ABC: Always Be Coding

How to Land an Engineering Job.

Be honest. Are you a good engineering candidate? How are you measuring yourself? How many companies have you interviewed at? What is your onsite-interview to offer ratio? Try the following formula (that I've totally made up in a vacuum and ultimately means nothing):

```
# x = number of companies interviewed with onsite
# y = number of offers received
value = 100 * ln(x) * y / x
```

If your value is < 90, you should read this. If it's > 120, then you probably don't need this, but should read it anyway.

Who am I?

I don't have a college degree. I started programming professionally at the age

of 19 after leaving Chicago for Southern California. Everything I owned fit in my car; I had $400 in my pocket and a job offer as a junior programmer for a cool $40,000 a year. That was 12 years ago. But that's another story.

Since then I've worked at Double Helix, Namco Bandai, Google, Obvious and Square.* I've also received offers from companies such as Naughty Dog, Activision, Riot Games, Blizzard, Pinterest, Goldman Sachs, and more. For what it's worth, my score from the above formula is 132.

I've interviewed at least 500 engineering candidates. Roughly 10% were given offers. Less than 3% I considered "rockstar" candidates, and I remember all of them.

I will tell you that there is absolutely no sure-fire way to getting hired. There are too many variables, especially at a company like Google where you are placed with 5-7 random software engineers and it's up to them to come up with an appropriate set of questions to ask, usually involving whiteboard coding. Some engineers are terrible interviewers, they ask unfair questions and create snap judgments. But that's OK, it happens to the best of us. You're generally allowed to flub a single interview in a panel.

The best I can do is tell you how you can be adequately prepared. So without further ado, here are the tips I can give you.

Technical Tips

ABC (Always Be Coding). The more you code, the better you'll get—it's that simple. By coding, you're practicing. But the best practice is *focused practice*. Have goals in mind, explore new areas, and challenge yourself. Over time, you should develop a portfolio of both unfinished and finished projects. GitHub is a great place to put this portfolio on display, but just having an eclectic body of work is huge.

Master at least one multi-paradigm language. Mastering a language gives you a great sense of perspective. To do this, you must write a lot of code, read a lot more, and learn the gotchas and best practices. Ideally the language has a vibrant community, runs a lot of production code and is reasonably en vogue. Some good candidates are C#, C++, Java, PHP, Python, and Ruby.

There's a famous leading question that C++ interviewers like to ask other C++ programming candidates, *"On a scale of 1-10, 10 being the highest, how would you rank your knowledge of C++?"* I hate this question. And god help anyone who answers a 9-10, because the claws will come out. Bjarne Stroustrap himself would rate himself probably an 8 or less. The language is simply too complex, too rich, and has evolved too much over time. I digress.

Know thy complexities. Read this cheat sheet. Then make certain you understand how they work. Then implement common computational algorithms such as Dijkstra's, Floyd-Warshall, Traveling Salesman, A*, bloom filter, breadth-first iterative search, binary search, k-way merge, bubble/selection/insertion sort, in-place quick sort, bucket/radix sort, closest pair and so on. Again, ABC. This article is also a good, thorough primer.

Re-invent the wheel. You should implement the most common data structures in your language of choice. Do not rely on common libraries. Implement the following and write tests for them: vector (dynamic array), linked list, stack, queue, circular queue, hash map, set, priority queue, binary search tree, etc. You should be able to implement them quickly.

Solve word problems. Forget queries like this. It all comes down to fundamental programming concepts. Spend at least 40 hours coding solutions to different types of problems. One of the best resources is TopCoder. Read this. Then try solving problems. Pick those that test your ability to implement recursive, pattern-matching, greedy, dynamic programming, and graph problems. Just go through a bunch of archived problems.
This is probably the number one reason I was hired at Google. I spent literally two weeks obsessed with TopCoder. After that, I could code Dijkstra's algorithm with my eyes closed and one arm tied behind my back. I could solve almost any kind of graph-problem under the sun. It was all problem-solving repetition. And as Eric Schmidt says, "Repetition doesn't spoil the prayer."

Make coding easy. At least, make it look easy. Over time, I've learned that programming is the most straightforward and simple part of being an engineer. I often use the phrase "a simple matter of programming" because I believe the harder parts of being an engineer is before and after

most of the coding takes place. For example, designing what you're about to code and ensuring what you've already coded is shippable and production ready. Make your interviewer understand that you know that programming is just a means to an end.
Note, coding in front of others can be daunting. Find a way to practice both white-boarding and pair-programming. Google is basically all about coding at a whiteboard, whereas Square is effectively all pair-programming at a real machine with your language and IDE of choice. Read this article from my friend and former colleague Dan.

General Tips

I can't claim to be an expert here. In fact, some would say that I'm not even very good with people. But I should probably speak to some non-technical tips, many of which are probably quite obvious.

Know why you're there. If you're interviewing at a company and you don't fully understand why they exist, who they are, or what they do; then don't do it. Engineers who care about the hires they make will smell it a mile away. You may be able to get away with this at bigger corporations, but it won't fly at smaller ones.

Be passionate. If you don't care, then nobody else will. Be passionate about something. It might be programming, but what about it? Do you enjoy building compilers in your spare time? Do you build and fly RC helicopters? It doesn't really matter because if you're passionate about it, then you can make it interesting.

Don't make assumptions. Ask questions if you're not sure. If you're asked a question and you aren't 100% certain what the problem is, then ask. There are a number of times where I've seen a candidate go down some path, never ask a question and ultimately waste time solving the wrong problem.

Smile. Be excited, happy and positive. But don't overdo it. As I mentioned before, people will make snap judgments. Make sure your first impression is a good one. Smiles are infectious, I've often walked into an interview in a bad mood or feeling overwhelmed with other priorities, but a well-placed smile from a candidate quickly snapped me out of it.

As I said before, there's no silver bullet to getting hired. But, as an engineer, the best thing you can do is to **ABC: Always Be Coding**.

Love or hate this article? Let me know @davidbyttow.

. . .

*Today, I'm building a product for teams called, **Bold** (https://bold.io). If you work at a company, you should join the waitlist because more details and beta is coming soon!

http://bold.io