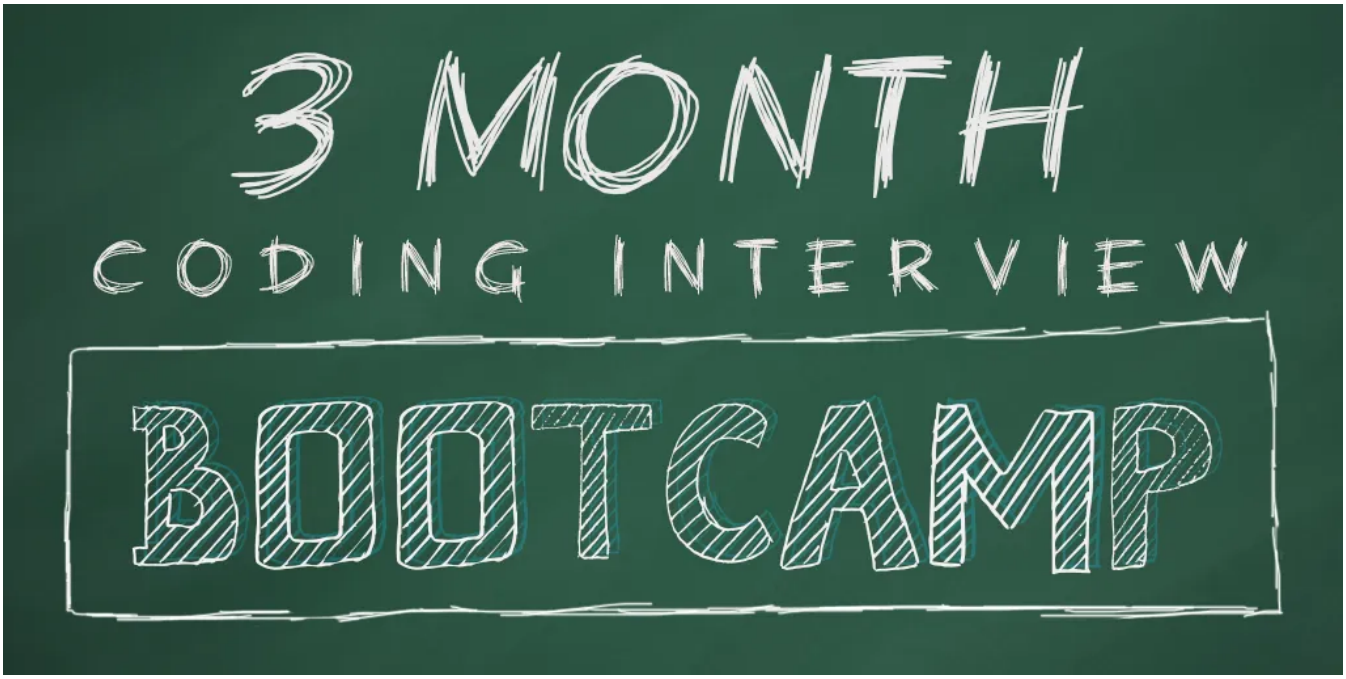
Published in Grokking the Tech Interview

Fahim ul Haq Follow
Oct 27, 2018 · 11 min read · Listen

Save

3 Month Coding Interview Preparation Bootcamp

The 3-month Self-Study Roadmap to Success at Software Engineering Interviews



(Some background: I've interviewed hundreds of candidates for software engineering jobs at Facebook and Microsoft. I've also failed several coding interviews myself when I wasn't prepared.

I originally started writing this as an answer to a Quora question about the roadmap for software engineering interviews. Eventually, the answer became so long that I thought it warranted a blog post of its own.)

Three months — Really?

Let's get this out of the way. People ask me a lot: what's a reasonable timeframe to crack the coding interviews if you're starting from scratch? (i.e., you haven't interviewed in the past several years.)

> *I would say that three months is a reasonable timeframe. Yes, really, three months. And barring that, at the very least dedicate 4–6 weeks if you haven't interviewed in a while. You can probably get away with less than that if you have interviewed in the last 12 months or so.*

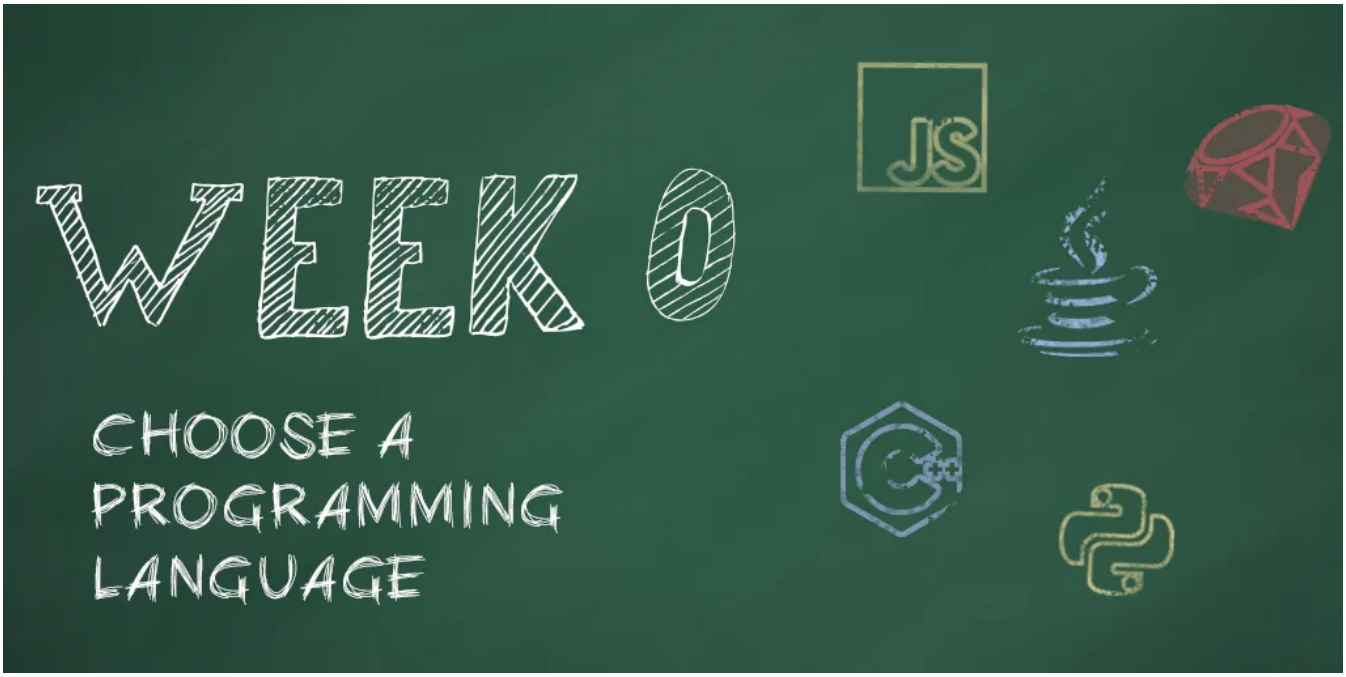
Now for the meat. Below are the five main sections that software engineering interviews at "Big Tech" companies like Facebook, Google, Microsoft, etc. will consist of:

1. Coding Interviews (focusing on problem-solving, data structures and algorithms).
2. OS and Concurrency Concepts
3. System Design Interviews
4. Object-Oriented Design Interviews
5. Cultural fit interviews

Like any other long-term goal that requires consistent work (such as getting ready to run a marathon), following some kind of structure is crucial, as it will encourage you to stay on track on days that your motivation might be waning.

To help with that, I've created a 12-week preparation plan that you can follow to prepare for your next coding interview. If you follow the plan over these 12 weeks, you'll cover all of the topics mentioned above in a structured way. Let's get started.

Week 0 — What Programming language should you use?



Pick a programming language and then stick with it. I'm often asked: what if I know more than one? Would, for example, Python be better than Java?

The answer, of course, is that the best programming language for your coding interviews is the language that you're most comfortable with. Most companies/interviewers don't care as long as you can show proficiency in any one mainstream programming language.

In some of the worst cases, I've seen people deciding to "switch" to a different programming language in the middle of the interview. That's a big turn-off and a waste of time. Don't do that. Pick one early, and stick to it throughout.

Week 1 — Brush up on the basics of your favorite Programming language

Brush up on your chosen programming language. There are going to be a lot of things that you've forgotten when you have been coding for your day job even using your preferred language. I've seen people struggling to remember things like:

- How to read/write from/to files
- How to read input from the console
- How to split strings
- Is string length a function or a property (answer: it doesn't matter, but still reflects poorly on you)
- How to declare and use 2D arrays
- In C/C++, how to handle null-terminated strings

Once, I saw a candidate get confused and not be able to remember how to figure out if a number was positive or negative. (I'm sure they knew it — they just had a brain freeze).

> The time and mental energy that you spend on trying to recall the nuances of your chosen programming language would be much better spent on actually solving the problem and exhibiting your

> problem-solving skills. That's what interviewers want to see.

Some companies, like Lyft and Salesforce, require you to solve problems on a laptop. You are expected to write complete programs that pass given test cases. In these cases, you might have to:

- Process command line arguments
- Parse CSV or text files

Of course, you could just Google those, but that would be time spent on trivial tasks that are necessary but don't help you stand out.

Most other companies, including Amazon and Google among countless others, will require you to solve problems on a white board. This is a very different experience, requiring some different skills than coding in an IDE. Now is the time to start practicing actually writing out code (while talking through your thought process) to flex those muscle.

Weeks 2 & 3 — Data Structures and Algorithms

Start revising Computer Science concepts like Data Structures and Algorithms. You know, those concepts that you once studied in your undergrad and have never

looked at since — they're actually quite useful in coding interviews. Here are some relevant resources:

- For Python, look at Data Structures in Python for Coding Interviews
- For Java, look at Data Structures in Java for Coding Interviews
- For JavaScript, look at Data Structures in JavaScript for Coding Interviews
- For C++, check out Data Structures in C++ for Coding Interviews, as well as Algorithms in C++ for Coding Interviews.
- For Algorithms in various Coding Languages, check out Algorithms and Complexity Analysis: An interview refresher
- For Big O notation and complexity analysis, look at Big O for Coding Interviews and Beyond. You can also check out this post titled "The insider's guide to algorithm interview questions" which gives you a detailed look at complexity analysis and how to work with it.

Remember to revise topics like:

- Complexity Analysis (a.k.a BigO)
- Arrays
- Stacks
- Queues
- Linked List
- Trees
- Tries (They are effectively trees, but it's still good to call them out separately).
- Graphs (BFS and DFS)
- Hash Tables
- Heaps
- Sorting
- Searching

Weeks 4 & 5 — Practice simple Data Structure and Algorithmic challenges

As you are familiarizing (or re-familiarizing) yourself with data structures, start practicing relatively simple coding problems associated with these data structures and algorithms.

These questions are typically **not** asked in interviews at big tech companies. Even if they are, they're usually used as fizz-buzz type warm-up problems. Such questions are also common during phone interviews. However, practicing these coding interview questions will help you internalize the data structures and help you tackle the harder questions which you'll be practicing a few weeks from now.

Brush up your array skills with questions like:

- Remove Even Integers from Array
- Merge Two Sorted Arrays
- First Non-Repeating Integer in an Array
- Find Second Maximum Value in an Array

Brush up your Linked List concepts with questions like:

- Find Length of Linked List

- Search in Singly Linked List
- Reverse a Linked List
- Find Middle Value of Linked List

Brush up your Stack/Queue skills with questions like:

- Sort values in Stack
- Create Stack where min() returns minimum value in O(1)
- Implement Two Stacks using one Array

Practice Tree Problems like:

- Find minimum value in Binary Search Tree
- Find Height of Binary Tree
- Find kth maximum value in Binary Search Tree

Practice Graph Problems:

- Implement Breadth First Search
- Implement Depth First Search
- Detect cycle in Graph

Practice basic Trie Problems:

- Total number of words in Trie
- Find all words stored in Trie

Practice basic Heap problems:

- Find k smallest elements in a list
- Find k largest elements in an array

Weeks 6, 7, 8 — Practice more complex coding interview problems

Now that you've been practicing more straightforward problems for a couple of weeks, it's time to get real and start practicing harder questions that are more likely to be asked during coding interviews.

For practice and automated challenges along with interactive solutions, look at **Grokking Coding Interview Patterns** (in **Python**, **JavaScript**, **Java**, **C++**, and **Go**).

1. .

Here are some guidelines to keep in mind as you solve these problems:

1. Now is the time to start timing yourself. Ideally, you shouldn't spend more than 20–30 minutes solving any given problem. (This probably won't be possible for all questions right away.)
2. Don't be discouraged if you are not able to solve a problem within the allocated time. Solve it even if takes you a couple of hours, without looking at the solution. This will help you build the confidence that you *can* solve it and then you can focus on solving them faster later.
3. Start thinking about the Runtime and Memory complexity of each solution. You will have to articulate the complexities in the actual interview clearly, so it's better to start now.

Here are some sample problems to consider:

- Implement Binary Search

- Find the Intersection point of two linked lists
- Reverse words in a sentence
- Check if two binary trees are identical
- Clone (deep copy) a directed graph
- Find solutions to a Boggle game
- Determine if there are any three integers in an array that sum equal to the given value.

You will have to spend 2–3 weeks here. Don't worry if you hit roadblocks and get stuck often — you will get the hang of it after a while. Trust me, questions that look impossible in the first few days start to seem easy after you've had practice.

Weeks 9 & 10 — System Design Interviews

System design interviews are now an integral part of the software engineering interview process — particularly if you are applying for a senior role. These interviews have a significant impact on your "hiring level."

Learn distributed systems concepts like Cap Theorem, Consistency, Partitioning, Load-Balancing etc.

Look at the course **Grokking Modern System Design for Software Engineers and Managers** for more design interview practice.

As part of your System Design Interviews, you are asked to design a “web-scale” service. Interviewers are interested in evaluating your ability to describe the different parts of a scale-able service, such as:

- How are web-servers load-balanced?
- How are databases shared?
- How are large files stored?
- How is the network set up for redundancy and maximum throughput?

You’ll want to practice questions like:

- Design Instagram
- Design Facebook Newsfeed
- Design Uber

Check out my article the How to Prepare for the System Design Interview for more example questions, tips, and resources for the System Design Interview.

Week 11 — OS and Concurrency Concepts:

Today, even “budget” laptops and mobile phone have multiple cores. Understanding concepts like Threads, Locks, Synchronization, etc. are beneficial whether you are building a mobile app or a web-scale service.

Just like System Design interview questions — Multi-Threading and Concurrency Interview questions are useful in gauging your level. A junior engineer will struggle with these questions (and are expected to learn more on the job). A relatively senior engineer is supposed to do better in such questions as they would be responsible for writing a lot of code that takes advantage of multiple cores/threads.

C.H. Afzal’s “Multithreading and Concurrency for Senior Engineering Interviews” series is an important one to check out, and available in multiple programming languages.

Week 12 — Object-Oriented Design Interviews:

Some common questions include:

- Design an ATM
- Design an elevator
- Design a Parking System

For more Object Oriented Design interview questions, check out this article: The Top 10 Object-Oriented Design Interview Questions for Developers.

In Object Oriented Design Questions, interviewers are looking for your understanding of Software Design Patterns and your ability to transform the requirements into comprehensible Classes. You spend most of your time explaining the various components, their interfaces and how different components interact with each other using the interfaces.

Take a look at Grokking the Object-Oriented Design Interview to learn more about questions that are typically asked during OOD interviews.

For learning more about some best practices for object-oriented programming with regards to software design patterns, look at Software Design Patterns: Best Practices for Software Developers.

Cultural fit interviews

This is the one that many think won't matter, although this is the interview that sometimes matters the most. For example, at Amazon, culture is deeply rooted in their hiring process, where a "Bar Raiser" (someone who lives and breathes Amazon culture) can have the final say over you getting hired.

The thinking for that is quite simple: if you have the right attitude, you can learn new skills so minor shortcomings in your coding or system design interviews can be overlooked. However, if a person seems to be dispassionate about the product or doesn't look like a team player, they are probably worth hiring even if they are great hackers.

There's also a famous book called The No Asshole Rule. Companies try not to hire people who can be toxic — the long-term cost of doing so can be enormous. Companies also don't want to hire engineers who are not passionate about the product. Cultural fit interviews are there to weed out such people.

Some of the basic rules of Cultural fit interviews are:

1. Show interest in the product, and demonstrate an understanding of it. (I once had a candidate who told me that Facebook sells cloud services like AWS (Storage/Compute). He had even used one of those. Now, Facebook did buy Parse.com and kept it alive for a while, but Cloud Infrastructure was never Facebook's primary/core business).

2. Be ready to describe scenarios where you had a conflict with your teammates or managers and how you resolved it. Please don't say that you never had a conflict if you've been working as a software engineer for a few years.

3. Talk about what you want to accomplish in the company

4. Talk about some of your recent / most significant accomplishments as an engineer

5. Talk about some particularly crazy/difficult bugs that you encountered.

Conclusion

Preparation for coding interviews takes a lot of time and effort, but if that helps you stand out and prove that you're ready for a complex job, it's worth it. I've found it helps to keep in mind the value of the end-goal throughout — in this case, the personal satisfaction and financial compensation of landing a big-ticket software job.

Resources

For your reference, here are a consolidated list of the resources for software-engineering interviews that I've mentioned throughout the post:

1. Grokking Coding Interview Patterns in Python
2. Grokking Coding Interview Patterns in JavaScript
3. Grokking Coding Interview Patterns in Java
4. Grokking Coding Interview Patterns in C++
5. Grokking Coding Interview Patterns in Go
6. Algorithms and Complexity Analysis: An Interview Refresher
7. Mastering Data Structures: An Interview Refresher
8. Big O for Coding Interviews and Beyond
9. Grokking Modern System Design for Software Engineers & Managers
10. Grokking the Object-Oriented Design Interview

11. Grokking Dynamic Programming for Coding Interviews
12. Multithreading and Concurrency for Senior Engineering Interviews
13. Software Design Patterns: Best Practices for Software Developers

I'd love to know if you have any feedback. Please reach out to me at fahim@educative.io if you have any questions or feedback.

Coding Interviews | Programming Interviews | Programming | Software Engineering | Coding