

**developer.\*****A Web Magazine for Software Developers**

## Making the Cut

By Daniel Read

As I mentioned in a previous article about resumes (“The Art of the Developer Resume”), I spent the past few weeks hiring two developers for our team. I read stacks of resumes, spent several hours on the phone screening candidates, and spent several hours in interviews with the people who did well in the phone screenings. (Thankfully, I also had the help of two colleagues, without whom the process might have been unbearable—or impossible.) We even gave each developer an “audition,” in the form of a sample program. We would sit each candidate in a room with a laptop, hand him a one-page spec for a simple program, close the door, and come back in an hour.

We were not sure if we would find them, but we were looking for developers who were *serious* about their craft, who had reached an advanced level of general software development knowledge, who had a sophisticated understanding of design, and who were masters of their chosen tools. We were not looking for the smartest people, or the people with the highest degrees, or the people who knew the most languages, or the people with the best haircuts.

I worked with several recruiters from placement agencies, who brought the resumes and candidates to us. In the end, we only chose two people. Naturally, the recruiters whose people were not chosen for the job wanted feedback as to why their candidates did not make the cut. I did my best to explain it to them, but have been feeling there is more to say on the subject.

These are some of the reasons we hired the two people we did:

**They had a quality mindset.** A “quality mindset” is hard to define exactly, but to me it is one of the most important characteristics that a top developer must have. If you do not care deeply about quality, then you will not build quality software. In short, the most reliable way to achieve quality is attention to detail. However, quality can be a difficult thing to pin down since every person involved has different ideas about what quality means. Quality to you as a developer might mean something totally different than it means to the president of your company. That said, even when the definition of “quality” is shifting right beneath her feet, a true professional is always pursuing it by paying close attention to detail and *caring* that those details are attended to.

**They knew their tools and platforms.** It may seem painfully obvious that the developers we hired were experts with their tools, but you’d be amazed at how many people who took our audition could not write part of the code because they could not remember the names of certain functions. (That’s right, the language reference help file was not installed on the audition machine.) I’m not talking about advanced, deep-down language stuff—I’m talking

about basic parts of the language. I could go on with further examples of the glaring mistakes and inadequacies in the audition results, but I'll spare you.

Tool mastery was obvious in the candidates we hired—but the audition was not the only indicator of that. It was obvious from the way they *talked* about the use of their tools. When probed with specific questions about the tools in which they claimed proficiency, they would answer with a) correct answers, and b) detailed and/or complex answers that demonstrated that they really *understood* the tools, especially the shortcomings, limits, and tradeoffs that come with the tools.

I have also found that, in general, the developers who are masters of their tools never stop reading about the intricacies of those tools. Any major programming language and/or platform, for instance, will have books dedicated to the “inner workings” and “hardcore stuff” of the technology. Seeking out this kind of material is essential. Too many developers stop studying their tools once they have reached a basic proficiency that allows them to do their jobs. A craftsman must be master of his tools, and mastery is impossible without intimate knowledge.

**They understood design.** If nothing else, we wanted to make sure we hired developers who had a sophisticated grasp of design. This quality turned out to be difficult to find. Bad code in a good design can be cleaned up, but a poor design can be very difficult to overcome no matter how well the routine-level code is written. Design skills, in my opinion, come from a combination of study and experience. Study is required so that you can learn from the mistakes of others and keep from reinventing the wheels that others have already invented and refined. Study allows you to “stand on the shoulders of giants,” as the saying goes. Experience is required so you can learn from your own mistakes and successes.

**They had design experience.** I really think that getting burned by poor design decisions of your own makes all the difference in understanding the importance of design (not that I have any specific knowledge of either of these developers “getting burned”). In addition, by repeatedly building systems, if you pay attention you begin to see patterns emerge. Over time you will realize that from assignment to assignment you really are solving many of the same problems over and over again.

Design experience also comes from longevity in a position. By that I mean, if a developer does not stick around on a job for very long and to see the systems she has built go into production and through some “maintenance” releases, she never gets to see whether the design was successful or not—or whether it was successful in some ways, but not in others. Design is always about tradeoffs, and it really helps to experience those tradeoffs first hand.

**They were avid readers, especially of “outside” material.** Reading of outside material was not a stated qualification for the job, but I use this as a key indicator of what I can expect from a developer. When I say “outside” material, I mean materials outside of the scope of specific technologies and tools. Lots of programmers read language manuals and technology- or solution-specific books (and many don't even read those), but too few

of us read magazines, books, web sites, etc. devoted to more general software development topics. When I say “general,” I mean materials on requirements gathering, testing, design and design patterns, deployment, project management, “Peopleware,” coding standards and style, user interface design, architecture, methodologies, etc. Reading books from other disciplines, not to mention general self improvement books, can provide additional insights.

**They had good communication skills.** Communication skills are more and more important in today’s corporate world, where IT and business are finally learning to work together more closely. Developers will frequently interact with business managers, subject matter experts, and others outside the technical team. At the very least, if I’m leading a team, I want the developers I work with to represent the team well, to communicate clearly, to have at least a basic grasp of the political aspects of any given situation (which is not strictly a communication issue), and to express a positive attitude.

Not every developer has to be an expert communicator, but in today’s world, with the position of the pure programmer quickly becoming a commodity (at least in the perception of the people who pay money to hire programmers), any edge you can give yourself is worth the trouble. To anyone wishing to improve his or her communication skills, I would highly recommend the Toastmasters organization.

**They could talk in detail about systems they had built at previous jobs.** This one is a favorite test of mine: I ask a lot of questions about a candidate’s past assignments. If a developer has trouble describing in detail the systems he has worked on in the past, then how can I expect that developer to do a good job building me a new system?

**They excelled in the audition.** Of course the people we hired did well in the audition, but “doing well” is defined by so much more than just getting the answer right. Doing well means writing readable code with excellent layout and naming. Doing well means reading the specification carefully and fulfilling its guidelines. Doing well means designing an attractive user interface that conforms to basic standards. Doing well means going to the effort of attending to these details even though the program is just some silly throw-away audition thing that has no utility in the real world.

**They were experienced.** Ah, there’s the kicker. How do you get the top software development jobs without experience in top software development jobs? My short answer: you make each job a top development job, even when that job sucks. Make the most of every assignment you have. Look for opportunities to do more. Be assertive and suggest innovative solutions to problems that people don’t even know they have. Do more than you are asked to do. Treat every document, algorithm, and unit test like you’re going to hand it over to the president of the company. Read and learn. Ask questions and listen. Keep a positive attitude. People will notice, and experience will take care of itself. However, the converse is also true: if you are not conscientious and fail to perform well when your duties are menial and less than exciting, then your superiors will not risk giving you more important and exciting tasks to perform.

That said, keep in mind that in this job search we were looking specifically for “the best” developers, so experience was an important qualification. However, tons of experience is not always required. Teams are not always looking to hire the most experienced or “senior” developers. The team may already have plenty of senior developers. The required tasks may not call for a senior developer. Many times I have hired less experienced developers because of the other strong qualities of that person. Syntax and technique can be taught, but a quality mindset and pride of workmanship are so much harder to teach. This is why I listed experience last in the above list. Don’t get too hung up on experience.

It’s important to understand that it’s okay if you don’t want to be the best of the best. Not everyone needs to be, and when a team is hiring developers, they’re not always looking for the best of the best. Indeed, companies cannot always afford to pay for the best of the best. In many situations it would be strategically unwise to hire the best of the best. There is no shame whatsoever in reaching a certain level of competence as a developer, and just kind of staying there—writing good, solid code for employers, building good software, and really only learning new things when necessity dictates. The world needs plenty of developers who stay with their favorite language for years and years, who come to work every day and do a good job. These developers should be proud of what they do. People have families and obligations and outside interests. All this talk about “mastery” is not about judging anybody’s worth as a person. Software development need not be the center of everyone’s universe.

That said, there are those who, for whatever reasons, aspire to the advanced levels of software development. They are not satisfied with competence—mastery is the only acceptable level. The developers who rise to the highest percentiles do not arrive there by accident. They arrive there through purposeful determination and hard work. I have encountered many developers with several years (even decades) of experience, but who are not what I would call at an “advanced” level. I have even met developers with years of experience who were outright incompetent. Experience alone does not produce mastery—far from it. Experience is essential, but does not outweigh initiative, scholarship, and humility.

Software development mastery is as much a journey as it is a destination. At the risk of being immodest, I feel I have reached an “advanced level,” but by no means do I have illusions that I have reached any kind of pinnacle. Far from it. I know that I have years and years of study ahead of me, not to mention a lot of growing as a person. The list of things that I *do not* know is so much longer than the list of things that I do know—and I expect that it will always be that way.

**Association.** Seek out people (especially when searching for a job) who are smarter than you, who know more than you, and who have more experience than you. If you cannot meet them in person, read their books. In life, who you associate with does more to define you than almost anything else—the same applies to professional development. Someone once said, “You are the sum total of the five people you spend the most time with.” Not literally true, of course, but this aphorism contains more than a grain of truth. Be selective

about who you associate with. If you wish to be good at something, identify and associate with people who are good at that thing. If you want to be successful, identify and associate with people who are already successful.

**Goal-setting.** The setting of goals does not have to be anything dramatic or elaborate. Some “self-help” guru types will encourage you to write your goals down and pin them up where you will see them every day. I think the idea is that a), the act of writing a goal down makes it more “real,” and b), seeing it every day reinforces the goal and prevents you from slipping. I am not against this practice, and if you think it will help you, by all means, write your goals down and hang them up. However, I don’t think this is strictly necessary.

It is necessary, however, to have goals, even if they are simple or general. For example, a goal to read at least one book every month is a good simple goal. A goal to read three magazines a month is another worthy goal. You could also set a goal to learn a new language or technology. Pursuing a certification or taking a class is a more structured way to work towards a goal of learning something new. Having a more general goal to improve as a software developer is also valid, but you have to plan more specific actions when setting such a goal.

I have always liked this famous quote from Aristotle: “We are what we repeatedly do. Excellence, then, is not an act, but a habit.” My strategy for goal setting and personal improvement follows this model. I set general goals, and then try to do many small things every day that move me in the direction of those goals. These “many small things” eventually become unconscious habits (hopefully anyway). Over time, all those small actions build up. One book per month leads to 12 books per year. When you concentrate on repeating the small steps, the big goals take care of themselves. Socrates said, “Education is the kindling of a flame, not the filing of a vessel.”

**Study.** I’m trying not to belabor this point since we’ve already talked so much about reading, but reading is the single best way I know to absorb new information and expand your horizons. Reading is the best way to learn from other people’s mistakes. Reading is the best way to discover new information and insight that might have taken someone else years to learn and compile.

When you read a lot, over time you start to see patterns and connections that you never saw before. In a software development context, this means that your thinking will rise above the knowledge of the specific technologies and tools with which you are most familiar and into a higher plane, where you can see patterns that transcend the technology. Reading is also the best way to give yourself an edge over other software developers whom you are competing with in the marketplace. During our interviewing process, we only encountered two candidates who were serious readers. We hired both of them.

I especially encourage you to be a *critical* reader. Do not take everything everyone writes in a book as the absolute truth. People make mistakes. People latch on to ideas, philosophies, and methodologies that are later discredited or proven ineffective. Any writer

brings to his work his own biases and knowledge gaps. This is one reason why it is important to read a wide range of material—even material outside of the realm of software altogether.

**Practice.** This one is pretty self explanatory. If writing software is something you wish to master, then actually writing software is an essential activity. You may be lucky enough to work at a job where you get plenty of programming challenges. If not, or if you wish to leap ahead from where your current job keeps you, then practice outside of work is hard to avoid. One way to merge the activities of study and practice is to use software development magazines. You can read an article and download the accompanying code. Then you can spend time reading that code, compiling it, stepping it through the debugger, modifying it, etc.

Another excellent resource for study and practice is the Open Source community. There are countless projects out there, with millions of lines of code that you can learn from. As a bonus, you can study complete systems, which will give you a system-level view of software that you won't tend to get from the sample code that comes with books and magazines. Finally, the Open Source community is full of smart, generous people who can answer questions and offer guidance.

In closing, let me address an issue that may be nagging some people: I have heard a developer say something along the lines of, "When I look at the great developers I know, either in my class or at my job, I see people who seem to have a natural talent for designing and building systems. They can see the math and the logic and the abstractions and the patterns. They understand new information immediately. I don't have this talent (or don't think I do anyway), so how can I become a great developer?"

For an answer, I'd like to quote from an interview with a great writer of fiction, Theodore Sturgeon, who died in 1985.

Q: Could you explain your distinction between talent and skill?

TS: Anybody can do anything he wants to if he wants to do it badly enough. Now I know that's a vast oversimplification, but in principle, it's true. For example, I think that you or I or anyone could be a wire-walker for Barnum and Bailey, and all it would take would be an absolute determination and practice, practice, practice. You have to study your field and you have to find out how other people do it, and you have to keep working and learning and practicing and ultimately, you would be able to do it.

However, there are people who are able to do it in the first three tries. They're born with such coordination and talent, a construct of the semi-circular canals in the inner ear, that they're able to do a thing like that with great ease. Now skill is what you develop by that kind of practice and work and study. Skills can be developed and refined and brought to a very, very high pitch indeed.

Talents you are born with. There is simply no question about it. People with a high talent unfortunately don't have to continually practice. They don't have to, so they don't do it. It is a great loss because a lot of highly talented people never work at their talents at all. In the end, their product is lesser than somebody who has worked his buns off to get somewhere and refine the talent that he has.

Q: Does this apply to writing also?

TS: Writing is certainly the same thing. There are people who have a born facility with words. They absorb words, they see them correctly, never have any difficulty with spelling or grammar. They have a startling poetic view and a way of creating images and doing the brilliant unexpected. This is coupled with an observance of human beings and how they react, the things they are afraid of, the things they care about.

There are many people who don't have a particularly high talent in this kind of observation, this kind of facility with words. Yet they become superb writers almost because they don't have the talent and have to work a little harder at it.(1)

Sturgeon's comments here about writing apply equally well, I think, to programming.

## Additional Resources

If the Open Source idea interests you, check out SourceForge.net, which hosts thousands of Open Source projects. SourceForge has more free source code than you'll know what to do with. You can even join on with an existing project (or start your own). My good friend Rob MacGrogan started his own Open Source project called SourceJammer (see [sourcejammer.org](http://sourcejammer.org)), which he hosts at SourceForge. The project has been a great way for him to add to his already considerable skills, and he is making development contacts all over the world.

Gerald Weinberg has some great books that relate specifically to the subject at hand, namely *Understanding the Professional Programmer* and *Becoming a Technical Leader*.

A company called BT.Novations has an interesting model/metaphor for career growth and progression. They call it "The Four Stages of Career Growth." I recommend taking a look at the free PDF that they have available at

<http://www.bt.novations.com/pdf/4S%20of%20Career%20Growth%20.pdf>.

Also interesting is the "Construx Professional Ladder." Construx is the company founded by Steve McConnell, author of *Code Complete* and *Rapid Development*. According to the Construx web site, "Construx's professional ladder outlines a development path for software developers who work at Construx." The ladder is a list of numbered levels, which break the path to mastery down into specific steps, with specific experience, skills, and

knowledge for each level. Also included are recommended reading lists, a skills breakdown, and something called the “Professional Software Engineering Pyramid.” All this can be found at <http://www.construx.com/ladder/>. The reading lists alone are worth examination of this material.

Finally, the Links page at [developerdotstar.com](http://developerdotstar.com) has a list of recommended books. It is by no means complete, but it’s as good a place to start as any. Be sure to check out some of the magazine links too, especially *Software Development*.

## References

(1) From an interview with Theodore Sturgeon by David D. Duncan; copyright unknown, but published with the author’s permission at <http://glinda.lrsm.upenn.edu/~weeks/misc/duncan.html>.

###

Daniel Read is editor and publisher of the **developer.\*** web magazine. He lives in Atlanta, GA, where he works as a software architect. He is currently at work on a book about software development crafted for a business audience.