The Evolving SRE Engagement Model

Written by Acacio Cruz and Ashish Bhambhani
Edited by Betsy Beyer and Tim Harvey

SRE Engagement: What, How, and Why

We've discussed in most of the rest of this book what happens when SRE is *already* in charge of a service. Few services begin their lifecycle enjoying SRE support, so there needs to be a process for evaluating a service, making sure that it merits SRE support, negotiating how to improve any deficits that bar SRE support, and actually instituting SRE support. We call this process *onboarding*. If you are in an environment where you are surrounded by a lot of existing services in varying states of perfection, your SRE team will probably be running through a prioritized queue of onboardings for quite a while until the team has finished taking on the highest-value targets.

Although this is very common, and a completely reasonable way of dealing with a *fait accompli* environment, there are actually at least two better ways of bringing the wisdom of production, and SRE support, to services old and new alike.

In the first case, just as in software engineering—where the earlier the bug is found, the cheaper it is to fix—the earlier an SRE team consultation happens, the better the service will be and the quicker it will feel the benefit. When SRE is engaged during the earliest stages of *design*, the time to onboard is lowered and the service is more reliable "out of the gate," usually because we don't have to spend the time unwinding suboptimal design or implementation.

Another way, perhaps the best, is to short-circuit the process by which specially created systems with lots of individual variations end up "arriving" at SRE's door. Provide product development with a *platform* of SRE-validated infrastructure, upon which they can build their systems. This platform will have the double benefit of being both reliable and scalable. This avoids certain classes of cognitive load problems entirely, and by addressing common infrastructure practices, allows product development teams to focus on innovation at the application layer, where it mostly belongs.

In the following sections, we'll spend some time looking at each of these models in turn, beginning with the "classic" one, the PRR-driven model.

The PRR Model

The most typical initial step of SRE engagement is the Production Readiness Review (PRR), a process that identifies the reliability needs of a service based on its specific details. Through a PRR, SREs seek to apply what they've learned and experienced to ensure the reliability of a service operating in production. A PRR is considered a prerequisite for an SRE team to accept responsibility for managing the production aspects of a service.

Figure 32-1 illustrates the lifecycle of a typical service. The Production Readiness Review can be started at any point of the service lifecycle, but the stages at which SRE engagement is applied have expanded over time. This chapter describes the Simple PRR Model, then discusses how its modification into the Extended Engagement Model and the Frameworks and SRE Platform structure allowed SRE to scale their engagement process and impact.

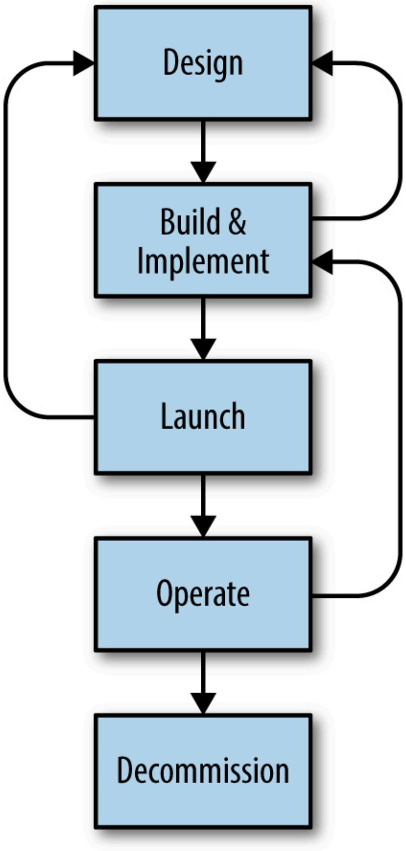

Figure 32-1. A typical service lifecycle

The SRE Engagement Model

SRE seeks production responsibility for important services for which it can make concrete contributions to reliability. SRE is concerned with several aspects of a service, which are collectively referred to as *production*. These aspects include the following:

- System architecture and interservice dependencies
- Instrumentation, metrics, and monitoring
- Emergency response
- Capacity planning

- Change management
- Performance: availability, latency, and efficiency

When SREs engage with a service, we aim to improve it along all of these axes, which makes managing production for the service easier.

Alternative Support

Not all Google services receive close SRE engagement. A couple of factors are at play here:

- Many services don't need high reliability and availability, so support can be provided by other means.
- By design, the number of development teams that request SRE support exceeds the available bandwidth of SRE teams (see Introduction).

When SRE can't provide full-fledged support, it provides other options for making improvements to production, such as documentation and consultation.

Documentation

Development guides are available for internal technologies and clients of widely used systems. Google's Production Guide documents production best practices for services, as determined by the experiences of SRE and development teams alike. Developers can implement the solutions and recommendations in such documentation to improve their services.

Consultation

Developers may also seek SRE consulting to discuss specific services or problem areas. The Launch Coordination Engineering (LCE) team (see Reliable Product Launches at Scale) spends a majority of its time consulting with development teams. SRE teams that aren't specifically dedicated to launch consultations also engage in consultation with development teams.

When a new service or a new feature has been implemented, developers usually consult with SRE for advice about preparing for the Launch phase. Launch consultation usually involves one or two SREs spending a few hours studying the design and implementation at a high level. The SRE consultants then meet with the development team to provide advice on risky areas that need attention and to discuss well-known patterns or solutions that can be incorporated to improve the service in production. Some of this advice may come from the Production Guide mentioned earlier.

Consultation sessions are necessarily broad in scope because it's not possible to gain a deep understanding of a given system in the limited time available. For some development teams, consultation is not sufficient:

- Services that have grown by orders of magnitude since they launched, which now require more time to understand than is feasible through documentation and consultation.

- Services upon which many other services have subsequently come to rely upon, which now host significantly more traffic from many different clients.

These types of services may have grown to the point at which they begin to encounter significant difficulties in production while simultaneously becoming important to users. In such cases, long-term SRE engagement becomes necessary to ensure that they are properly maintained in production as they grow.

Production Readiness Reviews: Simple PRR Model

When a development team requests that SRE take over production management of a service, SRE gauges both the importance of the service and the availability of SRE teams. If the service merits SRE support, and the SRE team and development organization agree on staffing levels to facilitate this support, SRE initiates a Production Readiness Review with the development team.

The objectives of the Production Readiness Review are as follows:

- Verify that a service meets accepted standards of production setup and operational readiness, and that service owners are prepared to work with SRE and take advantage of SRE expertise.

- Improve the reliability of the service in production, and minimize the number and severity of incidents that might be expected. A PRR targets all aspects of production that SRE cares about.

After sufficient improvements are made and the service is deemed ready for SRE support, an SRE team assumes its production responsibilities.

This brings us to the Production Readiness Review process itself. There are three different but related engagement models (Simple PRR Model, Early Engagement Model, and Frameworks and SRE Platform), which will be discussed in turn.

We will first describe the Simple PRR Model, which is usually targeted at a service that is already launched and will be taken over by an SRE team. A PRR follows several phases, much like a development lifecycle, although it may proceed independently in parallel with the development lifecycle.

Engagement

SRE leadership first decides which SRE team is a good fit for taking over the service. Usually one to three SREs are selected or self-nominated to conduct the PRR process. This small group then initiates discussion with the development team. The discussion covers matters such as:

- Establishing an SLO/SLA for the service

- Planning for potentially disruptive design changes required to improve reliability

- Planning and training schedules

The goal is to arrive at a common agreement about the process, end goals, and outcomes that are necessary for the SRE team to engage with the development team and their service.

Analysis

Analysis is the first large segment of work. During this phase, the SRE reviewers learn about the service and begin analyzing it for production shortcomings. They aim to gauge the maturity of the service along the various axes of concern to SRE. They also examine the service's design and implementation to check if it follows production best practices. Usually, the SRE team establishes and maintains a PRR checklist explicitly for the Analysis phase. The checklist is specific to the service and is generally based on domain expertise, experience with related or similar systems, and best practices from the Production Guide. The SRE team may also consult other teams that have more experience with certain components or dependencies of the service.

A few examples of checklist items include:

- Do updates to the service impact an unreasonably large percentage of the system at once?
- Does the service connect to the appropriate serving instance of its dependencies? For example, end-user requests to a service should not depend on a system that is designed for a batch-processing use case.
- Does the service request a sufficiently high network quality-of-service when talking to a critical remote service?
- Does the service report errors to central logging systems for analysis? Does it report all exceptional conditions that result in degraded responses or failures to the end users?
- Are all user-visible request failures well instrumented and monitored, with suitable alerting configured?

The checklist may also include operational standards and best practices followed by a specific SRE team. For example, a perfectly functional service configuration that doesn't follow an SRE team's "gold standard" might be refactored to work better with SRE tools for scalably managing configurations. SREs also look at recent incidents and postmortems for the service, as well as follow-up tasks for the incidents. This evaluation gauges the demands of emergency response for the service and the availability of well-established operational controls.

Improvements and Refactoring

The Analysis phase leads to the identification of recommended improvements for the service. This next phase proceeds as follows:

1. Improvements are prioritized based upon importance for service reliability.
2. The priorities are discussed and negotiated with the development team, and a plan of execution is agreed upon.
3. Both SRE and product development teams participate and assist each other in refactoring parts of the service or implementing additional features.

This phase typically varies the most in duration and amount of effort. How much time and effort this phase will involve depends upon the availability of engineering time for refactoring, the maturity and complexity of the service at the start of the review, and myriad other factors.

Training

Responsibility for managing a service in production is generally assumed by an entire SRE team. To ensure that the team is prepared, the SRE reviewers who led the PRR take ownership of training the team, which includes the documentation necessary to support the service. Typically with the help and participation of the development team, these engineers organize a series of training sessions and exercises. Instruction can include:

- Design overviews
- Deep dives on various request flows in the system
- A description of the production setup
- Hands-on exercises for various aspects of system operations

When the training is concluded, the SRE team should be prepared to manage the service.

Onboarding

The Training phase unblocks onboarding of the service by the SRE team. It involves a progressive transfer of responsibilities and ownership of various production aspects of the service, including parts of operations, the change management process, access rights, and so forth. The SRE team continues to focus on the various areas of production mentioned earlier. To complete the transition, the development team must be available to back up and advise the SRE team for a period of time as it settles in managing production for the service. This relationship becomes the basis for the ongoing work between the teams.

Continuous Improvement

Active services continuously change in response to new demands and conditions, including user requests for new features, evolving system dependencies, and technology upgrades, in addition to other factors. The SRE team must maintain service reliability standards in the face of these changes by driving continuous improvement. The responsible SRE team naturally learns more about the service in the course of operating the service, reviewing new changes, responding to incidents, and especially when conducting postmortems/root cause analyses. This expertise is shared with the development team as suggestions and proposals for changes to the service whenever new features, components, and dependencies may be added to the service. Lessons from managing the service are also contributed to best practices, which are documented in the Production Guide and elsewhere.

Engaging with Shakespeare

Initially, the developers of the Shakespeare service were responsible for the product, including carrying the pager for emergency response. However, with growing use of the service and the growth of the revenue coming from the service, SRE support became desirable. The product has already been launched, so SRE conducted a Production Readiness Review. One of the things they found was that the dashboards were not completely covering some of the metrics defined in the SLO, so that needed to be fixed. After all the issues that had been filed had been fixed, SRE took over the pager for the service, though two developers were in the on-call rotation as well. The developers are participating in the weekly on-call meeting discussing last week's problems and how to handle upcoming large-scale maintenance or cluster turndowns. Also future plans for the service are now discussed with the SREs to make sure that new launches will go flawlessly (though Murphy's law is always looking for opportunities to spoil that).

Evolving the Simple PRR Model: Early Engagement

Thus far, we've discussed the Production Readiness Review as it's used in the Simple PRR Model, which is limited to services that have already entered the Launch phase. There are several limitations and costs associated with this model. For example:

- Additional communication between teams can increase some process overhead for the development team, and cognitive burden for the SRE reviewers.

- The right SRE reviewers must be available, and capable of managing their time and priorities with regards to their existing engagements.

- Work done by SREs must be highly visible and sufficiently reviewed by the development team to ensure effective knowledge sharing. SREs should essentially work as a part of the development team, rather than an external unit.

However, the main limitations of the PRR Model stem from the fact that the service is launched and serving at scale, and the SRE engagement starts very late in the development lifecycle. If the PRR occurred earlier in the service lifecycle, SRE's opportunity to remedy potential issues in the service would be markedly increased. As a result, the success of the SRE engagement and the future success of the service itself would likely improve. The resulting drawbacks can pose a significant challenge to the success of the SRE engagement and the future success of the service itself.

Candidates for Early Engagement

The Early Engagement Model introduces SRE earlier in the development lifecycle in order to achieve significant additional advantages. Applying the Early Engagement Model requires identifying the importance and/or business value of a service early in the development lifecycle, and determining if the service will have sufficient scale or complexity to benefit from SRE expertise. Applicable services often have the following characteristics:

- The service implements significant new functionality and will be part of an existing system already managed by SRE.

- The service is a significant rewrite or alternative to an existing system, targeting the same use cases.

- The development team sought SRE advice or approached SRE for takeover upon launch.

The Early Engagement Model essentially immerses SREs in the development process. SRE's focus remains the same, though the means to achieve a better production service are different. SRE participates in Design and later phases, eventually taking over the service any time during or after the Build phase. This model is based on active collaboration between the development and SRE teams.

Benefits of the Early Engagement Model

While the Early Engagement Model does entail certain risks and challenges discussed previously, additional SRE expertise and collaboration during the entire lifecycle of the product creates significant benefits compared to an engagement initiated later in the service lifecycle.

Design phase

SRE collaboration during the Design phase can prevent a variety of problems or incidents from occurring later in production. While design decisions can be reversed or rectified later in the development lifecycle, such changes come at a high cost in terms of effort and complexity. The best production incidents are those that never happen!

Occasionally, difficult trade-offs lead to the selection of a less-than-ideal design. Participation in the Design phase means that SREs are aware up front of the trade-offs and are part of the decision to pick a less-than-ideal option. Early SRE involvement aims to minimize future disputes over design choices once the service is in production.

Build and implementation

The Build phase addresses production aspects such as instrumentation and metrics, operational and emergency controls, resource usage, and efficiency. During this phase, SRE can influence and improve the implementation by recommending specific existing libraries and components, or helping build certain controls into the system. SRE participation at this stage helps enable ease of operations in the future and allows SRE to gain operational experience in advance of the launch.

Launch

SRE can also help implement widely used launch patterns and controls. For example, SRE might help implement a "dark launch" setup, in which part of the traffic from existing users is sent to the new service in addition to being sent to the live production service. The responses from the new service are "dark" since they are thrown away and not actually shown to users. Practices such as dark launches allow the team to gain operational insight, resolve issues without impacting existing users, and reduce the risk of encountering issues after launch. A smooth launch is immensely helpful in keeping the operational burden low and maintaining the development momentum after the launch. Disruptions around launch can easily result in emergency changes to source code and production, and disrupt the development team's work on future features.

Post-launch

Having a stable system at launch time generally leads to fewer conflicting priorities for the development team in terms of choosing between improving service reliability versus adding new features. In later phases of the service, the lessons from earlier phases can better inform refactoring or redesign.

With extended involvement, the SRE team can be ready to take over the new service much sooner than is possible with the Simple PRR Model. The longer and closer engagement between the SRE and development teams also creates a collaborative relationship that can be sustained long term. A positive cross-team relationship fosters a mutual feeling of solidarity, and helps SRE establish ownership of the production responsibility.

Disengaging from a service

Sometimes a service doesn't warrant full-fledged SRE team management—this determination might be made post-launch, or SRE might engage with a service but never officially take it over. This is a positive outcome, because the service has been engineered to be reliable and low maintenance, and can therefore remain with the development team.

It is also possible that SRE engages early with a service that fails to meet the levels of usage projected. In such cases, the SRE effort spent is simply part of the overall business risk that comes with new projects, and a small cost relative to the success of projects that meet expected scale. The SRE team can be reassigned, and lessons learned can be incorporated into the engagement process.

Evolving Services Development: Frameworks and SRE Platform

The Early Engagement Model made strides in evolving SRE engagement beyond the Simple PRR Model, which applied only to services that had already launched. However, there was still progress to be made in scaling SRE engagement to the next level by designing for reliability.

Lessons Learned

Over time, the SRE engagement model described thus far produced several distinct patterns:

- Onboarding each service required two or three SREs and typically lasted two or three quarters. The lead times for a PRR were relatively high (quarters away). The effort level required was proportional to the number of services under review, and was constrained by the insufficient number of SREs available to conduct PRRs. These conditions led to serialization of service takeovers and strict service prioritization.

- Due to differing software practices across services, each production feature was implemented differently. To meet PRR-driven standards, features usually had to be reimplemented specifically for each service or, at best, once for each small subset of services sharing code. These reimplementations were a waste of engineering effort. One canonical example is the implementation of functionally similar logging frameworks repeatedly in the same language because different services didn't implement the same coding structure.

- A review of common service issues and outages revealed certain patterns, but there was no way to easily replicate fixes and improvements across services. Typical examples included service overload situations and data hot-spotting.

- SRE software engineering contributions were often local to the service. Thus, building generic solutions to be reused was difficult. As a consequence, there was no easy way to implement new lessons individual SRE teams learned and best practices across services that had already been onboarded.

External Factors Affecting SRE

External factors have traditionally pressured the SRE organization and its resources in several ways.

Google is increasingly following the industry trend of moving toward microservices.[151] As a result, both the number of requests for SRE support and the cardinality of services to support have increased. Because each service has a base fixed operational cost, even simple services demand more staffing. Microservices also imply an expectation of lower lead time for deployment, which was not possible with the previous PRR model (which had a lead time of months).

Hiring experienced, qualified SREs is difficult and costly. Despite enormous effort from the recruiting organization, there are never enough SREs to support all the services that need their expertise. Once SREs are hired, their training is also a lengthier process than is typical for development engineers.

Finally, the SRE organization is responsible for serving the needs of the large and growing number of development teams that do not already enjoy direct SRE support. This mandate calls for extending the SRE support model far beyond the original concept and engagement model.

Toward a Structural Solution: Frameworks

To effectively respond to these conditions, it became necessary to develop a model that allowed for the following principles:

Codified best practices

> The ability to commit what works well in production to code, so services can simply use this code and become "production ready" by design.

Reusable solutions

> Common and easily shareable implementations of techniques used to mitigate scalability and reliability issues.

A common production platform with a common control surface

> Uniform sets of interfaces to production facilities, uniform sets of operational controls, and uniform monitoring, logging, and configuration for all services.

Easier automation and smarter systems

> A common control surface that enables automation and smart systems at a level not possible before. For example, SREs can readily receive a single view of relevant information for an outage, rather than hand collecting and analyzing mostly raw data from disparate sources (logs, monitoring data, and so on).

Based upon these principles, a set of SRE-supported platform and service frameworks were created, one for each environment we support(Java, C++, Go). Services built using these frameworks share implementations that are designed to work with the SRE-supported platform, and are maintained by both SRE and development teams. The main shift brought about by frameworks was to enable product development teams to design applications using the framework solution that was built and blessed by SRE, as opposed to either retrofitting the application to SRE specifications after the fact, or retrofitting more SREs to support a service that was markedly different than other Google services.

An application typically comprises some business logic, which in turn depends on various infrastructure components. SRE production concerns are largely focused on the infrastructure-related parts of a service. The service frameworks implement infrastructure code in a standardized fashion and address various production concerns. Each concern is encapsulated in one or more framework modules, each of which provides a cohesive solution for a problem domain or infrastructure dependency. Framework modules address the various SRE concerns enumerated earlier, such as:

- Instrumentation and metrics
- Request logging
- Control systems involving traffic and load management

SRE builds framework modules to implement canonical solutions for the concerned production area. As a result, development teams can focus on the business logic, because the framework already takes care of correct infrastructure use.

A framework essentially is a prescriptive implementation for using a set of software components and a canonical way of combining these components. The framework can also expose features that control various components in a cohesive manner. For example, a framework might provide the following:

- Business logic organized as well-defined semantic components that can be referenced using standard terms
- Standard dimensions for monitoring instrumentation
- A standard format for request debugging logs

- A standard configuration format for managing load shedding
- Capacity of a single server and determination of "overload" that can both use a semantically consistent measure for feedback to various control systems

Frameworks provide multiple upfront gains in consistency and efficiency. They free developers from having to glue together and configure individual components in an ad hoc service-specific manner, in ever-so-slightly incompatible ways, that then have to be manually reviewed by SREs. They drive a single reusable solution for production concerns across services, which means that framework users end up with the same common implementation and minimal configuration differences.

Google supports several major languages for application development, and frameworks are implemented across all of these languages. While different implementations of the framework (say in C++ versus Java) can't share code, the goal is to expose the same API, behavior, configuration, and controls for identical functionality. Therefore, development teams can choose the language platform that fits their needs and experience, while SREs can still expect the same familiar behavior in production and standard tools to manage the service.

New Service and Management Benefits

The structural approach, founded on service frameworks and a common production platform and control surface, provided a host of new benefits.

Significantly lower operational overhead

A production platform built on top of frameworks with stronger conventions significantly reduced operational overhead, for the following reasons:

- It supports strong conformance tests for coding structure, dependencies, tests, coding style guides, and so on. This functionality also improves user data privacy, testing, and security conformance.
- It features built-in service deployment, monitoring, and automation for all

services.

- It facilitates easier management of large numbers of services, especially micro-services, which are growing in number.
- It enables much faster deployment: an idea can graduate to fully deployed SRE-level production quality in a matter of days!

Universal support by design

The constant growth in the number of services at Google means that most of these services can neither warrant SRE engagement nor be maintained by SREs. Regardless, services that don't receive full SRE support can be built to use production features that are developed and maintained by SREs. This practice effectively breaks the SRE staffing barrier. Enabling SRE-supported production standards and tools for all teams improves the overall service quality across Google. Furthermore, all services that are implemented with frameworks automatically benefit from improvements made over time to frameworks modules.

Faster, lower overhead engagements

The frameworks approach results in faster PRR execution because we can rely upon:

- Built-in service features as part of the framework implementation
- Faster service onboarding (usually accomplished by a single SRE during one quarter)
- Less cognitive burden for the SRE teams managing services built using frameworks

These properties allow SRE teams to lower the assessment and qualification effort for service onboarding, while maintaining a high bar on service production quality.

A new engagement model based on shared responsibility

The original SRE engagement model presented only two options: either full SRE support, or approximately no SRE engagement.[152]

A production platform with a common service structure, conventions, and software infrastructure made it possible for an SRE team to provide support for the "platform" infrastructure, while the development teams provide on-call support for functional issues with the service—that is, for bugs in the application code. Under this model, SREs assume responsibility for the development and maintenance of large parts of service software infrastructure, particularly control systems such as load shedding, overload, automation, traffic management, logging, and monitoring.

This model represents a significant departure from the way service management was originally conceived in two major ways: it entails a new relationship model for the interaction between SRE and development teams, and a new staffing model for SRE-supported service management.[153]

Conclusion

Service reliability can be improved through SRE engagement, in a process that includes systematic review and improvement of its production aspects. Google SRE’s initial such systematic approach, the Simple Production Readiness Review, made strides in standardizing the SRE engagement model, but was only applicable to services that had already entered the Launch phase.

Over time, SRE extended and improved this model. The Early Engagement Model involved SRE earlier in the development lifecycle in order to "design for reliability." As demand for SRE expertise continued to grow, the need for a more scalable engagement model became increasingly apparent. Frameworks for production services were developed to meet this demand: code patterns based on production best practices were standardized and encapsulated in frameworks, so that use of frameworks became a recommended, consistent, and relatively simple way of building production-ready services.

All three of the engagement models described are still practiced within Google. However, the adoption of frameworks is becoming a prominent influence on building production-ready services at Google as well as profoundly expanding the SRE contribution, lowering service management overhead, and improving baseline service quality across the organization.

[151] See the Wikipedia page on microservices at *https://en.wikipedia.org/wiki/Microservices*.

[152] Occasionally, there were consulting engagements by SRE teams with some non-onboarded services, but consultations were a best-effort approach and limited in number and scope.

[153] The new model of service management changes the SRE staffing model in two ways: (1) because a lot of service technology is common, it reduces the number of required SREs per service; (2) it enables the creation of production platforms with separation of concerns between production platform support (done by SREs) and service-specific business-logic support, which remains with the development team. These platforms teams are staffed based upon the need to maintain the platform rather than upon service count, and can be shared across products.

← PREVIOUS
Chapter 31 - Communication and Collaboration in SRE

NEXT
Part V - Conclusions