Conclusion

Written by Benjamin Lutch[162]
Edited by Betsy Beyer

I read through this book with enormous pride. From the time I began working at Excite in the early ’90s, where my group was a sort of neanderthal SRE group dubbed "Software Operations," I’ve spent my career fumbling through the process of building systems. In light of my experiences over the years in the tech industry, it’s amazing to see how the idea of SRE took root at Google and evolved so quickly. SRE has grown from a few hundred engineers when I joined Google in 2006 to over 1,000 people today, spread over a dozen sites and running what I think is the most interesting computing infrastructure on the planet.

So what has enabled the SRE organization at Google to evolve over the past decade to maintain this massive infrastructure in an intelligent, efficient, and scalable way? I think that the key to the overwhelming success of SRE is the nature of the principles by which it operates.

SRE teams are constructed so that our engineers divide their time between two equally important types of work. SREs staff on-call shifts, which entail putting our hands around the systems, observing where and how these systems break, and understanding challenges such as how to best scale them. But we also have time to then reflect and decide what to build in order to make those systems easier to manage. In essence, we have the pleasure of playing both the roles of the pilot *and* the engineer/designer. Our experiences running massive computing infrastructure are codified in actual code and then packaged as a discrete product.

These solutions are then easily usable by other SRE teams and ultimately by anyone at Google (or even outside of Google…think Google Cloud!) who wants to use or improve upon the experience we've accumulated and the systems we've built.

When you approach building a team or a system, ideally its foundation should be a set of rules or axioms that are general enough to be immediately useful, but that will remain relevant in the future. Much of what Ben Treynor Sloss outlined in this book's introduction represents just that: a flexible, mostly future-proof set of responsibilities that remain spot-on 10 years after they were conceived, despite the changes and growth Google's infrastructure and the SRE team have undergone.

As SRE has grown, we've noticed a couple different dynamics at play. The first is the consistent nature of SRE's primary responsibilities and concerns over time: our systems might be 1,000 times larger or faster, but ultimately, they still need to remain reliable, flexible, easy to manage in an emergency, well monitored, and capacity planned. At the same time, the typical activities undertaken by SRE evolve by necessity as Google's services and SRE's competencies mature. For example, what was once a goal to "build a dashboard for 20 machines" might now instead be "automate discovery, dashboard building, and alerting over a fleet of tens of thousands of machines."

For those who haven't been in the trenches of SRE for the past decade, an analogy between how SRE thinks about complex systems and how the aircraft industry has approached plane flight is useful in conceptualizing how SRE has evolved and matured over time. While the stakes of failure between the two industries are very different, certain core similarities hold true.

Imagine that you wanted to fly between two cities a hundred years ago. Your airplane probably had a single engine (two, if you were lucky), a few bags of cargo, and a pilot. The pilot also filled the role of mechanic, and possibly additionally acted as cargo loader and unloader. The cockpit had room for the pilot, and if you were lucky, a co-pilot/navigator. Your little plane would bounce off a runway in good weather, and if everything went well, you'd slowly climb your way through the skies and eventually touch down in another city, maybe a few hundred miles away. Failure of any of the plane's systems was catastrophic, and it wasn't unheard of for a pilot to have to climb out of the cockpit to perform repairs in-flight! The systems that fed into the cockpit were essential, simple, and fragile, and most likely were not redundant.

Fast-forward a hundred years to a huge 747 sitting on the tarmac. Hundreds of passengers are loading up on both floors, while tons of cargo are simultaneously being loaded into the hold below. The plane is chock-full of reliable, redundant systems. It's a model of safety and reliability; in fact, you're actually safer in the air than on the ground in a car. Your plane will take off from a dotted line on a runway on one continent, and land easily on a dotted line on another runway 6,000 miles away, right on schedule—within minutes of its forecasted landing time. But take a look into the cockpit and what do you find? Just two pilots again!

How has every other element of the flight experience—safety, capacity, speed, and reliability—scaled up so beautifully, while there are still only two pilots? The answer to this question is a great parallel to how Google approaches the enormous, fantastically complex systems that SRE runs. The interfaces to the plane's operating systems are well thought out and approachable enough that learning how to pilot them in normal conditions is not an insurmountable task. Yet these interfaces also provide enough flexibility, and the people operating them are sufficiently trained, that responses to emergencies are robust and quick. The cockpit was designed by people who understand complex systems and how to present them to humans in a way that's both consumable and scalable. The systems underlying the cockpit have all the same properties discussed in this book: availability, performance optimization, change management, monitoring and alerting, capacity planning, and emergency response.

Ultimately, SRE's goal is to follow a similar course. An SRE team should be as compact as possible and operate at a high level of abstraction, relying upon lots of backup systems as failsafes and thoughtful APIs to communicate with the systems. At the same time, the SRE team should also have comprehensive knowledge of the systems—how they operate, how they fail, and how to respond to failures—that comes from operating them day-to-day.

---

[162] Vice President, Site Reliability Engineering, for Google, Inc.

← PREVIOUS
Chapter 33 - Lessons Learned from Other Industries

NEXT
Appendix A - Availability Table