The Systems Engineering Side of Site Reliability Engineering

DAVID HIXSON AND BETSY BEYER

David Hixson is a Technical Project Manager in the Site Reliability organization at Google, where he has been for eight years. He currently spends his time predicting how social products at Google will grow and trying to make the reality better than the plan. He previously worked as a system administrator on High Availability systems and has an MBA from Arizona State.
usenix@dhixson.com

Betsy Beyer is a Technical Writer specializing in virtualization software for Google SRE in NYC. She has previously provided documentation for Google Data Center and Hardware Operations teams. Before moving to New York, Betsy was a lecturer in technical writing at Stanford University. She holds degrees from Stanford and Tulane.
bbeyer@google.com

In order to run the company's numerous services as efficiently and reliably as possible, Google's Site Reliability Engineering (SRE) organization leverages the expertise of two main disciplines: Software Engineering and Systems Engineering. The roles of Software Engineer (SWE) and Systems Engineer (SE) lie at the two poles of the SRE continuum of skills and interests. While Site Reliability Engineers tend to be assigned to one of these two buckets, there is much overlap between the two job roles, and the knowledge exchange between the two job roles is rather fluid.

The collaborative SWE/SE engineering approach was popularized in the Silicon Valley environment, but is now common to locations characterized by a high density of software engineering requiring an operational component. A hybridization of the two skill sets is also demonstrated by lone individuals who hold together complicated software systems by sheer force of will. While Software Engineering is generally well understood in the tech world, Systems Engineering remains a bit more nebulous. What exactly is Systems Engineering at one of these companies? This article takes a closer look at Systems Engineers: what they are, what they do, and how you might become one.

Characteristics of a Systems Engineer

The task of defining the exact characteristics of a Systems Engineer at Google, or at any other Silicon Valley tech company that uses classifications like "Developer Operations," is problematic. Informally, a Systems Engineer might be described as someone who enjoys discovering particularly difficult problems and applying their problem-solving skills in uncharted territory. A Systems Engineer regularly undertakes tasks like dismantling software or hardware, re-engineering and optimizing their design, or finding new uses for the components. Traditional wisdom dictates that "you know a Systems Engineer when you see one." However, this definition fails to either permit a Systems Engineer to self-identify or to become a better Systems Engineer. Nor does it help tech companies to create a satisfying job ladder and career progression for Systems Engineers.

Therefore, pinning down a set of characteristics particular to a Systems Engineer is a necessary and useful exercise.

A Systems Engineer

...uses the scientific principles of experimentation and observation to build a body of knowledge that affects the architecture and design of the system as a whole.

Complex and ambiguous problems can be solved by:

1. Breaking down the problem into smaller components.
2. Testing assumptions about these components.
3. Continuing in this vein of investigation until a root cause is identified.

In addition to this troubleshooting skill set, a Systems Engineer must have the willingness to chase a problem through the multiple layers beyond its surface, acquiring the knowledge necessary to conduct the investigation along the way. Willingness to learn new technologies or techniques is critical to pursuing each new investigation.

...has an actionable skepticism towards layers of abstraction.

At least a few layers of abstraction are necessary in order to deal with the complexity in the world around us. However, when a Systems Engineer's expectations of how a certain system should perform are violated, the engineer must figure out why. The investigative effort can focus on determining the causes of existing problems, avoiding future problems, or finding improvements where no one has looked before.

...focuses on the connections between the components within the system as much as focusing on the components themselves.

Understanding the interactions between each system element is critical to building or troubleshooting systems that scale. Decisions about how communications are passed between the elements can have extreme effects on the overall stability of the system.

...knows many ways to *not* solve a problem, rather than one perfect way to solve the problem.

A perfect solution to a problem is extremely rare. Instead, choosing the best engineering solution requires tradeoffs between many different elements. Deliberately evaluating and making these tradeoffs is key to building a stable and scalable system or to identifying problems in an existing system. To state this principle another way: success is probably a corner case of the possible failure modes of a complex system.

Differences Between Software Engineering, System Administration, and Systems Engineering

The fundamental differences among three core specializations in creating and operating software at tech companies—Software Engineering, System Administration, and Systems Engineering—fall into three main categories: approaches to problems, academic background and professional communities, and career progression.

Approaches to Problems

The scenario presented in the simple drawing to the right (Figure 1) would be approached in very distinct manners by an archetypical SWE, SA, and SE. In practice, a successful Site Reliability Engineer is expected to use a combination of these approaches.

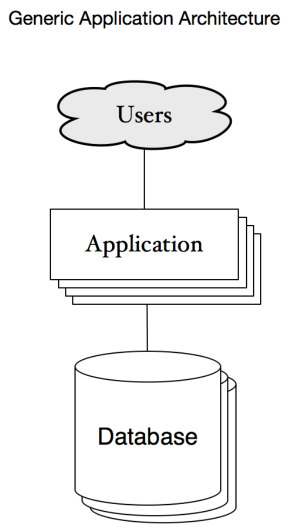

Figure 1: Generic application architecture

An SWE would focus on constructing boxes that have predictable behavior and that operate as efficiently as possible. Software needs to:

- Turn requests into responses.
- Call the database via the appropriate API.
- Optimize the schema to reflect the kinds of queries that will be made.

A combination of language choices, frameworks, unit tests, load testing, and a variety of best practices make these operations more likely to work effectively.

Sysadmins generally approach their operational responsibilities with operational solutions. This diagram would prompt an SA to think about the infrastructure required to actualize the service represented and manage the operational aspects, considering questions such as:

- What hardware is needed?
- What software is implied but not specified?
- How do we get all of these operations to run at the same time in a supportable way?

A wide variety of tasks not depicted in the drawing come into scope: OS deployment, backups, security, user administration, configuration management, logs, monitoring, performance tuning, capacity planning, and so forth.

Systems Engineers generally approach operational requirements with a software or system approach. This diagram would prompt an SE to focus on the lines connecting the sets of boxes and the overall experience represented in the diagram more than on the boxes themselves. For example, an SE might ask the following questions:

- If multiple front ends are employed, how do we shard incoming traffic?
- How do we replicate databases, and how do we manage connections, failover, hotspots, etc.?
- Do we replicate databases globally, in the same datacenter, or on the same machines?
- What changes are we sensitive to in terms of impacting availability or performance?

An SE's scope encompasses everything from language choice to networking to hardware platforms. However, the SE considers the code in less depth than would an SWE, and the machines in less depth than would an SA.

Academic Background and Professional Communities

Software Engineering is taught through a variety of academic paths, ranging from elementary school courses to doctoral programs. The academic community has produced a substantial body of respected work, and research into various facets of computing continues to advance the state of the art. Ongoing advancements in computer science, performed by a thriving community spanning multiple industries, continually facilitate the authoring of better code both through technology and best practices.

System Administrators usually receive some level of professional training, which normally doesn't occur through traditional academic institutions, or at least not at the level of a degree program. Much of an SA's training and certification is on the job and focuses on specific hardware or software configurations from specific vendors. Therefore, an SA's training risks becoming highly specialized and may be outdated fairly quickly. A large body of documentation focuses on the practice of the SA job (e.g., best practices for accomplishing specific tasks), but there is little writing that explains the philosophy behind the job. SA support communities exist, but are heavily fragmented because SA work is very specialized and concerns quite specific focus areas, such as backups, user management, storage, and so on.

Systems Engineering is more multi-disciplinary than Software Engineering or System Administration, meaning that it benefits heavily from academic study, but doesn't necessarily align with a typical degree program that focuses on depth rather than breadth. SEs benefit from increasing their skills in computer science or other areas of engineering, but these fields don't represent the whole of an SE's work. Academic disciplines dubbed "Systems Engineering" do exist, but typically don't focus on the kinds of work that information technology companies expect. There are communities which overlap with the situations faced by SEs, but the discussions of such communities tend to be far-ranging and cover the union of several different complex areas. As a result, the audience that can usefully sympathize or contribute to solutions tends to be rather limited.

Career Progression

SWE positions exist at every level in a job ladder, from those able to code "hello world" to the engineers in charge of inventing the technologies of the future. Similarly, SWE positions exist across an incredible variety of available technologies and scope—an SWE might focus on writing custom firmware for some exotic device, designing a new programming language, building Skynet, or writing an iOS app that issues reminders to buy groceries.

The SA ladder often starts with a help desk position or work servicing computers, advances to managing or tuning complex services, and further advances to managing networks of computers. Advancement is generally either one of scale—extending up to controlling thousands of computers—or depth—requiring expertise in managing a smaller number of much more complex systems.

The SE suffers from not having a low ladder rung from which to ascend. A productive SE must have the skills and experience that stem from working on real problems. Subsequently, an SE likely begins on either an SWE or SA ladder, at some point recognizing more of a cross-functional calling and branching off into an SE role. A typical SWE to SE trajectory entails either working on loosely coupled systems, or needing to tune a software project for a very specific role that crosses into computer hardware or networking territory. A typical SA to SE trajectory entails either fitting together a wide variety of components that operate outside of SA documentation, or needing to understand and modify software in interesting ways.

At the apex of the Silicon Valley job ladders, the SE and SWE jobs merge back together, since the largest and most complicated software problems cross so many disciplines and technologies that the leaders of such projects need to understand how all of their components fit together. These engineers are dubbed Software Engineers or Principal Engineers, but no element of a given system is outside of their scope—everything from software, to networking, to hardware choices are within their purview. Because pushing the frontiers of technology is rarely conducted in just one dimension, the tradeoffs between these choices need to be conducted with as much flexibility and understanding as possible. SRE at Google works to push this merger of skills early in engineers' careers in order to provide them with opportunities to impact or create globally distributed systems.

The SE Approach to Problems

The manner in which an SE approaches a problem varies from person to person and depends on the service being investigated, but a few SE-specific skills and thought processes are quite common. Problem-solving begins with figuring out how a given system is supposed to work and/or the expected outcome of the system. An SE's investigative approach starts at the unexpected output or outcome and then traverses the system until the problem is resolved. At each step in the investigation, the SE contemplates the expected event or outcome versus what actually occurs. When the expected and the actual don't align, the SE digs deeper.

An SE doesn't just investigate correctness, as problems frequently split off into much less obvious areas such as how to avoid latency tails, the construction of systems that are resilient to failure conditions, or the design of systems that can run under extremely high levels of load. At each stage of investigation, it is critical to not just understand what should happen, but to create tests to verify that a given event occurs in the expected manner and with no side effects worthy of consideration. Any unexpected event is an opportunity to deconstruct the component, be it software or hardware, and repeat the process in order to understand the event at a lower level of abstraction.

The skills espoused by an SE can be vital in bringing an idea to fruition in a way that is scalable, performant, and generally in alignment with the expectations of all players involved. Unfortunately, success in the areas of SE expertise is frequently difficult to detect, particularly by those who aren't deeply involved in the investigative process and resulting decisions. At the end of the day, success is rarely attributed to the integration between system components (an operation performed by the SE), but rather to whomever built its key components (the SWE). Although Google SRE works hard to correct this anti-pattern by raising the level of understanding and appreciation of SE tasks, Systems Engineering is a good career choice for those seeking the satisfaction afforded by intensive problem solving, but is not an attractive role for those seeking recognition outside their immediate team.

The deliverable of an SE is unlikely to be a body of code comparable to that of a software engineer. Instead, the job of an SE entails the thought process and work necessary to either make a given system function as intended, or to build elegant solutions to new problems. The important contribution of an SE is the improvement to the system as a whole, not the list of actions, and quality and effectiveness must be measured by the improvement of an entire system over time, rather than the delivery of a particular, narrow task. The deliverable may be just a few lines of code, or even a setting on some obscure piece of networking hardware that ends up providing value [1]. Subsequently, evaluating an SE's overall performance or contribution to any given project is difficult. While the SE role is potentially higher impact than that of an SWE or SA, it is a difficult role to manage, and career advancement paths aren't always obvious.

Becoming a (Better) SE

To pursue work (or improvement) as an SE, start by building enough depth in one area to provide a basis from which to understand how the pieces of your system interact. You can begin such an investigation from either an SA or an SWE background. Note that if you're pursuing an SE role at the launch of your career, a basis in software engineering may be advantageous, as the available education options are both more comprehensive and more readily available. That being said, building your SE skill set requires refusing to acknowledge that a system problem lies outside of your scope or control. Follow networking problems over the network, chase performance problems out of your code and into the hardware, or dissect an application to figure out how it works and then improve the application. Whenever a system violates your expectations of how it should perform, figure out why. Working with or contributing to open source projects is one great way to improve your SE-related skills. Systems Engineers tend to love open source, as it enables them to pry open a black box and shine a light upon its inner workings to figure out exactly why components behave the way they do. Growing your software engineering skills will help with career advancement, as these skills enable a deeper engagement with the software components that comprise the systems with which SEs work.

Fundamentally, skill as a Systems Engineer comes from satisfying your curiosity over and over again, and accumulating that experience to continually improve your investigative skills. Eventually, this accumulated experience can inform how you build new systems. From day one, incorporate the system monitoring intended for your final product. Design systems that expose their side effects in a way that makes those effects easy to understand later. Document expected outcomes and your assumptions about any given system, in addition to what might be expected if you violate those assumptions.

For those who possess an abundance of curiosity and a willingness to constantly dig into uncertainty and complexity, Systems Engineering can be a thoroughly enjoyable and rewarding career path. The SE role has matured into a viable profession, and the prospects for SEs will likely continue to grow in the future. More and more companies, particularly those outside the traditional tech world, will need Systems Engineers to build and improve the complex systems required to sustain their operations.

Resource

[1] For more context on SE deliverables, see snopes.com: Know Where Man: http://www.snopes.com/business/genius/where.asp.