During LISA ’13, John Looney taught something he called “SRE Classroom: Non-Abstract Large System Design for Sysadmins” [1]. I had little idea what that was, and I was teaching on the same day so I couldn’t drop in. But the description sounded too good to be true.

John had gathered ten fellow Google SREs to mentor people during class exercises, and the result was an experience that really had people talking. The class covered the design of large distributed systems, with exercises focused on building a detailed design. The designs are not specific to Google products but general enough for any large-scale system.

I was intrigued by the success of the tutorial, and wanted to learn more about where this class evolved. For me, John’s explanation of what it’s like to work at scale is fascinating and eye-opening.

Rik: Tell me about how you got involved with working with *nix and clusters.

John: In college, I loved messing around with the commercial UNIX workstations we had and, with a friend, secretly installed Linux on all the university Windows desktops. We repurposed one desktop with a broken monitor as server and covertly offered a Linux network to the students for their projects—we ran MySQL, NFS, etc. on Pentium 60s with VGA cards—it was so much more pleasant to use than SPARC ELCs with monochrome screens.

I did some Solaris support, various training and consulting services for Sun Microsystems after college and realized that though there were many training resources for Solaris sysadmins, there were none for Linux. So I started the Irish Linux User Group and arranged free community-led training programs every month. I made lots of great friends and learned so much. I got entangled in a few hilariously doomed startups, and eventually ran the network and systems of a Web hosting company with explosive growth for a few years. We never had much money, so had great fun building our own routers and servers with whatever hardware we could find.

This gave me the bug for supporting big systems, and I lucked out and joined Google’s “Clusterops” team in 2005. At the time, we were responsible for the Google File System and all low-level cluster functionality.

Rik: My early sysadmin experiences did include Sun workstations, but we were still happy to have any network at all—UUCP over serial cables. So my experience stops well short of understanding what it’s like administering clusters of servers. Could you tell us more about that?

John: In the Web hosting company, we had -15 racks of equipment, most of which I’d installed myself from CD. When one failed, I knew which customer was affected, and usually exactly how to fix it. My first day in Google couldn’t have been more different. “Here is a list of 18,000 broken machines. See how many you can get back serving, and if you think it’s hardware, bounce it onto the hardware guys.” My debugging skills needed to evolve. In the old days, I used to ssh in, poke around, work out what was wrong, write a small script to fix it, then run it on all other similarly broken machines.
A problem could be “Kernel X on platform Y in network Z emits 100 kernel log messages an hour filling the disk.” The workaround might be “Truncate /var/log/messages on all affected machines with a cronjob.” The permanent fix would be “File a bug with kernel team and get that kernel pushed out to affected machines once it passes testing.”

In my Web-hosting days, that was a five-minute fire and forget shell command. In Google, shepherding that process could take a week...but it was worth it, if it was affecting 1000 machines; it could take more time if the buggy kernel was rolled out to all machines. Running clusters makes you think much more deeply about root causes and how to resolve issues permanently.

We have some very smart upper management in Google who realized that taking many weeks to turn up new clusters, by assigning new engineers the task, was a colossal waste of money. They told us to do the impossible—“Go work out how to turn up a cluster in a week. And the deadline is three months, because we have a half-dozen new clusters coming online that week.”

We quickly built a technically beautiful system, based on Python unit test frameworks, to do cluster turn-ups. First, we wrote unit tests to verify that every aspect of a cluster was configured according to best practices. One unit test for “Does the DNS server IP respond to DNS queries?” and another for “Does it have an IP for its own address?” etc. There were thousands of tests that had to be written for everything from “Did anyone make a typo in the cluster definition database?” to “Does the Web search pass a load test?”

When a unit test failed, we had the automation framework run code that “repairs” the service, given that specific test failure: “DNS isn’t listening on port 53...so go install the DNS package on machines reserved for DNS.” “DNS is listening, but doesn’t resolve correctly...execute code to push the latest configuration files to the DNS server.”

It allowed geographically distributed teams to add tests and fixes for their own services, which could depend on each other (WebSearch depends on storage, which depends on Compute, which depends on Chubby, which depends on DNS). By the deadline date, we had a system that could execute months of manual, error-prone work in three hours. This saved a massive amount of money, but also meant we had up-to-date code that documented exactly how to turn up clusters, and could spot and repair configuration problems in running production systems. It was the application of engineering to an operations function, the definition of SRE.

Rik: You certainly make it clear that managing clusters is not like the system administration that I used to do. While there is still problem solving, automation becomes an important part of the solution.

What are some of the things that people interested in managing clusters can learn that will help them?

John: Try to keep abreast of the state of the art. I’m fortunate that in Google, if I have a problem I can usually find ten teams who have had that problem first and have taken a stab at solving it once or twice.

In the outside world, it’s common to think your specific problem is unique, and the existing tools don’t work. You are probably wrong%). Conferences and user groups are good places to find experienced folks who can talk you out of trying to build something new, but have ideas on how to customize and contribute to an existing system that will solve your problem.

Of course, that means that sysadmins need to be able to code in the common systems and automation languages—C, Python, Java, and perhaps one of the less common systems. “I only know AWK and Perl” isn’t good enough anymore. If you are on a very small site, with few coders, it’s really important to network and use the free software community as a resource to allow you to develop software engineering skills sysadmins need—writing code that can be tested, is efficient, and that others can maintain.

I’m a big believer in “whole stack” knowledge for a sysadmin. You should be able to talk about Arduino-level electronics, the latest CPU and RAM designs, and the physics of hard disk manufacture. Books like The Elements of Computing Systems: Building a Modern Computer from First Principles (www.nand2tetris.org) are wonderful for this “low-level” knowledge. You should know your way around the operating system kernel and be able to write simple device drivers (even if it’s just to talk to an Arduino over serial or something).

One of the ways sysadmins can surprise and intimidate software engineers is by being able to take a broken system they’ve never seen, and diagnose and suggest a fix. You get good at this by practicing—downloading random free software packages, trying them out, shouting aloud at dumb config systems, but making it work anyway. Go a step further—suggest or make changes to the software, and get the changes into the upstream code. Ask the author why it was done that way in the past. The modern sysadmin needs to be a software archaeologist when needed.

There are some powerful configuration systems out there these days. It seems most people use them in a very procedural way—“This is the directory for Web server configs, one file per machine, this is the directory for SSH daemons.” Learn how your whole network looks from above and build models (in a text file, SQL database, etc.) that describe everything about your system...then write tools that take that “model” and build the configs for you. Cluster admins shouldn’t edit text files, unless they configure systems that configure systems that write text files ;).
Lastly, I’d recommend that they try to think bigger. We often try to solve the problem in front of us the way we solved it last year. If you are in an environment that tolerates failure (aka innovation). At LISA last year, I had great fun getting 70 sysadmins to try to design a system, like imgur.com, in enough detail to work out how much it would cost to buy the hardware. By pushing our limits, in design, capacity planning, architecture, and presentation skills, we can learn a lot about ourselves.

*Rik:* That was a very popular class. Can you tell us more about that class and how it was run?

*John:* The interview process for SRE is pretty tough, as we grill people on networking, UNIX, software engineering, project management, and other skills we might find useful. The interview that most people struggle with is “non-abstract large-scale design.” It’s probably because it’s not something everyone does. If you do well in that interview, it’s a good sign you will make a good SRE. If you do poorly...we can’t draw conclusions.

Some SREs in London decided to run an experiment—locate SRE job applicants who were potential SREs, but were too inexperienced to succeed at a large-scale design interview. Invite them in for a one-day class on design, explain how such a skill would be applied in the real world, and if they felt up to it, have them interview for an SRE job the next day.

It’s one thing to be able to design, but it’s also important to be able to communicate that design to people. So they designed a class that would introduce the concepts of design, and allow people to practice them, and discuss them in groups. We discussed how to approach an SLA-based design, some common facets of large-scale systems, failure modes of distributed systems, and monitoring such systems. We interleaved the classes with discussions on design in small groups, each of which was led by an experienced Googler.

We’ve now run this class in a few offices, and helped a good number of people to get hired by Google (success!). I was asked to help with a tutorial at LISA and decided that I’d love to try to do a similar event. I panicked a little when I found out we had nearly 70 attendees signed up—in Google we had one “mentor” per five attendees. I made an impassioned plea to the Google SRE team, and ten volunteers traveled to Washington and gave up their Sunday to share their love of crazy big systems.

I think if you assume that your attendees are up for a real challenge, are willing to have their brains turned inside-out with some really good material, and are willing to try something they’ve never done before, you can have a lot of fun.

*Rik:* That’s quite a story. I really had no idea that members of Google SRE teams would give up part of their weekend to mentor students.

USENIX hopes to find other tutorials that will help people learn what it takes to work at scale. Do you have any ideas for other class topics that might be useful? Other people have suggested data analysis and release engineering.

*John:* Release engineering is vital. At scale, you are likely running custom software, so you can’t assume someone else has tested everything for you. It’s also not acceptable to have scheduled downtime, so building and testing software that can be incrementally drained and upgraded without a problem isn’t trivial.

Data analytics could be useful; it could also be useful to learn how to take “production quality” software and instrument it. Add the equivalent of an Apache “status” page that tells all the incoming and outgoing requests, latencies, RAM usage, etc. Something that can be aggregated by your monitoring system later.

I’d also love to build a distributed debugging class, but I fear debugging is so domain specific it wouldn’t be useful—or easy to run. We need to optimize for people’s time and energy. If you’d like to be involved, please contact LISA’s Program Committee for 2014.

Reference
Donate Today: The USENIX Annual Fund

Many USENIX supporters have joined us in recognizing the importance of open access over the years. We are thrilled to see many more folks speaking out about this issue every day. If you also believe that research should remain open and available to all, you can help by making a donation to the USENIX Annual Fund at www.usenix.org/annual-fund.

With a tax-deductible donation to the USENIX Annual Fund, you can show that you value our Open Access Policy and all our programs that champion diversity and innovation.

The USENIX Annual Fund was created to supplement our annual budget so that our commitment to open access and our other good works programs can continue into the next generation. In addition to supporting open access, your donation to the Annual Fund will help support:

• USENIX Grant Program for Students and Underrepresented Groups
• Special Conference Pricing and Reduced Membership Dues for Students
• Women in Advanced Computing (WiAC) Summit and Networking Events
• Updating and Improving Our Digital Library

With your help, USENIX can continue to offer these programs—and expand our offerings—in support of the many communities within advanced computing that we are so happy to serve. Join us!

We extend our gratitude to everyone that has donated thus far, and to our USENIX and LISA SIG members; annual membership dues helped to allay a portion of the costs to establish our Open Access initiative.

www.usenix.org/annual-fund