Writing Code On Whiteboards Is Hard

Eric Lippert April 15, 2004

87

0 0 0

Work has been crazy lately and I haven't had much time to work on SimpleScript. It's a lot of work starting from scratch! Right now I'm writing a templatized hash table for the binders and other lookup tables. I haven't written templatized C++ for a loooong time and its slow going. I hope to have the named item logic done over the weekend, so that I can explain how the module system works next week.

Until then, here's an article I wrote a while back on interviewing that I never got around to putting up on the blog before now.

Writing Code On Whiteboards Is Hard

As I've mentioned before, I occasionally interview candidates for development positions on my team and occasionally other Visual Studio teams. Plenty of people have written plenty of web pages on interviewing at Microsoft, so I won't rehash the whole story here. What I wanted to mention today was some words of advice for candidates for development positions. This is by no meanscomplete — I want to concentrate on one important aspect of the interview.

Dev candidates: if you've done any reading at all, you know that most of your interviews will involve writing some code on a whiteboard. A word of advice:writing code on whiteboards is HARD. Practice!

It's hard in part because you don't have intellisense or syntax colouring or any of the other tools at your disposal that you normally do when writing code. I know that, and I'll take that into account. I don't care if you write `memset(&b, cb, 0x00)` when you mean `memset(&b, 0x00, cb)` — I don't remember that stuff either. I don't care if you forget semis or make other syntactical gaffes. I'm not looking for people who can spit out syntactically perfect code at the drop of a hat; that's not the point of the coding exercise at all. I'm trying to find out how you solve problems. Do you:

- rush right in and start coding without thinking the problem through first?
- find areas where the problem is ambiguous and clarify them, or do you make a bunch of assumptions?
- break the problem down into pieces?
- do the easy pieces, paint yourself into a corner, and attempt to handwave your way out of the situation, or work on the hard stuff first?
- have any confidence that your solution is correct?
- do anything to prove to yourself and me that it is correct?

It would be wise to make it easy for me to see that you're a well-organized person with good problem solving skills. Leave lots of space — you might need to stick in some extra lines. Write slowly and clearly. Explain what you're doing as you do it. If the code is clear, well-organized, legible, and at least vaguely syntactically correct, it will be a lot easier on both of us to tell whether the code is well designed and bug free. Walk through some simple test cases — don't just dash down some code and say "yep, that's correct!"

The vast majority of the coding problems people pose do not require any "aha!" insights or rocket-science algorithms. No one is going to ask you to implement a 4-5 Runge-Kutta differential equation solver. We're going to ask you to implement a hash table or replace every instance of "foo" in a string with "bar".Eric Carter has a copy of Schaum's Programming With C++ in his office because that thing is a gold mine for interview questions. Got a technical interview coming up? Pick up a copy of any introductory programming text, pick a few problems at random, and solve them on a whiteboard. Heck, I'll flip through it at random right now. Here are some sample problems taken from various places in the book:

- implement a method which determines how many characters the string pointer must be incremented to point to the trailing null.
- implement the less-than operator for a class representing rational numbers as (numerator / denominator) pairs of integers
- implement function that takes an array of integers and returns the maximum, minimum and average.
- implement a template for generating stack classes
- implement a program that shuffles an array into a random order

Any of those would be a highly typical coding question. And before you say "that's easy!" about any of them, think about what's missing from those problem statements before you write the code.

That string pointer: What encoding is it pointing to? 7-bit ASCII chars, UTF-8, UTF-16, some ANSI encoding? Are you going to ask the interviewer, or are you going to assume that you're writing C code for some operating system that existed in the Before Time when there were no strings that contained Chinese characters? Hint: Microsoft writes very little C code targetting PDP-11 machines. Ditto for programs that never have to deal with non-Roman character sets.

That less-than operator: Do you have to handle nonreduced fractions like 2/4 ? Illegal fractions like 2/0? Can you assume anything about the size of the integers?

That array you're shuffling: do we care if a hacker can predict the sequence? Do we need crypto-strength randomness, or reproducible pseudo-randomness? Is this for an online poker game for real money or a test case for a sort algorithm?

And so on. None of these problems is well-defined enough that I'd feel comfortable writing production quality code without at least some clarification.

Candidates often make the mistake of concentrating on the code, like the code exists in a vacuum for its own sake. That's a very unrealistic assumption! Code not only has to be well engineered and correct, it has to be maintainable, testable, and solve a real customer problem. In your interview, when you're done writing the code, think about:

- how would a tester attack it? Is the design even testable?
- does it handle stuff that hostile/buggy callers are going to throw at it? null pointers, large denominators, huge arrays?
- does it work well with other technologies? that is, does it use the conventions of COM or ATL, or does it work against them?
- is it correct, robust, maintainable, debuggable, portable, extensible?
- how would you know whether this was the code the customer wanted or not?

Now, this is not to say that someone who automatically thinks char * when they hear "string" is an automatic no-hire. But I am a lot more inclined to believe that "experienced in COM programming" on your resume if you acknowledge the existence of BSTRs! Also, I recognize that fresh-out-of-school candidates often have very little

experience with these sorts of "real world" considerations; I cut them some slack in those areas and concentrate on their raw intellectual horsepower, coding talent and long-term potential.

Finally, let me reiterate that technical interviews are hard, and even bright people screw up on them sometimes. Two teams no-hired me when I interviewed for full-time positions here! But that's another story.