LAND YOUR DREAM JOB!

A-Z PLACEMENT GUIDE FOR SOFTWARE DEV'S

BY HRISHIKESH SUSLADE

From choosing the Coding language to applying for Jobs. This guide cover's everything!

A-Z Placement Guide for Software Dev's!

HRISHIKESH SUSLADE

DEDICATION

Dedicated to my parents, teachers, friends, and mentors who helped me in my journey! From getting started with coding to landing my first job it would not have been possible without their support!

CONTENTS

ACKNOWLEDGMENTS

A huge thanks to geeksforgeeks.com, YouTube, CarreerCup.com, and Cracking the Coding Interview (book) for helping me clear my coding and data structure concepts and helping me in landing my first job at Amazon! These sources were instrumental in making this book possible!

1 INTRODUCTION

When I started preparing for Software Development roles I was in a complete mess. I didn't know where to start, what to study, why am I doing this? How will the interviews be? And millions of other questions!

I'm sure you are also currently going through the same phase or must have gone through it!

My seniors and my college coding club helped me a lot in getting started. Right from writing the first line of code to interview tips I had someone to guide me! But, that's not the case with everyone.

Another thing that I observed was there is not a source from where you can get all the guidelines and help. You need to move from one source to another for different stuff. Study from website A practice on B, give a test on C. I can't change it, but what I can do is at least share the knowledge about it with you. Trust me, if you have proper guidance at the right time it can put you miles ahead in life.

Fortunately, I received that on time, but I have seen many people fail because of a lack of guidance.

So, what is this book exactly about?

Well, it's your answer to Why? How? What? Questions related to getting a software development job.

It will tell you why you can be a software developer, how you can prepare for it and what you'll need to do for landing a great Software Development job!

The book is divided into 11 modules, each module covers an important aspect one faces during a software development interview!

2 WHY SOFTWARE DEVELOPMENT?

In the last 10-15 years, we have seen a huge boom in the Software Development industry. The number of jobs have increased exponentially and so have the number of people who want to be a software developer. And the major reason behind it is the great pay this position offers!

Many students blindly take up Computer Science or related subjects just for the sake of getting a high package, without giving a thought to what the job will be?

Well, it is a lot different than the traditional office jobs that we imagine. We build and develop software's and products which will reduce someone's effort and increase their productivity, making a particular process efficient and smooth.

Well in layman's terms we are the people who program Google Maps to help you find the shortest path, we are the ones who make your Amazon Orders deliver within 60 mins by optimizing the supply chain!

It might seem complicated/simple on a high level but when you deep dive it's full of wonders, there are scenarios and cases that we don't even think of but are very important and very crucial to the system.

So, can I be a Software Developer or not?

YES, anyone can be a software developer. But, but, but there are some conditions.

You should be a person who can think a lot, always willing to find a way to make things simpler and the most important one always ready to learn new things!

Because Software Development is about Learning to Learn…

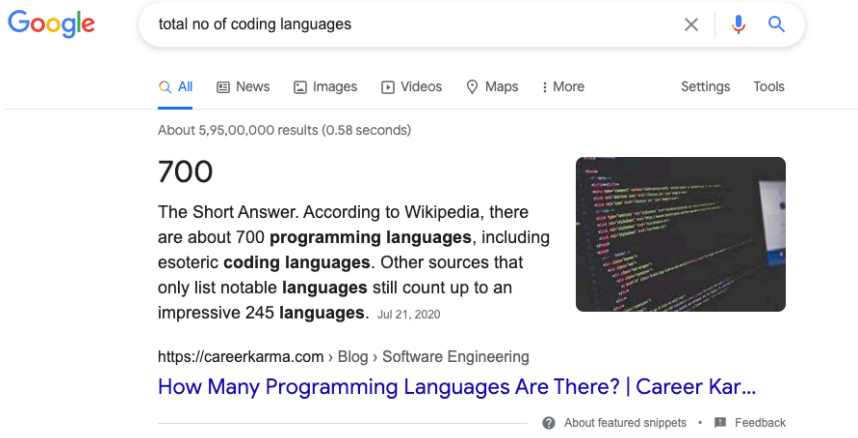
3 GETTING STARTED WITH CODING

Why should you code?

We are living in an age where everything is becoming automated. From cars to coffee machines there is some or the other kind of automation there in all devices around us. And all this is possible because of Coding. This is a skill that you surely don't want to miss out on. Whatever be your profession coding will help you in some or the other way. And especially if you want to make your career in Tech, coding is a must. It will be beneficial to you a lot!

So, CODE!

Which Language to choose?



There are over 700+ programming languages available!

And it's not an easy job to choose from where you should begin?
But, there's one thing

"If you have good command over one language, you can easily switch to different languages"

So, you don't need to learn all these languages! But, still which is one to choose?

For this I'll suggest choosing a language that can implement Object-Oriented Programming, some are C++, JAVA, Python, etc. The reason being the industry requires it, OOP based languages are used widely. Why we'll cover that later!

Now we have narrowed down our search to 3 languages C++, JAVA, and Python.

My personal favorite is C++

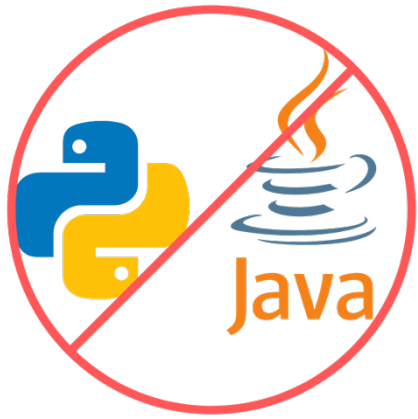
Why C++?

There are many reasons to choose C++, but some of the important ones are:

- Easy to Adapt
- Lots of resources and documentation
- Fast and reliable
- And the STL Library

STL Library makes a lot of things easier for you, especially in coding rounds and interviews. It is widely used in competitive programming also.

Why not JAVA or Python?



Java is not bad but it is a completely Object-Oriented based language. So, it is difficult for beginners to adapt to it easily. But, if you are comfortable with it you can go ahead with Java also, it is completely your choice.

Python is a relatively very easy language to code in. So why not start with the easy things? Because it will make you lazy. You will find it very difficult to write lengthy syntax and methods if you switch to Java or C++ from Python and also it is relatively slow.

Where to study?

There are many resources to study C++.

- Books
- Blogs
- YouTube Channels
- Paid Courses

But the resources I'll be suggesting will be completely free of cost.

So, let's find these resources.

- New Boston YouTube Channel - C++ Playlist
- Geeksforgeeks (free and public articles)
- cpluscplus.com

At Least for learning a language I don't think one needs a paid resource!

By the way, I have used these resources. So, yes, they are good and have quality content.

- Now you have chosen a language
- Know the resources
- But, how to study?

"Remember, coding is studied by doing it!"

- So, Study the concept
- Understand it
- Implement it
- Repeat and also Practice. That's how you study coding

What's next?

Now once you know a language properly the next steps get a bit complicated.

You should study some basic Object-Oriented Programming and Data Structures and Algorithms.

Don't feel burdened with all the jargon and new concepts. It can be very difficult for a newbie but with time and proper practice and efforts, you'll become very comfortable with this stuff. If you are planning to apply for Product based companies, starting with some Competitive Coding will also help. The earlier you start, the greater the lead you get!

Irrespective that you should practice on online platforms. At all stages of learning, with the time you'll be able to solve all the questions. but keep solving!

- Hackerrank
- Codechef
- Hackerearth

Even I solve some problems regularly. The reason being, this keeps me in touch with my basics and keeps giving my brain something to work on!

4 DSA'S

Data Structures and Algorithms are a core part of your work. They are the tools that will be used by you to make things work faster, efficiently, and effectively!

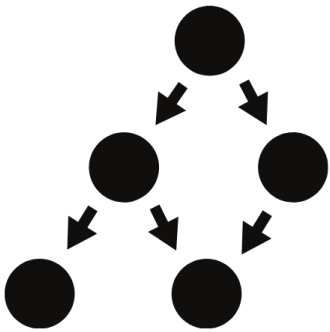
DSAs also enhances our problem-solving skills and help in solving many complex problems.

The majority of coding interviews are around DSAs.

So, in all DSAs are very much important when it comes to software development

But what I have observed after reading 1000s of interview experience is not all data Structures and Algorithms are needed. You don't need to know all of them by heart!

If you master the key and core DSAs properly you can solve a majority of the questions. But there will be exceptions.



The following DATA STRUCTURES and ALGORITHMS are the ones I'll suggest one should master nicely.

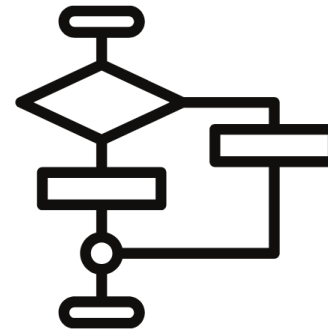
Data Structures

1. Stack
2. Queue
3. Tree
4. AVL Tree

5. Binary Tree
6. Graph
7. Hash Table
8. Linked List

Algorithms

1. Searching Algos (Binary)
2. Sorting Algos (Merge, Quick)
3. Graph Traversal
4. Travelling salesman
5. Sieve Algorithm
6. Dynamic Programming (not exactly an algo, but an imp concept)



I'm telling it again these are the important ones you should have proper mastery over them. But that doesn't mean you should ignore other concepts. **Knowledge is never wasted!**

How to master DSA?

I used this technique to master DSA. It works in three-phase Understanding, implementation, and practice and repeat

We pick 1 DS or algo at a time!

1. Understanding

Here try to gain a conceptual understanding of how a Data Structure or Algorithm works. The best way to understand is to try to relate it to an example.

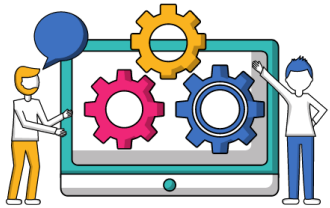
Why do we need it? How are we using it? Try to find answers to these questions.

This will help you get a basic understanding of that DSA. Seems tricky? Don't worry we'll go through a series of examples, which will explain to you clearly how to do it!

2. Implementation

Imagine if you could build a car on your own. You know how to

repair it when it fails. You know how to modify it when needed after as and when required. Well, that's the same purpose here. Implementing anything on your own gives you a great edge on understanding the concept and working on that topic!



What does implement a data structure or Algorithm mean?

Well choose the language of your choice

Preferably C++, python, or Java

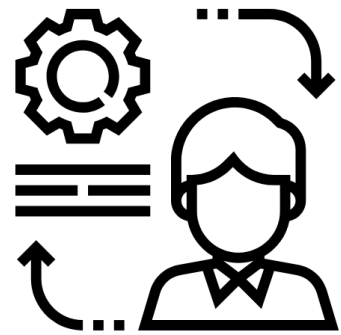
And write the code of that DSA on your own. You can refer to resources like GFG etc. But, remember that don't just copy-paste. Refer it and write on your own!

Try doing this till you don't need to refer to the resources. By now you must have a good understanding of that topic.

You don't need to include all the functionalities of that DSA in one go, go step by step. (Let's say you are implementing stack, first implement push and pop operation, the size, followed by the top and so on)

I know you must be thinking well I'm doing this. There are inbuilt libraries for these. And yes you are correct we don't need to implement data structure and Algorithms in our job. But knowing how it works under the hood, help you enhance the performance and modify it according to your need. Also, it helps you boost your problem-solving skills!

3. Practice



What's the use of all the knowledge if you don't know how to use it?

And don't be under the impression that "I know how it works; I can solve any questions" I failed miserably in multiple interviews because of this thinking.

Unless and until if you can't apply these concepts, you can't master them!

So, practice is a must.

Solving around 50-60 questions on a particular topic will help you gain a good command over that topic. But these questions should be of varying difficulty.

10-15 easy
20-30 medium
8-10 hard
Hard questions are not generally asked, but they are to just develop your problem-solving skills.
GFG, InterviewBit, and Hackerrank is a good place to practice.
Try to solve questions where you can apply those DSA.

Eg. Balancing brackets - Stack

Follow this process for the above mentioned DSAs. There's no hard and fast rule for solving 50 questions if you are confident enough on a particular topic you can move ahead!

Try to stick to a single programming language throughout your practice. Also, while you are learning to implement the DSA on your own, don't forget to check out how to use the inbuilt libraries also. Because many times you'll have to solve a complex question that needs multiple DSA.

Remember one thing your logic matters!

Let's say you want to study Stack Data Structure.

So, we will follow the above approach

Step 1: Understanding

Where is the stack used?
Ummm, bracket balancing?? Yes, it is a good example of where the stack is used!

How does it work?
A stack of books in front of a professor!
Perfect! You can only place the books from the top and the professor will only check the books from the top unless he/she wants to crash the stack ;)

So, in short, Stack is Data Structure where the Last person to submit the book is the person who's book will be checked first LAST IN FIRST OUT

Step 2: Implementation

Before we start to code, let's devise a rough algorithm first and before that let's decide what functionalities are we going to implement

We will create a very simple stack
Just Push, Pop and topElement and size of the stack be n

Let's devise an algo

Push Operation:

The only edge condition we have to look after is when we try to push even when the stack is full.
This will result in StackOverflow!!!!!!

```
if(Stack.size > n ):
    Return: Stack Overflow
Else:
    push(element)
    Return status
```

Pop Operation:

Now, for pop operation, we have to think of just one special condition, when there is no element in the stack!
I.e when topElement is NULL or empty

```
if ( topElement == NULL):
    Return : Can't pop stack empty
else:
    Change the topElement
Return: TopElement / TRUE status (for successful
pop operation)
```

Now let's code
I love to code in C++ when it comes to DSA, so I'll pick it, but you are free to choose whatever you like!

```
#include <bits/stdc++.h>
using namespace std;

#define MAX 10

class Stack {
    int top;

public:
```

```
    int a[MAX];
    // construtor initially top in NULL
    Stack(){
           top = -1;
    }

    bool push(int x){
if (top >= (MAX - 1)) {
        cout << "Stack Overflow!!!!";
        return false;
    }
    else {
        a[++top] = x;
        cout << x << "SUCCESSFULLY PUSHED IN STACK";
        return true;
    }
    }

    int pop() {
            if (top < 0) {
        cout << "Stack Underflow";
        return 0;
                  }
    else {
        int x = a[top--];
        return x;
    }

    }

    int peek(){
           if (top < 0) {
        cout << "Stack is Empty";
        return 0;
    }
    else {
        int x = a[top];
        return x;
    }
    }

    bool isEmpty(){
return (top < 0);
    }

};

int main() {
    Stack s1;
    for(int i=0; i<10; i++) {
           s1.push(i*i);
           if((i*i)%2 == 1) {
                  s1.pop();
           } else {
```

```
                    s1.peek();
                }
    }

    if(s1.empty())
        cout<< "I'm empty now";

    return 0;
    }
```

Now, find the output of this code and check whether this code works perfectly, play around with the code on IDE till you understand how it works!

Step 3: Practice

Now we know how a Stack works, how to implement it, now it's time to practice.

Practicing doesn't mean just implementing a particular algorithm over and over again!

One should practice questions where a particular DSA is applied. Now as we are studying Stack, find questions where you can apply stack DS to solve it!

Bracket balancing is one of the most trivial ones. You can go to websites like GFG and Hackerrank to find data structure and algorithm-specific questions and solve questions of varying difficulty levels. Some easy, few hard, and majority of them should be of medium level.

Pro Tip: Try to solve the most common questions first it's very much likely that these questions might show up in your test

NOTE: Though I suggest solving 50 questions for a particular DSA, sometimes you might need to solve less - because it is easy or sometimes you might need to solve more than 50 questions. So, keep a note of this!

Now, your task is to follow the above methodology to study the mentioned DSAs. This can take some time anywhere between 3-4 months, it can vary though!

Also, at the same time, I'll suggest giving some time for the DSA not mentioned by me, they are less like to be asked but you should check them out.

Some of them are RedBlack Tree, Trie, Segment Tree, etc.

The advantage of this process is you are gaining knowledge and at the same time you are learning how to apply it, and trust me this is what we have to do as Software Developers, apply knowledge problem-solving skills, and technical skills to solve problems!

5 PRACTICE, PRACTICE AND PRACTICE

Now you have mastered the key data structures and also practiced a lot of questions specific to those DSA. But, have you observed one thing during practice you knew what kind of Data Structure or Algorithm I need to use to solve it? But, in reality during your coding rounds, interviews, and even during actual work we have to figure it out on our own, and this needs a lot of intuition that comes with Practice, to be honest, a lot of practice!

And there is no shortcut to this, you'll have to work hard and practice many questions to get that intuition of how to solve any question!

Before you jump off on how to practice, I'll suggest you ***follow this methodology to approach any coding or any other problem***

1. Determine what you have, determine what should be your desired output
2. Look for the special or edge cases
3. Break the problem into sub-problems
4. Solve these sub-problems to solve the main problem

This strategy has two advantages

1. It makes your task easy. Many of the times questions are filled with ambiguous or useless information just to confuse, but by using this methodology you only focus on what's needed.
2. It puts a good impression on the interviewer. Well, this is just a personal experience for and many of my friends. This approach shows that you solve your questions properly and understandably!

You have the methodology; you know now it's time to put your knowledge to the test!

There are many online coding platforms from where you can practice. But I'll suggest you stick to only 2-3 platforms max. Reason being when you go to multiple platforms you end up solving similar questions of similar difficulty levels. Whereas when you stick to 2-3 platforms you solve a good variety of questions of varying difficulty levels.

Practicing Platforms
I'll list the coding platforms that I have used with their pros and cons

1. **GeekForGeeks**
   Pros:
   (a) A lot of questions to solve from
   (b) Questions have proper difficulty level defined also they are tagged with company and data structure or algorithm
   Cons:
   (a) IDE is not so Good
   (b) Test cases are very easy to pass
   (c) Time Complexity is not tested for the majority of questions'

2. **Hackerrank**
   Pros:
   (a) Excellent UI
   (b) Good test cases
   (c) Great collection of questions for Data Structure practice
   Cons:
   (a) Company-Specific questions are not found much
   (b) Questions are not updated regularly

3. **Leetcode**
   PA great
   (a) Great collect company-specific questions
   (b) Excellent Discussion Forum
   Cons:
   (a) Very less DSA specific questions
   (b) Compilation time is much more for free users

4. **InterviewBit**
   Pros:
   (a) Very few platforms to have a time limit while solving questions
   (b) A decent collection of high-quality questions
   Cons:
   (a) Code expected is very rigid, especially the functional questions expect you to follow a strict methodology. It's good and bad, but I feel we should have the freedom of choosing how to solve the question
   (b) No of the questions is limited

5. **Codechef**
   Pros:
   (a) Quality of question is great
   (b) Frequent competitions and events
   Cons:
   (a) Questions are too demanding, at least when you are planning to get a job. Questions are good if you like Competitive Programming
   (b) IDE is not good

You can try them out and fix which 2-3 platforms you will be using. I'll suggest GFG, Hackerrank, and Leetcode, and sometimes CodeChef, just to twist your brain.

And don't forget to practice using pen and paper also, because you'll need them. In later chapters, you'll understand where!

How to Practice?

You know DSA and you have also practiced them. So, know you should solve competitive programming questions on either Hackerrank or CodeChef. Start with the basics, start with easy questions. It's very much likely that you will be able to solve them easily, owing to your rigorous practice of DSA, but it's okay if you are not able to do so!

I was not able to solve more than 1 question in my first 3-5 long challenges on CodeChef.

Everyone starts with small, but with practice, you'll be able to solve more and more questions with increasing difficulty. Practice for a few days on

Hackerrank and CodeChef. If you are not able to solve questions on CodeChef it's okay you can skip it, I too skipped it after a few months!

But, try to solve at least 2-3 coding questions daily. Coding is the exercise for the brain, if we stop doing it, we just lose our stamina. But, keep increasing the difficulty level and in between try to solve some tougher and not so straightforward questions!

Company Specific Questions

You have practiced many questions now. You have great confidence also. And the best thing you landed an interview with a tech company. We will talk later about how you can land an interview. But, what now should you keep practicing with the same strategy as you have been following before? Should you start working hard? What should you do, when you have an interview lined up in the next 1-2 months?

Well, I got a perfect strategy for you!

Believe it or not, every Company has a hidden pattern in their questions. I'm not saying that questions will repeat, they can though. But, companies ask questions in a pattern, and they change very less. If you can find what this pattern is you get a great edge over other applicants!

How to find and use these Patterns?

Well, here also you'll have to practice a lot of questions. The reason why I said you need many questions is that the bigger our dataset, the greater are chances that our prediction of that pattern is accurate.

I solved around 100-150 questions that were asked in Amazon and observed that it asks questions on Graphs and Trees much more than other questions!

And guess what it turned out to be true, I was asked 2 questions related to Graphs and Trees.

So, try to find interview experiences of that particular company, GFG is

an excellent resource to find it. And go through as many interview experiences as you can before your interview. After a point, you'll realize what kind of pattern that company follows, and you can then focus on that topic more!

Let's say a company is asking questions about Graphs, you can focus on it and practice more questions related to it. And it helps, even though the questions might not be asked directly, but your previous practice can help you in devising the approach to solve the new question. I'm speaking from personal experience, in my Amazon Interview, I was asked a question on Trees that I had never seen, but my practice on trees helped me out. I created an approach by combining two different questions and voila I got a working code!

This methodology does not guarantee that you'll get questions as you predicted, but it's a good approximation and helps a lot! Especially when a company's favorite questions are from your weak topics!

From my personal experience, what I have observed is Product based companies like FAANG ask questions related to graphs, Trees more and Banking Companies ask questions around Dynamic Programming!

You should start company-specific practice around 1-2 months before the interview and GFG Interview experiences and Leetcode are some excellent resources to practice!

6 STUFF OTHER THAN DSA

DBMS, OS, SYSTEM DESIGN, NETWORKING, OOPS

Software Development is much more than just coding. It is not just solving complex problems using DSA, but to give solutions that are scalable to millions of customers! And that cannot be done alone by just using DSA, we need to use different tools and techniques to make it possible. System Design is one tool.

Also, companies expect you to be a full stack developer so you should be able to work with any stack that you are given to work on, one day you are managing the database the next day you are defining the network configurations of the hosts! Thus, we need to know DBMS, Networking, and even Operating Systems.

Another thing is they are asked in interviews! So you need to study them!

System Design is no doubt the most important topic among these because it uses knowledge from all the rest of the topics, so we'll go through the rest topics first and then come to System Design.

Database Management System

Before we begin I have a question for you.

What is the difference between Database and Database Management System?

Research on this one and find an answer! It won't be asked in interviews but knowing doesn't hurt 😊

Most questions are around SQL databases, so I'm focusing on it currently.

- **What you should know?**
  (1) Designing a SQL database
  (2) ACID properties
  (3) Writing and executing Queries (Medium-Hard level, Joins are very famous ones so study them properly)
  (4) Normalization up to 3N
  (5) Other basic stuff

- **From where shall I study?**
  (1) Abraham Silberschatz - Database System Concepts (If you want to do a deep study)
  (2) GeeksForGeeks
  (3) Tutorialspoint
  (4) sqlzoo.net

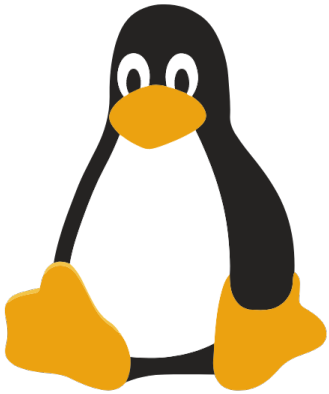
- **What shall I study and practice?**
  (1) Writing SQL queries
  (2) Designing Database
  (3) Applying Normalisation
  (4) How ACID properties can be implemented?

Databases are one of the most important topics and you can't skip this one. And writing queries or finding the answer to SQL queries is one of the most common questions asked in Coding Interviews. If you are applying for internships in your second year or third year it is very much likely that a DBMS question will be asked to you, because at that stage you don't have much knowledge about System Design, so the next closest thing they can ask you is about Databases.

If you have done any project related to DBMS, it is very much likely that you'll be asked about it. I had done a simple DBMS project and one of the coding interviews was 60-70% on that project. The questions were mostly about how can you take this system and make it work in the real world, what challenges can you face, how will you solve it? them that time I did not know these things, but learned from my mistakes, if you are mentioning a DBMS project in your resume, you should have a thorough study of that system, researching how you can scale and deploy it will also help you!

DBMS is also a core part of the System Designing process, so knowing it will help you in those questions also!

Operating Systems



The Operating System is the backbone of Software development! Every software is developed on an Operating System, so you can't ignore it! Even though the frequency of questions asked is very low, but I feel having OS knowledge helps a lot in your software development job.

- **What Should I know?**

(1) Scheduling Algorithms
(2) Deadlocks
(3) Multithreading and threading concepts

(These topics are the most widely asked common topics, but do study other basic OS concepts also don't expect very high level or complex questions on this topic)

- **From where shall I study?**
  (1) GeeksForGeeks
  (2) Operating System Principle by By Peter Galvin
  (3) Ravindrababu Ravula Gate Lectures on OS

- **What shall I study and practice?**
  (1) Practice implementation of various Scheduling Algorithms
  (2) Don't make the mistake of considering this an OS concept and ignoring there are many coding questions asked on this topic in coding rounds and interviews!
  (3) Dealing with Deadlocks

System Design

As a fresher System Design is not asked much in the interviews, but if you are applying for a senior role such as SDE 2/3 you will be asked about System Design. Not just that, System Design is very much useful when you join a company as a software developer. Because Software Development is a lot more than just coding, it comprises of

- Deployment
- Scalability
- Consistency
- Backups
- Load Balancing
- Caching and much more

And all these things come under System Design
Thus you can't ignore it!

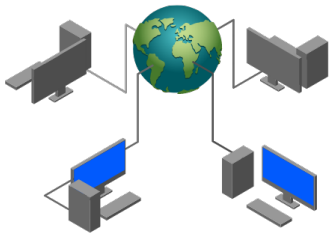
- **What should I know?**
  (1) Core system design concepts
    - Scaling
    - Caching
    - Backups
    - Load Balancing
    - Database Sharding

  (2) Understanding of HLD and LLD

- **From where should I study?**
  (3) Gaurav Sen System Design Playlist on Youtube
  (4) Systems Expert Course by Clement Mihailescu
  (5) Grokking the system design interview Course on educative

- **What Shall I study and practice?**
  (1) The concepts I mentioned above
  (2) Practice Designing various real-world systems like Instagram, WhatsApp, Parking Lot, Lift system, etc.
  (3) Practice LLD and HLD o a few use cases

Networking



Networking enables communication between various systems. Although a software developer rarely needs to interact with network settings, sometimes you need to configure such things and there the knowledge of networking comes into play.

Coming to the interview part, networking questions are asked with very little frequency, and most of them are very basic, not much preparation is required other than the college curriculum.

- **What should I know?**
  (1) Networking Basics
  (2) Networking layers
  (3) Concept of subnets
  (4) IPs/ MACs/ etc
- **From where should I study?**
  (1) Computer Networking A top-down approach
  (2) GeeksForGeeks.com
  (3) Computer Networks by Ravindrababu Ravula YouTube
- **What Shall I study and practice?**

  To be honest nothing much here just has a good understanding of the theoretical concepts and understand how they are used in thal-world applications and systems.

Object-Oriented Programming and Design Patterns

Coming in last doesn't mean this topic is not important. To be honest, this is the most important topic other than DSA, and the most widely asked questions other than DSA are asked on OOP, so pay attention now!

OOP and Design Pattern makes coding very much easier and efficient. As a fresher, you are expected to code, and thus working with OOP and Design patterns is your daily job. Consequently, a lot of questions are asked on this topic.

- **What should I know?**
  (1) OOP concepts
  (2) Inheritance
  (3) Polymorphism
  (4) Encapsulation
  (5) Abstraction
  (6) SOLID Principles
  (7) Design Patterns
  (8) UML/Class and other types of Diagrams

- **From where shall I study?**
  (1) New Boston C++ YouTube (If you are a beginner)
  (2) Head First OOP and Design Principles
  (3) Sourcemaking.com
  (4) Refactoring.guru
  (5) Design Patterns in OOP by Christopher Okhravi YouTube

- **What Shall I study/practice?**
  (1) A thorough understanding of core OOP concepts
  (2) Practice and implement all these OOP concepts
  (3) Practice applying Design Patterns to various use cases
  (4) Practice Drawing UML Diagrams for use cases

The majority of the questions asked will be around Object-Oriented Programming, so have a greater focus on those topics.

A simple suggestion is to remember a short example of each concept, this will help you recall the concept and even explain it!

Anything Else?

We have covered almost every important topic that can be asked in a coding interview. But, Software development is a vast topic! And no one can predict what will be asked.

So, it's okay sometimes if you don't know the answer.

Other than the above-mentioned topics do solve lots of puzzles, they are asked very frequently in the interviews. Sometimes you might be asked questions on Digital Logic Design, State Machines, regex, etc. So, if you study these topics in your academics properly. I don't recommend spending extra time on these topics!

7 PROJECTS

"If you haven't done Projects then, what have you done?"

Don't get disheartened by the above statement, but this is the real truth! As an aspiring software developer, it is very much important that you have very good projects in your portfolio. DSA is the knowledge and projects are where you apply this knowledge.
Projects display your practical knowledge, they display your teamwork if you have done projects in a team, they display your expertise in a particular domain. Especially if you are applying for a Data Science, frontend, web dev, etc type of job roles then it is very much important to have projects.

How should you begin and What kind of Projects you should do?

Start with the basics, build simple projects Web Development is a great place to start, Python also offers great project options.
Start with simple projects, build simple websites such as notepads, simple arcade games, frontend websites. Gradually move towards complex projects build REST APIs, integrate with websites, deploy them, use multiple APIs, use OpenSource APIs to do simple tasks, etc
Start with simple vanilla.js and gradually move your way towards frameworks. Make use of documentation provided by the frameworks and many of the frameworks have a "Hello World" kind of projects which can help you get started!

Do explore Machine Learning, Data Science, and Blockchain domains, these are some of the most developing areas in software development and thus it will surely help you!

1. Web Dev
   i) Flask, Django
   ii) NodeJS
   iii) REST APIs
   iv) GraphQL
2. Python scripts

How to build projects?

YouTube is an excellent place to learn. Learn by doing things, follow the simple tutorial available, build a basic model, and then try out your version. If you are stuck try Googling it! This might not sound fancy, but to be honest this is the best way to learn!

Another Method is - learn by Blogs, blogs are also an excellent way to implement things and understand them. The major advantages of a blog are you can directly copy-paste the code, no need to pause the video and write the code, you can just do everything at your own pace!

Disclaimer: I do not support Copy-pasting of code!

There are various blogs on Medium and Freecodecamp which are excellent sources of learning various tech and making great projects!

Coming to YouTube channels, well these are the few channels that I refer regularly for project-related stuff!

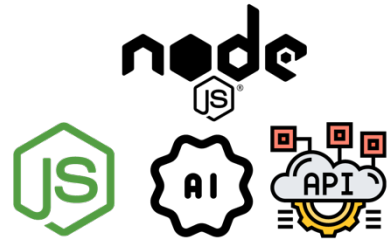
- Freecodecamp.org
- Fireship
- Dev Ed
- Hitesh Choudhary
- Programming with Mosh
- Traversy Media
- Telusko
- Clever Programmer

Now that you have seen so many channels, do check out my YouTube Channel also :)

Projects and Resume!

I'll be covering the Resume Building in the coming chapters in detail. But for now, let's focus on the projects sections of the resume.
Depending upon which kind of role and job you are applying you need to showcase that type of project on your resume.
If you are applying for a software development role projects such as

- Web Dev
- Mobile Application
- Machine Learning or Deep Learning
- Database Projects
- Website or Application clones



Are some good projects that you can have!

Ideally, you should have around 10-20 projects with you on your GitHub profile, but on your resume, you should mention only 2-3 best projects that you have done!
These projects can also be a part of your curriculum or internships. Try to put in projects which have system design stuff used in them, they are very good discussion topics for your interview.

While mentioning your projects in the resume don't forget to mention key aspects of your project in the description

- Accuracy
- Efficiency
- Speed

aka metrics that show the impact of the project

E.g. My Project at Amazon helped reduce the workflow plan file creation time from 3 months to 2-3 weeks.
And yes, don't forget to mention your GitHub link!

But, one should be very thorough with the projects they are mentioning in the resume, be prepared to be grilled heavily during the interviews!
To be honest, don't just copy-paste it, understand the logic behind it, try to document it, keep a design document if possible, this will help you a lot in the later stages!

Some good project ideas that you can build!

- Social Networking website clone with backend
- Written text converter using open-source API
- Object Detection projects
- Malicious Mail / Spam Mail detector
- Ride-Sharing Application
- Food Delivery App
- Movie Ticket booking system

These are just a few suggestions, you can always do more complex or simple projects according to your capacity!

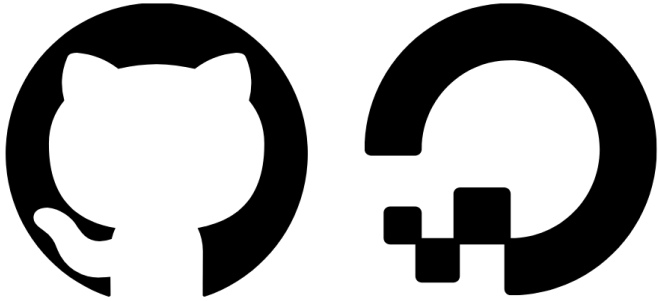
8 OPEN SOURCE AND HACKATHONS

Projects, DSA, and Competitive Coding will give you knowledge but Open-Source contributions and Hackathons will teach you how to apply this knowledge! These two are the best place to learn, see, and do what an actual software developer does!

And guess what many software developers full-time open-source contributors! So, don't underestimate these programs because they can have a huge impact on your resume and skill set!

Open Source

Hacktoberfest is the best place to get started with Open-Source development. It happens every October every year, but you don't need to wait till October to get started! If you know even some basic git commands and some knowledge about a particular Language or a framework is more than enough to get started!



And even if you don't know about this it's never too late to get started! There are some blogs by Digitalocean* on how to get started with Open Source programs. Do check them out and get started with Open Source programs!

Once you get hold of the basics, explore new and different programs such as Google Summer of Code, MLH Fellowship, GirlScript Summer of Code, FOSSASIA Open-Source Program, etc. These are many advanced open-source programs that will give you great exposure and you'll learn a lot of new things! Do check out the respective websites and their social media channels for timely updates so you don't miss the openings!

*(https://www.digitalocean.com/community/tutorial_series/an-introduction-to-open-source)

Hackathons

If OpenSource development is a Marathon then Hackathon is a 100m Sprint Race! The learning curve in a Hackathons is super steep! You'll learn about so many new things in such a short period that you'll be surprised a lot!

And you'll be more surprised when you see the output of your just 2 days of effort! And the best part of a Hackathon is the Goodies :)

I attended just a handful of Hackathons in my engineering but, I request you attend as many hackathons as you can! Don't be afraid to fail, Don't think about winning and losing! Think about what new things you could learn from such a short period!

Me at Amazon CTPS (internal) Hackathon2019
Winner of Peoples Choice Award

It can take days, even months to build a website, but in a hackathon you can, build the same things in hours, just because of team effort and hard work!

Hackathons teach you how to work in a time-constrained manner, how to work in a team, and how to deliver results. This is exactly what a Software Developer does in his job!

9 RESUME BUILDING

What is the use of all the skills that you have, but you cannot properly represent them? Well, a Resume is something like that! It gives the recruiter/interviewer a summary of you! So, a resume plays a very important role in getting a software development job. Especially when you apply through Job Portals your resume goes through an automated screening process and if your resume is not compliant with that system it is very much likely that you'll be rejected without even an interview call!
It's a harsh thing but you have to summarize all your efforts and experience in just 1-2 pages!

- **For Freshers**

a. Name and Basic Information
b. Education Details (Engineering/ MCA)
c. Internships or any experience
d. Projects
e. Technical Skills
f. Achievements (Hackathons, coding competition, etc)
g. Anything else (Position of responsibility or some great achievement in some extracurricular activity) - Does not matter much

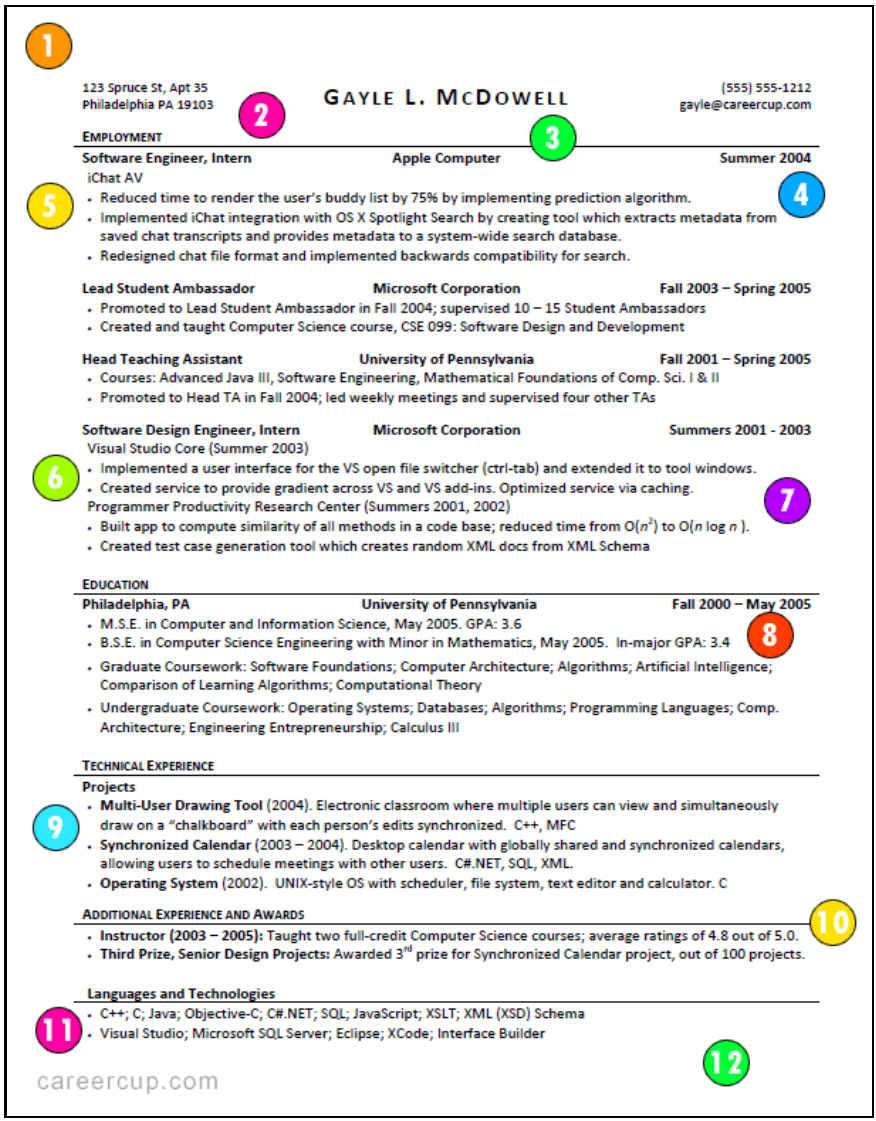
- **For Experienced Professionals**

a. Name and Basic Information
b. Work Experience and Internships
c. Education Details
d. Projects
e. Technical Skills
f. Achievements

Be precise, on your resume. It is not your history exam! Write what is relevant and write what is true is just the right number of words!

Use a single page format, as it is easy to view and understand! Do not overdo it! Keep it simple and sweet.

I'll personally suggest that the resume template suggested by Gayle Lackman in the famous book Cracking the Coding Interview is a great template for resumes.



123 Spruce St, Apt 35
Philadelphia PA 19103

GAYLE L. MCDOWELL

(555) 555-1212
gayle@careercup.com

EMPLOYMENT

Software Engineer, Intern — **Apple Computer** — **Summer 2004**
iChat AV
- Reduced time to render the user's buddy list by 75% by implementing prediction algorithm.
- Implemented iChat integration with OS X Spotlight Search by creating tool which extracts metadata from saved chat transcripts and provides metadata to a system-wide search database.
- Redesigned chat file format and implemented backwards compatibility for search.

Lead Student Ambassador — **Microsoft Corporation** — **Fall 2003 – Spring 2005**
- Promoted to Lead Student Ambassador in Fall 2004; supervised 10 – 15 Student Ambassadors
- Created and taught Computer Science course, CSE 099: Software Design and Development

Head Teaching Assistant — **University of Pennsylvania** — **Fall 2001 – Spring 2005**
- Courses: Advanced Java III, Software Engineering, Mathematical Foundations of Comp. Sci. I & II
- Promoted to Head TA in Fall 2004; led weekly meetings and supervised four other TAs

Software Design Engineer, Intern — **Microsoft Corporation** — **Summers 2001 - 2003**
Visual Studio Core (Summer 2003)
- Implemented a user interface for the VS open file switcher (ctrl-tab) and extended it to tool windows.
- Created service to provide gradient across VS and VS add-ins. Optimized service via caching.

Programmer Productivity Research Center (Summers 2001, 2002)
- Built app to compute similarity of all methods in a code base; reduced time from $O(n^2)$ to $O(n \log n)$.
- Created test case generation tool which creates random XML docs from XML Schema

EDUCATION

Philadelphia, PA — **University of Pennsylvania** — **Fall 2000 – May 2005**
- M.S.E. in Computer and Information Science, May 2005. GPA: 3.6
- B.S.E. in Computer Science Engineering with Minor in Mathematics, May 2005. In-major GPA: 3.4
- Graduate Coursework: Software Foundations; Computer Architecture; Algorithms; Artificial Intelligence; Comparison of Learning Algorithms; Computational Theory
- Undergraduate Coursework: Operating Systems; Databases; Algorithms; Programming Languages; Comp. Architecture; Engineering Entrepreneurship; Calculus III

TECHNICAL EXPERIENCE

Projects
- **Multi-User Drawing Tool** (2004). Electronic classroom where multiple users can view and simultaneously draw on a "chalkboard" with each person's edits synchronized. C++, MFC
- **Synchronized Calendar** (2003 – 2004). Desktop calendar with globally shared and synchronized calendars, allowing users to schedule meetings with other users. C#.NET, SQL, XML.
- **Operating System** (2002). UNIX-style OS with scheduler, file system, text editor and calculator. C

ADDITIONAL EXPERIENCE AND AWARDS
- **Instructor (2003 – 2005):** Taught two full-credit Computer Science courses; average ratings of 4.8 out of 5.0.
- **Third Prize, Senior Design Projects:** Awarded 3rd prize for Synchronized Calendar project, out of 100 projects.

Languages and Technologies
- C++; C; Java; Objective-C; C#.NET; SQL; JavaScript; XSLT; XML (XSD) Schema
- Visual Studio; Microsoft SQL Server; Eclipse; XCode; Interface Builder

careercup.com

Resume Template by Gayle L. MacDowell (source Careercup.com)

1 One Page Resume: Recruiters do not read your resume; they do a 15 - 30 second "spot check" of your resume. When your resume is too long, it just takes your best stuff - the stuff that would have made the "one page cut" - and dilutes it with more mediocre content. Lengthy resumes do not make you more impressive, and there are many other reasons to keep your resume short too. A good rule of thumb is to keep your resume to one page if you have less than 10 years of experience or at most two pages if you have more than 10

years of experience. And if you think you can't get your resume to just one page, trust me, you can! You just need to think about what is important for a recruiter to see.

2 No Objectives: All an objective does is the state, in a wordy way, what position you're interested in. The company already knows that because you applied for a particular position. At best, it'll just waste space. At worst, it'll limit you since it'll exclude other positions that might have been interesting to you.

3 Use a Resume Template with Columns: Unless you're great with design, you probably shouldn't be creating your resume template. It'll most likely look sloppy. Use a template, and make sure it has multiple columns. Using three columns, for example, will allow you to put the company name, position, and date all on one line. This makes it easier to read and saves space.

4 Use Tables: If you're using Microsoft Word to create your resume (which you probably should), use Microsoft Word's "tables." Just make sure to hide the borders afterward.

5 Short Bullets: Because resume screeners only spend 15 - 30 seconds on your resume, length bullets - anything that feels like a paragraph - just won't get read. Keep your bullets to one to two lines (with one line being better than two).

6 Accomplishment Oriented: Your bullets should focus on your accomplishments - that is, the impact you had - rather than your responsibilities. What did you build, create, design, optimize, lead, etc?

7 Quantify: Whenever possible, you should quantify your accomplishments. If you optimized something, by how much? If you won an award, out of how many people?

8 Resume: The general rule of thumb is to list your GPA if it's at least 3.0 or higher, but there are two important rules to know here: (1) You may choose to list your in-major GPA if it's higher than your overall GPA, but you need to specify that it's your in-major GPA. (2) If your school uses a different scale (such as a 10-point scale), you may want to convert your GPA to a 4.0-scale which will be more widely understood.

9 Projects: Most candidates should pick their top 3 - 5 projects to list on their resume. These can be the academic required project or independent projects. They do not need to be completed or launched either. As long as you've done a "meaty" amount of work on them, that's good enough!

10 Additional Experience: You can put additional experience, like leadership activities or awards, in a section like this (changing the name of the section depending on what you list). Be careful here to focus on what matters. If you're applying for a coding role, your role as an eagle scout in high school is probably not very important!

11 Languages and Technologies: It's a good idea to list your languages and technologies, but remember that anything you list here is "fair game" for the interviewer to test. If you want to list a language but you happen to be a bit rusty in it, consider listing it as something like: "C++ (Proficient), C# (Prior Experience), ..."

12 What did you NOT include?: Is there anything impress impressive/interesting you've done that you left out? About 50% of candidates leave out an important project or another component of their experience because it wasn't finished / "official" / etc. If you've done it, and it's impressive enough to "cut" (you shouldn't just list everything you've ever done), then it belongs on your resume!

Source: Careercup.com

Applying Online and ATS Scanners!

When we apply for jobs online, the majority of resumes are rejected via an automated tool! This tool is known as ATS. What it does is a simple thing it matches the keywords of that particular job posting with that in the resume, if the score is low you are rejected, if it is high you are in!

The most common mistake people make while applying for jobs is they don't make a resume ATS friendly. Now, what does one mean by making a resume ATS friendly?

- ***Using the proper File format:*** PDF and word files are the best ones. (PDF which has parsable text)
- ***Using proper keywords in the resume:*** If the job requires a C++ developer and you don't mention C++ not even once! It is 110% sure you'll be rejected! So make a resume after going through the Job Description! Try to include as many keywords from the job application in your resume! Don't overdo it also, your resume should make sense even after adding keywords!

If one follows these above methods it is very much likely that he/she will be able to clear the job portal test and you'll receive a call from the recruiter for the further stages!

Something Extra!

Even though I'm endorsing the above resume template, but I had created my template which looked a bit more appealing to the eyes, plus it had all the other qualities that I have mentioned above.

You can also do something like that, there are some great websites like Novoresume and others which have great resume templates. Or you can design one in Photoshop!
But, I'll suggest this only if you have time in your hand, don't waste your time doing this stuff!

10 APPLYING FOR JOBS

You have studied, you have practiced, you have made your resume! Now the only thing left is applying for your dream job! This can be the most tedious or tiring process but might be the most important one. You need to deal with multiple rejections, multiple interviews, and a lot more!

If your college provides a good placement opportunity then trust me getting a job will be very easy for you. Since you have prepared very well! You do not have to search for referrals, no waiting for HRs reply, and a lot less hassle. So, if you have a good opportunity for placements from college don't miss it!

Whereas if you are applying via campus you have a lot more work to do! Applying via job portals, searching for referrals, connecting with HR, sending follow up emails, etc.

Let's discuss this process in detail now

Choosing a job?

This might sound a bit naïve, but choosing the right job is also as important as anything else! You do not want to regret getting that job. So go through the job description thoroughly, go through company reviews on Glassdoor and other portals, if possible interact with someone you know from that company who will give you an insight into the company culture and other things that you need to know before you apply or start the job!

Take a thorough look at the technology stack that you'll be working on! You

do not want to end up with a tech stack with which you are not comfortable. There are plenty of jobs in the market so do not compromise for the sake of a good company. If you are good with C++ and you want to work with it only ahead don't settle up for a front-end job based on JS and TS

Applying directly or Get a referral?

When you are applying off-campus is the biggest dilemma that you might face. Should I apply directly or should I try to get a referral?
Well, let me tell you! **Having a referral puts a huge weight on your application**

But every company has a different way of handling a referral so you need to check that out beforehand! Some companies count the referral when you apply to the job after getting the referral, whereas some companies give you the benefit of referral in situations where you have applied for the job beforehand and got the referral after a few days

LinkedIn is an excellent place to search for a referral. Try connecting with the people who are working in companies you wish to apply to. Send a nicely framed message directly to the point asking for a referral and do not forget to attach your resume to the message or mail.
It is very much likely that you might not get a reply but it's okay, do not take it personally, just move on and keep trying someone will give it if you are worthy enough!

Keep an eye out for LinkedIn posts where someone is willing to give a referral upfront this is your best chance at getting a referral. I got a referral for Google from such a post only! So leverage the power of LinkedIn

This thing goes without saying, your LinkedIn should look great and should try to stand out from the crowd! Post a nice and decent Profile Pic, keep an updated resume, your work ex, education, certification, and all should be kept updated. The reason behind it is when you request for a referral your resume and LinkedIn are the first impressions so they are very important if you want to land that referral

PS: Getting a referral does not mean you'll get the job for sure; it does not even guarantee an interview call. So do not go loose on your interview prep

Sometimes you might get an interview call without even a referral so do not give up if you are not getting the referral! Applying directly on the portal can also land you your dream jobs.

Just remember that when you apply via job portals the importance of resume increases ten folds and it being ATS friendly increases 1000 times!
So make a nice ATS friendly resume and hope for the best

Also, another important thing to note about it, many companies have a no-application period! Let's say you have applied for a job and unfortunately you are not selected, so you can not apply directly for the next job in the line you'll have to wait for a set amount of period before applying again. This period is generally for 6 months. So, keep this point in mind before applying for jobs!

11 D-DAY TIPS

After preparing for months, practicing for hours, and multiple rejections you have finally landed an interview! You will be lying if you say you are not nervous. But trust me it's completely normal, I have given almost 20+ interviews in different companies but the nervousness never goes away! The only thing that changes is the confidence level and the interview questions!

Some tips might sound a bit naïve, but they do work so do not ignore them!

Sleep well!

I have seen many students who do not take proper sleep a day before their interviews, this might due to anxiety or nervousness. But sleep is very much important you have prepared for months and you are going to be tested in just 60 mins! You need to recall all the stuff that you have practiced and studied and apply that knowledge to solve a problem, trust me it's a difficult task, and having a clear and fresh mind can help you a lot in the interviews!

Ask Questions! Please!!! And Communicate

Google Interviews are known for leaving questions open-ended and you cannot fill the open-ended part of the question based on the assumption you have to ask questions, then and then only you can solve it! Asking questions is very much important! Any doubt that you have in the problem statement, space and time complexity, edge case related doubts, or anything like that! Ask your questions!

The clearer understanding that you have about the problem statement the better. But don't go too far also because you can end up in the constraint loop and mess up the question.
Keep discussing your approach with the interviewer, the interviewer will direct you in the right direction and indirectly help you out so keep talking about what you are thinking and what you are planning to do!
Especially in these current times when most of the interviews are virtual, it becomes very important to establish proper communication!

Be clear!

Try to be as clear as you can in your approach. Make use of diagrams illustrations or short examples to explain your idea. Your job is to solve the question in such a manner in which the interviewer can understand it!
Writing a clean and understandable code is also very much important. Seeing the current scenario mostly the interviews will be online and you will be asked to write code in either an Online IDE or Google Docs.
The challenge is writing code in Google Docs because it does not have auto-indentation, auto-completion, and all the common features you can expect in an IDE. So, if your interview is going to be held on Google Docs do practice writing code in it

Behavioral Interview

This can be a tricky place for sure! You have cracked all the rounds now it's an interview of your personality and not your knowledge and skills. So, look at this interview differently but do not under look at it is as important as others!
Before going to interviews of a particular company look into the tenets and culture of the company every company has them on their websites. Go through them try to understand what they are, how they benefit the company, and most importantly do you have these tenets in you or can you implement them?
The behavioral interview is a test of whether this candidate is a good match for our company culture, can he adopt to the way we think? can he handle pressure situation and deliver too?
And here it is all about selling yourself, selling your skillset. The great you are at selling yourself the great you'll be at this interview

Do not lie!

The interviewers are not fool! You have given 10 interviews but they have taken 100s of interviews. They know when you are lying or bragging!
If you do not answer a question and it is a theoretical question just try for a minute and if not simply tell "Sir, I do not know it!"
There's no harm in it! Be true and yourself!
Do not lie on your resume also! Many a time interviewer checks your GitHub repos and asks questions on it.
You have your lesson now.

There is always the next Company!

You have tried your best, and now the ball is in someone else's court! Believe in yourself, belief in your preparation, and leave the decision in God's hand. If you make it great congratulations! But if you could not, it's okay!
I was rejected by 4 companies before landing on Amazon. So just move ahead and take this as a learning experience.

12 LANDED THE JOB! WHAT'S NEXT?

First of all, Congratulations you deserved this! You have worked hard for this and you deserve the job. Don't kill yourself if you couldn't that FAANG job getting a job in a product-based company is also a big achievement and slowly and steadily you'll reach your goal. Just keep Hustling!

If you are in college and have some 6-7 months before the job starts well you can use it to get the job ready for your new job! You might be already having those skills but software development jobs are a lot different than what we study in colleges and our engineering. I will try to highlight a few skills and technologies on which you can brush up on before starting your job. And if you don't have a big gap then I'll suggest enjoying the vacation!

Technical Skills you can work on

1. **Cloud Computing:**
   In recent times most of the infrastructure is managed via Cloud Computing and it is very much likely that you'll have to work on some or another cloud computing technology. Thus, knowing how it works will surely help you out. Few important things that you can check out our EC2, S3, Lambda function, CDK, CloudWatch, CloudFormation,

IAM roles, etc. (I have used AWS services as a reference but you can use other cloud providers also)

2. **Design Patterns and OOPs:**
   When you work in a large company most of the code is written using Design Patterns these can be custom or the most common design pattern we use. Having practical knowledge about them will surely help you out!

3. **JAVA and Spring**
   Mostly you'll have to work with Java and Spring when you work in large organizations. But the thing is Java is much more advanced from what we study in college and for our interviews, and Spring is again a few steps ahead of it! So, it can be a bit difficult for a newbie to get used to this tech stack. Especially with Spring, I faced a lot of issues. I would suggest that study Spring if you have the time in your hand!

These are a few things you can work on in your free time. There are many open-source programs such as MLH Fellowship, GSOC, etc. where you can apply and g hands-on experience of working with large tech companies!

Also do not forget to enjoy your life!
Software Development is fun, but it can be hectic too sometimes so try to inculcate healthy habits such as exercising, having a healthy diet, reading, etc. Habits that will help you stay healthy and happy even after you start your job!

We have covered almost all the important aspects that you should cover if you are planning to get a software development job. This book guides and you can leverage from it only if you put it into practice otherwise it is just a collection of papers! So, get in action start working hard and your dream job is just a few months of effort away.

Also do not get dejected after getting rejected take this process as a learning process. Learn from your mistakes. Try to find what you can improve from this failed attempt. You'll surely improve in the next attempt!

If you have put in the efforts, you'll get a suitable return, when I don't know but the fruits of your hard work will come so don't give up! All the best for your hustle!

ABOUT THE AUTHOR

I'm from a small-town Nashik, Maharashtra in India. I have graduated from Veermata Jijabai Technological Institute (VJTI), Mumbai with a degree in Information Technology. I have interned at Amazon India and currently working as a Software Development Engineer at Amazon India.

Apart from my job I also make YouTube videos regularly, write blogs, and very much active on LinkedIn. The reason for doing all of this is to help the community who has the skills but lacks guidance! And my goal is to see the upliftment of this community!

ABOUT THE AUTHOR

Currently working as a Software Developer at Amazon Development Centre India. I have also interned at Amazon India. Apart from this, I was a Microsoft Student Partner and also run a YouTube Channel where I post Content about Software Development and Life! I'm very much active on LinkedIn and Twitter and like to share my journey and learnings.

The Reason Why I'm doing all this? When I received my job offer I realized that many students have the skills but What they can't find is proper guidance and that's what I'm trying to do!

Trying to give as much guidance, mentorship, and help to those who need it!