Writing Better Code

How to Interview a Programmer

by Bill Venners

February 24, 2003

Summary

Recognizing good programmers among job applicants is not easy. This article contains interview techniques, garnered from a recent summit on writing better code, that can help you can find the most qualified programmers for your project.

In January 2003, I attended a Writing Better Code summit in Portland, Oregon, organized by Scott Meyers and Bruce Eckel. At the three-day summit, 15 people gathered to discuss code quality and how they could improve it. Throughout this discussion, one theme was clear: good code is written by good programmers. Therefore, one great way to improve the code quality within an organization is to hire better programmers. The trouble is, recognizing a good programmer among a pool of job applicants is not easy.

Finding good programmers is hard because good programming is dependent on much more than just knowledge of programming language syntax. You need someone who, despite wearing striped pants with a polka dot shirt, has a good sense of taste in OO design. You need someone who is creative enough to find innovative solutions to problems, yet anal retentive enough to always line up their curly braces. You need someone who is humble enough to be open to suggestions for improvement, but arrogant enough to stand firm and provide leadership when they are the best person to provide it. How can you tell all this about a stranger by spending 30 minutes with them in a conference room?

The final morning of the Writing Better Code summit, Bruce Eckel announced he was "hijacking" the meeting. Bruce wanted each person at the table to share his or her interview techniques. He wanted to know how *we* recognize a good programmer in an interview. In this article, I highlight some interview techniques discussed that morning. If you have more ideas or would like to discuss any techniques presented here, please post a comment to the News & Ideas Forum Topic, How to Interview a Programmer.

Explore an Area of Expertise

Although various interview methods were tossed about that morning, a few fundamental techniques

emerged from the discussion. For example, rather than simply look for expertise and experience in the exact area in which the candidate will work, you should look for general programming talent and ability. One way to explore and judge a candidate's talents is to explore an area of their expertise:

Dave Thomas: Hire for talent. One of the biggest mistakes companies make is to recruit from a shopping list: I need a programmer with six years Java, three years Oracle, and two years EJBs. The world changes, so you need to hire folks who change with it. Look for people who know *computing*, not necessarily particular narrow niches. Not only will they adapt better in the future, they're also more likely to be innovative in the present.

Chris Sells: To identify how good the candidates are technically, I let them choose an area in which they feel they have expertise. I need them to know something well, and I ask them about that. I ask them why. I want them to know why something in their area of expertise works the way it does. I'm not necessarily after an expert in the area I need. If they learned why in the past, I have confidence they'll learn why in the future.

Have Them Critique Something

Another technique involves the importance of creating a dialog with the candidate. To get to know the candidate's talents and personality, you can't merely ask questions that have short factual answers. You have to find a way to engage a conversation. To stimulate dialog, you can ask the candidate to critique some technology:

Josh Bloch: I ask candidates to critique a system or platform that we both have in common, preferably something they will use on the job. For example, I might ask, "What parts of Java don't you like and why?"

Pete McBreen: I give candidates samples of our current code and ask them to explain and critique it. This gives me a sense of their skills, but also lets them know what they can expect.

Ask Them to Solve a Problem

Another way to foster an open-ended dialog is to ask the candidate to perform a task: to solve a problem or create a design. Although everyone at the meeting seemed to agree that this was important and useful technique, it also generated a lot of concern. People felt that asking the candidate to solve puzzles and problems needed to be done with care:

Josh Bloch: I like to ask a candidate to solve a small-scale design problem, finger exercises, to see how they think and what their process is: "How would you write a function that tells me if its argument is a power of 2?" I'm not looking for the optimal bit-twiddling solution `((n & -n) == n)`. I'm looking to see if they get the method signature right, if they think about boundary cases, if their algorithm is reasonable and they can explain its workings, and if they can improve on their first attempt.

Bruce Eckel: I ask candidates to create an object model of a chicken. This eliminates any problems with uncertainties about the problem domain, because everyone knows what a chicken is. I think it also jars people away from the technical details of a computer. It tests to see if they are capable of thinking about the big picture.

Scott Meyers: I hate anything that asks me to design on the spot. That's asking to demonstrate a skill rarely required on the job in a high-stress environment, where it is difficult for a candidate to accurately prove their abilities. I think it's fundamentally an unfair thing to request of a candidate.

Matt Gerrans: I don't like when I'm asked to write a program that does X on a piece of paper. Don't ask the candidate to write a program on paper. That is a waste of time and sweat. People don't write software on paper, they do it with computers using auto-completion, macros, indexed API documentation, and context-sensitive help. They think about it, refactor it, and even rewrite it. If you want to see a person's work, ask them to write some small module or implement some interface *before* the interview and bring the code on a notebook PC or on hard copy. Then you can review it and discuss the design, coding style, and decisions that went into it. This will give you a much more realistic and useful assessment of a person's work and style.

Kevlin Henney: I like design dialog questions that don't have a single fixed answer. That way they have to ask me questions, and this sparks a discussion. It's good to have a whiteboard available in the room. A dialog lets the interviewer see how the interviewee works, whereas a question of fact is just that: it is great for TV quiz shows, but doesn't tell you how someone will work and approach things over time. A puzzle question requires knowing a trick, which is in essence something that is either known or unknown. I dislike puzzle questions, because they don't require dialog.

Josh Bloch: What constitutes a reasonable question depends a lot on the candidate's experience and maturity.

Dave Thomas: I look for people with curiosity. Present problems, not puzzles.

Look at Their Code

Josh Bloch suggested one technique we all seemed to like: Have the candidate bring a code portfolio to the interview. Look at the candidate's code and talk to them about it. Although we were concerned that some candidates may not have code they could legally bring to the interview, we figured most candidates could probably come up with something. It can't hurt at least to ask a candidate to bring to the interview a sample of code they had written in the past.

Josh Bloch: I want to see their code. You get to see what they pick. You learn what they value. You learn how they communicate.

Find Out What Books They Read

Several people indicated that they ask candidates about the programming books they read to see if a programmer is self-motivated or concerned about improving their own programming skills:

Matt Gerrans: I ask candidates, "What books have you read about programming?" If the book is beyond syntax, that's important.

Randy Stafford: I find out what books they read because it's important to me that they educate themselves of their own volition.

Ask About a People Problem

As important as technical ability, or perhaps more important, is personality. How well would the candidate fit the team? How well would they fit the work environment? People used various techniques to judge personality:

Randy Stafford: Good citizenship is probably more important than technical prowess, because if you have people with the right kind of attitude and demeanor, you can help them gain the technical knowledge and software development habits. But if you have people who lack humility and maturity, it can be extremely difficult to get them to cooperate in reaching a goal, no matter how bright they are or what they've accomplished in the past.

Chris Sells: I ask candidates, "Tell me about a problem you had with a boss or teammate. Tell me how you've dealt with a problem with a boss."

Jack Ganssle: I check references. Now, I know these people are the candidate's five best friends, and will not say anything negative. But I ask those references for names of people who know the candidate, and go to these others for insight. This way I spread the net beyond anything the candidate ever imagined.

Kevlin Henney: I try to imagine if I would go to a pub and talk non-tech with them—not if I like them, but whether I could get along with them. Are they pubbable? Could I talk to them in a non-office situation?

Dawn McGee: The most likable person is often not the best person.

Dave Thomas: I think every team of a certain size needs a professional pain in the ass, because teams get complacent, fixed in their ways. They need nudging out of their comfort zone once in a while, so they can look at problems from a different perspective. There are two kinds of pains in the ass: the obnoxious boor—to be avoided on all teams—and the person who never learned that grownups shouldn't ask "Why?" all the time. The latter is a treasure.

Get to Know Them

Perhaps the prominent theme of our discussion was that you need to try to get to know the candidate as best you can. Talk to the candidate in the interview. Try to get a feel for them. If possible, bring them in on a trial basis or for a probationary period. That would give you more time to get to know the candidate, and give the candidate more time to get to know you:

Chuck Allison: I talk to them. I get a feel for them. I always ask about what they've done. I have found that by discovering what a person is excited about technically, you can learn a lot of important things about them. In the past I've asked people to describe a project that was especially interesting to them, or that was challenging and successful. On occasion I've asked what they've done that they're the most proud of. This usually reveals the depth of one's understanding and mastery. It also gets them to turn on the fire hose verbally, and you can sit back and get most of the answers you need.

Randy Stafford: I look at past projects listed on their resume, and ask them to talk about those

projects—how was the team organized, what the technologies and architectures were used, was the software successful in production, etc. In their answers I'm listening for what lessons they learned from those experiences, and whether those lessons match with lessons I've learned from my experiences and from professional literature. I get a glimpse into how they perceive themselves in relation to the world around them. Some come off as arrogant, some ignorant, some helpless. Others sound humble, intelligent, and motivated. I often ask them what software development literature they read. Continuous education is very important to me.

Angelika Langer: In Germany, hiring is like marrying someone. It's "until death do us part"—a marriage without the backdoor of a divorce, because you can't fire employees. The only chance for firing someone is during a three- to six-month probationary period, or when the company goes out of business.

The major filtering is done *before* the interview, based on the curriculum vitae (CV) and submitted papers, such as evaluations from former employers. (In Germany, employers must provide every employee with a written evaluation when they leave the company.) The interview itself is usually brief. The main tool in filtering is scrutinizing the CV and papers; 98 percent of all applicants are disqualified in this phase. The interview should confirm the impression you gain from the applicant's papers and allows you to sense their personality. The lucky winner then goes on a probationary period.

Probation definitely does not replace the filtering; it just keeps a last exit open until you really must commit.

Andy Hunt: We've hired people who interviewed well, but they were terrible at the job. If possible, hire them in for a trial period.

Dawn McGee: You could also bring candidates in for half a day, and have them do what they would be doing on the job.

Conclusion

To sum up the overlying themes from our hour-long discussion in Portland: You should look for talent and fit more than specific skill sets. Ask open-ended questions to initiate revealing dialog. Ask candidates to critique something. Ask them to design something. Investigate their past experience. Review their code. And through conversation and, if possible, a trial period, you should try to become familiar with the candidate's technical abilities, talents, and personality.

How Do *You* Interview a Programmer?

Do you have an opinion on the techniques mentioned in this article? Do you have a tool or technique you use to find good programmers? Please share your ideas in the News & Ideas Forum Topic: How to Interview a Programmer.

About the Author

Bill Venners is President of Artima Software, Inc. and Editor-In-Chief of Artima.com. He is the author of *Inside the Java Virtual Machine* (Computing McGraw-Hill), a programmer-oriented survey of the Java

platform's architecture and internals. His popular columns in *JavaWorld* magazine covered Java internals, object-oriented design, and Jini. Bill has been active in the Jini Community since its inception. He led the Jini Community's ServiceUI project, whose ServiceUI API became the de facto standard for associating user interfaces to Jini services. Bill also serves as an elected member of the Jini Community's initial Technical Oversight Committee (TOC), and in this role helped to define the governance process for the community.

Resources

Chuck Allison is a professor of computer science at Utah Valley State College in Orem, Utah. He has degrees in mathematics and practiced software development professionally from 1978 to 2001, working for defense contractors and other large corporations. He spent most of the 1990s as a contributing member of the C++ Standards Committee and designed the standard bitset class. He has been a columnist for the *C/C++ Users Journal* (C++ and Java) and the editor since 1992, and is now senior editor of the journal. He is the author of *C & C++ Code Capsules* (Prentice Hall) and co-author with Bruce Eckel of *Thinking in C++, Volume 2* (Prentice Hall). He has taught C++ and Java extensively throughout the U.S. and is available for training and consulting from May through August:
http://www.freshsources.com/

Josh Bloch is an architect in Sun Microsystem's Core Java Platform Group. He has designed major enhancements to the Java APIs and language, specifically for the Java Collections API and the java.math package. Most recently, he led the expert groups that defined Java's assert and preferences facilities. In his 2001 book, *Effective Java Programming Language Guide* (Addison Wesley), Josh distilled his wisdom into 57 concrete guidelines for designing and implementing Java programs:
http://java.sun.com/docs/books/ef fective/

Alistair Cockburn is the founder of Humans and Technology and Cockburn and Associates. Alistair is internationally known for his work in object- oriented software development. Alistair does technical facilitation, process and organization design, project setup, requirements gathering, and OO design:
http://alistair.cockburn.us/

Bruce Eckel is the author of *Thinking in Java* (Prentice Hall), the Hands-On Java Seminar CD ROM, *Thinking in C++* (Prentice Hall), and *C++ Inside and Out* (McGraw Hill), among others. He's given hundreds of presentations throughout the world, published more than 150 articles in numerous magazines, was a founding member of the ANSI/ISO C++ committee, and speaks regularly at conferences. He provides public and private seminars and design consulting in C++ and Java:
http://www.mindview.net/

Jack Ganssle helps developers build better embedded systems faster. He started, developed, and sold three electronics companies; including one of the world's leading producers of embedded development tools. Jack is *Embedded Systems Programming*'s technical editor, as well the magazine's monthly "Breakpoints" columnist. He has published more than 300 articles on different aspects of embedded development, as well as two books (*The Art of Designing Embedded Systems* and *The Art of Programming Embedded Systems* both published by Newnes) on the subject. Online, he writes the weekly "Embedded Pulse" column on embedded.com and is the editor of *The Embedded Muse*, a free bi-weekly email newsletter:
http://www.ganssle.com/index.htm

Matt Gerrans began his professional life as an electronic engineer but quickly saw the light and switched to software development. He now has more than 12 years professional software development under his belt, including work in C++, Java, Python, and yes, even JavaScript. He is the C# columnist for Artima.com:
http://www.cyclethere.com/

Kevlin Henney is an independent consultant and trainer based in the UK. Prior to becoming the founding director of Curbralan Limited, he was a principal technologist for QA Training. He has developed and delivered training course material and consultancy on many aspects of OO development, which he has practiced across a number of domains for more than a decade:
http://www.curbralan.com

Andy Hunt and Dave Thomas are the "Pragmatic Programmers," recognized internationally as experts who develop high-quality software -- accurate and highly flexible systems. They helped write the now-famous Agile Manifesto, and regularly speak on new ways to produce software. Their best- selling book, *The Pragmatic Programmer* (Addison Wesley), describes their software development best practices. They have more than 40 years combined experience in the industry. They know practical software development:
http://www.pragmaticprogrammer.com/

Angelika Langer works as an independent freelance trainer, mentor, and consultant with a course curriculum of her own. Her current work is backed by more than a decade of experience as a software engineer working for German and US companies and several years as a trainer and consultant. She enjoys speaking at conferences all over the world. Her consulting and mentoring assignments focus on code reviews, audits, project evaluations, special purpose workshops, and more. Together with Klaus Kreft, she wrote the authoritative book *Standard C++ IOStreams and Locales* (Addison Wesley) http://www.langer.camelot.de/ios_treams.htm. Angelika also served as a columnist for *C++ Report* http://www.langer.camelot.de/Articles/Articles.htm#C++_Report and *C/C++ Users Journal* for many years, and currently writes a column entitled "Effective Java" for the German *JavaSPEKTRUM* magazine:
http://www.langer.camelot.de/

Pete McBreen is the author of *Software Craftsmanship* and *Questioning Extreme Programming* (both by Addison Wesley). He is an independent consultant who actually enjoys writing and delivering software. Despite spending a lot of time writing, teaching, and mentoring, he goes out of his way to ensure that he does hands-on coding on a live project every year. Pete specializes in finding creative solutions to problems that software developers face:
http://www.mcbreen.ab.ca/

Dawn McGee is a Business Consultant and Attorney. Formerly, she served as the Lead Analyst for Underdog Ventures, LLC, in New York City and Global Partners, LLC, in Napa, California, focusing on investing in companies that have a high social impact. Ms. McGee provides clients with legal and business advice on start up issues, business plan reviews, business formation, venture capital financing, technology development agreements, licensing agreements, intellectual property protection, private placements, joint venture agreements, real estate issues and estate planning and business succession. She speaks frequently on the issues of angel investing, venture financing and women's entrepreneurship. She can be reached at:

dawnmcgee@excite.com

Scott Meyers is one of the world's foremost experts on C++ software development. He wrote the best-selling *Effective C++* series (*Effective C++*, *More Effective C++*, and *Effective STL* (all Addison Wesley)), wrote and designed the innovative *Effective C++ CD*, is consulting editor for Addison Wesley's *Effective Software Development* series, and is a member of the advisory board for *Software Development* magazine. He also sits on technical advisory boards for several start-up companies. A programmer since 1972, he holds an M.S. in computer science from Stanford University and a Ph.D. from Brown University. Scott offers consulting and training services to clients worldwide:
http://www.aristeia.com/

Chris Sells is a consultant, author, speaker, and general technology wonk. He specializes in component-based and distributed systems in a Windows environment, including .Net, Web Services, COM, C++, ATL and Win32:
http://www.sellsbrothers.com/

Randy Stafford is a professional software developer and chief architect at IQNavigator, Inc. He wishes he had more time for writing about software development, but has recently contributed chapters to Martin Fowler's *Patterns of Enterprise Application Architecture* (Addison Wesley) and Floyd Marinescu's *EJB Design Patterns* (John Wiley & Sons). He's been developing enterprise applications in Smalltalk or Java for 15 years in government and commercial sectors, for companies large and small. His homepage on Ward Cunningham's Wiki Wiki Web, where he has been a longtime community member, is:
http://c2.com/cgi/wiki?RandyStafford

Dave Thomas and Andy Hunt are the "Pragmatic Programmers," recognized internationally as experts who develop high-quality software -- accurate and highly flexible systems. They helped write the now-famous Agile Manifesto, and regularly speak on new ways to produce software. Their best- selling book, *The Pragmatic Programmer* (Addison Wesley), describes their software development best practices. They have more than 40 years combined experience in the industry. They know practical software development:
http://www.pragmaticprogrammer.com/

Last Updated: Wednesday, September 14, 2005

URL: http://www.artima.com/wbc/interprogP.html
Trouble with this page? Contact:
webmaster@artima.com