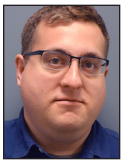
Hiring Site Reliability Engineers

CHRIS JONES, TODD UNDERWOOD, AND SHYLAJA NUKALA



A computational daemonologist, Chris Jones works in San Francisco as a Site Reliability Engineer for Google App Engine, a platform serving over 28 billion requests per day. He was previously responsible for the care and feeding of advertising statistics, data warehousing, and customer support systems, joining Google in 2007. In other lives, Chris has worked in academic IT, analyzed data for political campaigns, and engaged in some light BSD kernel hacking, picking up degrees in computer engineering, economics, and technology policy along the way. cdjones@google.com

Todd Underwood is Site Reliability Director at Google. Prior to that, he was in charge of operations, security, and peering for Renesys, a provider of Internet intelligence services; and before that he was CTO of Oso Grande, a New Mexico ISP. He has a background in systems engineering and networking. Todd has presented work related to Internet routing dynamics and relationships at NANOG, RIPE, and various peering forums tmu@google.com

Shylaja Nukala is a Technical Writer at Google, and she has been leading the Site Reliability Engineering (SRE) technical writing team for six years. She is involved in documentation, communication, and training for SRE. Prior to Google, she worked at Epiphany and Sony. She has a PhD from the School of Communication and Information, Rutgers University. She also taught at Rutgers, Santa Clara, and San Jose State universities. snukala@google.com

Operating distributed systems at scale requires an unusual set of skills—problem solving, programming, system design, networking, and OS internals—which are difficult to find in one person. At Google, we've found some ways to hire Site Reliability Engineers, blending both software and systems skills to help keep a high standard for new SREs across our many teams and sites, including standardizing the format of our interviews and the unusual practice of making hiring decisions by committee. Adopting similar practices can help your SRE or DevOps team grow by consistently hiring excellent coworkers.

Google's Site Reliability Engineering (SRE) organization is a mix of software engineers (known as SWEs) and systems engineers (known as SEs) with a flair for building and operating reliable complex software systems at an incredible scale. SREs have a wide range of backgrounds—from a traditional CS degree or self-taught sysadmin to academic biochemists; we've found that a candidate's educational background and work experience are less predictive than their performance in interviews with future colleagues. Google's hiring process intentionally prevents teams' managers from making hiring decisions, instead using a hiring committee of engineers from across the organization to assess the merits of each potential hire on a case-by-case basis.

Who We Look For

Ben Treynor, Google Vice President and Site Reliability Tsar, describes SRE as being "what you get when you treat operations as if it's a software problem": a software engineering philosophy ("write software to solve problems") hybridized with an operations mission ("keep the service running"). These two influences can be seen in the dual job titles within SRE—SRE-Systems Engineer and SRE-Software Engineer—reflecting the different emphasis with which individual SREs may approach the same problems: one may be most comfortable writing new software, while the other may tend to prefer fitting existing components together into new and exciting architectures, but everyone can do some of both.

By "systems engineering," we mean a discipline that takes a holistic approach to the *connections* between distinct software systems or services rather than either (1) the internals of how to build a piece of software, where software engineering has tended to concentrate as a field, or (2) how software artifacts are deployed onto specific hardware, which has been the historic domain of system administration. Instead, systems engineers view the collection of individual pieces, which may be built by many separate product development teams, as a whole with properties distinct from its components. SEs tend to focus on how to monitor services, identify and remove bottlenecks, manage and balance connections, handle data replication, ensure data resiliency, and so on. This skill becomes essential at the scale at which Google and other large software organizations operate.

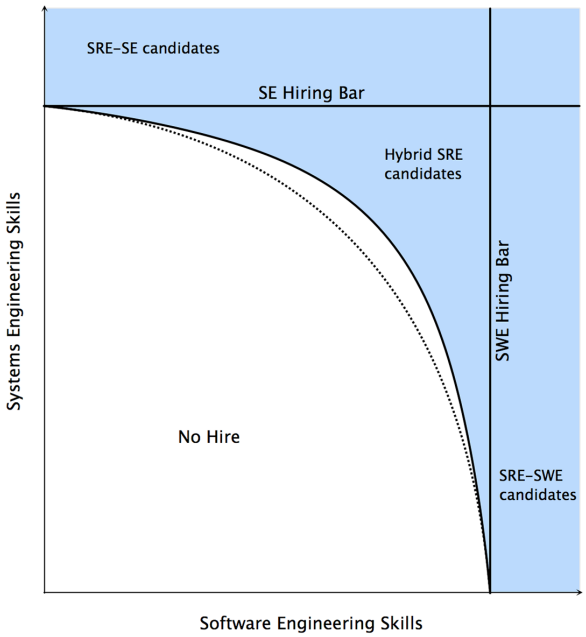


Figure 1: Skills and hiring for SRE candidates

Google's preference for generalists and internal mobility meshes well with our hiring bar for SWE candidates within SRE being the same as that for our product development organizations—engineers are able to move freely from the reliability organization to other groups. We've found, however, that there's a third category of candidates who form a particularly valuable pool: those who trade off some *depth* of experience in one field for a *breadth* of experience in both fields. In Figure 1, the area in the shaded region between the curve with acceptable tradeoffs (which we call the "Treynor Curve"), the SE hiring bar, and the SWE hiring bar shows this pool of "hybrid" SRE candidates.

What We Look For

We look for candidates who are smart, passionate about building and running some of the largest and most complex software artifacts on the planet, and able to quickly understand how something works that they may never have seen before. Since we would like to scale the size of our systems *much* faster than Google can hire SREs to work on them, the SRE approach to problem resolution emphasizes automation, improving system design, and building resilience into our systems so that we don't have to repeatedly fix the same problems; it's also much more interesting to find new failure modes, usually due to newly launched systems or features. Accordingly, we try to find and hire candidates wherever they are, regardless of background: there are simply too few people with the right mindset and skills for SRE to limit ourselves to candidates with conventional backgrounds.

Every SRE, regardless of whether they're an SE or SWE, needs to have an understanding of the fundamentals of computing. Unsurprisingly, we look for the ability to solve problems with software, whether that's been acquired from a textbook or at the school of hard knocks. Similarly, troubleshooting skills and the ability to unpack a problem into smaller pieces, identify possible causes, triage, and do so systematically are essential, whether that's been acquired through debugging code, operating a network, building hardware, or in other, entirely unrelated domains; the cognitive skills and approaches to problem-solving are subject-matter agnostic and critical to have, regardless of a candidate's background.

We specifically do not look for "architects"—that's a role that simply doesn't exist at Google: everyone in our engineering organizations both designs and implements. Similarly, prospective candidates for managerial roles in Site Reliability must meet the same technical bar as individual contributors, as well as understand that management at Google is a different proposition from that at many other companies: SRE line managers are typically also technical contributors to their team, including being part of an on-call rotation, in addition to their managerial responsibilities in coordinating skilled and highly autonomous individual contributors.

How We Interview

SRE interviews follow Google's typical engineering interview pattern: much like elsewhere in the industry, there is first a short technical pre-screen with a recruiter; next, an initial phone interview with an engineer, perhaps with a follow-up phone interview; and then a day at one of our sites, doing four or five interviews, each with an engineer. Each interview is intended to be a conversation between peers rather than an interrogation: we strenuously discourage brainteasers and trivia questions, as they provide minimal insight into how a candidate thinks about problems.

Each SRE interviewer has a specified topic to cover—e.g., programming, UNIX internals, networks, or troubleshooting and problem-solving—to ensure that we have a wide range of assessments from interviewers, while minimizing duplication [1]. The mix of topics varies based on the candidate's self-assessed strengths and weaknesses: there's no point in spending valuable interview time asking someone about their weaknesses, only to discover that they were right when they said they didn't know much about a topic. Similarly, we try to match candidates' strengths with those of their interviewers, so that they have a more interesting conversation and there's a better quality signal in the resulting assessment. Ideally, each interviewer will discover the limits of the candidate's knowledge in their topic and see how the candidate reasons and reacts when faced with problems they have not previously encountered—that is, can they make reasonable assumptions and extrapolations from what they *do* know?

At least one interview will involve programming in the candidate's preferred language: while Google uses C++, Go, Java, Javascript, and Python for most of its projects, we have SREs who can read pretty much any language a candidate might want to use. Candidates do not need to use one of the five languages in their interview, as we expect that anyone who meets our hiring bar is likely able to learn at least one of those languages fairly quickly.

One of the interviews will be on "non-abstract large system design" [2], in which they're asked to concretely design a large-scale system, such as a system to join different types of log entries written in multiple datacenters for analysis. Simply laying out boxes on a whiteboard and invoking magic technologies ("I'll store everything in BigTable, since that's what Google uses") to solve a problem isn't sufficient: silver bullets are rarely found when building real software systems, so it would leave too much mystery about a candidate's quality to accept answers depending on them. Instead, we're looking for candidates to be able to provide realistic estimates of throughput, storage, and so on for each component—while considering various tradeoffs for reliability, cost, and difficulty of building the system. The ideal candidate can not only reason about how each high-level component fits together, but work through each layer in the design, right down to the hardware underpinning it.

Afterwards, the interviewer provides a hire/no-hire recommendation along with detailed written feedback explaining how a candidate answered the questions and the strengths or weaknesses of those responses compared to others'.

If the interview feedback for a candidate is borderline, the recruiter can ask a group of engineers to perform some quality control and validation: Are additional interviews likely to be needed? Is the interview panel sufficiently senior for the candidate's experience? Does another topic need to be covered or an interview topic repeated for some reason?

How We Decide

An unusual feature of Google's engineering hiring process is that the hire/no-hire decision is *not* made by a manager; instead, it's made by a hiring committee before going to senior management for approval (if the decision was to hire). The committee members are drawn from across the organization, including multiple locations and teams within SRE.

A mix of SRE managers and individual contributors will read the interview feedback each interviewer wrote and come to a joint decision about whether the hiring bar for the role has been met. Hiring committee meetings are characterized by extensive debate on whether each candidate meets our organization-wide hiring bar. The hiring committee has access to past scores and hiring decisions for each interviewer, so it can decide how much weight to put on an interviewer's feedback given their past predictive track record.

Using a committee to make hiring decisions is a critically important part of our process because it ensures that we have an assessment that reflects the skills and capabilities we expect our engineers to have, while maintaining common standards between offices and parts of the organization to ensure internal mobility. The committee's diverse perspectives can also provide a broader assessment of candidate strengths and weaknesses.

Removing the (prospective) hiring manager from the process prevents the common management pathology of taking the first warm body who seems vaguely competent to fill a vacancy or a short-term need, compromising hiring standards at the expense of the long-term health of the organization. In fact, allocation of a new hire to a specific team in SRE always happens separately, *after* the hire/no-hire decision is made. As a result, we can expect our hiring quality across SRE to stay consistent over time—or at least, be changed *intentionally* by management in response to headcount availability—rather than simply choosing a candidate who happened to apply for a particular opening at the discretion of that team's manager. As it happens, Google's practice is to hire candidates we believe to be better than our average current employee [3], consciously accepting a higher risk of false negatives (incorrect no-hire decisions) to reduce the chance of false positives (incorrect decisions to hire).

Conclusion

Talented future SREs are scarce and hard to find; it's often difficult to make a confident prediction about whether a given candidate will succeed as an SRE. We've found that standardizing our hiring process so that we consistently cover a range of skills essential for success in SRE and ensuring that all SRE candidates are able to code regardless of their background in system administration, systems engineering, or software engineering are critical to guaranteeing a high level of mobility between SRE teams and within organizations at Google.

Finding people who are *simultaneously* generalists comfortable with encountering novel software systems and specialists with sufficient technical depth in particular fields (e.g., software engineering, networks, distributed systems) is even more difficult: by building an organization that takes each SRE's individual strengths—regardless of his or her place on the Treynor Curve between systems engineering and software engineering—and combines them, we're able to have an *organization* which can paradoxically bridge the two skills.

On Interviewers

Ideally, we would like every interview to be performed by long-tenured, senior SREs who have done thousands of interviews and have a perfect track record of predicting hiring decisions; unfortunately, the volume of interviewing and other demands on Senior Engineers' time make this an impossibility—and this would also make it impossible for anyone else to become an experienced interviewer. Instead, we try to populate an interview panel with a majority of reasonably experienced interviewers whose feedback has good predictive value, while still providing an opportunity for newer interviewers to get practice in one of the interview slots. Very new interviewers may "shadow" experienced engineers' interviews or "reverse shadow," in which one conducts the interview while the other observes: both submit feedback, but only the experienced interviewer's feedback is used.

As engineers gain experience interviewing, they become better able to determine candidate strength through more exposure to interview candidates and common interview responses, both good and bad; increased time working with their peers to understand the skills expected of new hires; and the opportunity to write assessments and receive feedback from colleagues on those interviews. After some time, we are able to evaluate their hiring recommendations and feedback for interview quality, consistency, and predictive value.

Because phone interviews are a single point of failure—a candidate's rejection at this stage generally precludes further consideration for that role for some time—we choose phone interviewers from a relatively small pool of particularly consistent interviewers trusted by the hiring committee, to try to make sure that we make good decisions about who to invite for on-site interviews. This is intended to ensure that candidates who make it to that stage have a realistic prospect of making it through the interviews and being hired, reducing the cost of interviewing: each on-site candidate costs at least four hours for the interviews themselves, plus time spent on writing feedback and reviewing it in the hiring committee.

We have an organized pool of interview questions with canonical answers we've seen from past candidates for interviewers to draw upon. This makes it easier for newer interviewers to get started and provides a consistent subset of questions for the hiring committee to use in comparing candidates, although interviewers are free to add their own technical questions. Over the course of an interview, an interviewer refines and increases the technical depth of the conversation to determine the candidate's depth of understanding, so that the pool is used more as a starting point for further discussion and elaboration rather than being a list of trivia questions to be checked off in sequence. Each interview is thus unique, though it follows a common pattern.

Several locations with SRE teams have a regular "Interview Club" group, where SREs can try out potential interview questions to see how they work in practice and to get feedback from experienced interviewers. SREs are also encouraged to occasionally observe hiring committee meetings. They may also receive comments from the hiring committee on their interview notes to help make their future feedback more useful or might mention that a particular approach to an interview question worked well.

Practices We've Found Helpful for Hiring SREs

Structure interviews to cover the topics essential to the SRE role, as appropriate for the candidate's skills and strengths; assign a specific topic to each interviewer.

- Build a pool of interview questions along with "gold standard" responses, to provide a consistent subset of questions across candidates.
- Ask about how to build *concrete* large-scale systems; avoid brainteasers and trivia.
- Ask *every* SRE candidate to code *something*.
- Separate interviewing from hiring decisions.
- Make hire/no-hire decisions by a committee of engineers.

On Process

TODD UNDERWOOD

It's reasonable to ask why Google uses such an elaborate process to hire people. Some other companies manage to hire people somewhat or much faster than Google. Perhaps there should be a model of quick hire and, if things aren't working out, quick fire. There are a number of reasons why this cannot work well at Google and probably doesn't work well at most other places, either.

As we pointed out, we hire generalists who will likely be part of several teams over their careers at Google. It's critically important that our hiring standard not be lowered by an individual hiring manager's short-term need for staffing. The easiest way to avoid this temptation while maintaining uniform and high standards is to make the hiring decision through a committee that excludes the hiring manager.

Additionally: the learning curve at Google is quite high. Our software stack is sophisticated, fragile, complex, and powerful, and it takes quite a while to learn it. It therefore takes months before it is apparent whether a new hire is doing well. By the time a bad fit is obvious, we may have made an investment of many months. This necessarily encourages us to be much more conservative than some other employers about hiring decisions.

Finally, there's the very serious issue of bias. Hiring decisions made quickly by individuals often result in hiring people who are just like those doing the hiring. The technology industry has a bias problem, and we are committed to doing what we can to fix it. Some of the things we have learned about avoiding bias in decisions, especially where that bias is unconscious, is that making decisions as a group according to articulated standards helps. It also helps to justify the decision and know that you'll have to justify the decision in advance. By incorporating these aspects into our process, we hope to make decisions that add diversity.

Hiring decisions made quickly by a hiring manager according to no articulated standards might work well at some organizations, but we have come to believe that consistently hiring well is critically important to us, and employment is critically important to most of our employees. Taking an appropriate amount of time to make sure there is a reasonable fit makes good sense for us.

Acknowledgments

This article was based on a conversation between Gary Arneson, Zoltan Egyed, Chris Jones, and Todd Underwood. Thanks to Dermot Duffy, David Hixson, Alex Matey, Niall Murphy, and Lucas Pereira for their invaluable comments.

Resources

[1] See Daniel Kahneman, "Thinking, Fast and Slow" (2011) for further discussion of using a structured hiring format instead of making intuitive judgments; also Lazlo Bock's *Work Rules*: https://www/workrules.net.

[2] Google SREs often teach classes on non-abstract large system design at LISA and other venues, featuring exercises where small groups solve design problems much like those encountered by interview candidates.

[3] Peter Norvig: googleresearch.blogspot.com/2006/03/hiring-lake-wobegon-strategy.html.