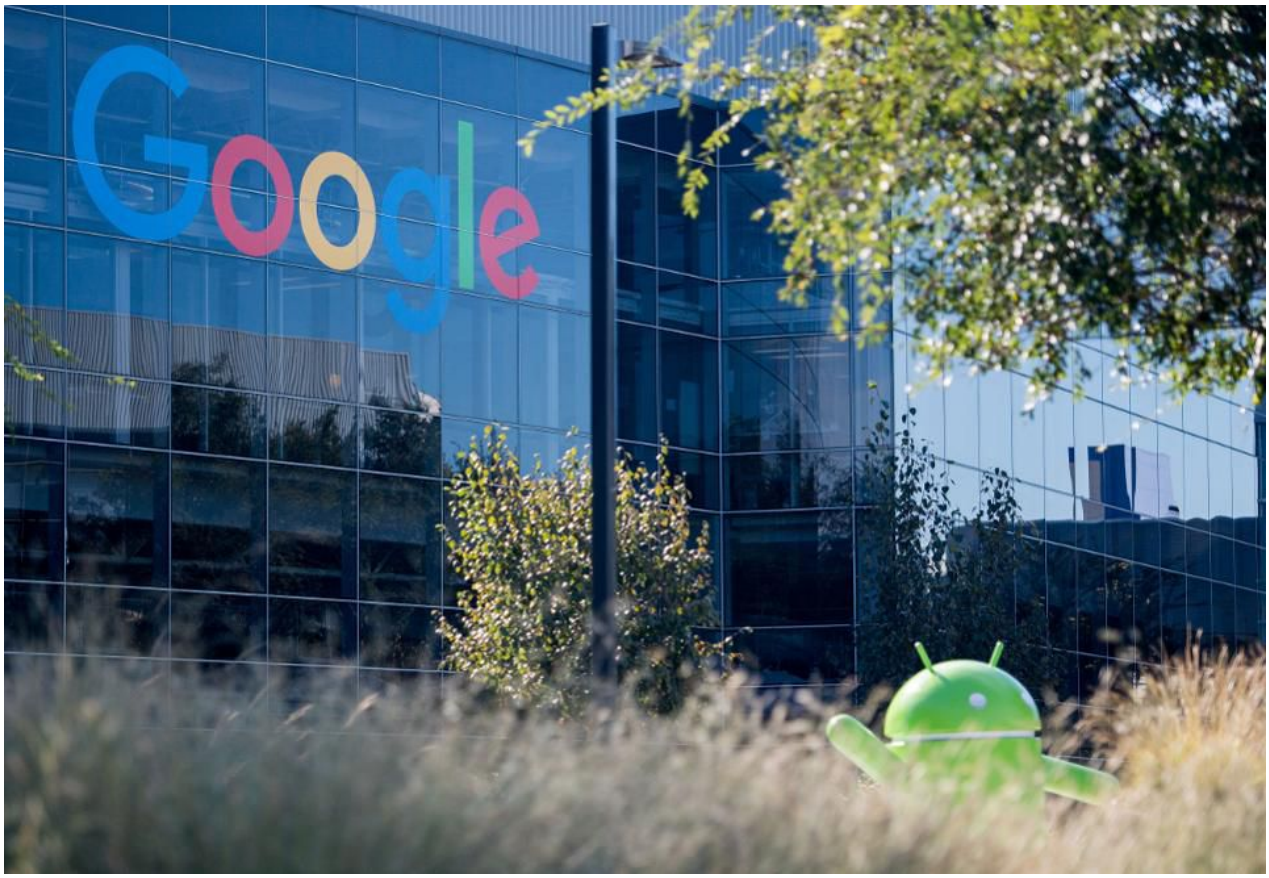
Jul 31, 2017, 01:21pm EDT

Is Working At Google Really The Gold Standard For Software Engineers?

Quora Contributor
Tech

This article is more than 2 years old.



(Photo: JOSH EDELSON/AFP/Getty Images)

Does working at Google really make you deliver your best as a software engineer? originally appeared on Quora: the place to gain and share knowledge, empowering people to learn from others and better understand the world.

Answer by Edmond Lau, Former Googler, Author of *The Effective Engineer*, on Quora:

I joined Google's Search Quality team right out of college. During my two years there — from 2006 to 2008 — I learned many things about how to become the best software engineer I could be.

I learned programming best practices from industry veterans who had distilled decades (probably even centuries) of collective experience into treasured documents of do's and don'ts and the rationales behind them.

I learned how to design good APIs from Joshua Bloch (the lead architect behind the Java collections API) and gathered wonderful insights from Guido van Rossum (the inventor of Python). High-quality tech talks were going on all the time.

I learned how critical it was to invest in simple building blocks. So much of Google was built on top of shared abstractions like Protocol Buffers and MapReduce, and so much tedious plumbing and hard problems became easy by assuming that these primitives were available.

I learned how having lots of data can trump smart algorithms. Peter Norvig calls this the "unreasonable effectiveness of data." [1]

I learned how simple, seemingly trivial choices like standardized conventions around how to indent your code matter at scale, when over ten thousand engineers are contributing to a 2+ billion-line code base [2] and anything that reduces confusion and complexity provides huge payoffs.

I learned how useful code reviews were in scaling code quality and how important it was to get iterative feedback on the code that I was writing.

I learned how powerful it can be to spread engineering cultural values. Every week, a group of Googlers would plaster bathroom stalls with one-page "Testing on the Toilet" flyers that shared the week's testing tip, and those tips would get incorporated into our work. What a quirky and fun way to spread knowledge.

I learned how important it was to invest in onboarding materials to ramp up new members of a team. I soaked up all the codelabs, best practices guides, and design docs that I could find, and I was thankful that these resources were so readily available.

All of these contributed to my effectiveness today as an engineer.

But there were also many things I didn't learn at Google — things I learned elsewhere and that I consider a core part of what it means to work effectively as an engineer.

I didn't learn how to truly be rigorous about prioritizing my work. Google was a fun playground, and my projects never had any real deadlines. Since

then, I've worked at startups where the choices engineers made directly affected the survival of the company. Those experiences have pushed me to think harder about where I can create the most impact with my time.

I didn't learn what it meant to have a really fast development loop. Resource-intensive problems at Google are generally tackled with C++ [3] — including its web server — which meant that changes to a web application could easily take minutes to compile. Nowadays, any development loop longer than a few seconds feels slow.

I didn't learn the level of empowerment and ownership that's possible at a smaller company — where the decisions you make and conversations you have as an engineer can shape the growth and direction of the business.

I didn't learn how to trade off building the “right” thing against moving more quickly. Most things at Google are engineered for scale. That means it's harder to do simpler things that don't scale, in service of getting a product or feature in front of users sooner — so you can learn if you're even building the right thing. This was a pattern I had to unlearn from my time at Google.

I didn't learn how to push the limit of how fast we could iterate and deploy production code. At Google, my team's code deployed once per week. When I worked at Quora, every commit went straight to production, and we were deploying code 40-50 times per day. It opened my eyes to an entirely different way of working that I hadn't realized was possible.

I didn't learn how to reason about user growth and engagement as a system to be optimized — a perspective that I've picked up from Quora and Quip and that provides a valuable lens on how to reason about

impact.

I didn't learn how to trade off code quality with iteration speed. At Google, every piece of code was reviewed because the audience was large, but that also meant code reviews often became a bottleneck. At every startup I've worked at since then, engineers selectively reviewed code based on risk and impact, and it has allowed us to iterate more quickly.

I didn't learn how to ask for and give direct feedback to accelerate my own growth and the growth of my team. Google's feedback system was tied to performance reviews, and getting textual feedback from peers twice a year is too slow if you want to build a remarkable career. At Quip, we've been experimenting with lightweight, weekly feedback sessions, and it has been amazingly valuable in connecting and growing the team.

All of these lessons (and more) that I didn't learn at Google I learned later. Some I learned during my time at startups — Ooyala, Quora, and Quip.

The rest I learned during a two-year quest while I was interviewing engineering leaders around Silicon Valley for my book, *The Effective Engineer*. I'd ask people, “What separates the most effective engineers you've worked with from everyone else?” “What's the most valuable lesson you learned in the past year?” Some themes in their answers would line up with my own experience at Google. Other themes — though they would show up again and again — surprisingly did not.

Google can teach you a lot on your journey to becoming the best engineer you can be — and it can be a great place to start out — but there are also many valuable lessons that are much harder to learn within Google’s culture and environment.

If you want to dive deeper into lessons I shared above — as well as what I learned from some of the best engineers at Google, Facebook, Twitter, LinkedIn, Airbnb, Reddit, and other top tech companies — I've put together a free collection of the best resources and proven techniques I gathered for becoming the best engineer you can be. They're based on the interviews I conducted for the book.

Footnotes

[1] Peter Norvig - The Unreasonable Effectiveness of Data

[2] https://www.wired.com/2015/09/go...

[3] https://www.quora.com/Why-did-Go...

This question originally appeared on Quora - the place to gain and share knowledge, empowering people to learn from others and better understand the world. You can follow Quora on Twitter, Facebook, and Google+. More questions:

- Google: What kind of company should you go to work for if your eventual goal is to get hired at Google?
- Software Engineering: What kind of process did great software companies go through to establish their engineering values and best practices?
- Career Advice: What do developers/programmers expect from their manager/supervisor?

Quora

Follow

Quora: the place to gain and share knowledge, empowering people to learn from others and better understand the world.