I just got a developer job at Facebook. Here's how I prepped for my interviews.

Andyy Hope
Nov 9, 2017 · 9 min read

I just finished seven on-site interviews at Silicon Valley tech companies. I ultimately accepted an offer for a software engineering job from Facebook.

Here's how I prepared for these interviews, and what I learned along the way.

My multi-year journey toward Silicon Valley

When I was studying Computer Science at my university in Australia, I always envisioned my future as a software engineer in Silicon Valley.

I loved the idea of being in the heart of all the tech industry's innovation — as well as its blunders. This goal kept me motivated. It kept me focused.

I left my post as Lead iOS Engineer at an amazing company in Melbourne and headed back to my home city of Perth in order to study. There I would prepare for the interviewing process ahead of me in Silicon Valley. I knew would be incredibly difficult and arduous.

If you mention the tech interview process to a room of software engineers, many will speak out against common interviewing practices. A lot of the argument comes from the reasoning that solving algorithms on a whiteboard doesn't actually represent, or translate to the day to day tasks of a software engineer.

For the sake of this article, I won't go into that conversation. Instead, I'll explore these different types of interview practices from a candidate's perspective, I'll also focus on what I learned from the process.

Interviewing is a skill

During my preparation, I always knew that interviewing would be challenging. But I honestly had no idea how hard it would be until I was knee-deep into my first interview.

In the lead-up to the interviews, I had used both paid and free services, which simulated coding and whiteboarding interviews over the phone with people who had industry experience interviewing candidates. Those practice interviews were essential for priming me for the pressure involved. But as I later realized, they only amounted to a fraction of what a real interview consists of.

I'd advise against interviewing at your dream job without having a few mock or real interviews under your belt. The nervousness can be incredibly overwhelming, and it can only be dulled through practice.

As with many other things in life, practice will improve your confidence.

The different types of interview I encountered

If you prepare and perform well enough in the preliminary phone screens, you'll be given the opportunity to come on site and conduct full days worth of interviews. These interviews will typically last four to six hours depending on the company for which you're interviewing with.

During my trip to Silicon Valley, I managed to line up seven on-site interviews in total. This gave me a unique perspective of the current landscape for interviewing.

Typically, an on-site will cover three main subjects: algorithm, architecture design, and behavioral, which is what I had studied and prepared for. However, there are some companies which seem to be bucking this trend and expanding their interviews to cover more practical skills.

I'll briefly go over each of the topics I encountered.

Algorithm Interviews

The most common type of interview you will encounter. The interviewer will ask you to solve a problem on a whiteboard which will assess your knowledge of data structures, sorting algorithms, recursion, time/space complexity analysis as well as pattern and edge-case recognition. In this interview, you will most typically come up with a brute-force solution, and then try to improve upon that solution and discuss the tradeoffs, if there are any, with the different solutions you propose.

This was the bread and butter of my preparation, every day for six weeks, I solved algorithms on a cheap hang-up whiteboard, analysed their time/space complexity as well as really trying to understand what happens at each line of code.

Personally, I really enjoy whiteboard algorithms because I don't necessarily have to worry about writing compilable syntax (most of the time), which lets me focus solely on the problem at hand. Other people may dislike whiteboarding, but to them I'd say to practice it consistently and it may change their mind.

Architecture Design Interviews

This is an interesting interview and one that I sorely underestimated. The interviewer will ask you to design a system (on a whiteboard of course) such as a car park ticketing system, chat messenger, twitter feed, amongst other common systems.

What you're being assessed on is how you take a broad concept and design a system which meets all the requirements and constraints. But it's up to the candidate to ask the right questions, which define the requirements and constraints. This interview is more of a conversation mixed in with some drawing diagrams and perhaps even class structuring. Everything is quite high-level, so you won't be writing any actual implementation code.

Naturally, you should steer the conversation to cover your knowledge of how systems work. If you're a backend engineer, you wouldn't really go into the mechanics of the client application details unless you had some previous expertise in that area. I'm an iOS engineer, so I talked about architecture patterns, modularisation of functionality, design patterns instead of how to scale the API endpoints, adding workers, AWS and such.

Behavioral Interviews

The interviewer will ask you questions about yourself and how you deal with certain types of situations. The preparation for this one isn't as difficult as the others but does require a lot of introspection on your own behalf.

The questions are typically along the lines of:

- *How do you deal with failure?*
- *What is your biggest weakness?*
- *How do you resolve conflicts?*
- *What would you do differently?*

I feel it would be pretty hard to screw this one up but I've heard a lot of people do. They try to disguise their strengths as weaknesses, engineer their response to something that they think the interviewer would want to hear or even just pass the blame of failed projects onto other people.

- *"My weakness is that I'm too focussed"*
- *"It was all Jerry's fault, he was sick for most of the project"*

These interviewers are trained and calibrated to identify crappy people and have an acute attention to bullshit. It's a quick way to get your candidacy thrown out the window. Just be genuine, show passion for your work, own your flaws, show initiative for improvement and you'll do fine.

Culture Fit

This is usually paired with the Behavioural interview and is focused on finding whether you are aligned with the company's values. For example, Facebook follows the hacker-like culture of being bold and shipping new ideas, trial by experimentation, not being afraid to break things. Whereas Airbnb wants to create a world where people feel like they belong anywhere they go, so they look for people with great hospitality skills.

A lot of the big tech companies put a lot of emphasis on the culture and hire people based on that person's alignment with their values. If you're interviewing at one of these companies, it's important that you look up their values and find past experiences which you're able to relate and communicate to your interviewer.

Pair Programming

An interesting category for which you will be paired with another engineer in front of a computer which has been set up with a development environment, much like what you would be using in the real world. You're given a basic task with a list of requirements which you must complete, as you finish each task the interviewer will ask you to implement more functionality until the time limit is reached. You're free to use whichever resources you want, such as Stack Overflow or online documentation.

I feel like a lot of a candidate's success in this interview would be determined by exposure to real-world experiences. Unlike whiteboarding, writing syntactically correct code is required, so you should know your language and environment inside and out because you don't want to be spending too much time on the internet or documentation searching for answers.

During my previous role, I would write clean code while I was working on a task, followed by optimisation once I felt the task was complete. This kind of workflow was not beneficial to this type of interview. I managed to clean-code myself into a corner by optimising too early which made it trickier to recover from. I found that writing scrappy code and mentioning to the interviewer that I would do it differently in production was considered sufficient than writing clean and optimised.

Finding and Patching Bugs

A lot of what we do as engineers centers around finding and patching bugs which are reported to us from various sources. In this interview, you will be given a list of bugs to find and patch as well as identifying other potentially problematic code along the way.

I only saw one instance of this type of interview and I feel it would be quite difficult for someone to truly prepare for, especially if they're a junior. Each coding environment has its own little quirks and nuances, a lot of the patchwork I did came from previous experiences with the IDE (Integrated development environment) and the related frameworks which I had accumulated over the years.

Testing Domain Knowledge

Programming is fundamentally the same across most of the common languages we see today. Chances are if you know object-oriented programming in one language, those skills will mostly transfer to another.

However, this interview focuses on the aspects that cannot be transferred between languages or frameworks. You will be interviewed on environment specificities relating to API, memory management, capabilities, constraints, history and so forth.

Practicing can be challenging for this particular topic. Similar to the Bug finding and patching interview, I feel a lot of the answers would stem from previous experiences. Depending on the level of the role you're applying for, the answers you provide may be weighted differently. For example, if someone applying for a junior role doesn't know the history of why an API is structured a particular way, they may be given a concession. However, if a candidate applying for a senior role doesn't know, then they may be marked against more harshly.

Understanding Operating Systems

Depending on the role or team you're interviewing for, you may have an interview which focuses solely on operating systems. In this interview, you'll be asked questions which will assess your understanding of the lower-level mechanics of a computer's operating system.

Admittedly, this interview caught me off-guard. Operating systems was something I had learned during early years at university, but my knowledge has since become hazy on the subject which was reflected in my performance.

How you should prepare

As I wrote earlier, interviewing is a skill of its own. Even if you're already a great programmer in your day job or getting great grades in your studies, those skills won't exactly transfer 1:1 when you're in a tiny interview room. Persistence, repetition, and consistency with interview preparation and practice will be the key determining factors of your outcome.

Minimum knowledge

If anyone were to ask me what I felt would be areas to focus on, I'd suggest the following:

- **Learn to write code by hand** on paper and a whiteboard first and then throw it into an IDE for syntax highlighting, this should become second nature to you.
- **Develop deep knowledge of data structures**, their strengths, and weaknesses in comparison to each other. I discovered that implementing data structures and their behaviours from scratch taught me so much more than what I knew from their abstract concepts.
- **Completely understand Big O notation** for both time and space complexities, this will pair perfectly with your algorithm and sorting questions.
- **Grasp all major sorting algorithms** because the difference in time/space complexities have the potential to derail your optimum solution for an algorithm you're trying to solve.

When to start

Depending on your timeline, you may want to start sooner than later. A lot of the companies I interviewed with had a 12 month cooling period before a failed candidate could reapply. On the flip side, if you know you won't be ready within a year, you may as well start the process now and get a small taste of what it's like to go through the interview process so when you are ready, it won't be nearly as scary.

Don't worry

You've got this.

. . .

Resources

Mock Interviews

- interviewing.io (beta), Free
- Pramp, Free
- CareerCup, Paid

Algorithms

- Cracking the Code Interview, Book

- byte by byte, Website and YouTube
- CS50, YouTube
- Interview Cake, Website
- HackerRank, Website
- LeetCode, Website

Operating Systems

- Operating System Concepts, Book

Architecture Design

- Intro to Architecture and Systems, YouTube

Behavioural

- Intro to Behavioural Interviews, YouTube

. . .

If you like what you've read today you can check our my other articles on iOS and Swift development, or if want to get in touch, please send me a tweet or follow me on Twitter ***@andyyhope****, it really makes my day. I'm also building a text diffing app for macOS. Be sure to follow its development,* ***@SameSame_app***

Andyy Hope (@AndyyHope) | Twitter

iOS Engineer. Blogger/Speaker of Swift & iOS

twitter.com

SameSame - Text Diffing for Mac (αlpha)

The latest Tweets from SameSame - Text Diffing for Mac (αlpha) (@SameSame_app). macOS Application for text diffing...

twitter.com

Thanks to Benjamin J. Dietzkis.