Interviews

# What You Need to Know to Ace Your Technical Interview

Posted by Emily Moore
Last Updated March 21, 2018
|12 min read


If you've ever gone in for a technical interview, or even just done a Google search on the subject, chances are you've heard of Gayle Laakmann McDowell, author of the best-selling book *Cracking the Coding Interview*. McDowell's book has quickly become a sacred text for those hoping to work for tech titans like Facebook, Amazon, and Salesforce. And with a resume that includes roles at Google, Microsoft, and Apple, she certainly has the credentials to back it up.

McDowell recently chatted with Glassdoor's Emily Moore to share some of her top tips. Whether you're just starting out in the field or are already a seasoned pro, McDowell offers invaluable information — read on to learn how to secure and then ace a technical interview at the company of your dreams.

**Glassdoor: Between HackerRank, whiteboarding, paired programming, and more, there's no limit to what form technical interviews take today. What can candidates expect when going into your average technical interview?**

**McDowell:** A [typical] process starts off with one or two phone interviews (at least one of which is technical), followed by an onsite interview with four to six interviews. Of the onsite interviews, one will typically be primarily behavioral questions. The others will be technical, which typically means coding/algorithms, design, or a deep-dive into technical knowledge and skills.

A typical technical interview is 45 to 60 minutes and starts with one or two quick behavioral questions. This is done in part to get additional information about the candidate, but also to ease the candidate into the interview. If an

interviewer walks in and immediately throws out a technical question, this can be a bit unnerving.

The coding interview is often conducted using a whiteboard, but some companies offer a laptop instead. The goal of these interviews is to evaluate the candidate's problem-solving skills and to see if they can translate their thought-process into reasonably correct, well-structured code. Generally, interviewers will be relaxed about minor syntax issues (especially on a whiteboard), but I still encourage candidates to shoot for flawless code. You're unlikely to actually achieve perfection here, but if you get too sloppy or hand-wavy about the details, it can concern interviewers.

Some companies will also have you do some sort of offline assessment, either using a tool like HackerRank or completing an independent project. This is typically just before or after the phone interviews, but in some cases can take place after the onsite interviews. Some companies use this as a way to gather extra information when the interviewers failed to gather a strong enough signal.

**9 Interview Questions You Should Be Prepared to Answer This Month**

**Glassdoor: What are the programming fundamentals you feel a candidate should have down pat before heading into an interview?**

**McDowell:** Assuming the company's going to do some sort of coding exercise, having a foundation in data structures and algorithms is really important. These are fundamental concepts like binary search trees and breadth-first search. They're easy to learn, but a lot of interview questions assume knowledge of these things. A candidate should also be comfortable coding in one or more languages. It's okay if you forget the exact parameters for the substring method, but you should be able to write pretty reasonable code — for loops, functions, classes, etc — without relying on internet searches.

**Glassdoor: Since data structures and algorithms are so important, do you need a CS degree to land a coding job at one of these top tech companies?**

**McDowell:** Not really. With a CS degree, a candidate will hopefully already know the core data structures and algorithms. But that only takes candidates a week or so to learn, so it doesn't really give those candidates a huge advantage.

The biggest disadvantage that candidates without CS degrees have is just a lack confidence.

I've seen many candidates without degrees who panic as soon as they get a question that feels "academic." It's what I call the "secret sauce fear." I ask a question that involved trees or graphs, and even though they do have the knowledge to tackle it, they give up immediately. They think everyone has something "more" than them, and they don't even try.

**Glassdoor: What's the best way to practice for a technical interview?**

**McDowell:** *Cracking the Coding Interview* is probably the most popular resource. It's all-in-one and provides a lot of different specific examples for you to go over. If you don't have a CS degree, it'll help you learn some of the fundamentals. HackerRank can give you practice problems as well.

It's really important to study on your own, but remember that oftentimes when you go in for the technical interview, they won't give you a computer — they'll give you a whiteboard. They do that in part because it encourages people to think and communicate a little more, but it requires that you know what to do before you write the code down. You need to practice being in that environment, so you might want to buddy up and practice interviewing with a friend. It not only gives you more experience as an interviewee — it also gives you experience as an interviewer. Lots of things people don't understand about the interview process beforehand make sense once they do that.

**Glassdoor: Is there any danger in sounding too rehearsed when you come to a technical interview? And do people typically judge the presentation of your answer just as much as the content of it?**

**McDowell:** When done properly, the questions you get in a technical interview are ones that you won't have heard before. In fact, the point of these

questions is to assess how you tackle problems that you haven't seen. If I asked you to design a city or create a new way to do something, there's not necessarily a 'wrong' answer, but there are lots of bad ways of doing it. Now that being said, you do sometimes have companies look up well-known Google questions and ask those, and it can trip up the people with answers that are too well-rehearsed. But those are generally companies who don't already know what they're doing.

Interviewers might judge you partially on how you communicate, but they tend to be forgiving there — unless there's a red flag like arrogance. What is most important is your ability to tackle the problem effectively and your fundamental problem-solving skills.

**Glassdoor: Is it okay to not know an answer? And if you don't know an answer, what's the best way to respond to it?**

**McDowell:** It's totally okay and normal! I assume you don't know how to solve it, so that's fine. What I encourage is to take a moment to process and make sure you understand the problem correctly and all of the details. Then use the whiteboard to write down the details and come up with examples of input and output, which will help you figure out how to go about solving the problem. Think of if you took someone who didn't know anything about coding and asked them a coding question, they'd have no idea what to do — but if you told them "find instances of this thing in this list," they'd be able to do it. So look at the input and walk through how to get the output. Once you have some approach, even if it isn't very fast, think about what the slowest pieces are and how to optimize it. If your approach actually fails, then think carefully to define why it's not correct.

Once you get to a solution, don't rush into coding it. Ironically, one of the things that slows people down is rushing — they get a basic idea of how to solve something, then they start immediately trying to write the code for it. To make an analogy, imagine you're driving somewhere two miles away. You understand more or less how to get there, but should you start driving? Well you can, but you're probably going to make a bunch of wrong turns. It makes

more sense to just take three minutes to really understand the directions before you leave.

**Glassdoor: What do you need to know for a technical interview that's not a standard data structure/algorithm question?**

**McDowell:** Developers should also be able to give a walk-through of their resume. Nothing long — maybe 30 seconds to a minute. The goal is to give an overview of their career. Some great introductions or walkthroughs also tell a story of sorts about what drives them.

Additionally, developers should be prepared to talk about 3-5 projects in detail. They should be able to give an overview of the architecture, and discuss the hardest parts. When thinking about the challenges, you should focus on the hardest technical challenges — not just times when you had to learn a lot. So often, when I ask developers about the challenges in a project, their answer is some version of "look how much I had to learn." I understand, as a developer myself, that working in unfamiliar technologies is challenging. But that's challenging whether you're a good or bad developer, so this doesn't impress interviewers. Better answers really talk about a technical challenge — an optimization, a tricky bug fix, a new algorithm. These are things that really show you're a great engineer.

**How to Master the Art of Bragging Like a Pro**

**Glassdoor: What do you do if you're having a hard time even getting a technical interview in the first place?**

**McDowell:** Besides *Cracking the Coding Interview*, I have a book called *Cracking the Tech Career*, which is for the people who think "Gosh, I'm

a programmer for an insurance company but I'd love to work for Google, how do I do that?" Or "I'm in marketing, how do I transition to working at a tech company?" One of the points I make in those books is that, to land an interview, you want to show not only that you would make a great developer, but also that you have the requisite knowledge to pass the interviews.

To show that you have the right knowledge, a CS degree will certainly work. But if you don't have a CS degree, taking a data structures and algorithms class on Coursera will help. You could also do a little competitive programming since it requires competitive programming. I've also seen people who provide a link to their implementation of problems from *Cracking the Coding Interview.*

To show that you'd be a good employee, projects (open source, independent, hackathons — anything!) help a lot. You should have three or four bigger projects on your resume that allow you to show off your skill set, passion, and initiative.

Don't get too tied in with one programming language (this happens a lot with boot camps). You're not a .NET developer; you are a developer who happens to currently use .NET. You could learn a new language and your skill set would certainly transfer well. Tech companies don't care about being strong in a particular language as much as they care about fundamentals. And, in fact, they often have a stigma against people who identify as a developer in one language too much, so diversifying that skill set is really important.

**Glassdoor: How has technical recruiting changed since you wrote your last book? Are there any earlier trends that are on their way out?**

**McDowell:** The latest edition of my last book came out a year and a half ago, and the earliest edition [was] released around 2011. Honestly, as much as people talk about how technical interviews are changing, they're really not. The styles of questions have stayed basically constant for 15 years, other than additional focus on scalable systems and more focus on web technologies. They want to know if you can write good code, if you're a smart

person, if you're good at problem-solving, and if you can work effectively on teams. That way of assessing doesn't change — it's just the languages and technologies that change.

But even though I don't think the interview process has changed very much at top companies, it does seem to be filtering down to lower-tier, non-tech companies. A lot of people recognize "Hey, look at Google, Microsoft, Amazon, Uber, Lyft. These companies are building cool tech — they know what they're doing. Let's do what they do."

**Glassdoor: Some say the technical interview is broken due to factors like implicit bias and the fact that they don't always accurately capture the sort of work you'd be doing on a day-to-day basis. Do you agree?**

**McDowell:** It's an interesting question. I don't disagree that there's bias on average — if someone sees a woman or underrepresented minority, they might assume they're less technical. However, I suspect coding interviews are less biased, as evaluation is more objective. Even a prejudiced interviewer will recognize that a candidate came to an optimal solution quickly.

I suspect that less objective interviews, such as behavioral interviews, have more bias. Women, for example, are often reluctant to promote their own accomplishments, in part because they can be effectively punished for doing so. And even when they do, an interviewer might be more likely to question how much the person really did or how challenging the task actually was.

Some companies have experimented with alternatives to live coding interviews, such as with homework projects. But these come with their own issues. How much time does a typical mom with two kids at home have to work on a project like that?

We should address implicit bias, but it'd be a mistake to move away from a biased process to something that is even more biased.

Is the process broken overall? It's true that there are lots of false negatives — people who do poorly in a technical interview who might actually be great

developers. But I will say this: Technical interviewing, when done effectively, can produce top notch results and talent.