acmqueue

Interviewing Techniques

Separating the good programmers from the bad

Dear KV,

My work group has just been given approval to hire four new programmers, and now all of us have to interview people, both on the phone and in person. I hate interviewing people. I never know what to ask. I've also noticed that people tend to be careless with the truth when writing their resumes. We're considering a programming test for our next round of interviewees, because we realized that some previous candidates clearly couldn't program their way out of a paper bag. There have to be tricks to speeding up hiring without compromising whom we hire.

Tired of Talking and Not Coding

Dear Tired of Talking,

My preferred model for judging prospective job candidates is the one used by the U.S. Marines: beat the candidate down until he or she has no remaining shreds of individuality or self-respect, and if the candidate still wants to work for you, hire that person because at that point you own his or her soul. Unfortunately, every time I suggest "boot-camp hiring" at the office, our legal department has a conniption fit, and so I have yet to implement and test this method. In the absence of being able to judge people's abilities under fire, you have to use less direct, subtler approaches.

The true goal of any interview is for both parties—the interviewer and interviewee—to work out whether the person can do the work and is a good fit with the rest of the group. There are many brilliant programmers out there whom I would never hire because the detrimental impact of their character defects on the rest of the team would outweigh their abilities as coders. The question to ask is, "How do I determine, in 30 to 60 minutes, if this person can do the work we need, and in a way that I can put up with him or her for 10 hours a day, five days a week, and possibly for years on end?" That's a lot to ask of such a short meeting.

Figuring out if someone has the knowledge you think is needed for a job is probably the easiest component of an interview. You don't really need to give the candidate a programming test at this point; all you need to do is to ask questions that you've recently answered at work. This assumes that the person will be doing the same kind of work you've been doing, and that's usually a safe assumption (programmers are, happily, rarely asked to interview people for the accounting department). I tend to start with basic questions first, and you should not feel that a senior person is above answering simple questions. Just because someone has a lot of experience does not excuse him or her from knowing how to work with, for example, linked lists. If a person has a lot of experience, then he or she will exhaust your trivial questions pretty quickly and you can move on to more difficult problems.

Once you've established that the person has the basic knowledge for the job, you need to figure out, at least at a high level, how he or she solves problems. At this point I find a whiteboard to be the best tool for the interview. I always make candidates for programming jobs describe, in block diagram form, a system they're familiar with. If a programmer or software engineer cannot describe

a system as a block diagram, I am pretty unlikely to hire that person. A candidate might be brilliant and might understand the system he or she worked on, but a candidate who cannot explain it to another person will be useless in any work group.

After a candidate has described a system to my satisfaction, however, I always ask the following question: “If you had had more time, what would you have changed or what feature would you have added?” This type of open-ended question is extremely important. Someone who cannot answer this question is pretty much an automaton that simply implements the will of others. I don’t like working with automatons; I like working with thinking human beings who have opinions on what they’re building and who are always thinking about how they can extend the systems they’re working on. Good programmers understand that their systems are never really complete and that there is always something else they could have done, if only they’d had more time.

While some companies use a programming test to evaluate candidates, I prefer two different ways of achieving the same thing. Many years ago I worked for a company that asked you to submit a piece of code it could compile and run. The code needed to be well documented and simple enough for someone to figure out in less than an hour. I much prefer real examples of code to asking someone to code a bubble sort on paper. With the advent of widespread participation in open source projects, this kind of test is less necessary, because you can often just search for a programmer’s name and see some code, somewhere, that the programmer has added to an open source project. However you do it, make sure to get a sample of the code, because that will tell you far more about the programmer than how he or she codes on a legal pad. If you’re the paranoid type, and you don’t trust the person not to send a friend’s code, then ask about the code during the interview. If the person is bluffing, then you’ll know it pretty quickly; and if the person did submit a friend’s code but can bluff his or her way through explaining it, then you should hire that candidate, anyway.

The other way that I like to quiz programmers on code is to provide a piece of code that’s broken in some way and ask them to find the bugs. Much of a programmer’s workday is spent debugging, and I value that ability almost as highly as I value the ability to write bug-free code. Even if the programmers, themselves, write bug-free code, a rather highly unlikely occurrence, they’re going to work on other people’s code, and it’s important that they have the ability to quickly analyze code that’s not their own.

Some interviewers like to give brainteasers, but I don’t find these to be useful, unless they relate to coding in the real world. Brainteasers fail on several levels. On one level, a person can simply memorize most of the popular ones. A quick search shows that there are more than 2,000 books on how to ace interview questions, with some books specifically about software. If candidates find out that your company is the type of company that uses brainteasers, then they’ll just brush up before the interview and quite possibly get by your interviewers.

Another reason to avoid the brainteaser approach is that there are some very good programmers who are terrible at brainteasers, so you’ll miss out on some good people because they didn’t seem smart on your tests. Let’s face it, most programmers do not spend their days looking at coding problems and trying to relate them to brainteasers. They look at problems and try to work out in code how to solve them. You’re hiring programmers, not game-show contestants.

I realize that you originally asked for a way to speed up the hiring and interview process. I’ve already made one suggestion that can speed things up: have the interviewee send you code before

coming to the interview. I suggest you do this after the phone screen but before the in-person interview, because if the person sends you code that's complete crap, you can forgo the interview.

The second thing to do to avoid wasting people's time is to evaluate, after each of your team members talks to the person, whether to send that candidate on to the next team member. While it might be embarrassing to send someone home after the first one or two interviewers have talked to the candidate, it's a hell of a lot less stressful than having the whole team go through the motions of interviewing someone, just to be nice. Just as in code, it's better to fail early.

KV

KODE VICIOUS, known to mere mortals as George V. Neville-Neil, works on networking and operating system code for fun and profit. He also teaches courses on various subjects related to programming. His areas of interest are code spelunking, operating systems, and rewriting your bad code (OK, maybe not that last one). He earned his bachelor's degree in computer science at Northeastern University in Boston, Massachusetts, and is a member of ACM, the Usenix Association, and IEEE. He is an avid bicyclist and traveler who currently lives in New York City.

© 2011 ACM 1542-7730/11/0500 \$10.00