A defence for the coding challange

May 6, 2022

Let's talk about code challenges. Code challenges are a topic with many opinions and something I have been unsure if I liked or hated. Still, I would like to make a case for why I think there are situations where this practise is beneficial, not only for the interviewer but the candidate as well.

But before getting that far, I would like to point out some of the downsides to code challenges because it isn't one-size-fits-all, and you may want to steer completely clear of them or only use them in specific circumstances.

Downside 1 The primary issue with coding challenges is that they may be built in a way that prevents the candidate from showing their strength. I have, for instance, often seen those logic-style code challenges being applied to all development positions, so a front-end developer would be quizzed on his ability to solve sorting algorithms. What he would be supposed to do after being hired was to align stuff correctly with CSS. This skill test, which ultimately assesses an entirely different set of skills than what is needed, will alienate the candidate and allow a candidate with skills in the quizzed topic to overshine one with the basic skills required.

Later I will talk a bit about some requirements that I think need to be considered in a good code test, so if used, at least would give a better indication of a candidate's skill concerning the specific role, not just as a "guy who does computer stuff".

Downside 2 The second one is one I have mentioned before, but in a competitive hiring market, being the company with the most prolonged hiring process means that you might very well miss out on some of the best candidates due to them not having the available time to complete these tasks in their spare time, or because another company was able to close the hire quicker.

0.1 Why you may want to use code challenges

Unfortunately, many people don't perform well in interviews. Without a technical assessment, the only place for a candidate to showcase their skills is in the interview itself.

The IT space has historically been associated with an introvert stereotype. While not always the case, they are definitely out there, and there is nothing wrong with that, but they are usually not the strongest at selling themself, and that is basically what most job interviews are. So if we give a candidate only the ability to showcase their skills through an interview, it stands to reason that the guy we end out hiring isn't necessarily the strongest candidate for the job but the best at showcasing hers/his skill.

Using a code challenge alongside the interview allows you to use the interview part to assess the person, get an idea about how they would interact on the team, have time to explain to them what the job would be like, without having the "hidden" agenda, of trying to trip them up with random technical questions, to try to see if they can answer correctly on the spot.

So instead of the on the spot question style, the candidate would get the time to seek information and solve the tasks more reminiscent of how they would work in the real world.

Additionally, if done right, the code challenge can also help the company/team prepare for the new candidate after the hire. For example, suppose your code challenge can indicate the candidate's strengths, weaknesses and knowledge level with various technologies. This can help put the "training"-program together to support the new hire to be up and running and comfortable in the position as quickly as possible.

What makes a good code challenge It isn't easy to answer, as it would vary from position to position, team to team, and company to company. Some jobs may require a specific knowledge set, where the "implement a sorting algorithm" may be the proper test and be something you would expect any candidate to be able to.

But here are a few questions I would use to evaluate the value of a code challenge:

- Does it cover all the areas you are interested in in a candidate? This is not to evaluate if the candidate has ALL skills but rather to see if he has some skills which would add value to a team. For instance, if the role is for a front-end team that does both the front-end development, back-end for front-end, QA, DevOps, etc., the test should allow a candidate to showcase skills. If, for instance, your test is too heavily focused on one aspect, let's say front-end development, you may miss a candidate that could have elevated the entire team's ability at QA.
- Does it allow for flexible timeframes? Some candidates may not have time to spend 20 hours completing your code challenge, and the test should respect that. So if you have a lot of different tasks, as in the example above, you shouldn't expect the candidate to complete all, even if he has the time. Instead, make a suggested time frame, and give the candidate the possibility of picking particular focus areas to complete. That way, you respect their time, and you also allow them to showcase the skills they feel they are strongest at.

Another bonus thing to add is to give the candidate the ability to submit additional considerations and caveats to their solution. For example, a candidate may have chosen a particular path because the "right" approach wasn't clear from the context, have made suboptimal solutions to keep within the timeframe, or even skipped parts because of scope but still want to elaborate. This way, you get closer to the complete picture, not just the code-to-repo.