How To Develop Your Problem Solving Skills

To become a better software engineer

Omar Rabbolini Follow
Aug 10, 2019 · 6 min read ★

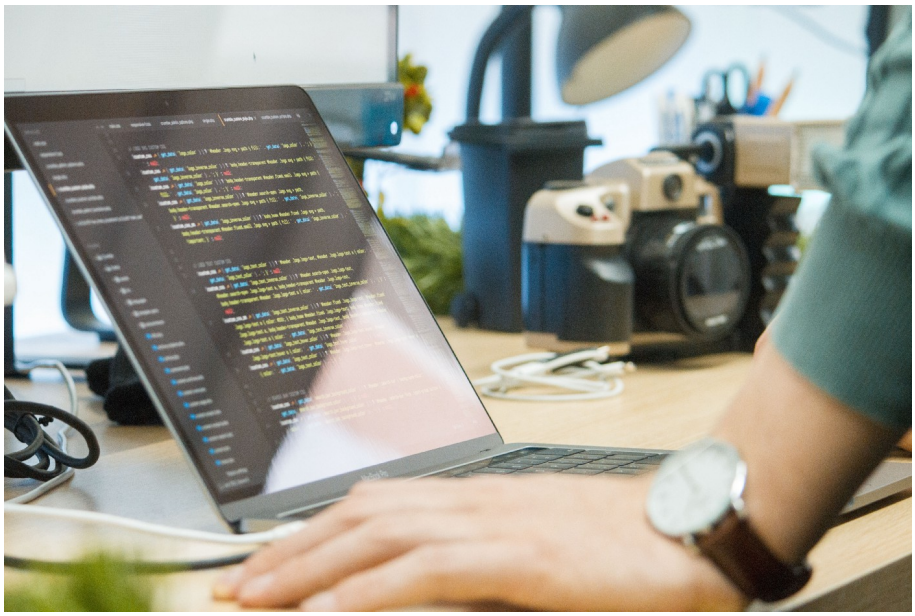

Photo by Fatos Bytyqi on Unsplash

Many, if not all, software engineering job adverts commonly found online list "*good problem solving skills*" as a requirement, and quite rightfully so. Problem solving skills are one of the key traits of great software engineers, and one that should be developed early in your professional life to pave the way to a successful career in the field.

Before we look at how to get better at solving problems, let's take a look at how *Junior Engineer Me* used to do it many years ago.

To put it simply: I sucked.

. . .

In the early 2000s, I got a job at a software house in London, working on a web-based report generation system. The engineering team was quite small, less than a dozen people in total, and the web team was even smaller, with just two of us covering both back-end and front-end development.

Both my teammate and I were quite junior at the time. I had been working in web development for a couple of years at most, while Matt had graduated just the year before with a bachelor degree in Mathematics.

One day, Matt sought my advice as he was trying to write a component to render a tree in the browser from information stored in a database. He had been going around in circles for a couple of hours already, and could not find an efficient way to translate the tabular data we had into a good looking tree.

I quickly wrote a recursive function that would hit the database every time a node was visited. We tried it, and it worked. Matt was amazed at the speed I came up with the solution. Modern day me is horrified at the code that I wrote.

. . .

Was my solution to the problem an example of good problem solving? No, not at all.

The process of problem solving

Photo by Kaleidico on Unsplash

The process of problem solving is traditionally divided into four steps:

- **Understand** the problem
- **Devise** the plan
- **Implement** the solution
- **Verify** the results

Furthermore, the step of ***devising*** the plan can be further divided into two distinct phases:

- **Research** for an existing solution
- **Adapt** it to our specific case

The ***Research*** phase identifies preexisting solutions for problems similar to the one under examination and considers the pros and cons of each. Nowadays, this can be performed quite easily thanks to the slew of online resources available for this purpose (StackOverflow, Quora, Arxiv, etc.), somewhat shortening the time required to come up with a decent number of alternatives to solve the problem.

The ***Adapt*** phase looks at optimizing the solution found so that it better fits our own problem. Sometimes, especially in the case where the research phase yielded poor results, this phase is more ***creative,*** and we might have to come up with a novel technique to satisfy our requirements.

If we look at the story above, although the problem was simple and therefore the ***understanding*** of it was trivial, the mistake I made was not to do any ***research***, and instead go for the first thing that came to mind. This meant creating code that worked at the unit-test level, but was destined to fail under load, which is what happened in practice.

Top highlight

Therefore, the first take-away for improving one's problem solving skills would be:

> Take your time to understand the problem and to research available solutions

The more you practice, the quicker you can do your research, but in any case you should take as much time as needed to ensure you understand both the problem and the proposed solution, along with its strengths and weaknesses.

I am going to say it again: **time is important**. This is why I hate giving brain teasers during engineering interviews: you can't expect somebody to come up with, let's say, a shortest path algorithm at point blank. Not everybody is Dijkstra. At most, brain teasers can give an indication of performance and

creativity under stress, but do not offer any insight into any problem solving ability.

Creativity and confidence

Photo by Alice Achterhof on Unsplash

Aside from rushing through the research phase, I found that most engineers (including myself) get stuck in the ***adaptation*** phase. There are two opposite cases here which frequently surface, especially when coming up with a completely new solution:

- **Lack of self-confidence** : You actually have a valid solution, but you think it is not good enough
- **Overconfidence** : You quickly find a solution, and you think it is the best invention since sliced bread

Although opposite, these behaviors both stem from the same problem: ***subjective bias***, or in other words: you are looking at your solution from a non-objective standpoint.

Hence, the second take-away tip for improving one's problem solving skills would be:

> When creating or adapting a solution, consider its merits objectively

How do we do this? The best way is to ***implement*** and ***verify***, as we saw earlier. However, we often do not have such luxury in software engineering as the implementation of each possible solution can be costly and

excessively time consuming.

What we can do instead is to create a ***spike*** and analyze its performance (a spike is the implementation of a mock-up / simplified version of the solution to explore its benefits). If this is not feasible, we can ask for opinions from other engineers to offset our own bias, bearing in mind that other engineers will have their own bias too!

Implementation and Verification

Photo by Nicole Wolf on Unsplash

Let's now take a look at the final two steps of the problem solving process.

Implementation is inevitable, unless a valid solution could not be found and the project was aborted. Any improvement in this phase falls under *coding skills* and is therefore out of scope for this article.

Verification, on the other hand, is often skipped as the team moves on to the next big thing straight after implementation is completed. This is a pity. The verification step can help the engineer to build their ***gut feel*** for good solutions, tying back with the "practice" point I made earlier on the topic of research.

Therefore, our third take-away for improving problem solving skills is:

> Remember to learn from past experience

Verifying your solution does not need to be a formal step to be done immediately after implementation. One can verify the effectiveness of their

work some time after it is released, by keeping an eye on issues raised on the topic and / or any modification requested and developed.

Code review feedback is also useful here, of course while keeping in mind the *subjective bias* point we saw earlier.

More things you can do

Photo by russn_fckr on Unsplash

The three tips we looked at so far are strictly related to the process of solving problems. There are many more things you can do to improve your problem solving skills, here are a few ideas:

Expand your horizons

You do not need to limit yourself to solving problems at work. Consider joining an open source project and see how you can help there. This will also allow you to learn from other engineers outside your immediate circle, and get exposure to a wider range of problems.

Challenge yourself

Although I do not like brain teasers during interviews, they are useful as a sort of gym for the mind when worked on in one's own time. You can challenge yourself through coding exercises by using sites such as Coderbyte and HackerRank.

Build something new

Start from scratch, build a new app and open source your code. You can do this alone or in a group with other engineers, and grow together. Opening the source code can invite other engineers to comment on your decisions, further improving your problem solving skills.

If you are out of ideas on apps to build, check out this list from Codementor for some inspiration.

. . .

That's it for today. I hope you enjoyed reading this article and that these few tips help you to up your problem solving game. Feel free to get in touch to let me know how you get on.

Until then, keep on making good software!