Joel on Software

The Guerrilla Guide to Interviewing

by Joel Spolsky

Thursday, March 23, 2000

This is a very old version of an article that has since been extensively rewritten. The latest version is The Guerrilla Guide to Interviewing, Version 3.0.

This version is here for historical reasons only.

Hiring the right people is extremely crucial to Fog Creek Software. In our field, there are three types of people. At one end of the scale, there are the unwashed masses, lacking even the most basic skills for this job. They are easy to ferret out and eliminate, often just by reviewing a resume and asking two or three quick questions. At the other extreme, are the brilliant superstars who write lisp compilers for fun, in a weekend, in Assembler for the Palm Pilot. And in the middle, you have a large number of "maybes" who seem like they might just be able to contribute something. The trick is telling the difference between the superstars and the maybes, because at Fog Creek Software we only hire the superstars. Here are some techniques for doing that.

First of all, the #1 cardinal criteria for getting hired at Fog Creek:

> **Smart, and**
> **Gets Things Done.**

That's it. That's all we're looking for. Memorize that. Recite it to yourself before you go to bed every night. Our goal is to hire people with *aptitude*, not a particular skill set. Any skill set that people can bring to the job will be technologically obsolete in a couple of years,

anyway, so it's better to hire people that are going to be able to learn any new technology rather than people who happen to know SQL programming right this minute.

Smart is hard to define, but as we look at some possible interview questions we'll see how you can ferret it out. **Gets Things Done** is crucial. People who are **Smart** but don't **Get Things Done** often have PhDs and work in big companies where nobody listens to them because they are completely impractical. They would rather mull over something academic about a problem rather than ship on time. These kind of people can be identified because they love to point out the theoretical similarity between two widely divergent concepts. For example, they will say "Spreadsheets are really just a special case of programming language" and then go off for a week and write a thrilling, brilliant white paper about the theoretical computational linguistic attributes of a spreadsheet as a programming language. Smart, but not useful.

Now, people who **Get Things Done** but are not **Smart** will do stupid things, seemingly without thinking about them, and somebody else will have to come clean up their mess later. This makes them *liabilities* to the company because not only don't they contribute, but they soak up good people's time. They are the kind of people who copy big chunks of code around rather than writing a subroutine, because it gets the job done, just not in the smartest way.

The most important rule about interviewing:

Make A Decision

At the conclusion of the interview, you have to be ready to make a sharp decision about the candidate. There are only two possible outcomes to this decision: **Hire** or **No Hire**. Turn to your computer and send immediate feedback to the recruiter. The subject line should be the name of the candidate. The first line of the email should be **Hire** or **No Hire**. Then you should spend about 2 paragraphs backing up your decision.

There is no other possible answer. *Never* say, "Hire, but not in my group." This is rude and implies that the candidate is not smart enough to work with you, but maybe he's smart enough for those losers over in that other group. If you find yourself tempted to say "Hire, but not in my group," simply translate that mechanically to "No Hire" and you'll be OK. Even if you have a candidate that would be brilliant at doing 1 particular thing, but wouldn't be very good in another group, that's a **No Hire**. Things change so often and so rapidly that we need people that can succeed anywhere. If for some reason you find an idiot savant that is really, really, really good at SQL but completely incapable of ever learning any other topic, **No Hire**. They don't have a future at Fog Creek.

Never say "Maybe, I can't tell." If you can't tell, that means **No Hire**. It's really easier than you'd think. Can't tell? Just say no! Similarly, if you are *on the fence*, that means **No Hire.** Never say, "Well, Hire, I guess, but I'm a little bit concerned about..." That's a **No Hire** as well.

An important thing to remember about interviewing is this: it is much better to reject a good candidate than to accept a bad candidate. A bad candidate will cost a lot of money and effort and waste other

people's time fixing all their bugs. If you have any doubts whatsoever, **No Hire**.

While you are conducting the interview, don't worry that if you reject a lot of people, Fog Creek won't be able to find anyone to hire. That's not your problem. It's the recruiter's problem, it's H.R.'s problem, it's Joel's problem, but it's *not* your problem. Keep asking yourself which is worse - that we grow into a big, lousy software company with lots of coconuts, or that we stay small but high quality? Of course, it's important to seek out good candidates and everybody should see it as a part of their mission to find and recruit smart people who get things done. But once you're actually interviewing someone, pretend that Fog Creek has plenty of great candidates. Never lower your standards no matter how hard it seems to find great candidates.

But how do you make this difficult decision? You just have to keep asking yourself during the interview: *is this person **smart**?* Does this person ***get things done**?* In order to be able to tell, you're going to have to ask the right questions.

Just for fun, here is the worst interview question on Earth: "What's the difference between varchar and varchar2 in Oracle 8i?" This is a terrible question. There is no possible, imaginable correlation between people that know that particular piece of useless trivia and people that Fog Creek wants to hire. Who cares what the difference is? You can find out online in about 15 seconds!

Actually, there are some even worse questions. I'll get to that later.

So now we get to the fun part: interview questions. My list of interview questions comes from my first job at Microsoft. There are actually hundreds of famous Microsoft interview questions. Everybody has a set of questions that they really like. You, too, will develop a particular set of questions and a personal interviewing style which helps you make the **Hire/No Hire** decision. Here are some techniques that I have used that have been successful.

Before the interview, I read over the candidates resume and jot down an *interview plan* on a scrap of paper. That's just a list of questions that I want to ask. Here's a typical plan for interviewing a programmer:

1. **Introduction**
2. **Question about recent project candidate worked on**
3. **Impossible Question**
4. **C Function**
5. **Are you satisfied?**
6. **Design Question**
7. **The Challenge**
8. **Do you have any questions?**

Before the interview, I am very, very careful to avoid anything that might give me some preconceived notions about the candidate. If you think that someone is smart before they even walk into the room, just because they have a Ph.D. from MIT, then nothing they can say in 1 hour is going to overcome that initial prejudice. If you think they are a bozo, nothing they can say will overcome that initial impression. An interview is like a very, very delicate scale -- it's very hard to judge

someone based on a 1 hour interview and it may seem like a very close call. But if you know a little bit about the candidate beforehand, it's like a big weight on one side of the scale, and the interview is useless. Once, right before an interview, a recruiter came into my office. "You're going to *love* this guy," she said. BOY did this make me mad. What I should have said was, "well, if you're so sure I'm going to love him, why don't you just hire him instead of wasting my time going through this interview." But I was young and naïve, so I interviewed him. When he said not-so-smart things, I thought to myself, "gee, must be the exception that proves the rule." I looked at everything he said through rose-colored glasses. I wound up saying **Hire** even though he was a crappy candidate. You know what? Everybody else who interviewed him said **No Hire**. So: don't listen to recruiters; don't ask around about the person before you interview them; and never, ever talk to the other interviewers about the candidate until you've both made your decisions independently. It's the scientific method!

The *Introduction* phase of the interview is intended to put the candidate at ease. I spend about 30 seconds telling the person who I am and how the interview will work. I always reassure the candidate that we are interested in *how* he goes about solving problems, not the actual answer. By the way, in doing the interview, you should make sure that you are not sitting across a desk from the candidate. This creates a formal barrier which will not place the candidate at ease. It is better to move the desk against a wall, or to go around and sit on the other side of the desk with the candidate; this does help put the candidate at ease. This results in a better interview because it is less distorted by nervousness.

Part 2 is a question about some recent project that the candidate worked on. For interviewing college kids, ask them about their senior thesis, if they had one, or about a course they took that involved a long project that they really enjoyed. For example, sometimes I will ask, "what class did you take last semester that you liked the most? It doesn't have to be computer-related." Actually I am usually pretty happy if they choose a non-computer related course. Sometimes you look at their schedule, and it looks like they are taking the bare minimum number of Comp Sci courses, but every elective is something related to Music. Then they will tell you that their favorite course was Object Oriented Databases. Yeah, right. I'd be happier if they admitted that they just liked music more than computers, instead of sucking up.

When interviewing experienced candidates, you can talk about their previous job.

In this question, I'm looking for one thing: *passion*. When you find a project that the person worked on recently, these are all good signs:

- They get very excited talking about it; they tend to talk more quickly and get animated. This shows that when they are interesting in something, they will be passionate about it. There are far too many people around that can work on something and not really care one way or the other. Even if they are passionately negative, this can be just as good a sign. "I was working on installing Foo Bar Mark II for my previous employer, but he was such a dope!" These are good candidates

that we want to hire. Bad candidates just don't care and will not get enthusiastic at all during the interview. A really good sign that a candidate is passionate about something is that when they are talking about it, they will forget for a moment that they are in an interview. Sometimes a candidate comes in who is very nervous about being in an interview situation -- this is normal so I always overlook that. But then when you get them talking about Computational Monochromatic Art they will get extremely excited and lose all signs of nervousness. Good. I like passionate people who really care. (To see an example of Computational Monochromatic Art try unplugging your monitor.)

- They are careful to explain things. I have rejected candidates because when they talked about their previous project, they couldn't explain it in terms that a normal person could understand. Often engineering majors will just assume that everyone knows what Bates Theorem is or what Peano's Axioms are. If they start doing this, stop them for a minute and say, "could you do me a favor, just for the sake of the exercise, could you please explain this in terms my grandmother could understand." At this point many people will *still* continue to use jargon and will completely fail to make themselves understood. GONG!
- If the project was a team project, look for signs that they took a leadership role. A candidate might say: "we were working on X, but the boss said Y and the client said Z." I'll ask, "so what did *you* do?" A good answer to this might be "I got together with the other members of the team and wrote a proposal..." A bad answer might be, "Well, there was nothing I *could* do. It was an impossible situation." Remember, **Smart and Gets Things Done**. A good way to tell if somebody **Gets Things Done** is to see if historically they have tended to get things done in the past. In fact, you can even ask them directly to give you an example from their recent past when they took a leadership role and got something done -- overcame some institutional inertia, for example.

OK, the third thing on that list is the *impossible question*. This is fun. The idea is to ask a question that they have no possible way of answering, just to see how they handle it. "How many optometrists are there in Seattle?" "How many tons does the Washington Monument weigh?" "How many gas stations are in Los Angeles?" "How many piano tuners are there in New York?"

- Smart candidates will realize that you are not quizzing them on their knowledge, and they will enthusiastically leap into trying to figure out some back-of-the-envelope answer. "Well, lets see, the population of LA is about 7 million; each person in LA has about 2.5 cars..." Of course it's OK if they are radically wrong. The important thing is that they leapt into the question enthusiastically. They may try to figure out the capacity of a gas station. "Gee, it takes 4 minutes to tank up, gas stations have about 10 pumps and are open about 18 hours a day..." They may try to figure it out by area. Sometimes they will surprise you with their creativity or ask for a Los Angeles yellow pages. All good signs.
- Not-so-smart candidates will get flustered and upset. They will just stare at you like you landed from Mars. You have to coach

them. "Well, if you were building a new city the size of Los Angeles, how many gas stations would you put in it?" You can give them little hints. "How long does it take to fill up a tank of gas?" Still, with not-smart candidates, you will have to drag them along while they sit there stupidly and wait for you to rescue them. These people are not problem solvers and we don't want them working for us.

For programming questions, I ask candidates to write a small function in C. Here are some typical problems I would ask:

1. Reverse a string in place
2. Reverse a linked list
3. Count all the bits that are on in a byte
4. Binary search
5. Find the longest run in a string
6. atoi
7. itoa (great, because they have to use a stack or strrev)

You don't want to give them any problems that take more than about 5 lines of code; you won't have time for that.

Let's look at a couple of these in detail. #1: reverse a string in place. Every candidate I've ever interviewed in my life has done this wrong the first time. Without exception, they try to allocate another buffer and reverse the string into that buffer. The trouble is, who allocates the buffer? Who frees the buffer? In giving this question to dozens of candidates I found out an interesting fact. Most people who think that they know C really do not understand memory or pointers. They just don't get it. It's amazing that these people are working as programmers, but they are. With this question, here are some ways to judge the candidate:

- Is their function fast? Look at how many times they call *strlen*. I've seen O(n^2) algorithms for strrev when it should be O(n), because they are calling *strlen* again and again in a loop.
- Do they use pointer arithmetic? This is a good sign. Many "C programmers" just don't know how to make pointer arithmetic work. Now, ordinarily, I wouldn't reject a candidate just because he lacked a particular skill. However, I've discovered that understanding pointers in C is not a skill, it's an aptitude. In Freshman year CompSci, there are always about 200 kids at the beginning of the semester, all of whom wrote complex adventure games in BASIC for their Atari 800s when they were 4 years old. They are having a good ol'; time learning Pascal in college, until one day their professor introduces pointers, and suddenly, *they don't get it*. They just don't understand anything any more. 90% of the class goes off and becomes PoliSci majors, then they tell their friends that there weren't enough good looking members of the appropriate sex in their CompSci classes, that's why they switched. **For some reason most people seem to be born without the part of the brain that understands pointers.** This is an aptitude thing, not a skill thing – it requires a complex form of doubly-indirected thinking that some people just can't do.

For #3, you can see how well they learned the bitwise operators in C.... but this *is* a skill, not an aptitude, so you can help them with

these. The interesting thing is to watch them write a subroutine that counts all the bits in a byte, then ask them to make it much, much faster. Really smart candidates will create a lookup table (after all, it's only got 256 entries) that they only have to create once. With good candidates, you can have a really interesting conversation about the different space/speed tradeoffs. Press them further: tell them you don't want to spend any time building the lookup table during initialization. Brilliant candidates might even suggest a caching scheme where bits are counted the first time they are used, and then stored in a lookup table so they don't have to be counted if they are used again. Really, really brilliant candidates will try to devise a way to compute the table using some kind of a shortcut taking advantage of the patterns that occur.

When you watch somebody write code, here are some techniques that may be helpful:

- Always reassure them that you understand that it's hard to write code without an editor, and you will forgive them if their paper gets really messy. Also you understand that it's hard to write bug-free code without a compiler, and you will take that into account.
- Some signs of a good programmer: good programmers have a habit of writing their { and then skipping down to the bottom of the page and writing their }s right away, then filling in the blank later. They also tend to have some kind of a variable naming convention, primitive though it may be... Good programmers tend to use really short variable names for loop indices. If they name their loop index CurrentPagePositionLoopCounter it is sure sign that they have not written a lot of code in their life. Occasionally, you will see a C programmer write something like **if (0==strlen(x))**, putting the *constant* on the left hand side of the == . This is a really good sign. It means that they were stung once too many times by confusing = and == and have forced themselves to learn a new habit to avoid that trap.
- Good programmers plan before they write code, especially when there are pointers involved. For example, if you ask them to reverse a linked list, good candidates will always make a little drawing on the side and draw all the pointers and where they go. They have to. It is humanly impossible to write code to reverse a linked list without drawing little boxes with arrows between them. Bad programmers will start writing code right away.

Inevitably, you will see a bug in their function. So we come to question 5: **Are you satisfied with that code?** You may want to ask, "OK, so where's the bug?" The quintessential Open Ended Question From Hell. All programmers make mistakes, there's nothing wrong with that, they just have to be able to find them. With the string functions, they'll almost always forget to null-terminate the new string. With almost any function, they are likely to have off-by-one errors. They will forget semicolons sometimes. Their function won't work correctly on 0 length strings, or it will GPF if malloc fails... Very, very rarely, you will find a candidate that doesn't have any bugs the first time. In this case, this question is even more fun. When you say, "There's a bug in that code," they will review their code carefully, and then you get to see if they can be diplomatic yet firm in asserting that the code is perfect... In general, it's always a

good idea to ask the candidate if they are satisfied with their answer before moving on. Be Regis.

Part 6: the design question. Ask the candidate to design something. Jabe Blumenthal, the original designer of Excel, liked to ask candidates to design a house. According to Jabe, he's had candidates who would go up to the whiteboard and immediately draw a square. A square! These were immediate **No Hires**. In design questions, what are you looking for?

- Good candidates will try to get more information out of you about the problem. Who is the house for? As a policy, I will not hire someone who leaps into the design without asking more about who it's for. Often I am so annoyed that I will give them a hard time by interrupting them in the middle and saying, "actually, you forgot to ask this, but this is a house for a family of 48-foot tall blind giraffes."
- Not-so-smart candidates think that design is like painting: you get a blank slate, and you can do whatever you want. Smart candidates understand that design is a difficult series of trade-offs. A great design question: design a trash can for a city street corner. Think of all the trade offs! It has to be easy to empty, but impossible to steal; it has to be easy to put things into, but hard for things to fly out of on a windy day; it has to be solid, yet inexpensive; in some cities, it has to be specially designed so that terrorists can't hide a bomb in it.
- Creative candidates will often surprise you with an interesting, non-obvious answer. One of my favorite questions is *Design a Spice Rack for Blind People*. Inevitably, candidates will put Braille somewhere on the spice bottles, and it usually winds up being on top of the lid for various reasons which you'll discover after you've asked this question 100 times. I had one candidate who decided that it would be better to put the spices in a drawer, because it is more comfortable to scan Braille with your fingertips horizontal than vertical. (Try it!) This was so creative it surprised me -- in dozens of interviews, I had never heard that answer. And it really took a major creative "leap" outside of the bounds of the problem. On the strength of that answer alone, and no negatives, I hired the candidate, who went on to be one of the best program managers on the Excel team.
- Look for *closure*. This is part of **Get Things Done**. Sometimes candidates will drift back and forth, unable to make a decision, or they will try to avoid hard questions. Sometimes they will leave difficult decisions unanswered and try to move on. Not good. Good candidates have a tendency to try to naturally keep things moving forward, even when you try to hold them back. If the conversation ever starts going around in circles, and the candidate says something like "well, we can talk about this all day, but we've got to do something, so let's go with decision X" that's a really good sign.

Which brings us to #7, **The Challenge**. This is fun. Throughout the interview, you look for the candidate to say something that is absolutely, positively, unarguably correct. Then you say, "wait a minute, wait a minute," and spend about 2 minutes playing devil's advocate. Argue with them *when you are sure they are right*.

- Weak candidates will give in. **No Hire**.

- Strong candidates will find a way to persuade you. They will have a whole laundry list of Dale Carnegie techniques to win you over. "Perhaps I'm misunderstanding you," they will say. But they will stand their ground. **Hire**.

Admittedly, in an interview situation, you are not equal parties. Thus there is a risk that the candidate will be afraid to argue with you because you are in a position of power over him. **BUT**, good candidates will tend to get fairly passionate about the argument, and they may momentarily forget that they are in an interview, and they will get very involved in trying to convince you. These are the people we want to hire.

Finally, you should ask the candidate if they have any questions. Some people like to look to see if the candidate will ask intelligent questions, which is a standard technique in the interviewing books. Personally, I don't care what questions they ask; by this point I've already made my decision. The trouble is, candidates have to see about 5-6 people in one day, and it's hard for them to ask 5-6 people different, brilliant questions, so if they don't have any questions, fine.

I always, always leave about 5 minutes a the end of the interview to sell Fog Creek. This is very important *even if you are not going to hire them*. If you've been lucky enough to find a really good candidate, you want to do everything you can at this point to make sure that they want to come to Fog Creek. Even if they are a bad candidate, you want to get them excited about Fog Creek Software so that they go away with a positive impression of the company. Think of it this way: these people are not just potential hires; they are also customers. They are also salesmen for our recruiting effort: if they think that Fog Creek is a great place to work, they will encourage their friends to apply.

Ah, I just remembered that I promised to give you some more examples of really bad questions to avoid.

First of all, avoid the illegal questions. Anything related to race, religion, gender, national origin, age, military service eligibility, veteran status, sexual orientation, or physical handicap is just **illegal**. If their resume says they were in the Army in 1990, don't ask them, even to make pleasant conversation, if they were in the Gulf war. It's against the law. If their resume says that they attended the Technion in Haifa, don't ask them, even conversationally, if they are Israeli. It's against the law. There's a pretty good discussion of what's illegal here. (But the rest of the interview questions at that site are pretty stupid).

Next, avoid any questions which might make it seem like we care about, or are discriminating based on, things which we don't actually care about or discriminate based on. The best example of this I can think of is asking someone if they have kids or if they are married. This might give the false impression that we think that people with kids aren't going to devote enough time to their work or that they are going to run off and take maternity leave.

Finally, avoid brain teaser questions like the one where you have to arrange 6 equal length matches to make exactly 4 identical perfect triangles. If it's an "aha!" question, you don't get any information about "smart/get things done" by figuring out if they happen to make the mental leap or not.

Interviewing is more of an art than a science, but if you remember the **Smart/Gets Thing Done** principle you will be in good shape. When you get a chance, ask some of your co-workers what their favorite questions are and what kinds of answers they look for. In the Building 16 cafeteria in Redmond this is a perennial favorite topic of lunchtime conversation.

Want to know more? You're reading Joel on Software, stuffed with years and years of completely raving mad articles about software development, managing software teams, designing user interfaces, running successful software companies, and rubber duckies.

About the author. I'm Joel Spolsky, co-founder of Fog Creek Software, a New York company that proves that you can treat programmers well and still be highly profitable. Programmers get private offices, free lunch, and work 40 hours a week. Customers only pay for software if they're delighted. We make FogBugz, an enlightened bug tracker designed to help great teams develop brilliant software, Kiln, which simplifies source control and code review, and Fog Creek Copilot, which makes remote desktop control easy. I'm also the co-founder of Stack Overflow.

joel@joelonsoftware.com