisition and application of skills. Another way to think of instructional design is as a framework. Although there are many techniques that instructional designers use to develop training, here we highlight some of the instructional design principles used by Google's SRE EDU team. These principles have helped build our training materials, experiences, and activities, so that our students gain the ability to hit the ground, confidently running, during their first few weeks at Google.

Identifying Training Needs

Before you design any training program, you need to establish what the problems are that you want to solve with training. In the case of SRE EDU, we were faced with two problems. The first was the steep learning curve in SRE. This was thought of as just a part of the job, which resulted in lots of smaller initiatives to help lower that learning curve. This led to onboarding that was inconsistent, nonexistent, or very targeted toward certain services. The ad hoc onboarding programs that did exist varied widely by team and by region. The second problem was to standardize on what and how SREs were trained, in a global, scalable, and dedicated effort.

Here are some questions to ask yourself and/or your training-minded team:

- What problem are you trying to solve?
- What are the goals of the training?
- Who are the people who can make this happen? Who are the key influencers in your organization? Who are the stakeholders who can back you?
- What are the root causes of the problems you are trying to solve?
- How will training help mitigate identified problems?

If you answer these questions fairly thoroughly, you will have a foundation on which to build. Spend a significant amount of time here. Understanding and enumerating the problems help guide you in building out the rest of the training content, and to a larger extent, the training program.

Build Your Learner Profile

Ideally, you already know your audience. However, we've found that our audience is often bigger than we think. Therefore, it helps to identify who you are training and to describe what they do.

Here are some questions to ask yourself and/or your training-minded team:

- What are the skills/knowledge/activity/behaviors you want people to do?

- What are the skills/knowledge/activity/behaviors you want people to stop?
- What is the current working environment of the audience you are training?
- What resources/training is the target audience currently using?
- Who are the people you are planning to train? Are they new to the company? Industry experts? Recent college graduates?

During this activity, you'll discover a long list of items that you should curate based on who you are training. For example, if your primary target is new employees, you'll want to decide on the most important skills that are appropriate for new employees to know, to hit the ground running. A major pitfall in business training, as mentioned earlier, is information overload. When everything is important, nothing is. For SRE EDU, we negotiated, discussed, and decided on what key things we wanted our students to be able to do at the end of the training, that had the biggest payout to the teams they were joining.

Finding out these answers should not be done in a silo. Get out into your organization and start interviewing. Talk to leaders, managers, individual contributors, and others. SRE EDU met with staff to identify these things. What came out of those meetings became very valuable to the development of the training. Following is a sample of resources we discovered:

- Ramp-up strategies in individual teams
- Best practices that teams had developed
- Resources and materials that would be useful to a larger group
- Graphs and data analytics dashboards that have broad use cases for training
- Tools and system use cases that were common across multiple areas

Consider Your Culture

The intersection between culture and training is reflected in attitudes, practices, environments, and beliefs about how one acquires and uses knowledge in the workplace. What we mean by this is that *how* you train is just as important as what you train. With that in mind, having a functional knowledge of the culture of the workplace becomes paramount to effective training. There are two main principles to culture that SRE EDU focuses on:

- The SRE culture (the way we do things and why) must be explained, demonstrated, and reinforced throughout all of the training.
- SRE EDU is a vehicle for early introduction of these cultural norms, by those who are living the culture.

Throughout the entire curriculum, we weave cultural classes that introduce cultural topics to students. As an example, we have a class called "The Role of the SRE". In this class, we explain Google's mission, and how we as SREs fit into that mission. We use real examples to demonstrate who we are, what we do, and how we do it. This class sets a foundation from which we build a knowledge base to work from, a vocabulary to build off of, and an immediate identification with something tangible in a globally distributed organization. It answers a common question from our students—who am I and why am I here?

Culture is the most fragile part of training. Authenticity in what you say and how you train culture matter. If what you train is not reflective of the real organizational culture and practice, you have effectively lied to your students and set them up to fail. When that trust is broken, it's difficult to get it back. It is entirely reasonable to present the culture as it is, while communicating where the culture should be. For example, we noticed that developer and SRE relationships in some teams were not as strong as we would have liked. After following our process of identifying the training needs, root-causing problems, and interviewing other teams, we developed a culture class centered on what healthy developer and SRE relationships look like. We acknowledged that not all teams are working with each other as effectively as they should, while stating that we need to change this within the culture, and here is how you can help.

Where did we get this information from? From within the culture, through teams that measurably worked well together. Interviews with these teams highlighted a common theme to work from—and teach. Each of these teams had excellent strategies for how they communicate, align, and agree on tasks, goals, and work. These teams built and maintained trust with one another. The authenticity of this class is enhanced by students who might have come from other industries and can attest to and/or provide additional insight into what we are teaching.

Storytelling

Another aspect of how we teach culture is through the use of storytelling. Storytelling is a cultural or social activity of sharing narratives that can be used for education, entertainment, or preservation. What makes storytelling so compelling as an instructional tool is the ability for recalling (or retelling) the story in ways that keep the lessons learned alive. When constructing the training program, we first started asking ourselves what is the story we want to tell? We went through a simple exercise of looking at what makes stories such an important part of culture. We identified what key things SREs learned in the past, that we wanted to bring to the present, to positively influence our future.

We framed training around telling a story. One story that framed our training is that things break, and they break in interesting and novel ways. You learn how these things happen, but more important, you learn strategies to combat that. What makes a story worthwhile is that it can be retold by the listeners; there are visuals, word queues, vocabulary, and actions, that make the stories memorable while imparting lessons.

Let's see how well using word queues work to get you to guess the story or event:

- Glass slippers, midnight, fairy godmother, prince, stepsisters (Cinderella)
- King Triton, Sebastian, Flounder, Ursula, Ariel (The Little Mermaid)
- "I have a dream...", Lincoln Memorial, Civil Rights March (Martin Luther King, Jr. speech)

So how does this work with SRE training? During the training, we build up a story around the many ways things break in interesting and novel ways. These are reiterated through storytelling by those who were there to experience it or were passed down this knowledge well enough to tell it. The important part of this is that we impart the lessons we learned to our students.

Here's an example of word queues that happen in a class we teach around causes of outages: nature, humans, power, cooling, bad config pushes.

Each of these words are tied to stories we tell, real-world examples of these things happening, and in some cases, humorous ways we've experienced outages we would have never expected. Students remember these things when they move into their teams and have a model around reliability to plan for. We further this experience through our hands-on breakage activities, which gives students the ability to use real tools and real systems, to solve real breakages and learn through the process.

In our orientation program, the students' first breakage activity happens on their second day of training. Throughout the experience, we stop and see what students have discovered. They tell us the beginnings of the narrative or story behind what's happening. We do this a few times during the experience, which continues to build the story. By the end of that first breakage, after they've solved the problem, we ask two questions:

- Can you succinctly tell me what the problem was and how you triaged, mitigated, and resolved the problem?
- What went well and what could have gone better?

In every instance, students are able to tell the story back to the instructor, identifying the tools and systems they used to triage the problem, explaining why the problem was happening, and detailing what they did to mitigate the problem and resolve the breakage. They further help to tell the story by adding their own personal ideas around how things went.

Why is this important? A primary principle of SRE is that we learn from failures. Most of our failures have valuable lessons that can be applied to system design, monitoring philosophies, code health, and/or engineering practices. By turning these lessons learned into stories, the memory of actions, words, and visuals become a method for recall in future work. These storytelling activities are also a part of the SRE culture at Google. They happen in various events throughout the SRE organization, whether it's through Wheel of

Misfortunes (see Chapter 15 of the *SRE Book*), Ops Reviews,[1] lunch conversations, and so on. Retelling these stories and lessons learned, much like world cultures, keeps our collective memory of history, practice, and culture alive.

Build the Vocabulary

Think back to your very first meeting at your organization. What do you remember? How many words, three-letter acronyms, and jargon did you understand? All of these are the vocabulary of both the company and the culture. They play an important role in identity and establishing a common understanding of meaning within a company. Google is no different. Throughout the training, we give students a common vocabulary from which they interact with the rest of the company.

Consider Your Learners

Closely related to building your learner profile is considering the learner attending your training. Who are the people you are planning to train? Building a profile of your target audience gives you an idea on how to design your training. Things to consider about your learners include the following:

Roles
: How many different roles is your training going to target, and how widely varying are those roles?

Geography
: What locations are your learners located in?

Population
: How many people will you train?

Prerequisite knowledge
: What should students already know?

Travel
: Do people need to travel for training?

1 Ops Review is an activity where participants meet weekly to discuss all the stuff that went wrong in production. The emphasis is on learning by talking through failures openly.

We identify this early because all of this influences the design choices we make when developing training content. Of course, these are all logistic questions. There are other aspects of the learner that we have to consider, as well.

Adult Learners

Training programs often overlook that adult learning is different from how children or teenagers learn.[2] How adults learn is a question of motivation. Adults choose to learn things for different reasons, mostly because they see value in learning something that is of interest to them. Adults have a readiness to learn because of that value. Adult learners in corporations also come with some level of industry or educational experience to build from. There is a entire field of study related to this called *andragogy,* as opposed to *pedagogy*. SRE EDU has worked within three main premises of adult learning:

- Learning is experiential—adults learn what they do.
- Adult learners transfer what they learn from one context to another.
- Adult learners must identify why the content they are learning is important.

Adult learners also deal with fears, uncertainties, and doubts about their abilities. Fear shows up in the form of questioning whether they made the right choice to be an SRE. Uncertainty shows up in questioning whether they have the skills to do the job, or how they will be viewed by colleagues. Finally, doubt shows in the form of imposter syndrome. We've seen this in our daily work lives, and some of us have personally experienced these feelings. In our training, we considered each of these cases in the choices we made when designing our curriculum—from the content depth to the activities.

2 Stephen Paul Forrest III, & Tim O. Peterson. (2006). It's Called Andragogy. Academy of Management Learning & Education, 5(1), 113. Retrieved from *https://oreil.ly/Ami2h* Roumell, E. A. (2019). Priming Adult Learners for Learning Transfer: Beyond Content and Delivery. Adult Learning, 30(1), 15. *https://oreil.ly/ANNeJ*

Learning Modalities

SRE EDU has various learning modalities, depending on the content. When choosing a training mode, think of what your students will be doing and learning. Different people learn best through different modes of learning. Different training topics also lend themselves to different modalities.

Instructor-Led Training

SRE EDU uses instructor-led training for our orientation program. It's the first training students go through. We've seen that having classes in person is the most effective way to introduce our students to the SRE world. Culture classes work well here because of how we teach culture through storytelling and vocabulary building. We also have a lot of hands-on activities that are better executed through instructor/facilitator observation. Observation by experts in their field becomes important for in-the-moment evaluation of how well students are progressing, especially new hires.

Reasons for choosing instructor-led training:

- Your students benefit from expert advice/Q&A from a diverse group of people in your organization.
- Reinforcement of new skills and knowledge.
- Peer-learning and networking are an important part of your company culture.
- Real-time feedback/observation is crucial to your training.
- You have exercises that would benefit from facilitation by an expert.

Instructor-led training does require a higher upfront investment in preparing your students and prospective trainers. We've found it vitally important to set expectations with our students early and be upfront about the experience. They should know what is required of them, how long the training will be, where they should be and at what time, and what to expect throughout the experience. This led us to create a short introduction to the entire program that starts off their training. You'll also want to prepare your students to know where things like restrooms and break areas are. For instructors, we prepared items such as lesson plans, slides, and help guides. We also do a Train the Trainer session for our new trainers. We discussed

Train the Trainer in context of the hands-on activities, earlier, in Chapter 3.

Self-Paced Training

Self-paced training shows up in various forms. It could be a video, document, or some form of elearning module. The point is that self-paced training is contingent on the student guiding the pace of learning. SRE EDU's primary use of self-paced training is in videos. SREs have lots of knowledge about many things. In SRE at Google, we have a culture of sharing knowledge. Once each year, SRE EDU hosts an education program in which any SRE can build and teach a class for their fellow SREs and other interested parties. We record all of these talks and post them to a page. The content varies widely, but what this allows us to do is broaden our training topics by using our own team members. This also allows us to quickly create a library of courses for varying skill levels. These videos are now available for anyone to watch, whenever they want to. Students control their own pace of learning.

Another aspect of self-paced training is scale. Our SREs are globally distributed. Getting SREs into one place for training is not always efficient, and generally difficult. Think about visas, travel, hotels, and transportation, and then multiply that by the number of SREs you have. The costs alone are prohibitive for a large organization. SRE EDU has made self-paced study a primary mode of training for our continuing education efforts.

Here are some reasons for choosing self-paced training:

- A globally distributed team
- Remote employees
- Content with high reusability and little change
- Activities that are best done solo
- Target audience is experienced staff
- Less prescriptive training
- Don't have a large enough population to train, to justify an in-person class

Some examples of the types of courses that tend to work well with self-paced training include account setup courses, tutorial classes, programming classes, and product overview classes. This is by no means exhaustive, just examples that we've seen work well.

Mentoring and Shadowing

Mentoring and shadowing is a training mode we encourage individual teams to do. It's a good way to get our employees accustomed to how their team operates. For teams that are highly specialized, this type of training mode often gets new hires up to speed faster. The interactions are more personal and can be tailored to individual needs. We use this technique for the SRE EDU program, when bringing in new instructors. When we have new people who want to teach one of our existing instructor-led classes, we ask that they shadow a class with an experienced instructor first.

Shadowing gives our new instructors an example of how class is run. They see how topics are taught and what examples or stories are used, and can ask the instructor questions, learn what types of interaction happens between students and instructors, and watch how classroom management is done. By having a shadow/mentor process, we generally ensure that the content is taught consistently, and share responsibility for that consistency.

Here are some reasons for choosing mentoring and shadowing:

- Onboarding new hires into the practices and norms of a team
- A team has use cases or practices for a highly specialized tool
- You have just a few people with specific knowledge you want to pass on

Create Learning Objectives

Learning objectives are generally defined as outcomes or actions that you want your learners to be *able to do* when they complete the training. To help with this, we use a model or framework for building learning objectives. We frame all of our classes and activities with "Students-Should-Be-Able-To's" in mind. Learning objectives in SRE EDU use the following formula:

<DO SOMETHING> <USING SOMETHING> <AT SOME LEVEL (if applicable)>

At the end of this training, a student should be able to do the following:

- Use <job tool> to identify how much memory a job is using
- Interpret a graph in <monitoring tool> to identify the health of a service
- Move traffic away from (i.e., drain) a cluster, using <drain tool> in five minutes

In these short examples, notice a few things. We use action verbs to establish the behaviors that we want to observe or measure. Action verbs imply an activity, something that someone is doing. We avoid words like "understand," "know," and "be aware of," because we can't measure those effectively. The objectives also contain the conditions under which the learner does the tasks. In some cases, it is a system or a tool. Finally, if the tasks the students are doing is time bound or accuracy dependent, we assign a value for the level a student needs to hit in order to perform the task successfully.

Concrete learning objectives (behaviors) lead to better measurability and observability of those behaviors. At this point, the data that you gathered from identifying the problem and training needs becomes important. During that phase, we asked what are the skills/knowledge/activity/behaviors that you want people to do? Notice we asked, *to do*. This is important, because now you have a framework for developing learning objectives.

Learning objectives that are clearly defined and measurable help designers develop training content and materials that meet those objectives. It's also a clear way to present the intentions of the training to stakeholders, and how you'll measure the effectiveness of training. We know training is working as intended when students demonstrate what the learning objectives detail. Because we frame them as actionable, we observe in real time students meeting or not meeting those objectives.

Designing Training Content

Training materials take many forms. The common ones include slides, workbooks, handouts, lesson plans, etc. Instructional designers tend to design with at least one of these in mind. However, the word *design* is more than just the tangible materials that students or instructors use. In this section, we highlight a few of those less-tangible items and how we designed training content with them in mind.

ADDIE Model

One of the most common models for creating training content is the ADDIE model. Instructional designers tend to be well versed in this model for development. ADDIE stands for Analyze, Design, Develop, Implement, and Evaluate. It is a staged process for how to design training as well as things to consider while designing.

Even if you don't have an instructional designer, you can employ the ADDIE model on your own. This is not an exhaustive study of the ADDIE model, but we want to keep the material in the realm of practical use, so we cover some key concepts here. Note that each stage of the ADDIE model is generally a linear process, but you can revise the model according to your training needs. Let's talk about the ADDIE model in detail next.

Analyze

In the *analysis stage* of the ADDIE model, instructional designers do a fairly exhaustive inventory of training needs. This includes identifying problems training intends to solve, and defining learning objectives, behavioral outcomes, delivery options (learning modalities), and timelines. Analysis takes a large amount of development time because getting the analysis right saves time and resources as you progress further.

Design

The *design phase* is primarily focused on the learning objectives. Through the learning objectives, you are determining evaluation strategies and exercises, and identifying the content needed, such as instructional guides, media, and the student experience. During the design phase of our more hands-on curriculum, SRE EDU created a design document similar to software design documents. We did this because we wanted a single source of truth for what the training would look like and how the training would progress.

Design documents get very detailed and definitely do not make for good leadership discussions. However, they are a useful tool for instructional designers and teams to comment, revise, and collaborate on the training content plans. In the design document, we also created our timeline and launch strategy. Start with a basic idea of what you are planning to do. You can see a more detailed design document in Appendix A.

Begin with an overview of your training. Think of this as your elevator pitch, a succinct description of the problem that you are trying to solve, why this training solves the problem, and how this training solves the problem. Create a learner profile. Identify who your intended audience is and why. Things to consider here are the roles or job positions your training is targeted for, where they are located, and how many people you plan to train.

Review the content that you are planning to train and give detailed information about each course. Include the timing of each class (60 minutes, 30 minutes), your learning objectives, and an outline of the content. If you have source material that you are referencing, document it so that you have a repository for all your source materials. If you are using subject matter experts, document who they are. Just as you identified who your learners are, you also want to identify your possible trainers. List who will teach the courses. You do not need to identify specific individuals at this stage; instead, identify the skills you are looking for, what knowledge and skills the prospective trainer should have, and what teams these individuals should come from.

Another thing that you want to identify early are the risks. Risks are things you are aware of that could derail or delay your training project. When identifying risks, talk to others who have gone through a similar project. The insight that others have is invaluable when designing training programs. One risk that Google SRE EDU found early was that some designs we had for our hands-on activities were not possible because the systems would not allow it. That meant we had to work with our partner teams to figure out hands-on activities that *were* possible.

Success metrics are closely tied to risks. Success metrics are measurable data points that you use to determine how well your training is performing. One way to do this is to look at how you evaluate your training (see "Evaluating Training Outcomes" on page 78 later in this chapter). Split success metrics into the overall program and another set of metrics to determine go/no-go decisions for launch. Next, document a launch plan.

Launch plans should include criteria for launch, and the decision to launch should meet those criteria. The launch plan should also have a general plan for training your trainers, piloting your classes, and a plan for keeping your training fresh (updated, accurate, and reflective of reality). It should also include any administrative tasks, and a training schedule for students. Closely related to launch plans are communications plans.

Communications plans are often overlooked, causing a last-minute panic to get the information out to the intended audience. By planning for communications early, you can draft, review, and test your communications strategy early and get feedback to improve before launch. Think through how you'll communicate to your audience. If you plan on using emails, draft them beforehand and get feedback.

Finally, documenting sign-off partners and timelines are helpful to keep track of how the development phase gets done. Sign-off partners are individuals who are key decision makers and influencers in your organization that help implement your training. This list of people also identifies key stakeholders with whom you want to keep in constant communication during your design phase.

Timelines at this stage can be a little difficult to determine with complete accuracy. Instead of focusing on dates, pay attention to the amount of time tasks might take. Refine and update those tasks as you progress. When developing timelines, give yourself some padding because unexpected things always come up. Having that extra padding is a way to mitigate risks to your schedule or development cycles.

Storyboarding during the design phase is also useful. Storyboarding is walking through the training ideas visually in a systematic or linear approach. Think in terms of a movie. You sketch plot points, interactions, action scenes, and so on. Various elements in the training get sketched out. After that's done, you step back and look at the flow. You find missing points and gaps in the training that require a link. More important, you play around visually with the flow. What would happen if I moved one particular aspect of training somewhere else? How does that work? What content support would this change need? One word of caution is that storyboarding works well if you are planning a training that has a linear or step-based process, but nonlinear or branching processes can inadvertently become linear by following this method. In some cases, using a storyboard in nonlinear training unintentionally makes your training linear—or seem linear—to your students. For example, take monitoring training. We introduce monitoring via a philosophy course, but students (later in their career) might choose to go further into monitoring, via other training topics and tools related to monitoring. There is no sequential path to follow.

Develop

The next stage, *development*, is where instructional designers begin to assemble the training. We don't recommend making a large amount of changes to your design plans during this stage. The development stage includes material creation, integration of any technology pieces, the building of activities, surveys or evaluative instruments, and graphics. We take the things learned in the analysis phase, learning objectives, the items decided on from design ideas in the design phase, and turn them into learning activities or experiences. This varies depending on the modality of training you chose.

Implement

Implementation deals with timelines and launching. It's all about getting the content, materials, rooms, trainers, and things you've planned to be ready for use. Implementation has a feeling of finality. In SRE EDU, we used implementation to start our pilot plan. One of the criticisms of the ADDIE model is the inflexibility of iteration and changing midstream. To combat this, we planned for multiple pilots of the content, gathering feedback, and incorporating it back into the curriculum before a final launch. Pilot plans vary depending on the size of the training you are building and the type of training. Hands-on training requires a pilot that tests not just your content and learners, but also the systems/tools you are using. Think of what you are building, and plan for pilots ahead of time. Launching and discovering major problems afterward, which you don't have time or resources to fix, is frustrating.

Evaluate

Evaluate is the last step in ADDIE. Often, evaluation is considered as the final step in training. Ideally, you want evaluations to happen throughout your training, which is when your learning objectives come into play. SRE EDU creates points of evaluation through the activities we build, to give students and instructors an idea about how they are progressing. This can be through storytelling, activities with a definite outcome we expect all students to experience, or hands-on practice.

Final evaluation in the form of surveys are another way SRE EDU has gathered feedback. In the beginning, we surveyed our students just once, post training. Now we gather feedback at various points, pre- and post-training (see "Evaluating Training Outcomes" on page 78 later in this chapter). The idea behind this evaluation is to use the feedback to improve content and stay in tune with the experiences of the SREs, as time progresses.

We've covered some basic ideas around the ADDIE model. There are, of course, other things we've learned from developing SRE EDU that we feel would be helpful, outside of the instructional design models that we use.

Modular Design

When designing a curriculum, we've found that designing for modularity is important for flexibility. Much like designing systems, we want to reduce single points of failure. Tools, technologies, priorities, goals, and even leadership, changes meaning, so tying yourself to any of these items gives you points of failures when they change, and can cause unnecessary toil as the years pass. Therefore, what SRE EDU has done is create a modular style of development, based on reducing single points of failure. We modularize around those principles, to incorporate the best practices and tools at the time.

Let's take our monitoring course as an example. Our class focuses on monitoring philosophies rather than teaching the entirety of a specific monitoring tool. What this enables us to do is teach the principles of monitoring first, and what we mean by monitoring. Then, we apply those principles to the current tool of choice. The principles (generally) should not change for quite a bit a time, and the strategies for what and how we monitor should be stable. We generally leave those in place and keep those principles as a core part of the class.

The hands-on, practical application of those principles can then be placed in a module within the class, which can be replaced as tools, best practices, and so on change. This happened a few years ago, when we transitioned from our legacy monitoring tool to a new tool. The principles still applied. What changed was how we incorporated the new features, layouts, dashboards, and so on. The ability to pull and replace was much easier as we kept the training experience modular, by design.

No Single Points of Failure

A single point of failure is a part of an overall system that, should it fail, the entire system stops working. For each of our hands-on activities, there are at least three points or clues that can be used to help solve the problem. We do this because we don't want students to just discover a single point of failure and then find the solution from that. If there is only one clue or one way to discover the problem and students miss it, they effectively cannot get to a resolution. When designing hands-on training, plan for how students discover problems, and eliminate single points of failure.

Designing for modularity comes back to your learning objectives. Modular design is about designing around your learning objectives and not about content coverage. If you have content that is not covered, don't add it to a course because it fits with the topic. Verify that the content meets your learning objectives. Keep your learning objectives at the forefront while you arrange all the content and classify them. Identify what are principles, what are tool specific, what are best practices, what are patterns, antipatterns, and so on. Identify what content changes frequently and distill the general principles in that content. Based on what you identify, relate them back to your learning objectives. After you have that categorization complete, you begin to see how this topic fits with your overall training program.

Train the Trainers

For instructor-led courses, preparing your trainers is important. It gives them two things:

- Confidence to teach the class, especially hands-on activities
- Ability to set the larger context of the class(es)

SRE EDU orientation is a collection of classes developed and cadenced in a way that builds skills with our learners. Because our instructors are SREs, they might teach as few as one of our classes per session. New trainers might look at their class in a vacuum, meaning that the class stands alone, and trainers might teach it that way. SRE EDU developed Train the Trainer to address this. Each class builds and connects to each other, to support our hands-on activities. Due to this, we wanted our trainers to develop teaching strategies using the overall curriculum story (things break in interesting ways / we teach you how this happens and methods to combat that). The overall curriculum story acts as a guide for the trainers.

When creating courses, think about your trainers. During Train the Trainers, we cover the following:

- Overview of the curriculum (purpose and overview of the classes)
- Schedule that students follow, to go through the classes
- Rundown of each class—its pitfalls, gotchas, lessons learned from pilots, and so on
- Learning/teaching strategies
- Practice time for hands-on activities

Some of our learning objectives for the Train the Trainer course are as follows:

- Demonstrate facilitation skills used in a hands-on scenario, using the teach-back method in class
- Identify how the overall curriculum supports the hands-on activities, using the Triage, Mitigate, and Resolve model
- Use the lesson plans, one-pagers, and/or guides to plan your teaching strategies

When trainers identify the relationship between the courses they teach, the objectives and purpose of the curriculum, and the activities, trainers have an easier time teaching. To help with some of this, we also create lesson plans that highlight talking points. It's important to note that these are not scripts. We don't want our instructors to read verbatim from the lesson plan; that defeats the purpose of classes. What we really want is for the SREs who are teaching the class to share their domain knowledge and experiences, within the constraints of the program. The lesson plan helps our trainers identify the purpose of the slide, video, and other training materials, and gives them some ideas on what could be said, not what they should say. Trainers should bring their own stories and experience to the class.

Pilot, Pilot, Pilot

Much like launching a service or software program, we want to canary test our training. When we launch new training, we do multiple pilots and staged rollouts. Piloting is a process in which you are testing everything in your training. You are identifying how well your training materials work, and how well the activities work. The benefit of pilots are finding those unanticipated bugs in your training. This is particularly useful for observing students' reactions, getting a pulse on what the class flow is like, and seeing whether your timing is correct. We let our students know up front that they are participating in a pilot.

We also give students time at the end to give feedback. We ask questions such as: What worked? What could be better? What was not clear? What would you do differently? All of these questions help guide our iterations and design, before a final launch. Because SREs at Google are also globally distributed, we pilot in different locations. We learn a lot about what gets lost in translation. An example of this is when an American trainer made an analogy to a rodeo for one of our team members in Zürich. A look of bewilderment washed over the student's face because he did not know what a rodeo was. Had this training gone out to others with that analogy included, a large portion of our teams would have spent their time using Google Search to figure out what a rodeo was instead of focusing on the class.

One pilot for medium and large training programs is not enough. Doing multiple pilots saves you time by avoiding the toil of launching and iterating at the same time. SRE EDU always has a pilot plan for each training. We pilot in multiple locations, with different audiences, to get as much feedback as we can. We plan a few days to a week for update time, post the updated content, based on feedback, and then run another pilot. Generally, for large programs, three pilots give you a lot of information to work with and will address most, if not all, major concerns.

Making Training Hands-On

SRE EDU Orientation started as a collection of classes that were deemed important to all SREs at Google, during their first two weeks on the job. The topics ranged from culture classes, to general courses on tools used by a majority of SREs. These classes were put together into a collection of courses that became the first iteration of SRE EDU Orientation. The classes were lecture heavy, with little to no hands-on experiences for students to apply what they were learning.

The most frequent feedback we received early on was to make the training more hands-on. What the students were telling us is that they wanted a way to immediately apply learned skills. Complex and oftentimes abstract principles stick better through hands-on activities. It's one thing to talk to students about draining a cluster, it's quite another when a student can drain a cluster themselves, view the monitoring data to validate that the drain worked, and analyze how they know that it worked. We investigated more about what problems we were trying to solve. It wasn't enough to create something that was a gimmick or fake. We wanted to create something that was real, used real production services, and the exact same tools that experienced SREs use at Google. We wanted to give an experience that was as real as possible. For SRE EDU to move from lecture-based training to hands-on training, we started—again—with defining learning objectives, and the idea that we should break real things and fix them, using real tools.

We defined a few learning objectives for the hands-on activities:

- Use the page-receiving device to acknowledge a page within two minutes of paging
- Use the monitoring tool to investigate what "things" are affected by the outage
- Use learned skills to identify the problem from each breakage
- Mitigate the identified problem using taught tools and processes
- Resolve the issue and validate the resolution using monitoring graphs and dashboards

From these learning objectives, we created a service in production for the specific purpose of breaking. It turned out that this was very difficult to do at Google because our systems tend to prevent breakages/outages from happening in the first place. We were, however, able to create four breakages that we use in the class.

The Breakage Service

As mentioned earlier in Chapter 3, "Current state of SRE orientation at Google" on page 39, we created a very simple but functional photo upload service called the Breakage Service, which contains many parts of a real, scalable service. We wanted to mimic a real production service as much as possible. The frontends and backends of the service are distributed over multiple datacenters and we also have some load generators running, to make sure the service looks "alive." The breakage service has a storage component, complete with quota and redundancy, and is housed in multiple data centers, in multiple regions. It also has a user-facing front end, monitoring, code, and network components, pager queue, on-call rotation, and so on. All of this runs on real Google production systems, and just like SRE, the SRE EDU team gets paged if dependent systems go down.

To give students an experience similar to a real SRE team, we don't have just one installation of the service (which we call a stack)—we actually have multiple complete instances. These stacks were built using Google's frameworks, which enforce SRE best practices; for example, for continuous integration (CI)/continuous deployment (CD), and monitoring and alerting. When students work with the breakage service, they have full ownership over their stack. They work in groups of three, and can bring down jobs, erase storage if they want to, and manipulate the service just like any other production service. They do this without causing harm to other students or "real" Google production jobs.

After the students get used to the service, we hand them a mobile phone that acts as a pager, and instruct the system to break. The students then receive a page. We designed the curriculum such that classes are followed by a breakage exercise in relation to that class. That way, the students are able to practice working with the tools used when handling an outage, and they follow Google's incident management procedures to Triage, Mitigate, and Resolve (TMR) the problem.

Part of an SRE's job is troubleshooting and finding root causes. Therefore, the breakage service we built has some deliberate weaknesses engineered into it, to simulate outages and have the new SREs exercise troubleshooting. For example, we have a deliberate bug in the "search by hashtag" code, in the backend that triggers on a specific hashtag; the binary crashes. When we “turn on” the breakage, we instruct the load generators to change the mix of hashtags it requests the service to look for. That way, more random crashes occur, and eat away at the service’s Service-Level Objectives (SLOs, see Chapter 4 of the *SRE Book*).

The breakage service is one important aspect of our training. The service, when paired with the curriculum, became a powerful teaching tool. To take this service and design around it, the SRE EDU team went back to instructional design principles to learn how best to implement it. We came away with some two key principles for how hands-on training becomes effective in practice: play to learn and collaboration among learners.

Play to learn

One of the ways hands-on learning works to reinforce and expand on skills is through play. When we developed the breakages and training, we wanted learners to feel that they were playing with the service. Students are given the space and time to explore, learn through failure, apply the scientific method to a problem space, and work through TMR. As mentioned earlier, adult learners in business environments are often stressed with being new to a company, pressure to “pass” a class, and confidence concerns. We wanted to limit this as best as we could. Hence the breakage system feels more like play, especially in the first encounter.

Collaboration among learners

Playing is generally a social activity. Think of sports, board games, parties, and so on. Therefore, we wanted to make sure that learning in this environment maintained a social aspect to it. Our hands-on activities were done in groups of three. We believe that this collaboration is both reflective of the work environment, and also a healthier way to learn new skills. We break students into groups, based on their self-reported industry experience. We make sure to mix the industry experience level within teams. No one team should have all the industry veterans, nor all the newbies.

During our pilots, we tested with various group sizes and settled on three being the ideal. The reason for this is that with too large of a team, you have individuals who won’t participate as much. If you have too few, getting stuck becomes easier due to less ideas shared. This breakdown also helps with imposter syndrome. When teams comprise individuals from different levels and types of experience—newbies, industry veterans, and internal transfers (refer back to Figure 2-3), teaching is happening in all sorts of ways between the team members. When a newbie hears an industry veteran say, "I don't know," they realize that it's okay to not know everything, even after 10 or more years of experience.

Scaffolding: Build, Stretch, and Reach

The hands-on activities are built to scaffold the learning experience. In other words, we start with very strong hand-holding to build skills and confidence. Then, as we progress through the breakages, we do less and less hand holding, until the final breakage, when the vast majority of students are progressing on their own.

We use a model for each of the breakage scenarios that students might not consciously feel (but our trainers do). We do this with the understanding that in each progressive scenario, we want our students to build, stretch, and reach their skills.

In Table 4-1, we show four of our breakage scenarios and what teaching style we use. In Breakage 0, the instructor is driving the entire time (*build*). They run through all the steps from start to finish, from pager to resolution. This demonstrated behavior is a model we want students to build from. In Breakage 1, the instructor allows students to drive, from pager to resolution, but there is a lot of guidance provided here. We remind students of the tools they've learned and ways to navigate it, suggest looking at certain monitoring views, and *level-set* frequently.

Table 4-1. Hands-on activity teaching style

Hands-on activity	Teaching style[a]	Instructor tasks
Breakage 0	Sage on the Stage	• Drives entirely • Follow along with me • Build skills
Breakage 1	Coach	• Get you started • Collective level setting • Remind students of tools/systems • More interaction with teams • Stretch skills

Hands-on activity	Teaching style[a]	Instructor tasks
Breakage 2	Guide on the Side	• Less interaction with teams • Answer more tool/service use questions • Collective level setting • Stretch skills
Breakage 3	Consultant	• Answers questions from individual teams • Less collective level-setting • Watching teams, more than interacting • Reach skills

[a] Teaching styles obtained from this article: "From 'Sage on the Stage' to 'Guide on the Side': A Good Start"

Level-setting is when we take a pause in the breakage. Level-setting happens throughout each breakage. During that time, we ask students a few questions, as a group, such as these:

- What tools have you used so far?
- How did you discover what you found?

We ask questions in this way so that we don't have students spoiling the activity, while allowing students who might not be quite as far along to try out some of the suggestions other students have provided. Level-setting allows instructors to get a pulse on the class, and encourages students to teach one another as a larger group.

In Breakage 2, we move to less hand holding, but still answer questions as students have them (*stretch*). We continue to do level-setting. Finally, by the time students get to Breakage 3, they are operating independently (*reach*). Often, some students finish early and can help other students, if needed. The cadence of training was built this way to help students confidently apply the learned skills that they will use in their day-to-day work.

We let students know that the hands-on activities are there to help build their confidence in what they've learned and apply it in a safe environment (see "Build confidence" on page 29 in Chapter 2). We want students to make all their mistakes here with the activities and learn from those mistakes. We preface every hands-on activity, every time, with a brief conversation highlighting that troubleshooting is a series of failures, and it's ok to go through a process of *not* solving the problem, especially if it helps you rule things out—just like the scientific method. We also create moments of discovery—called eureka moments—throughout the triage and mitigation steps, and have our SREs simulate working very closely with their developer teams by role-playing SRE to Developer communication, directly in the training. Our hands-on trainers act out the role of SRE or Developer, to give students practice on escalating issues to a developer or SRE.

Evaluating Training Outcomes

As mentioned earlier, evaluating training is a process that happens throughout the experience—not just at the end. During training, structure your learning objectives to help assess how students are progressing through the courses. Because the learning objectives are based on things students are doing (action verbs), you observe those in real time. That is one data point you use to evaluate your students' progress and training outcomes.

When We Evaluate

We also want to know how our students are perceiving their training outcomes. Therefore, we ask them, multiple times. We like self-reported data because it gives us an idea of how the training affects the growth of students, as they perceive it. We survey our students at different times: before their training, after their training, 30 days after, and six months after. We also survey the hiring managers who receive our students. We ask our managers the same questions and get their take on their employees' skills, post training. As with the students, we survey our managers 30 days after, and six months after receiving their employee. Finally, we also survey our instructors around their training experience. We use all this data for the explicit purpose of improving our program. We are not evaluating the students here—it's all about how the program is working or not working, and gives us data points to justify making changes as needed. We explicitly tell our students that.

We evaluate at different points because as students ramp into their job roles, their ability to recall and apply concepts becomes real. Sometimes, what they thought they could do was not as solid as they thought, which we capture through feedback. Often, they realize they *can* apply and recall things very well, signaling that we are teaching the right things. In some cases, we discover that we need to improve or add something to the training that was not there before.

How We Evaluate

Surveys often take the form of a "rate yourself on a scale" model. In our early iteration of SRE EDU Orientation, we used a point scale model for our questions. We discovered that the signals developed were too noisy. If you have a seven-point scale, you might ask someone a question like, "How well can you identify the health of your service by using monitoring graphs?" They might rate themselves a five. However, what does that mean? Does that mean yes? No? Maybe? The result is that we make presumptions about the student's responses and find no actionable results.

So, instead, we adopted a more measurable "scale" for our surveys. We partnered with Google's broader engineering training team to adopt the same verbiage around survey data. Each question we ask is tied back to skills that we know are an important part of the job and the learning objectives. The scale is reflective of actions that students think about and answer. Our scale is as follows:

1. No idea where to start
2. Could do it with someone's help
3. Could read documentation and figure it out myself
4. Could do it mostly from memory
5. Could teach others mostly from memory

For example, we ask students to rate their ability to use monitoring (and alerting) to make rational decisions about the production service that they support. Students choose from the five choices. Notice also that the five student response options are action-based, much like the learning objectives. Having the responses worded this way helps identify a level of proficiency for each surveyed task. One thing to note: SRE EDU believes that teaching is a function of learning (see Chapter 1, "Teaching to Learn" on page 12). We know how well someone understands something if they can pass that information on to others. Because of this belief, our highest rating is that a person could teach others from memory. We built that principle into our survey response scale. Also, note that none of the options are necessarily bad. Having a student read documents to solve a problem is not a bad thing. It implies they know where to look and how to use a company's knowledge base of resources to find answers. Having a student get help from another, is also not bad. It means they have an idea who to talk to, whether that other person is on their team or not.

Now, if a majority of new SREs choose "no idea where to start" in the post–SRE EDU Orientation survey, we know we have a problem. There is something in our monitoring class that we need to change, so we start looking at the content, the trainer involved, and the hands-on activity session's feedback. In other words, based on both individual and collective responses, we have an idea of what to do next to improve the training.

Of course, there are many ways to evaluate training, and we've only shared one way which the Google SRE EDU team has chosen. Evaluating training outcomes is something you want to think about early in your training development, and tie them back to your learning objectives. Also, think about how you present this data to your colleagues. In our case, we have regular meetings with our stakeholders, to show the data and guide decisions for our entire program. It's much easier to make decisions and changes based on data and evidence than on presumption.

Instructional Design Principles at Scale

Although the preceding discussions relate to large, hands-on training systems like Google's SRE EDU program, and are appropriate for large company trainings, small and medium-sized companies might not have the ability to do all this. Google is a large corporation, with resources that many companies do not have. For smaller companies and SRE teams, our recommendation is to focus on four main things:

- Create learning objectives
- Follow a learning design model (ADDIE or another)
- Create a few simple activities that allow students to "play" and apply learned skills
- Create an evaluation method that measures how well your training is meeting objectives

Learning objectives are critical to a training program and cannot be ignored. Defining your learning objectives is the first step to designing training. You always refer back to your learning objectives, when designing training. When cornered with decisions or problems in your design, ask yourself, “Does this meet my learning objective?” If it doesn’t, either the learning objective needs to change or you discard or change your design idea.

We also recommend following a learning design model like ADDIE because it gives you a framework and basic instructional design tools to develop your training, in a generally linear fashion. Following a model helps you define your workflow in an organized way and keeps you moving forward. The cost for following a model is low, and how you organize your tasks falls in-line with the different stages of the model.

Hands-on activities should be relevant to the job your trainee will do. It can be something simple but impactful. For example, if you are trying to teach your students mitigation techniques for a problem, use the existing outage information you have and turn it into a case study or a game. Have students identify where things went wrong and why. Have students describe different ways in which they could mitigate the problem and the pros/cons of each. Use your learning objectives to align your activities with expected behavioral outcomes.

Creating an evaluation method tells you how your learning program and students are doing. Use your learning objectives to build the evaluation model, and implement a rating system to give you actionable results. In other words, based on what your students report, you should be able to identify areas to improve your training.

These tips for instructional design principles are meant to guide you. It’s not the only way to develop or design your training, but it’s one way we found that worked for SRE EDU.

Conclusion

In this chapter, we explored instructional design principles and how to create training that sticks with SREs. We talked about identifying training needs, building your learner profile, and specifics on how to design training content. We discovered the advantages of hands-on training, and the importance of piloting and evaluating your training. In the next chapter, we take evaluation a step further by learning how to “SRE” your SRE training program.

CHAPTER 5
How to "SRE" an SRE Training Program

In this chapter, we discuss how you can apply the SRE principles, practices, and culture, which you are likely covering in your SRE education program, to your *training program itself*, to deliver efficient program operations. Program operations cover things like logistics, reporting, and material curation. Let's discuss in detail how to "SRE" an SRE training program.

Applying SRE Principles to Your Training

Teaching SRE best practices means that you need to set an example—live by the rules you set.

"SRE'ing" your SRE Training Program

In *Site Reliability Engineering: How Google Runs Production Systems*, the Service Reliability Hierarchy is used to describe the health of a production service. In a similar way, the health and maturity of an SRE training program can also be framed in the context of the SRE *Training* Reliability Hierarchy (see Figure 5-1).

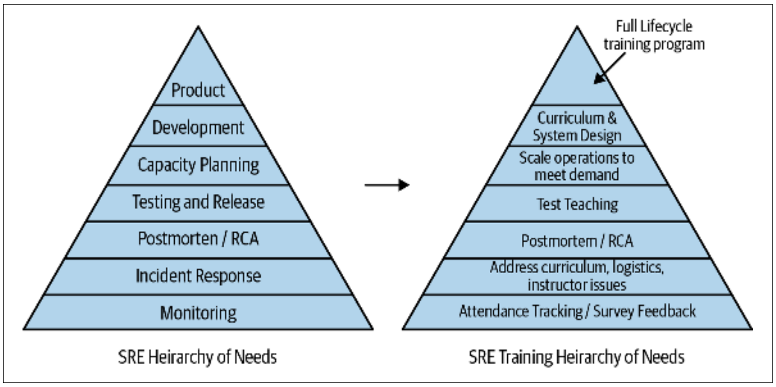

Product
Development
Capacity Planning
Testing and Release
Postmorten / RCA
Incident Response
Monitoring
SRE Heirarchy of Needs
Full Lifecycle training program
Curriculum & System Design
Scale operations to meet demand
Test Teaching
Postmortem / RCA
Address curriculum, logistics, instructor issues
Attendance Tracking / Survey Feedback
SRE Training Heirarchy of Needs

Figure 5-1. SRE Training Reliability Hierarchy

Although the Service Reliability Hierarchy is usually discussed in a bottom-up fashion, important insights and understanding can be had by looking top-down. Starting with "Product" at the pinnacle, the product needs the development process in order to exist. The development process needs a testing and release process to ensure that the product works, is fit for a purpose, and can be deployed to production. Testing needs postmortems and analysis of root causes to inform what issues need to be proactively looked for and tested. Postmortems are informed by rigorous incident response or the ability to reactively adapt, based on data. Incident response or adaptation is possible only with good monitoring. If you don't have one of those, you can't possibly hope to build a reliable and effective product.

Now let's look at how each element in the Service Reliability Hierarchy maps into the training space to ultimately build a reliable and effective training experience.

Monitoring and Measuring

Without monitoring, you have no way to determine whether your training program is effective. For a training team, the training program itself can be viewed analogously to the production service that is supported.

Analogous to best practices for an SRE team supporting a production service, define Service-Level Objectives (SLOs) for the training program and communicate them. Think about what's important. SLOs are critical because this is what you ultimately monitor and measure the program against. For the SRE EDU team at Google, it's things like coverage (what percentage of the target audience takes the training) and response time, because we owe it to our students and volunteers to acknowledge their feedback in a timely manner (e.g., replies to email expected in two business days). Another SLO target might be around student satisfaction, or the delta of reported confidence before and after the training. You could also benchmark confidence on specific, job-related tasks or even test competence on these tasks explicitly.

The simplest form of training program monitoring and alerting is survey and attendance tracking data. We recommend starting with simple metrics that are easy to measure, and build from there (e.g., Net Promoter Score,[1] self-reported confidence, parsing open comments). You can get more sophisticated by measuring time to on-call, time to tackle their first bug, and time to first meaningful code submitted. However, it's important to weigh the toil of measuring the thing versus the benefit of having that measurement. Do your metrics map to the student experience and desired learning outcomes?

If attendance drops below the SLO threshold, this could trigger an analysis and even a postmortem to troubleshoot what went wrong:

- Is the training not effective, and people are voting with their feet?
- Is there an operational issue that the students don't know where to be and when?
- Did the instructor not show up?

By reacting to your monitoring and understanding things that are suboptimal with respect to the student experience, you continuously drive change and make a conscious decision about what to address.

Consider also the importance of observability. Observability enables a team to gain important insight into the behavior of the software systems it supports.[2] By making the training program hands-on, instructors and program owners can observe concrete, job-related

1 The percentage of favorable minus unfavorable divided by total answers to the question: "Would you recommend this training to others?"

2 The Importance of Observability https://oreil.ly/cTN9Z

behaviors. Building in observability to a training program is foundational to success. If students perform concrete, job-related behaviors in class with ease, you know you are setting them up for success. If they cannot, you get valuable feedback on ways to improve the program.

Similar to running a production service as an SRE, it's important to point out that you should never set an SLO of perfection for a training program, or you are setting yourself up to fail. At some point, the cost to deliver improvements to the student experience outweighs the potential benefits. A training team has a finite number of cycles to spend, and should be thoughtful in how they spend them. For example, on the SRE EDU team at Google, we set a target coverage of 99+% for all students globally (99+% of students complete SRE EDU Orientation in a year). Meanwhile, for our SRE EDU Going On-call curriculum, we target a lower coverage and a lower Net Promoter Score—we recognize that a distributed training experience is more difficult to score high on for student satisfaction compared to an in-person training experience. Choose to make the curriculum and student experience "good enough" so that you don't burn out the team, and allow the team to cover a wider scope.

Incident Response

"Incident response" in the context of an SRE training program should be considered over a longer time horizon than a production outage. Even though teams strive to resolve a production outage or issue within minutes or hours, it can take days or weeks to uncover and address a training "incident." One example in which incident response can be invoked is by monitoring survey data. Because of the longer time horizon, we aren't talking about incidents in the "someone gets paged" sense. Someone on the training team is designated as the "on-call" on the team, who regularly monitors the survey results. If a survey response comes in for which certain questions are scored negatively by a student (e.g., "How likely are you to recommend this training program to others?" rated "Unlikely" or "Very Unlikely"), this calls for investigation and follow-up to understand what went wrong. Was it a curriculum issue, a logistics issue, or an instructor issue? It's important to Triage, Mitigate, and Resolve (TMR) student experience issues. We do this to drive improvements to the training program in ways that matter to our students.

A training program might even have issues that require reactive and fast action. Having an escalation point in each location, where the training is delivered, helps to ensure that professional/corporate code of conduct is followed by students and instructors, and any inappropriate behavior is safely escalated. This ensures a more inclusive environment for all.

In addition to "incident response" for the operational elements of the training program, a more traditional incident response model should be deployed. If you are using hands-on exercises that rely on underlying infrastructure, having a traditional on-call rotation and incident response protocol is important. The on-call rotation should be staffed, leading up to and during the training program, and downgraded to "on-duty" when there is less urgency and impact to the student experience during off weeks (e.g., if you only offer the program once per month).

Postmortem and Analysis of Root Causes

If issues arise that are serious enough, a postmortem might be warranted. As with SRE in general, blamelessness should be maintained at all times. No one should ever feel like their job might be in jeopardy if they make an honest mistake. Writing a postmortem when there is an issue that significantly affects the student experience allows the training team to define action items that drive real improvements to the program. Similar to running an SRE production infrastructure service, the actions should be focused on eliminating entire classes of issues. For example, if there is an incident around instructor quality, an action item in the postmortem might revolve around how to upskill the entire pool of instructors, or give the best instructors first dibs on teaching each time, in an automated way.

Testing and Release Procedures

Testing in the context of an SRE training program is about making sure that the training material is fit for its purpose. As discussed in Chapter 4, *Instructional Design Principles*, any new class materials should be piloted with a small group, with success criteria defined in advance. For the test teaching session (pilot), make it clear to the students that they are the guinea pigs for this new material, and leave time at the end of the session for feedback. If the pilot proves successful, introduce the class in production, in one site, if you run in multiple locations. If it goes well there, you deploy to the remaining sites in an accelerated fashion, or all at once.

Training release processes should also invoke a well-defined Train the Trainer process. The Train the Trainer itself is also a test, one that provides valuable feedback. In addition, Train the Trainer sets you up for success by familiarizing instructors with the learning objectives, materials, and the key points to convey. Train the Trainer might involve watching a reference video, shadowing and reverse-shadowing, a short lecture with Q&A, or an all-day or multiple-day session, depending on how complicated the material is and how difficult it is to facilitate.

Capacity Planning

Capacity planning is super-important in the SRE Training Reliability Hierarchy. In the case of training, capacity planning focuses on how to ensure that you have enough space to train everyone who needs to be trained. It also ensures that you get the best return on investment (ROI) for the program. For example, if there is extra capacity beyond those required to take the training program, how do you fill extra seats? Who gets priority among those who are interested but not in the target audience?

Capacity planning involves making sure that the location distribution and timing of the training is appropriate, you have enough facilitators to cover the load, and you have enough core team cycles to handle the communication overhead and logistics so you don't burn out the core team. In the case of core team capacity, the SRE principle of beating down toil through targeted automation comes into play, to free up capacity for more value-added work.

Development

Development in the SRE Training Reliability Hierarchy focuses on curriculum and associated system design. The SRE EDU team at Google started our orientation program by assessing what we thought every SRE needed to know during their second week on the job. In many cases, there were multiple slide decks covering this content. We assessed the best of the existing materials and launched a minimum-viable curriculum, so that we could get feedback from students, to drive improvements. From here, we worked with the team's instructional designer on a new model for the curriculum and built a new and improved version from the ground up. The development process involved building both educational materials (slide decks, instructor guides, Train-the-Trainer decks) and building educational infrastructure. We built a stack using production best practices that could be broken on demand. The curriculum relied on plausible storytelling—why the infrastructure was broken, and reasonable ways to fix it, which students on their second week of the job could reasonably be expected to solve, in small groups.

Product

The culmination of the training development process is a product that meets the needs of our students. At Google, the SRE EDU Orientation product encompasses short lectures that give students just enough breadcrumbs to TMR issues, using their initiative and other self-directed methods. The SRE EDU photos stack and associated breakages, discussed earlier, might arguably be the most important element in our product. Other important additions to the Google SRE EDU Orientation training product include the supporting documentation for instructors and operational components (mobilizing volunteers and getting folks to the right place at the right time, giving out SRE EDU branded swag, and copies of the *SRE Book*—our "textbook").

As discussed earlier in this report, we piloted and then launched a hands-on training program that drove considerable improvements in Net Promoter Score (+5 percentage points) and student confidence (+14 percentage points in students reporting one-point increase in confidence, and +13 percentage points increase in students reporting 3+ point increase in confidence, in survey results). We also saw survey comments, such as "please make it more hands-on," drop down to zero. Survey comments are analogous to systems logs—helpful for troubleshooting.

Other Considerations

Looking beyond the SRE Training Reliability Hierarchy, there are a few additional ways to "SRE" an SRE training program. We do this by applying the following SRE principles:

- Don't be a hero
- SRE is about culture not tools

- Ruthlessly automate toil

In SRE, heroism tends to mask problems until they become even bigger. In fact, this is a key point of SRE culture that we reinforce in our classes. There can be a temptation to be a hero for your training program. Guard against this by being very clear on scope and available resources. Create a decision tree for any additions or changes, to take the emotion out of responding to someone's request—for example, to add their pet project to the training. If something clearly adds toil to the team, push back or ask for more resources as a condition of taking it on.

Remember that SRE is about culture and not tools. Make sure to imprint that culture on students early. We do that with specific classes in our orientation program, such as "Role of the SRE" and "Launches, Rollbacks, and Postmortems". All of the SRE EDU Going On-Call curriculum classes also have an element of SRE culture. Make sure that volunteer instructors model the behavior you want students to exhibit. Don't just teach the tech, teach the culture.

Also, the more you scale, the more important operational considerations become. A key SRE principle is to generously implement automation to ruthlessly eliminate toil. In addition to reducing toil, use automation to improve reliability. For example, having a human send out a survey to class participants is not as reliable (e.g., due to forgetfulness, vacation time) as having a solution to deploy the survey automatically, without human intervention, that alerts when it fails to run. Another example when automation might be beneficial for improving reliability is with instructor sign-ups. We rely on a spreadsheet for instructors to sign up to teach classes. Originally, human intervention was required to parse the spreadsheet and add the classes to the instructors' calendar. This was done about once each week. We now have developed automation that automatically adds classes to people's calendars when they sign up, on a daily basis. This decreases the likelihood that an instructor becomes double-booked between the time they sign up in the spreadsheet and when the class appears on the calendar.

Another example of toil-busting is around scheduling sessions in our learning management system. The process was originally manual and toil-filled, and took a good chunk out of one of our program manager's time, to manage each month. We now have a solution that automatically schedules orientation sessions each month, based on a template calendar.

Just like SRE teams for any production service, look for opportunities to free up cycles, to work on more value-added things and “make tomorrow better than today” for the team, and for your students. In our case, the time saved on instructor management and session-creation automation was reinvested in improving the process for communicating with students. We eliminated single points of failure by routing communication through a ticketing system rather than individual emails.

Managing SRE Training Materials

Whether you are just one person flying the flag for education at your organization, someone responsible for ramping up new hires on the team, or a member of a training team within a larger organization, managing your SRE training materials is super important. How do those who need them find self-study materials or classes? How do you ensure that there is a single source of truth? How is the content managed for freshness and reliability so that students trust in the material and have confidence that they are teaching themselves the appropriate things?

Strategies for Discoverability

Discoverability is one important element when it comes to SRE training materials and classes. You can create all the great content you want, but if no one knows about it or uses it, what’s the point of having it? For a small company, consider creating a simple, one-page index of training materials, with pointers to the materials themselves. For larger collections of materials, ensure that it’s searchable. We’ve found that Google Drive is a simple and searchable shared repository for training materials. For larger corpuses of material, at larger companies, consider buying an off-the-shelf learning management system (LMS), or use an in-house solution to surface training materials. At Google, we have an in-house built platform for hosting information about instructor-led and self-study classes. An LMS is also useful for collecting registrations for live classes, and information about who attended the training class or completed a self-study module. In addition, an LMS helps collect feedback on both class materials and instructors, which you can incorporate into your monitoring to drive improvements to your program.

Content Curation

Content curation is equally important. How do students know that the material is endorsed and can be trusted? No matter the size and maturity of your organization, lock down the materials to prevent unauthorized forking of online training materials and copies being released into the wild within your organization. SREs often have strong opinions and nuanced views on technical content. If you're not careful, you can find yourself with 20 copies of the same deck, each slightly different, and none that are completely up to date.

Make sure that the owner of the deck and when the content was last updated is clear. When does the material "expire," or when is it up for a refresh? Limit the ability to edit to your training team, and possibly a small group of subject matter experts responsible for keeping the material up to date when infrastructure changes. Bonus points if you implement automation to alert you if view or edit permissions are incorrect (e.g., signaling that an editor is inadvertently added).

The Google SRE EDU team manages our training materials in Google Presentations. Options for viewers to copy, print, or download the decks are disabled.

Content Freshness and Reliability

For rapidly changing systems, thinking through your freshness strategy is also important. How often do the materials need to be evaluated and updated? How can students tell when the materials were last updated and if they are still "fresh"? The Google SRE EDU team sets the permissions of our training materials to "everyone with the link can comment (within the google.com domain)." That way, students and instructors can provide feedback on the spot if a) something is unclear, and b) something appears to be out of date. When more invasive changes are required, we fall back to filing bugs against the training team. Bugs are created automatically when the freshness of the training materials expire, so that an assigned owner either makes the updates, or verifies that the content is still good. In addition, if problems with the material are discovered between review periods, anyone can file a bug against the training owner to trigger an update (e.g., if a key tool is deprecated or a dependency changes suddenly).

Conclusion

In this chapter, we learned about how to apply SRE principles, practices and culture to the SRE training program. Parallel to the Service Reliability Hierarchy, we followed similar processes for training, including monitoring and measuring, incident response, and testing the training program. In addition, we applied SRE principles to the training program, such as "Don't be a hero," "SRE is about culture," and "Ruthlessly automate toil." Lastly, we looked at how to manage SRE training materials.

We've come a long way. In our final chapter, we wrap up what we've learned and summarize key takeaways from this report.

CHAPTER 6

Summary and Conclusions

We've looked at a variety of training practices for ramping up SREs and keeping them up to date with continuous education. We've seen that for students, oftentimes, the scariest part of their job is going on-call. Their worries and imposter syndrome can actually hinder them from ramping up. This is why it's important to concentrate on instilling confidence. Make sure there are plenty of hands-on activities in the training program to give students confidence.

Setting up a safe exercise environment, be it large (like Google's setup) or small (in one or more virtual machines on a laptop PC) is important. The students get to practice what they will do in their jobs when they actually have to troubleshoot, without the pressure of time or fear that they might break something.

If possible, have students practice SRE culture: reach out to other SREs or developers, and read and write blameless postmortems. This is especially important when you hire people from an environment without an existing SRE culture. If even a small number of people do not buy into the blameless postmortems, for example, you get a situation that is difficult to recover from.

Continuous education is also important, but it has less to do with confidence building and more to do with getting people to learn specific techniques. For this situation, having slide decks and recorded videos available can help students quickly absorb new material, but here, too, exercises help accelerate the learning and retention of the new material. Of course, you should have some structure in place to regularly verify the freshness and continued validity of the training materials. In general, SRE principles should also be applied to your training program: monitoring results, and automating toil away are two important factors we've discussed.

We've also discussed how your classes can be designed using instructional design best practices. No matter the size of the environment, you should develop classes for wherever the students are in their SRE journey. Therefore, it's good to clearly specify your learning objectives before designing a class.

On a final note: when you find that education is lacking, it's tempting to "just build some slide decks" and be done with it. It's important to take a step back, however, and think about what it is you really want to achieve, and design a curriculum based on that. Keep a finger on the pulse of your students. Monitor how they are doing, and what their feelings are about the curriculum. Notice where you can improve the program, and act on it.

APPENDIX A

Example Training Design Document

World of SRE Design Doc

Date: 2019-08-21

Author: evebob@

Status: Draft | In Review | Final

Overview: The purpose of the World of SRE class is to equip new hire SREs globally with a baseline knowledge of Google's SRE culture—its philosophy and ways of working. SREs come into Google with little practical knowledge of how SRE works at Google, thereby causing increased ramp-up time to their teams. This training targets all new hire SREs globally, as a class to take to help reduce the ramp-up time. This training takes the foundational tribal knowledge within SRE at Google and passes that down to new hires.

Learner Profile

Roles

- Primary Audience: One-week-old Site Reliability Engineers (primary audience)
- Secondary Audience: New SREs transferring in from the Software Engineering ladder

- Other Audience: Existing SREs who want a refresher

Population

- Primary Audience: We anticipate to hire about 300 new SREs in FY2019
- Secondary Audience: We have a potential for transfers from 100 open headcount that could be anywhere from 10 to 60 transfers out of the 100
- Other Audience: There are about 200 existing SREs that may enroll in the class, either out of curiosity or to refresh

Geography

Primary attendees will be located in the following locations:

- Mountain View (MTV)
- Pittsburgh (PIT)
- Dublin (DUB)

Experience with Content

- None
- Minimal

Course Overview

Introduction to the SRE organization. Focus on why SREs are an important part of Google, who SREs are, what SREs do, and how SREs do it. The importance of this class is to address the concern that new hire SREs didn't have a clear understanding of what the SRE role was and how it operates within the organization, leading to increased ramp-up time globally.

Total Class Time: 90 minute

Delivery Style: In-person class

Topics Covered	Reference Material
• Begin with company's mission statement.	SREs at Google
• How does SRE support the mission?	
• SREs at Google	Reference Doc
— Who we are?	HR data
— ~xxxx people tasked with maintaining Google's reliability	
— X% of overall engineering	
• Quotes from leaders with the SRE org	SRE Culture
• Why and How do we do that?	Reference Doc 2
• SRE Culture	
— Philosophy	High-Profile Examples
—— Hope is not a Strategy	
—— What we do	System A goes down
—— Shared learning/teaching	
— Support	Customer X impacted
—— What we support	
—— Availability science	Online News Clipping from
— Engagement	incident A
—— How we work with each other and our partners	
— Respect	Meet with Subject Matter
—— Team Members	Expert (SME) Bob who
—— Developers	knows all about incident A
—— Users/Customers	
• High-Profile Examples	
— Share stories of awesome SREs doing great things	Other
— Discuss public outages that were reported in the press	Reference Doc 3
— Excite new SREs: "This is the world you are joining and the types of impact you can have."	Reference Doc 4
• Getting Involved	
— Things you should do in the first month on the job	
— Attend Ops Review	
— Attend a Wheel of Misfortune	
— Have lunch with other SREs	
• Recap activity	
• Q&A	

Instructional Materials

Component	Purpose/Content	RACI[a]	WHO
Lesson Plan	• Instructor teaching strategies • Instructor speaker notes • Images of the slides • Gotchas—things to look out for	R	person1@ person2@
	• IMPORTANT: This is not a script, call that out • Use lesson plan <$template> for this training Place everything here that is important for the lesson plan	I,C	persona@ personb@ personc@ SME Bob@
Slides	• Graphic assets • Speaker notes in the slide deck • Use slide deck <$template> for this training • Link to <$graphic_assets> we can use	R	person1@ person2@
	Place everything here that is important for the slides	I,C	persona@ personb@ personc@ SME Bob@
Video	• Used only the Train the Trainer • Record one of our instructors to give a point of reference for new instructors • Pilot videos are ok to use (identify edit points)	R	person1@ person2@
		A	persona@ personb@ personc@

[a] RACI = Responsible, Accountable, Consulted, Informed. This is a method for defining roles and responsibilities in cross-functional work.

Learning Objectives

At the end of this class, a student should be able to:

- Explain what is meant by Site Reliability Engineering (SRE)
- Explain who make up the pool of SRE by using HR definitions and data
- Explain what we do by reflecting on stories told by our expert SREs
- Convert how we do things into scheduled tasks students should accomplish within their first month on the job

Trainer Profiles

- Trainers will be recruited from the existing SRE population.

- SREs selected must have been in role for at least 1 year.
- Trainers must have manager approval to participate.
- Trainers must take a Train the Trainer for this course.

Risks

We're aware of some risks that could impact our ability to deliver a successful training experience:

- Content hands-on activity not complete.
- Trying to cram too much information in the course.
- Trainers using words/vocabulary/acronyms without explaining what they are.

Success Metrics

- 95% of all new hires complete this training within their first month on the job
- A rating of 95% in self-reported satisfaction with the course
- At least 30% reduction in management self-reported ramp-up times

Pilot Plan

Before going live with the training, we would like to pilot this training with a handful of SRE EDU alumni and trainers. We will pilot the content, materials, and exercises with our alumni and hold a large feedback session, where their feedback will be translated into actionable items to fix before the global launch. Prerequisites for the pilot include the following:

- All instruction content (lesson plans, slides, etc.) must be complete
- Training of the trainers must be complete
- SRE EDU Alumni should register for the pilot in each location
- We plan to pilot in three locations
  - — Mountain View
  - — Pittsburgh
  - — Dublin

Train the Trainer Plan

We plan to host a 60-minute Train the Trainer for this course. The Train the Trainer will go over the entirety of the course content, lessons learned from the pilots, and tips/tricks for how to teach this class. We will also send out to our trainers the following:

- Video of the course being taught by one of the experienced instructors
- Lesson plans and slides

Logistics

- Work with our program managers to schedule and book rooms for the session
- Email communications to our participants at least 2-weeks before the Train the Trainer

Launch Plan

We plan to launch this training in a global, phased, roll-out. We will start with Pittsburgh as we have a smaller population, which should allow us to catch anything we may have missed from pilots. Dublin will be the second wave location, and then Mountain View.

- <date> Pittsburgh
- <date> Dublin
- <date> Mountain View

Timelines

These are developmental timelines. For a larger project timeline see <$link_to_project_plan>

Timeframe	Actions	Notes
Xx days	• Slides created • Lesson plan created	
Xx days	• Sent out for feedback to RACI participants • Incorporate first round SME and RACI feedback	See Instructional Materials for RACI group

Timeframe	Actions	Notes
Xx days	• Pilot in three locations • Instructional designer attends and incorporates quick action feedback during the pilot • Possibly record all pilots for instructional designer use	
Xx days	• Incorporate any other feedback from the pilot and send out for review again to RACI and SMEs	
Xx days	• Host a Train the Trainer	See Train the Trainer Plan
Xx days	• Launch training in a global rollout	See Launch Plan

Communications Plan

Below are the comms we are planning to send to teams, as we progress through development and launch.

Come teach the new World of SRE class

From: TBD

To: recent volunteers and shadowers

cc: some other group,

Subject: [Volunteer!] SRE EDU Instructors needed for new World of SRE class

Hello SRE EDU volunteer!

We have a new class that is all about you and the role you play at Google. If you've been waiting to get involved with teaching, now is the time! Next steps:

Get extra training to teach our lectures.

Train the Trainer: Monday, June 10, 10:30am - 11:30am. Register here.

If you're already been teaching, thank you for all your help and continued efforts.

Thanks again!

Come see a pilot of the new World of SRE class

From: TBD

To: recent volunteers and shadowers

cc: foo-bar-group,

Subject: Come see a pilot of the new World of SRE class

Hello SRE EDU Volunteer,

We are in the process of developing a new class that is all about SRE and the role SREs play at Google. Before we launch this training to a larger audience, we are running a pilot so we can gather feedback from experts like you.

Sign-up to attend one of our pilots in your location.

Pittsburgh - Pilot - World of SRE: Monday, May 10, 10:30am - 11:30am. Register here.

Dublin - Pilot - World of SRE: Wednesday, May 12, 10:30am - 11:30am. Register here.

Mountain View - Pilot - World of SRE: Friday, May 14, 10:30am - 11:30am. Register here.

Thank you!

Sign-Off

Please provide your short general feedback in the section below.

Username	Date	Comment
persona	2019-07-30	Initial comments
personab	2019-07-30	LGTM

About the Authors

Jennifer Petoff is a Senior Program Manager for Google's Site Reliability Engineering team based in Dublin, Ireland. She is the global lead for Google's SRE EDU program and is one of the co-editors of the best-selling book, *Site Reliability Engineering: How Google Runs Production Systems*. Jennifer joined Google after spending eight years in the chemical industry. She holds a PhD in Chemistry from Stanford University and a BS in Chemistry and BA in Psychology from the University of Rochester.

JC van Winkel has been teaching and creating curricula for UNIX-related and programming language courses since 1991, while working for AT Computing, a small courseware spin-off of the University of Nijmegen, the Netherlands. In 2010, JC joined Google's Site Reliability Engineering team, working on production monitoring, and he is now a founding member and the lead educator of the SRE education team, SRE EDU. He holds a BSc in Computer Science from Fontys University of Applied Sciences, Eindhoven, the Netherlands, and a MSc in Computer Science from Vrije Universiteit, Amsterdam, the Netherlands.

Preston Yoshioka is a Senior Instructional Designer for Google's Site Reliability Engineering (SRE) team. He has over 15 years of instructional design experience. Prior to joining Google in 2015, he spent 13 years at Apple teaching and developing technical hardware and software training.

Jessie Yang is a technical writer for Google's Site Reliability Engineering team. She works on documentation and information management for SRE, Cloud, and Google engineers. Prior to Google, she worked as a technical writer at Marvell Semiconductor. She holds a Master of Science from Columbia University.

Jesus Climent Collado is a Sr. SRE at Google, where he has been working since 2008. After his tenure as System Administrator and Sr. Architecture Engineer during his 8 year tenure at Nokia, he is now a member of Google's CRE team, helping companies meet their reliability requirements.

Myk Taylor is a Site Reliability Engineer (SRE) at Google. He works to extend reliability beyond the Google Cloud Platform, guiding Google Cloud customers through the process of adopting and implementing SRE practices and culture. He also teaches Monitoring and Alerting Philosophy for Google's SRE EDU curriculum. In his free time, he enjoys eating good food and rock climbing (though not at the same time).