David Byttow

How to Pass the Engineering Interview in 2021

David Byttow Jan 6 · 13 min read

This is a follow-up post to [ABC: Always Be Coding. How to Land an Engineering Job](#) from 2013.

Most people dread the idea of going through the interview process, especially engineers. It can be daunting to try and solve complex problems on the spot with someone watching you. Many times I've heard coworkers say they probably wouldn't pass the interview today for the company they already work at. Rejection sucks, and there are too many variables you can't control when it comes to interviewing. I've seen plenty of exceptionally successful engineers fail at interviews. You won't get an offer every time, but there are principles you can apply and concrete steps you can take to maximize your chances. Below you'll find some tips and guidelines to gain confidence and prepare for the engineering interview process in 2021.

About you

This guide is for anyone planning to interview for an Engineering role, from entry-level to director, as either an individual contributor (IC) or manager. Depending on what you're interviewing for, you should expect a some combination of a few types of interviews:

- **Coding interviews.** You'll be expected to write code to solve a problem during the interview. This demonstrates to the interviewer that you have the necessary skills to solve technical problems. You'll find more of these in junior and individual contributor roles.
- **Design/architecture interviews.** These interviews are more open-ended and are designed to be collaborative. You'll be expected to solve a design problem, generally on a whiteboard and from both conceptual and practical levels, without writing code. You'll find these in more senior and individual contributor roles.
- **Behavioral/experience interviews.** In this interview, you'll discuss your background, recent work experience, and describe how you handle specific situations found in the workplace. You'll find this in more senior and management roles.

The entry-level or junior engineer may face four coding interviews and one behavioral. When interviewing as a senior engineer, you may have two coding interviews, two design/architecture, and one behavioral. A director or principal-level candidate may have only one coding interview, two design/architecture, and two behavioral. You get the idea.

About me

I'm a self-taught engineer that has worked at companies such as Google, Square, Postmates, Snapchat, and Bridgewater Associates. I've received offers from great companies, like Naughty Dog, Riot Games, Blizzard, Pinterest, Goldman Sachs, Two Sigma, and D.E. Shaw. That's not to say I haven't failed interviews–I certainly have, and those are the ones I've learned from the most.

I also have a lot of experience on the other side of the table, having interviewed at least 500 engineering candidates over the course of my career, where roughly 10% or fewer were given offers. I hope to share my insights and techniques with you. What follows are some of the most relevant tips to doing well on your technical interviews and avoiding common pitfalls.

Before the interview

Get as much information as possible before the interview. The more you know beforehand, the better prepared you'll be. Recruiters will sometimes spend ample time preparing you; but if not, you'll need to ask for information or do your own digging. While being informed is no substitute for performing well in the interviews, it will certainly help orient you in the right direction. The more you know what to expect, the more effective your preparation can be.

Get the names of people you'll be interviewing with and review their LinkedIn profiles. Sometimes reviewing someone's background can give you useful insights. You may learn that you share connections or interests. That can help establish rapport in the beginning of the interview and make the interview feel more relaxed. Small things like that can sometimes make all the difference.

For larger companies like Facebook, Amazon, and Google, there's already a lot of publicly available information on the interview process. But if you want the most accurate and up-to-date information, you should speak with people on the inside. At the very least, it's the internal recruiter, HR representative, or other points of contact. If you're working with a headhunter, you should ask to be connected directly with the recruiter that they're working with. That usually shouldn't be a problem. If it is, then you should ask why. If it's still an issue, make sure that person is working for you and getting the information you need.

It's important to remember that everyone you're working with wants you to succeed. If you get an offer, everyone wins. The headhunter gets paid, the recruiter gets closer to meeting their goals, and the company brings on more talent. But you need to do the preparation work first in order to set yourself up for success.

Coding interviews

There will probably be at least one interview where you're expected to produce some code for any role in the Engineering org. Depending on the role and scope, it may be several interviews or just one. Either way, failing here can sink your chances entirely, so it's important to be prepared.

Can you pass coding interviews without preparing for them? Of course, but you shouldn't assume you're prepared because you spend your free-time coding or do so on the job. Focused practice will best prepare you for coding interviews. This type of practice is deliberate and goal-oriented, where you set aside time and focus on improving your skills. What you practice should be informed by everything you know about the upcoming interview. If you're unsure, then ask your recruiter what you should expect. Will you have to write code that executes? Or will you write pseudocode on the whiteboard? What types of coding questions should be expected? Sometimes the recruiter will provide sample problems and that's a great place to start.

Coding interview preparation tips

ABC (Always Be Coding). The more code you write, the more prepared you'll be for your technical interviews — it really is that simple. The key here is to make sure you've spent at least 4x as much time preparing for a coding interview as you think you'll spend coding. In other words, if you expect a single 60-minute interview writing code, then you should spend 4 hours practicing.

Be comfortable with at least one dynamic programming language. Dynamic programming languages like Python and Javascript are both well-known and conducive to writing short, powerful programs with less boilerplate code than found in other languages like Java or C++. Coding problems don't typically involve writing a lot of code, and it's most important that your solution is correct, clean, and performant (often in that order). Using a language that allows you to focus on solving the problem rather than wrestling with the compiler or abstractions is often the best way to go. For algorithmic problems, I personally tend to lean toward Python, even though it's not my preferred language for the majority of my work.

Know thy complexities. Read this cheat sheet and make sure you understand at least 80% of it. Maybe you haven't ever had to implement a red-black tree, but you should understand when and how it's useful in various applications. When presenting a solution, interviewers will often follow-up with: "What's the time complexity of your solution?" Be prepared to speak to it.

Reinvent the wheel. Practice by implementing the most common data structures in your language of choice. Do not rely on common libraries. Implement the following and write tests for them: vector (dynamic array), linked list, stack, queue, circular queue, hash map, set, priority queue, binary search tree, etc. Interviewers won't typically ask you to do this directly. They will usually take the form of a word problem or some other abstraction (though I was once asked to pair program a circular queue with my interviewer).

Solve word problems. This is one of my favorites and one of the most important activities you can do. Interviewers will often present problems that seem complex but can be reduced down to simple algorithmic solutions. First, read about competitive programming. Make sure you cover your bases by solving problems that use recursion, pattern-matching, greedy/dynamic programming, and graph-based problems. Solve a few in each category using your dynamic programming language of choice. You can find sample problems on sites like HackerRank and TopCoder. I promise this will have a big impact on how well you perform in the coding interviews.

During the interview, make sure you are clear on the problem you're being asked to solve. Sometimes interviewers won't give you all of the information required for the optimal solution and expect you to ask clarifying questions. While that's more common in design/architecture interviews, you should be prepared to make sure you're approaching the problem correctly before attempting to solve it. When you do start to work on it, focus on solving it first and optimizing it later. Correctness counts the most, and getting to a suboptimal solution is better than failing to produce a performant one.

After you solve the problem, be prepared for follow-up questions that may require you to write more code. When a problem seems straightforward, you might anticipate there will be follow-up questions that constrain or complicate the problem somehow.

Design/architecture interviews

Here you will be expected to design a system from an architectural standpoint. This type of interview is more of a discussion with the interviewer. Your goal is to produce a design that could realistically be implemented and explain your thought process along the way. There are plenty of resources available on what to expect here, so I will highlight some of the common pitfalls and tips for preparing.

It's important to start from the perspective that there is no single solution to a design problem, though there are plenty of suboptimal ones. The interviewer will observe how you approach and think through the problem. To do this, you should be more proactive than reactive. Try to think and communicate a couple steps ahead, rather than waiting for the interviewer to ask questions or probe your design.

Spend the first 5–10 minutes making sure you understand all of the requirements and constraints. The interviewer won't lay out all of the constraints and criteria for you immediately, and you'll be expected to ask clarifying questions. For example, if you're asked to design a multiplayer Chess game, you should ask questions like "How many concurrent players should we design for?" or "Can a player be in multiple games at once?" and so on.

Start at a high-level (e.g., box and arrow diagrams) and work your way down from there. It's important to not to get too caught up in the details of a particular component before sketching the overall design. Though, the interviewer may ask explicit, level-down questions. If you start designing an API for a webapp, the interviewer may ask what protocol you'd use to expose it (e.g., GraphQL, REST, etc).

Ask questions and communicate your thought process along the way. As I mentioned above, it's easy to get distracted by a certain component or pull too much on a single thread before hitting on your design's key points. The interviewer may not intervene immediately to course-correct you, so you should feel comfortable asking questions when in doubt. After drawing some boxes and arrows, it's fair to ask, "Should we start with the API or database schema design?" As you describe components, it's good to communicate your thought process proactively. If you select a piece of technology (e.g., a relational data store), you may explain why you selected that over a key-value storage solution and the tradeoffs you considered. They shouldn't have to work hard to pull information out of you.

Finally, be ready to talk about trade-offs. You should be able to weigh the pros and cons of your design. This includes areas like security, availability, scalability, hardware limitations, dollar cost, extensibility, and so on.

Design interview preparation tips

Understand service-oriented architecture. These design problems typically involve drawing "boxes and arrows" to convey how your system works and is coupled together. You should be intimately familiar with what makes a good SOA design and how/when to abstract services in logical and physical ways.

Read about how some companies built their large-scale systems. I'm not suggesting you read and digest the BigTable white paper, but it's probably a good idea to read a primer on how systems like S3, Google File System, and Google Spanner work.

Watch YouTube videos on solving design problems. You can find many useful videos that break-down common problems. Head over to YouTube and search "{FAANG company} design interview questions."

Put pencil to paper. It's not enough to passively watch some YouTube videos; you should be able to solve them yourself with paper and pencil. You should speak to the overall system design (box diagrams), API design, and database design. Here are some good examples of these sorts of problems:

- Design a ticket reservation system (like Ticketmaster).
- Design an app like Twitter, assuming 100M active users. Make sure you cover how a user can post a tweet and also fetch their home timeline.
- Design an app like WhatsApp, assuming 100M+ active users. Consider end-to-end encryption as a design requirement.
- Design a distributed web crawler to run on N remote machines that you can run arbitrary software on.

Behavioral/experience interviews

These interviews are generally less technical, though you may be asked to describe a situation that involves a technical solution. These interviews are used to assess culture fit, work experience, background, and how you handle specific situations. If you're a senior engineer or manager, you should plan to knock these interviews out of the park.

When answering behavioral questions, consider first explaining how you think about it from a conceptual level. This helps to frame your answer and demonstrate to the interviewer that you can approach handling situations in a systematic way. Interviewers will ask questions that draw on your experiences, such as "Tell me about a time when you had to deal with a conflict. How did you resolve it?" Before jumping into a specific answer, you might start with something like, "Well, I believe that most conflict in the workplace can be avoided with good communication, and so it's important that people spend time getting in sync often." Then go on to explain a situation where a conflict arose because there was a lack of communication between two people and how it was resolved. In this case, taking this approach shows that you think systematically and can both perceive and handle problems like this in a repeatable manner. When describing the situation, you might find the STAR method useful.

Behavioral interview preparation tips

Write the script. Think about the types of questions that you expect in the interview (and ask the recruiter). Then write them down and answer them, ideally with anecdotal examples. You should never have to think hard when an interviewer says something like, "Tell me about an interesting technical challenge you've solved and how you did it." Here are some questions that I would make sure to pre-emptively answer:

- What are you looking for in your next role? Why this company?
- Tell me about a time when you had to work with a cross-functional team to solve a problem.
- Tell me about a time when you demonstrated a bias for action.
- Tell me about a time when you took ownership of a project or process and improved it.
- Tell me about a time when you had to deal with ambiguity.

- Tell me about a time when you disagreed with someone or had to resolve a disagreement.
- Tell me about a time when you led a project that failed. Why did it fail and what did you learn?
- What is your management style?
- How do you know that your team is working optimally?
- How do you handle low-performers?
- How do you approach solving complex problems?
- How do you evaluate buying vs. building technology?
- What is your approach to recruiting talent?

Write down your principles. You should try to enumerate some of your principles or virtues for dealing with common workplace situations. It's a good exercise to think about how you've approached problems and reduce them down to first-order concepts. You can then use these to frame your answers. Here are some examples of my principles:

- Perceive problems early and often, but don't tolerate them. Don't just call out problems without owning them.
- Be approachable and encourage others to probe by actively soliciting questions and challenges.
- Escalate when necessary to avoid getting stuck or to resolve a dispute or problem quickly.
- Act like an owner by putting the best interest of the organization and its people first.
- Be assertive and open-minded at the same time in controversial discussions or debates.
- Understand the disagreement in a situation before attempting to find agreement; this often reveals important information worth digging into.

- Get to the root cause to assess how a process is working, and/or to learn more about the people involved.

A good interview is a collaboration between the interviewer and the candidate. You should play an active role to help the interviewer get the information they are looking for.

A few words on "remote onsite" interviews

Throughout 2020, most interviews were conducted remotely and it seems as if this will continue into most of 2021. There are pros and cons to the remote onsite interview. For one, you can interview from the comfort of your own home, so take advantage of being at a desktop (such as with more screen space and a mouse and keyboard). You might be able to reference your script during behavioral interviews or have reference material open for coding interviews (unless explicitly asked not to).

After the interviews

Once you complete your interviews, relax and don't stress because whatever happens next is pretty much out of your hands. I find it useful to immediately write some of my thoughts on how the overall process went and share them with the recruiter. They find this information invaluable, and they may be more inclined to share more detailed information and provide more feedback to you about the interview results.

I hope you found this useful or interesting. Please subscribe for future posts and feel free to email me david.byttow@gmail.com or follow me on Twitter with any questions or comments you may have.

Good luck out there!

Want more?

Subscribe to my new blog here:
https://www.alwaysbecoding.com/#/portal/signup/free

Subscribe to my channel on YouTube for updates and tips:
https://www.youtube.com/user/guitardave24?sub_confirmation=1

Always Be Coding

Programming Interview Questions Engineering Interview