"Top Five (Wrong) Reasons You Don't Have Testers" by Joel Spolsky

Article Review and Commentary by
Bill Tepper, CEO of Provaré Technology

Coming from Joel Spolsky, this article has a high degree of credibility out of the gate. At last count, Joel has authored 3 books and over a thousand articles on software development. Joel's philosophy of how to recruit and retain talent is very similar to ours. In fact, we do agree with most of what Joel says in this article. We do have a couple of quibbles, but I suspect that if I were able to sit down with Joel over a cup of coffee, we'd find that they are more semantic differences than actual disagreements.

Joel notes (as I did in my 2005 white paper) that even after years of shared experience in software development, some development organizations either do not have an independent test team or have one that is vastly undersized and underfed. He then proceeds to share the top 5 reasons that he has heard over the course of his career for this unbelievable situation and to debunk each of them.

I'll list each of the 5 reasons, review Joel's counter arguments, and then add a few notes from our experience at Provaré.

(Wrong) Reason 1: "Bugs come from lazy programmers"

Joel: Joel says that he has encountered multiple managers that stated a belief that not having testers would force their programmers to test their own code more thoroughly. However, due to human nature and the force of habits, programmers will mainly test the program's functionality exactly as they intend it to work. Joel recounts an experience where a tester told him that a particular UI was completely not functional when Joel was *certain* he had verified it. It turned out to simply be a difference of mouse commands versus keyboard commands.

Provaré Observation: Really good test engineers approach product testing from a completely different perspective than programmers do. They look at the product from the customer's perspective and will explore functionality in new and creative ways. They look beyond the requirements to the intent and business case for the software. Ideally, they are plugged in to the sales and marketing organizations to be sure that a customer will experience the functionality largely as they expect it.

(Wrong) Reason 2: "My software is on the web. I can fix bugs in a second"

Joel: While web distribution is not as costly as packaged software for delivering patches, many companies underestimate the costs of fixing these bugs. The biggest danger is introducing new bugs when attempting to fix the first one. Plus, a close examination of your process for new versions may reveal this as more expensive than you realize.

Provaré Observation: Rapid distribution is a double-edged sword. Remember that web distribution ensures that a much larger pool of customers are more quickly affected by the presence of bugs in these releases. If you get a reputation for breaking something new in every patch release, you won't have customer issues for long because you won't have customers for long.

(Wrong) Reason 3: "My customers will test the software for me"

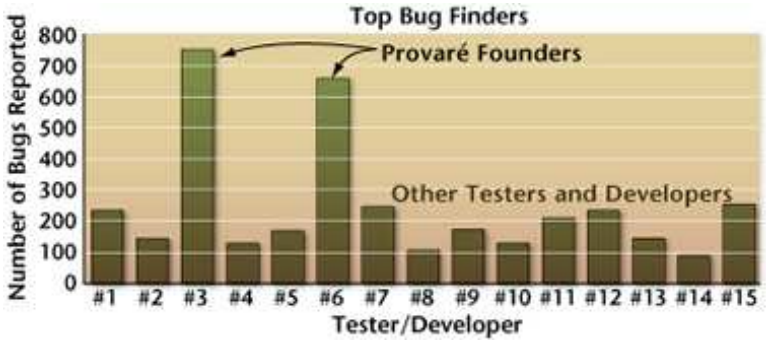
Joel: Nothing damages your product's and your company's reputation more than the perception of poor quality caused by customers finding your bugs. Netscape is a classic example of a company that used this methodology nearly to the point of extinction. The perception of poor code quality in Netscape's products is persistent to this day. Additionally, your customers may not uncover the truly insidious bugs until an absolute catastrophe occurs.

Provaré Observation: We totally agree. Your reputation is arguably the most valuable intangible asset your company owns. This is the foundational reasoning behind the millions of dollars spent on branding and trade marking. Damages to reputation last longer and are more costly than many people realize. Customer support, sales, and technical support, in addition to the programmers and testing staff, all incur increased costs due to bugs.

(Wrong) Reason 4: "Anybody qualified to be a tester doesn't want to work as a tester."

Joel says "With testers, like programmers, the best ones are an order of magnitude better than the average ones. At Juno, we had one tester … who found three times as many bugs as all four other testers, combined. I'm not exaggerating, I actually measured this. She was more than twelve times more productive than the average tester."

Provaré Note: At Provaré, we have found this to be demonstrably true. This chart shows actual stats for defects submitted by the test and development team over a period of 2 years (the data came directly from the defect reporting system of one of my former employers). Note that the two truly outstanding test professionals submitted almost 3 times the bug reports of any other submitter. The 3 other testers on this chart are practically indistinguishable from the development team in terms of the number of bugs found and reported.



Joel further says that he believes that the problem is that many of these exceptional testers will not be sufficiently challenged at most companies and will then move on. One suggestion is to allow these testers to begin the process of test automation to challenge them and to bring the overall quality of your test team up. One way ***not*** to handle this is to tell all programmers that they must start out in QA. Programmers do not make good testers.

Provaré Observation: This is where we may disagree to some extent with Joel. It is true that good test engineers need to be challenged, but not necessarily the way that Joel suggests. In our experience, when companies try to placate good (but bored) test engineers by allowing them to begin automation efforts, they often make a very poor trade. Creative manual testing by a top

notch tester is, in our experience, the absolute biggest bang for your testing buck. GUI level test automation is at the opposite end of the scale – the lowest return on your investment. This is why we're big advocates of Test Driven Development as the automation methodology of choice.

But another huge contributor to test engineering turn-over is the environment in which most test engineers find themselves. Often, high-value test engineers are not recognized for the contributions they make to the product. Or maybe the entire test team is seen as a "necessary evil" rather than as a strategic part of the product design and distribution. When good test engineers are treated as second class citizens, they'll naturally want either to move into development or to leave your company. Either way, you lose a good resource.

Test Engineers must be challenged and allowed to think outside the box to remain content as "testers". You might be surprised at how much just demonstrating to your test team that you value their work will improve their morale and performance. (By the way, this is one reason of many that we really like agile development models like Scrum. Implementing Scrum requires that test engineers are included in every step of the process. Any agile process that you consider should have the inclusion of testing and documentation specialists as a non-negotiable requirement.)

One of the foundational principals of Provaré Technology has always been to create an environment where truly outstanding test engineers would know beyond doubt that they are valued and respected. This has allowed us to recruit and retain the best engineers from organizations that do *not* value them.

(Wrong) Reason 5: "I can't afford testers!"

Joel says "Skimping on testers is such an outrageous false economy that I'm simply blown away that more people don't recognize it." The costs to your programmers and product reputation is simply too great to ignore testing. Testing is a vital part of product design and release. While you may not be able to afford 10 mediocre testers, one exceptional test engineer will give you a solid return on investment of quality, time to market, and optimal use of resources.

Provaré: We couldn't agree more. In fact, I spent considerable time in 2005 studying the economics of testing. I wrote a white paper demonstrating that there is an optimal amount of testing and presenting some ideas on how to know whether you're there and how to get there if you aren't.

For Joel's complete article, visit: **http://joelonsoftware.com/articles/fog0000000067.html.**