Hiring Software Testers in an Information Age

By Paul Carvalho. October 2007.

Over the last few years, more and more people have asked me for advice on hiring testers -- especially about senior testers for lead positions. Such advice, however, is not something that can be shared in 15 minutes over coffee.

Evaluating testers is tricky. Most learn their trade on the job. There aren't any particular educational requirements that you can look for on a résumé to help you discern the credible candidates from the rest, and every good candidate has unique experiences and skills that may or may not suit the needs of the available position.

This article is a guide to how I evaluate testers' abilities, credentials and work experiences. It also presents the idea of the 'schools' of testing, highlights my interviewing techniques, and includes my approaches to finding candidates and other hiring challenges. It is for hiring managers and testers alike.

Paul Carvalho
Owner/Consultant
Software Testing and Quality Services
(**STAQS**)

More about Paul

Index

Common Attributes of a Good Software Tester

In my experience, there are attributes common to good software testers that indicate potential for almost any testing position, regardless of industry or development technology. For almost a decade, I have often used the following as a guideline of the basic attributes I look for:

- Customer-focussed, Quality-minded
- Troubleshooting and Problem-solving skills
- Good communicator - written and spoken
- Thinks like a Hacker - Analytical and likes to break software for fun or the challenge.

For a senior tester, I would add the following basic attributes to the above list:

- Critical Thinking and Systems Thinking skills
- Reverse Engineering and/or Modelling skills
- Adaptable and flexible
- Understands the financial cost of Quality
- Ethical, has integrity
- Commitment to learning, open-minded

An implicit understanding of the Scientific Method is also very useful to a software tester. I believe that the Scientific Method is the basis for all good testing approaches. However, I look for that understanding in a slightly different way and I can teach it to someone as required.

The above lists identify the core attributes that interest me most in a candidate. Depending on the requirements of the position, I look for additional *experience* such as project management or leadership experience, domain knowledge, etc.

As you might guess, some of these attributes are next-to-impossible to identify in a candidate without some direct interaction. You can infer some of them based on the résumé contents. Unfortunately, this means that it takes time to review a large stack of applications.

When I review résumés, I dislike reading work experiences that list responsibilities or only tell me what they did. *Boring!* Some description of the general work environment, processes and technology is helpful to understand the context of the work experience. More importantly, I want to know what a candidate contributed, what they accomplished, and how they grew in the position.

At one time, I made it a habit to interview all the candidates that had interesting résumés, but that had its problems. Bringing someone into your office to decipher their experiences may set the wrong expectations and takes up a lot of your time. Now I use phone interviews. Within 30 minutes, I can make a better decision about whether or not to bring someone in for a more detailed interview than if I had the résumé alone to go on.

Is that it? Is that all there is to hiring good software testers? Unfortunately, no, that's just the start.

Skimming Through Résumés

When I hear people talk about hiring for tester positions, I often hear keywords thrown out like 'programming skills,' 'test terminology,' and 'certifications.' After all, these are readily identifiable on a résumé and they are clear indicators of a candidate's testing ability, right? Actually, no, they're not *clear* indicators.

Programming skills generally indicate that the candidate thinks like a programmer, not a tester. While understanding programming could be useful in *some* testing positions, a good tester thinks in a way that is sometimes opposite to that of a programmer. That is, a programmer thinks about how to *build* software while a tester thinks about how to *break* software. If programmers were great testers, there wouldn't be any need to have separate testers. Given the current state of software quality worldwide, it's quite apparent that we need good testers who think differently from programmers.

That doesn't mean that I would automatically rule out a candidate for a System-Testing position, for example, simply because they have some programming experience. However, I might if they have too much programming experience and we already have that knowledge within the team. It's like the difference between taking one donut from the box versus taking the whole dozen. Too much of one thing is generally not a good idea. (There might be exceptions.)

What about Test Terminology usage on the résumé? Knowing the correct jargon is part of an experienced person's toolset, right? Unfortunately, there is no standard glossary of terminology for the Software Testing profession today. So when I see terms like test plan, test cases, test scripts, etc., I can't *assume* that I know what these terms mean. I might have a general idea of what *I think* they mean, but until I ask the candidate to clarify the terms, I can't be sure. The word 'test' may be used as a noun, verb and adjective - sometimes all in the same sentence!

Many people sprinkle these terms across their résumé to make it look more appealing to hiring managers. This makes it hard to separate the noise from the signal. In the last ten years, I have reviewed hundreds of Tester résumés. Only once have I ever seen test terminology used in a particular way that made me want to interview the candidate. The majority of the time I see terms used in vague or uninteresting ways, and sometimes even incorrectly!

Here's a screening trick that I sometimes use. It might seem silly but it's simple and I find it helpful. For every test phrase or term you see in a résumé, replace the word with "blah". Does the sentence with the replaced words still sound appealing? Are you really interested in finding out more about what they meant? Most times, I'm not interested.

What about Tester Certifications? Debates have raged on email discussion forums and at public gatherings of Software Testers on the topic of Certifications.

This is some of what Wikipedia has to say about "Professional Certifications":

> *"A professional certification, trade certification, or professional designation (often called simply certification or qualification) is a designation earned by a person to assure that he/she is qualified to perform a job or task.*
>
> *Certifications are earned from a Professional society and, generally, need to be renewed periodically, or may be valid for a specific period of time (e.g. the life-time of the product upon which the individual is certified). As a part of a complete renewal of an individual's certification, it is common for the individual to show evidence of continual learning."*

That's a fine introduction. With that in mind, here are some of the major problems I have with Tester Certifications as they stand today:

1. It is “a designation earned by a person to assure that he/she is qualified to perform a job or task.” Unfortunately, the current Tester Certifications are all based on a person’s ability to memorise facts and regurgitate them on a multiple-choice exam. Demonstrations of skills or abilities are not required to obtain these certifications. I’m sure we wouldn’t certify Doctors without some demonstration of ability, so how can we ‘certify’ a tester simply because they passed a multiple-choice exam? I don’t think you can.

2. “Certifications are earned from a Professional society...” Tester certifications are currently provided by vendors interested in selling you their services and courses. This seems like a conflict of interest to me. Certifications are a business to some companies. It’s not about *assuring* that someone is qualified to perform the job or task.

3. Certifications don’t indicate the scope of applicability. Recall how I mentioned that there is no standard glossary of testing terminology within the IT industry. This means that the facts and terms memorised for the certification exams represent *specific* instances that are not commonly used everywhere. The certifications don’t indicate what they cover or how they might be meaningful. If I’m looking to hire someone as a Performance Tester, should I hire the candidate with the Tester Certification? That’s probably *not* the deciding factor. You wouldn’t hire a Proctologist when you need a Brain Surgeon simply because he was a certified MD. It’s the same with testers - the certification doesn’t indicate that the candidate has the skills, knowledge or experience you need. You have to *talk* to the candidate to try to find out in what context(s) the certification *might* be meaningful.

4. What was the *motivation* for the certification? Certification is not a requirement to be a tester today, so why did the person get it? Were they motivated by the promise of a higher salary? Did they get the certification to avoid being fired? Was it for some other reason? It’s like the example I gave earlier about test terminology on a résumé - in isolation it tells me nothing. I am more interested in *why* someone got it because it’s neither a requirement nor an indicator of someone’s ability to perform a task well.

5. Tester Certifications are never revoked. Where is the governing body that assesses the professionalism and performance of the certified individuals to ensure that they live up to an expected code of ethics and practice? Oh, right. There is none. So if no one monitors the people certified and certifications are never revoked, how are these certifications different from buying a university degree or diploma over the internet?

So, unfortunately, tester certifications are not good indicators of a tester’s abilities or capabilities either. They don’t help me narrow down the best candidates from a large pile of résumés. If the candidate looks good enough to warrant an interview and they happen to have a certification then I will ask about it. Otherwise, it might leave me with more questions and assumptions than answers.

Advanced Side Note: The Schools of Software Testing

WARNING! As with anything else, the information in this section can be misunderstood or misused. If you don't have the time to research the topic further, then just skip it and move on.

Please do ***not*** assume that this information is widely known or understood within the Software Testing community at this time. Since there are no common educational requirements for testers, it is up to each individual to develop their professional knowledge on their own time. Asking every tester which school they belong to might create more confusion than you are prepared to handle.

I find this information to be a useful tool in helping me understand a candidate's experiences compared to how we work. I also find it helpful in developing a new employee's training plan when they have some related work experiences. Here is a quick overview followed by how I use it during the hiring process.

What about the testing experience that someone claims to have on their résumé? Isn't testing done the same way everywhere? Aren't the methods and techniques transferable from one company to the next? Maybe, but not always. If you look closely enough, you might find that testing can be radically different from one project to the next even within the *same* organisation! The problem is one of *context*.

Back in 2003, author and Testing consultant Bret Pettichord had an idea. What if the field of Software Testing wasn't just one big group with conflicting practices and ideas? What if there were different schools of Software Testing like how we have different schools in Psychology (e.g. Structural, Behavioural, Psychoanalytic, Cognitive, etc.), Engineering (Civil, Chemical, Electrical, Mechanical, etc.), and Medicine (Paediatrics, Ophthalmology, Neurology, Radiology, etc.)?

By categorising the observed differences into separate schools, we might better understand why testing experts disagree. It is not simply a matter of personality or experience. A school is defined by specific common goals, includes particular values, exemplar techniques and a common vocabulary, to name just a few ideas.

So what are these Schools of Software Testing? Between Bret's presentation (see his web site at http://www.pettichord.com/) and that of another consultant's, here are the Schools as I currently understand them:

1) Analytic School	2) Standard School	3) Quality School
4) Context-Driven School	5) Agile School	6) Oblivious

This list has two major separations: those who care and think about how they do their jobs (#1-5) and those who don't (#6).

The last school, 'Oblivious', was added by another Testing consultant, James Bach, and is simply made up of all the people who don't give testing any thought as they go about their daily work routine. For example, many companies hire expendable 'Temps' to fill tester positions because they need the warm bodies. However, these people aren't generally motivated or qualified to give their assigned tasks any real thought. They just don't care. While generally an uninteresting group, I believe these people make up a large part of the workforce, so it is definitely worth bringing to your attention.

When I review a résumé, this is generally the first thing I try to identify in the candidate. Does this person with some testing experience really have a clue? Do they have any idea *why* they're testing? Have they taken *any* interest in what their job means or how they can improve their skills or knowledge in the field? I need to see some example of meaningful insight, growth or contribution to tell me that the person isn't completely oblivious. Otherwise, it's straight into the recycling bin. I don't need a drone to increase my headcount. I need someone who can and *wants* to think in order to help my team be successful.

So, how do you differentiate between the remaining Software Testing Schools? Here are a few points to highlight some of the major differences:

Analytic School:

- Sees testing as rigorous and technical with many proponents in academia
- Testing is seen as a branch of CS/Mathematics, and testing techniques have a logical, mathematical form
- Exemplar: Code Coverage
- Most prevalent in: Telecom, Safety-Critical, and in Academic institutions (courses)

Standard School:

- Sees testing as a way to measure progress with emphasis on cost and repeatable standards
- Costs may be reduced by using low-skilled workers. Mostly follows or forces the Waterfall development model.
- Exemplar: Traceability Matrix (i.e. make sure that every requirement has been tested)
- Most prevalent in: Enterprise IT, Government
- Encourages standards, "best practices," and tester certifications

Quality School:

- Emphasizes process, policing developers and acting as the gatekeeper
- Testing determines whether development processes are being followed; may alienate developers
- Exemplar: The Gatekeeper (i.e. the software isn't ready until QA says it's ready)
- Most prevalent in: large bureaucracies and organizations under stress

- Related institutions include: ASQ, SEI (CMMI model), ISO -- all of which also have individual and organizational certifications

Context-Driven School:

- Emphasizes people, seeking bugs that stakeholders care about, and providing information to the project
- People create software and people set the context. Testing is multidisciplinary and a skilled, mental activity.
- Exemplar: Exploratory Testing (i.e. a form of scriptless testing that involves rapid learning, concurrent test design and test execution)
- Expect changes and adapt the testing plans based on test results
- Most prevalent in: commercial, market-driven software

Agile School:

- Uses testing to prove that development is complete; emphasizes automated testing
- Developers must provide automation frameworks
- Exemplar: Unit Tests (e.g. as in those for Test-Driven Development)
- Most prevalent in: IT Consulting, ASP Development

These schools have different definitions for "Testing", different procedural approaches and documentation requirements, different ideas on how to perform Risk-Based Testing, and different ideas on Tester Certifications. So, while some test techniques might be the same regardless of who uses them, the *way* they are used and the reasons behind them may be completely different.

This is why I don't like to see test jargon listed on a résumé without some idea of the project context to help me understand what they mean. It would be like seeing a carpenter's résumé with words like 'hammer,' 'saw' and 'measuring tape' listed on it. Those are pretty words, but what did you do with these tools and why? What did you learn from them? How did you help the team, company or customer with them? Were you building a birdhouse or the Taj Mahal?

Using the Schools to Interpret Work Experiences

It is sometimes tricky to piece together from the clues in a résumé, but you can usually make a good guess of which school a person's experiences fit into best. If I have doubt, a few specific interview questions can confirm any guesses I might have.

Why is this important? Unfortunately, people from different schools don't always work together well unless the expectations are clearly defined. As a result, like Yoda, I sometimes have to help my apprentices *unlearn* old habits so that they can be productive on my team. One colleague of mine focuses on hiring people directly out of University or College. He says he is far more likely to find a superstar that way than to look at people who are floating around the industry with a few years of bad habits (er, experience).

Besides as a way of ignoring the *Oblivious*, there is another good reason to focus on college graduates. There simply aren't enough candidates out there to fill the available positions. Since most educational institutions still don't teach software testing well, if at all, we need to woo graduates away from becoming programmers simply because that's all they think they can do.

Returning to the Testing Schools, here are some tips and ideas I keep in mind when reviewing the experiences on résumés. For starters, it's probably a safe bet that most system-level testing experiences fall into the "Standard School."

I have a few issues with this school. Firstly, the underlying approach followed here is the Waterfall model. That might have been fine about 30 years ago, but it isn't very efficient when the development staff is following an agile development model. Here are some of the résumé keywords that stick out:

- Created Test plans, test cases, test reports
- Test documentation, templates, IEEE standards
- Traceability matrix
- WinRunner, RUP, and other enterprise tools

If these keywords are typical in your environment, then these keywords will likely be appealing to you. However, if you are in an agile environment where you might not have detailed specs to work from, the software builds change too often to make high-level test automation efforts worthwhile, or generating unnecessary test documentation adds no value, then tread carefully.

Another problem I have with the Standard School is the embedded social hierarchy. People testing this way generally fall into one of three roles: test manager, test designer or tester. The test manager is actually a project manager although he/she may deny it. The test designer does the thinking required to create bunches and bunches of documented test cases. The testers execute the test cases created by the test designer.

As the lowest level in the hierarchy, the testers here are not actually required to think. It happens sometimes by accident rather than by design. For this reason, this 'tester' group often includes many *oblivious* candidates. You need to watch for this.

True "Quality School" experiences are not as common as regular testing jobs, but I see them every now and then. I even worked in this role once. Some résumé keywords that may suggest this school include:

- Quality System, ISO standards
- Process improvement activities, audits, CMM
- Software Quality Assurance

In most situations, the title 'QA' and 'Tester' are used interchangeably and are synonymous. This is the one time when they are different and mean different things.

"Software Quality Assurance" (SQA or just QA) refers to someone tasked with defining and managing the *processes* required to design, build or support some product or service. In general, a QA person is interested in *Defect* ***Prevention***. A QA person may often be involved in developing Quality Systems, performing Audits, and participating in committees for Continuous Improvement initiatives.

By contrast, a Tester is someone who looks at a product, system or service and tries to help find things that *don't* meet somebody's expectations. In general, a Tester is tasked in some way with *Defect* ***Detection*** or finding bugs. There are many, ***many*** ways that someone can do testing.

Again, if these Quality School keywords are what you are looking for, then they might indicate a good match with the candidate. If you happen to be in an agile development environment, note that the values in this particular Testing School go completely against the Agile Manifesto (http://www.agilemanifesto.org/). 'QA' people here are ***not*** testers. You might want to consider a different candidate.

The last school that I want to touch upon briefly is the "Agile Testing School." Some popular résumé keywords here would be:

- Programming skills and experience, API testing
- TDD (Test-Driven Development), Unit Testing frameworks or harness
- (Low-level) Test automation (probably using a real programming language like Java or C/C++ rather than a scripting language like Perl or Ruby)

It's often for Agile Testing positions that failed programmers think they can become testers. It's also for these positions that some testers think they can become programmers. In some cases, no tester is actually involved and it is simply an integrated responsibility for the programmer.

There are times when automating certain tests makes sense and can provide value, and times when it simply isn't worth the return on investment of time and money. If you are in a situation where test automation provides value and you need these skills, then this candidate might be worth interviewing. If not, then move on to the next résumé.

Bottom line: can you still hire someone from one school to switch to another? Sure. It depends on the individual though. Recall the list of 'common attributes' that I look for in a good tester above. I listed "adaptable and flexible." This is exactly the kind of scenario when someone can demonstrate how well they can adapt to their working environment rather than forcing the rest of the department to accommodate to their testing methods and ideals.

When joining a new project, team or organisation, some good advice I offer any tester comes from the famous martial artist, Bruce Lee:

> *"Empty your mind, be formless. Shapeless, like water. If you put water into a cup, it becomes the cup. You put water into a bottle and it becomes the bottle. You put it in a teapot it becomes the teapot. Now, water can flow or it can crash. Be water my friend."*

I currently place myself within the Context-Driven School of Testing (see http://www.context-driven-testing.com/). I wasn't always in this group, though. Like most others, I started by following the practices of my employers and early mentors in different 'schools'.

The more I stayed in the testing field and accepted new positions in different contexts, the more I realised that "best practices" and "silver bullets" don't exist or lead to the promised successes. Developing software is a highly creative process accomplished through collaboration among very intelligent people. If I am going to provide value within this process, it won't be by following dogmatic methods with my eyes closed. I need to be adaptable and productive.

Conducting the Interview/Audition

When you have selected the candidates that interest you from among all the applicants, the interview plays a critical role in the hiring process. There is a plethora of advice out there on conducting interviews. Depending on your needs, some good ideas include: ask behavioural questions rather than true/false ones; have different people interview the candidate so you can compare notes afterwards; and be sure to (phone) screen your candidates in advance so you don't waste anybody's time.

Author, professor, and consultant, Cem Kaner, wrote a nice summary of his interview process for the Dr. Dobb's web site (see "Recruiting Software Testers, Part 2" at http://www.ddj.com/architect/184414561). In this article, Cem offers some nice tips for checking the following during an interview:

- Testing Philosophy
- Technical Breadth
- Project Management
- Staff Relations
- Tests and Puzzles
- Written Technical Communication (Bug Reports)

I can add little else to that advice. One thing that I have always done in my interviews is to have the candidate actually test something. I do this because it's not as if a tester can bring a portfolio with them of the best tests they've ever done on past testing projects. This is a testing position after all, and I need to see if the candidate can really walk the walk.

It never ceases to amaze me how many times I bring in an interesting candidate, they tell me many things that I want to hear and then they totally flub the hands-on testing activity. In this way, the interview is more of an audition. I wouldn't hire a dancer or singer for a stage production without seeing a performance first. Similarly, I am not interested in hiring a tester without seeing if they can actually test.

I have a set of nine or ten applications that I've collected from colleagues and around the internet. They are simple apps that don't take more than about 10 minutes to figure out. So the challenge for the tester candidate then becomes: what information will you tell me about yourself while you test? Is it how well you find bugs? How well you can model and predict the expected behaviour? How you apply specific test techniques when you test? Or something else? It's your audition - impress me.

Not every test manager uses hands-on testing exercises during their interviews. Many are content to rely upon the quality of responses to the questions asked as their primary selection criteria. Everyone has a different style and preference that works for him or her.

Personally, I like to see a tester in action. I learn a lot about the breadth of someone's understanding of testing techniques in this way. It also gives some candidates the chance to show me what they can do when they might not have the words to articulate their abilities. I can always help someone learn particular testing terminology, techniques and approaches if I need to. Finding someone with the right aptitude, motivation and ability to learn is the tricky part.

Audition trials may be more than just testing applications. For a senior or leadership role, you might also consider having the candidate organise a meeting, demonstrate prioritisation abilities, or show some other skill that is critical to your business. If you are concerned about finding the right person for the job and a candidate *says* they can do it, don't be afraid to ask the candidate to demonstrate their skills. If you are more concerned about personal fit, an important interview trial may include meeting the rest of the team in an informal environment - over lunch, for example.

Advice for the Tester Applicants out there

Now that I have described some of my interviewing tips, here are some final thoughts for testers looking for work.

1. Don't give up. Be persistent, in a good way. Set goals and work towards them.

2. Look for jobs that provide growth opportunities. This doesn't just mean training and education courses. It could also mean working with people, technologies or approaches that you have never worked with before.

3. Don't stop learning. Learning isn't something that happens in educational institutions, it's something that happens in life. When you stop learning, you stop living and growing; you become less useful in an ever-changing workforce.

4. Try new things. Show initiative. Leadership doesn't require you to manage people. Successes are great advertisers. Failure is a part of the learning process. Remember that Thomas Edison once said: "I have not failed. I've just found 10,000 ways that won't work."

5. Try the same thing in different projects. There are no silver bullets. You will never learn the context(s) in which something works if you don't try and risk failing. Understanding the factors in which certain practices might be successful will increase the value of your opinions and recommendations.

6. Don't stop testing. Learn to tell when it's the right time to stop testing.

a) If you are currently employed, consider what percentage of your day is spent actually testing or thinking about testing. Testing is an activity that requires an infinite amount of time. Do you know that the testing you have done is good enough? Are there other techniques you can learn or apply to get you more information and raise everyone's confidence in the quality of the product? Question. Learn. Apply. Discuss.
b) If you are currently unemployed, don't let that stop you from testing if that's what you want to do. Find some Open Source projects on the internet that interest you and help provide them with the feedback they need to develop better quality products. Use this as an opportunity to learn and try out new test techniques. Keep your spirits up.

7. Learn about Testing. Read a book or two, take a course, attend a conference or workshop, or join a local QA/Testing peer network. If you have at least a few years of Testing experience under your belt and you want to keep doing it, continuing your education in your chosen profession is a must. Every tester should know about http://www.testingeducation.org/ and have browsed at least a few of the topics freely available there. Practice what you learn.

8. Develop more than just technical skills (e.g. like programming). As a tester, you may interact with many people every day. For example: customers, developers, product managers, business analysts, project managers, senior management, support people or technical writers. Developing software solutions is a creative, intellectual, people-centred activity. Develop skills that improve multiple intelligences: logical, intrapersonal, interpersonal, linguistic, and visual/spatial. Develop business savvy.

9. Be assertive, not stubborn. Keep an open mind. Adapt and accept change. Be a proponent of change, not a stick-in-the-mud.

10. Have fun. Remember that "All work and no play makes Jack a dull boy." Adding joy to your day boosts morale and performance and also helps the day go faster. Harvey MacKay once said, "Find something you love to do and you'll never have to work a day in your life."

Where Are the Candidates?

Historically, I have had two major challenges when looking to fill available testing positions: getting enough good candidates to apply, and sifting through the applicants to find the best candidates for the job. Personally, I worry more about a lack of applicants than an abundance.

I have tried using popular internet job boards, peer networks, local colleges and universities, word of mouth, and Career Fairs. I have colleagues who have gone even further to advertise and attract candidates for available testing positions. Applicants come, but not always with the calibre and quantity that one might need.

This adds stress to the hiring process. The longer it takes to hire the required personnel to your test team, the greater the potential impact to the schedules and quality of the software development projects in progress. The greater the pressure you feel to hire someone right away, the more inclined you will be to lower your standards or requirements for the position and make hasty hiring decisions. It's a gamble. You win some; you lose some.

I have often wondered why there aren't as many good candidates applying for tester positions as there are for programming positions. Is it because colleges and universities don't prepare students to pursue this option as a career? Is it because the candidates out there just don't have the right communication mechanism(s) to find the available jobs? Is it because the demand for testers has surpassed the supply? Or is it maybe because the job postings themselves just aren't attracting the candidates I need? (i.e. a failure on my part to properly communicate my needs in a way that will attract the candidates I want?)

Software and system quality is a serious matter that is becoming ever more apparent in News media and everyday life. If we are to address this need with enough qualified testing professionals to make a difference, then we have to answer the right questions to help narrow the gap between supply and demand.

Hiring Challenges and Final Tips

Once you have a ready supply of applicants to sift through, the hiring challenge shifts. Now the onus is on you to find the right candidate that either has the attitude, skills and knowledge that you need, or has the potential to meet that need with a little guidance or training. Personal fit with the rest of the team is also very important and something you ignore at your own peril.

I find that it's easier to train someone with the technical skills I need if they have good motivation and the right intangibles (e.g. see the basic attributes of a good software tester that I listed above). It's harder to go the other way though. Someone with good technical skills who doesn't communicate very well or have the desire or ability to troubleshoot problems won't be a very good addition to the team.

The applicant often has more to tell you than just what you see in the cover letter and résumé. In this Information Age, it's hard not to leave a footprint of some kind somewhere on the Internet. For that reason, I also use Google as an important tool during the hiring process.

When I come across a candidate that I might be interested in interviewing, I do a Google search on their name. Searches may turn up: personal blogs, personal web sites, emails they have posted on public forums, participation in personal or professional activities, something other, or nothing at all. It's all additional information to help you better understand the candidate.

For instance, I'm sometimes curious when a good candidate turns up *nothing* at all in a Google search. The internet is a great tool for a tester, so I'm curious why a good candidate wouldn't have *any* kind of footprint. On the flipside, I have also eliminated candidates based on their internet footprint.

Look for examples of originality, creativity, imagination and accomplishments on résumés. This may be a good indication of the contribution they can make on your team.

The jury is still out on spelling and grammar mistakes. Some hiring managers never accept tester applications with these types of mistakes while others are more forgiving. If we, as testers, are paid to find and report the bugs in the work of others, should we not take the same pride in our own work and at least have someone else check it before you submit it?

With spell-checkers readily available in every word processor out there, I'm often in the camp that requires good spelling as a minimum baseline requirement for a tester application. Typewriters are gone people; it's time to make use of the tools and brains you have to produce reasonable-quality documents.

Your résumé and cover letter might be your one chance to impress me. I may be forgiving of grammar mistakes for candidates who don't have English as a first language, but not using a spell-checker is just lazy. A second pair of eyes should help you spot *incorrect* words that the spelling tool can't catch.

The final step is the interview itself. After you have whittled down the list of candidates using phone screens or some other approach, it's time to get to know the hopefuls. Questions are okay and often turn up interesting things. I wouldn't say that's my strong suit though – I'm a Tester, not a Shrink. I prefer to have the candidates *demonstrate* the skills they think they know. If I'm looking for a baker, I want the candidate to bake me something not just talk about baking.

It's the same with Testing. If someone says they can test, have them test something. If they say they can develop test strategies, have them do that. If they supposedly have great meeting management skills, have them organise and run a meeting. You don't have to copy Donald Trump's "Apprentice" show, but I think there's some definite merit in that approach. Actions speak louder than words.

Note that just because someone fails an interview task doesn't mean they should be eliminated from consideration. I put a lot of weight on the intangibles and whether or not someone has the desire, motivation and ability to learn how to do something. You can teach many things, but some things just aren't worth the effort.

Final Thoughts

If you are looking for some additional references, I recommend Cem Kaner's paper "Recruiting Software Testers", 2000 (http://www.kaner.com/pdfs/JobsRev6.pdf). I have gotten a lot of mileage out of this paper over the years. I also recommend Johanna Rothman's book: "Hiring the Best Knowledge Workers, Techies & Nerds: The Secrets & Science of Hiring Technical People", 2004. Be sure to browse her web site (http://www.jrothman.com/) for additional materials and advice.

It is as easy to underestimate the effort required to hire good testers as it is to underestimate the value a good tester provides to a software development project. Unfortunately, there are no shortcuts in the hiring process that I know of; no magic wells from which excellent testers spring forth.

Each testing position is as unique as the project contexts themselves. Good testers often have some good foundational skills, attitudes and knowledge, which are not as common as we might hope. When I can't find the candidate I'd like for a position, I look for the best candidate who demonstrates the motivation, creativity and capacity for learning, that I feel will make them great testers in time. These are the key attributes to watch for in an Information Age. Be creative in where and how you look for them.