Interviewing in Ten Minutes

Spencer Tipping

November 23, 2014

Contents

About this guide

Over the past few years I've worked for a lot of startups. (Because I job-hopped, not because I'm important.) For one reason or another, I ended up conducting technical interviews at about half of these companies. Usually we were looking for developers who knew some high-end functional language, so this guide is strongly biased by that – though I think it also applies to more standard startup jobs.

Everything in this guide is my own opinion. It does not reflect the views of any of my employers, past or present. It also doesn't always reflect my own views of what is ideal or ethical, just what the situational economics dictate. I assume you have a strong system of ethics and understand the implications of the Nuremberg trials. (If you don't, then stop reading now and fix it.)

1 The business protocol

For various reasons, partly tradition and partly expedience, companies tend to have a set of norms that dictate how people work together. From what I've been able to observe, the following commandments encode the business protocol:

1. **Thou shalt respect other people's time like thine own.** This is absolutely crucial. Business is about money, people convert money to time, and if you waste/take an employee's time then you're wasting the company's money. People holding the money to pay your salary want to see a positive return on their investment in you, which is undermined if you then make other people less productive. **Be the person who makes others more productive.**

2. **Thou shalt get things done.** By this I mean, if someone gives you a task it will be done. They don't have to think twice about it. If you hit difficulty, you'll figure it out. If you can prove it's impossible, change the problem to be possible and explain why you did. **Be the person who gives up last.**

3. **Thou shalt work only when it is productive.** Work $\neq$ productivity. The two are nearly orthogonal, in fact. Understand what is productive at as high a level as you can, and work on that stuff. Busywork is worse than doing nothing because you'll be unavailable but will be getting nothing done. **Be the person who does everything for a reason.**

4. **Thou shalt simplify.** This is a way to respect people's time. Every problem that comes your way should be simpler after it interacts with you. Make things look easy because to you they are easy. **Be the person who finds the best solution.**

5. **Thou shalt be fluent.** This goes two ways: you must communicate flawlessly both with people and with computers. Functionally, a programmer is an information channel that translates between human intent and computational action. You want this channel to be efficient, lossless, and high-capacity. **Be the best conduit of intent from people to machines.**

6. **Thou shalt not bikeshed.** Thou shalt not argue in general, in fact. But if you must, make sure it's about an issue that justifies a conversation costing the company more than $100/hour (nearly every conversation is likely to cost this, and some will cost far more). **Be the person who cuts through the noise and focuses on real problems.**

I'll go into more details about how to convey these things during an interview later on. For now, though, understand that the above values (or similar ones) serve as the conduit by which your skills can be applied to the company's problems. If this conduit is missing or sufficiently damaged, your skills won't matter.

2 The interviewer

Interviewers are not usually sadists. While the case could be made that the job demands a certain degree of malevolence, most of the confrontation during the interview is due to the economics underlying hiring.

Consider the common flow of resume screen, phone screen, and then in-person interviews. There's a progression of risk/cost associated with moving forward at each stage: screening a resume requires minutes, a phone interview requires an hour, and an in-person interview requires several people-hours, possibly with hotel and airfare. In dollar amounts, these could easily represent investments of $10, $100, and $1000. The worst case, a bad hire, costs potentially $10,000 or more.

Given this, the interviewer operates as an investor of the company's assets.[1] Their job is to evaluate risks for the company by making bets on people. Because of this, if you are an interviewee, your job is to sell such a bet to the person interviewing you. Perhaps more to the point, the interviewer is likely to be conservative because they don't get paid commission; so your goal is to be so awesome that they feel safer saying yes than no.

3 The résumé and cover letter

At $100, a phone screen is still fairly cheap. The stakes of a resume and cover letter are not usually very high, nor is there very much signal for the interviewer to work with. However, if you're about optimizing stuff (and what engineer isn't?), there are a few things you can do to communicate at this point:

1. TEX your resume. There's only upside here, and it's a nice positive signal that you know what you're doing. If you don't know TEX, use HTML and export as PDF. If all else fails, write one in plain text and claim that minimalism is a social virtue you take seriously. Don't use Word because it conveys that you have no standards. **Use a spellchecker.**

2. Keep your resume down to one page. This shows that you respect their time. **Use a spellchecker.**

3. Write a purposeful but informal cover letter. Be personable, reasonably enthusiastic, and talk about how the stuff you do for fun is relevant to the company's hard problems. **Use a spellchecker.**

4. Link to your github profile. If you don't have one yet, create one and upload something. If you don't have anything to upload, start solving Project Euler and upload that. **Use a spellchecker.**

5. Put some code in your cover letter. Nothing elaborate, but just something you thought was cool, or maybe a challenge problem they have on their website, or something. Do this only if it makes sense, but it's a great opportunity if you can. **Use a spellchecker.**

[1]Unlike most investors, however, their pay is constant; so they don't have a stake in the outcome of the interview. This creates a slight negative bias, but as far as I know it shouldn't influence their decisions otherwise.

6. Link to a project you worked on that really describes who you are. **Use a spellchecker.**

There are a few things you absolutely cannot do:

1. Talk about salary in any way. It's just too early and it sends the wrong message. It would be like someone asking about your finances before the first date. This is a question you ask when you're considering marriage, not dinner.
2. Lie. If you're called out on it, you're done for and people will hate you. Word might get around to other companies if you're unlucky. If the truth isn't good enough, fix the truth.
3. Be noticeably unprofessional. Startups aren't overwhelmingly formal places, but they're just as vulnerable to harrassment lawsuits as anyone else. You do not want to give anyone reason to believe that you're going to cause legal problems or alienate people.
4. Bring politics or religion to the table. These topics are emotional catalysts with no upside and enormous potential for problems. Brendan Eich, the inventor of Javascript, was forced to resign as CEO of Mozilla because he had made a $1000 contribution against gay marriage six years prior. If you have public political statements on a blog or elsewhere, take them down.

4 The technical phone interview

At this point you have cost the company $110 or so, and they're deciding whether or not to bump that figure by a factor of 10. This is real money, so the burden is on you to convince the interviewer (who may or may not be good at interviewing people) that you're the best bet they'll make this year. Depending on the job you're applying for, there are a few different ways to do that.

In this section I'm going to assume the technical interview is just talking on the phone. The next section is entirely devoted to how to handle a collaborative coding interview.

4.1 Technical literacy

This is an absolute must. At a minimum, you need to be able to answer questions like these with little or no thought:

- Write the FizzBuzz function.
- Reverse an array in place.
- What is the insertion time for a binary tree?

- What is the insertion time for a hashtable (worst, average)?
- What is memoization?
- Write the Fibonacci function without using recursion.
- What's the difference between the stack and the heap?
- Conceptually, what does a garbage collector do?
- Conceptually, how does virtual memory work?
- What happens under the hood when you go to `google.com`?
- Why should you profile before optimizing?

4.2 Development literacy

This is also important, especially if you're applying for a position that requires experience. You should be able to speak intelligently about things like:

- The bugs you most commonly write, and habits you've developed to avoid them.
- How expensive it is to fix a bug in production.
- Strengths and weaknesses of unit testing.
- The drawbacks of using powerful languages like Lisp. (You can extol their virtues too, but if that's all you do then people are unlikely to take you very seriously.)
- Your preferences about design and documentation, and why you do it this way.
- A project you've worked on that went really well, and why this happened.
- A project you've worked on that was a disaster, and why. Don't try to blame other people; you need one where you messed up and learned something. If you don't have a project like this, think of something really hard, try to solve it, push the failed results to Github, and talk about why that didn't work. It's fine if failure was inevitable; the main thing is to know how to translate it into improvement.

5 Collaborative coding interview