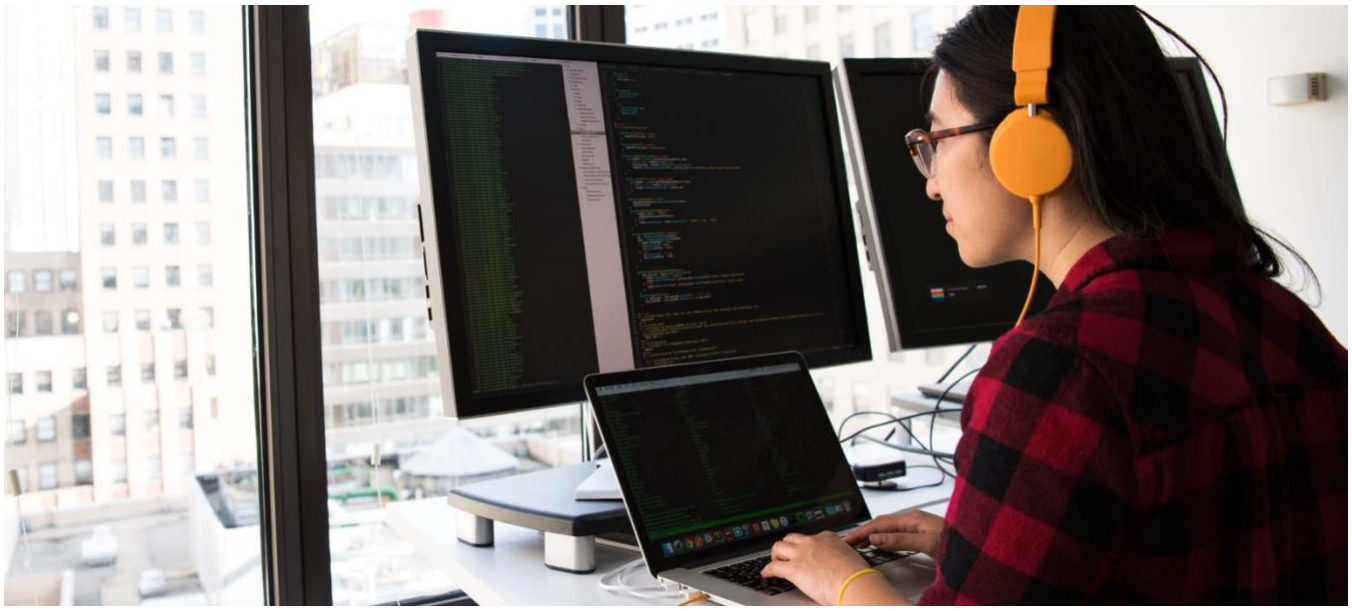

The proven method to technical interviews

While it's not a sure science, we've seen other software engineers succeed by following this same approach and wanted to share this straightforward, replicable formula for simplifying and completing complex technical interviews.

Before you begin, know your audience

Think of your interviewer as your customer. As they lay out the initial problem, imagine that you're sitting down with a potential client who is seeking your help with a specific challenge. Your goal is to solve their problem. It's as simple as that. There is a tendency to overcomplicate the problem you are asked in an interview, but framing the conversation in this light can help you keep the right challenge in front of you.

- Keep it simple and efficient
- Solve the problem they asked you to solve

Moreover, the more you know about your interviewer(s) in advance, the better you'll be able to design *to their needs.* This means understanding the goals and motivations of each person on the hiring panel. Glassdoor recommends learning as much as possible about your interviewers (search they're LinkedIn, Google them,

get a feel for what they're working on) so that you can better tailor your answers to their specific areas of interest. They are your customers, after all.

5-part formula to ace technical interviews

1. Understand the *scope* of the problem

Take the first few minutes to talk to your customer and get a better understanding of the problem, its limits, and its applications (use cases). If there are multiple people in the room, consider each person as an individual customer with a different agenda. What do they want to get out of this? How can you meet their needs?

Google Engineer Anthony Mayes suggests repeating the question (in your own words!) back to your interviewers to make sure you are on the same page. Ask questions, listen intently, and pay close attention to details. I recommend starting with use cases; outline *on the white board* the use cases you've come up with. Document everything very carefully; you'll want to refer back to these tests and use cases down the line. So much of the technical interview is actually about communication, so don't forget to turn back around **to your customers and check in with them regularly.**

2. Make reasonable *assumptions*

Before diving into code, develop a set of assumptions. The more you can get your interviewer(s) to agree with you, the better – their buy-in will help make your life easier as you design and iterate based on these pre-approved assumptions. Again, document everything. If you aren't physically in a room with a whiteboard, document on a screenshare or notepad.

3. *Design* a working version

Your design and the assumptions that support it need to be simple but comprehensive. Though you want to keep things as clear as possible, you're not trying to deliver an elementary version of what you're capable of. Think simple, elegant, but thorough.

Begin by drawing out the major components, with the assurance to your customer that you will iterate more targeted versions. The key is to make this

as easy as possible for you to succeed; don't over design, balance tradeoffs to best satisfy the set of requirements, and *avoid one-way doors.* Don't back yourself into a corner.

Once you've laid out the key components, start to elaborate your design strategy and clarify interfaces. Keep your scope tight for this initial version and plan to iterate in the next steps. Stick with simple data structures for a one-computer version before moving on.

4. ***Identify*** your design's current limitations

At this point you've mapped out your idea, ensured buy-in, and are on your way to a working solution to the problem. It's time you identify and share any limitations, weaknesses, or potential pitfalls with your customer. Don't shy away from confronting where your system can fail – it'll show that you understand the process and are willing to correct/iterate. Start laying out a list of things to fix, one by one.

5. ***Redesign*** (iterate) to address those limitations

Once your "fix-it" list is complete, start knocking items off as best you can. Iterate and elaborate. While you should definitely discuss the pros and cons when making tradeoffs – to clue the customer into your thinking process – it's important to be decisive. Every project will have limitations and the customer knows that. They want to see you reason through a complex challenge before coming to a final, conclusive decision.

Looking at the process as a simple system will help you successfully complete the coding challenge – even without knowing what the specific questions will be in advance. Remember, hiring managers aren't *only* searching for technical experts – they also want professionals who can clearly explain themselves, follow direction, and work under pressure.