Testers and Developers Think Differently

Understanding and utilizing the diverse traits of key players on your team *by Bret Pettichord*

Software development requires various talents and roles. This article takes a look at two of these roles: testers and developers. On an effective team, testers and developers complement one another, each providing perspectives and skills that the other may lack. A common mistake is to see testers as junior developers, and to thus encourage them to emulate the skills and attitudes of the developers. In fact, good testers have many traits that directly contrast with the traits good developers need. With understanding, managers can make the most of these divergent skill sets and use them to build a successful team.

Many developers don't realize how difficult system testing can be. It takes patience and flexibility. It requires an eye for detail and an understanding of the big picture. Many testers are frustrated working with developers who think testing is easy. My understanding of this dynamic was deepened when I got a chance to see several developers take over the system testing. They had been used to relying on a team of dedicated testers, who were no longer available to test the product.

Since I had experience testing the product, I trained them during this project and saw them struggle with tasks they had previously underestimated. In the process, I noticed some differences between the skills and attitudes of testers and developers.

QUICK LOOK

- Exploiting specific traits to the benefit of a project
- Allowing differing styles to complement each other

I want to avoid stereotyping. There are, of course, many good developers who can test well, and there are many testers who can write decent code. But in many ways, the skills that the jobs require are different and often in opposition to each other. Understanding this is important to getting teams of different people to work well together.

Embracing "Dilettantism"

A dilettante is someone who dabbles in a practice or study without gaining mastery. The term is often used to disparage a lack of commitment or seriousness, but good testers *are* able to make judgements even when they may not have mastered the specific subject at hand. Their strength is often their ability to be generalists; they need to have a broad understanding of many areas. This contrasts with developers, who are often required to be specialists in particular technical areas (networking protocols, for example, or database internals or display libraries). Testers, in contrast, need to be able to get up to speed quickly on any product that they test. They often have little time to learn a new product or feature.

On the other hand, developers need to have a thorough understanding of the libraries or protocols they'll be using before they can work on something new. Otherwise, they may break something. They're used to being allowed to gain mastery before being expected to deliver. On my project, I saw developers who were used to this kind of arrangement really struggle when they had to test dozens of features that were new to them in a short period of time.

Testers need to have the kind of knowledge that *users* have. This helps them use the product the way a user would, instead of the way the developer might like them to. Thus it can be

important for them to not be familiar with the internal architecture of the product or perhaps even to act as though they don't know it when testing. The tester has to value a certain kind of ignorance or naivete. Developers, of course, can't afford this kind of ignorance. Developers may see this kind of ignorance as a sign that a tester isn't smart enough or is even being difficult. I had great difficulty getting developers to test from a user's perspective and put aside the knowledge that they had about the product internals.

Good testers *are* dilettantes. They need to know something about the customer's domain, as well as information about computer systems. They need to be good at gaining superficial knowledge. They may even have to resist becoming too knowledgeable—or at least be able to pretend that they don't know things that they really do. Developers need to see the value in this. This doesn't indicate a lack of intellectual ability—it's just a different attitude about acquiring knowledge.

GOOD TESTERS
Get up to speed quickly
Domain knowledge
Ignorance is important

GOOD DEVELOPERS
Thorough understanding
Knowledge of product internals
Expertise is important

Modeling User Behavior

Having good models of the users' behavior—and of the system design—is extremely important to software development. Good developers and testers need both. But there are times when each discipline stresses different talents and calls for different emphases. As system designs become more complicated, developers have to spend more and more time and effort making sure that they understand the designs. At those times, having testers take responsibility for understanding user needs and behavior can be very helpful to making sure that the product is tested in realistic scenarios.

A good example of this is error testing. Good testers know that users make numerous errors when learning and using new products. We all do this. More people prefer learning by trying and seeing what happens than by reading a manual. Thus it is important to test software with small errors, typos, and nonsensical requests. Tests need to make sure that users can change their mind and undo partially completed work. On the other hand, the complexity of software products often requires developers to concentrate more on what needs to happen to fulfill a *well*-defined request. To do that, they need to focus on the system design.

PERSPECTIVE FROM A TESTER

JONATHAN BACH takes his job as Test Lead for Microsoft's Systems Management Server team seriously. But he describes his job at the software giant as pure play. "I love the diamond in the rough," says Bach. "Testing is like hide and seek, scavenger hunting, taking the metal detector onto the beach." From his perspective, testing is a treasure expedition, along with all the inherent tedium and risks. "Just like prospecting for gold," he says, "you're going to come up with a lot of useless dirt." The payoff, for Bach and his fellow testers, is finding that one thing buried in the rock that will prevent the customer from ever calling up Product Support.

That, says Bach, is something developers don't always understand. "Testers don't set out to break developers' software," he says. "As Cem Kaner has said, the software comes to us already broken. We just want to find the breaks that are already there."

He remembers what it's like to work on the development side, though—to create something out of nothing, and then have to hand over your child for critique. "When I was writing test apps for Visual Basic a few years ago," Bach remembers, "I had the same feelings when the time came to hand my work over to my tester." He understands the potential hit to the ego at that stage of the reviews, but also remembers how his pride of ownership made him *want* his apps tested thoroughly. "When that product was delivered to the customers, they thought it was really cool, and I wanted to be able to take the credit for a solid application. Good testing helped me do that."

Now back in his full-time tester role, Bach depends on the developer to help him build that same kind of partnership. "I exist to make their code better," he stresses, "to be their bodyguard, their advocate, as well as the customer's." Before the developer's code leaves the building, Bach says his team wants to do everything possible to make sure no one is going to be embarrassed by the quality of the product.

The development team can help, Bach says, by helping steer testers toward any potential danger zones in the code. "You [the developer] know the code better than anyone else," he says. "It's great when a developer can feel okay about telling me 'Here's where the risk is in my code' or 'This is new code.' "

Bach says that developing good relationships between testers and developers is vital if that kind of sharing is going to happen. "If we both understand that synergy, that two people with totally different perspectives will always find different bugs," Bach says, "then we'll invest in each other to take advantage of that resource." *—A.W.*

This difference in focus can also be seen in typical responses to bugs. Testers will often assess a problem in terms of the potential severity for a customer. This is done, perhaps tacitly, by having an intuitive model of how customers will use the product. How likely will customers encounter the bug? What might the consequences be?

Developers focusing on system design may respond to bugs in one of two ways. First, they may dismiss a problem. "Why would anyone want to do *that?"* On the other hand, they may see the bug as an interesting problem even if its effect on functionality is not that important to customers. In these cases, the developers are primarily working from their model of the system behavior. In the first case, the software is working in accordance with the design, so they are dismissive. It works as designed. In the second, the problem may show a flaw in the design or at least in their understanding of the design. Of course, it's possible that this small consequence indicates a serious design flaw. This is why it's wise to encourage developers to investigate these kinds of problems. Having different people take different approaches to analyzing the software increases the likelihood that serious problems will be detected and removed.

I saw these differences in approach on my project. It showed up in the test designs, where the developers sometimes didn't see why certain tests were important. The tests, for example, may have been redundant from a system design perspective even when they covered distinct user scenarios.

It also showed up in analyzing test failures. On a couple of occasions, a lot of time was spent tracking down what I perceived to be minor anomalies. Although the software behaved as a user would reasonably expect, the results were not what the system design would lead one to expect—so the developers were naturally driven to analyze them.

Testers need to be encouraged to understand the customers' needs. When testers are recruited from customer support this happens automatically; it also happens when testers are recruited from actual customers or from the targeted user community. Don't let your testers be treated simply as developers' assistants. Make sure that they get exposure to customers—and to the people at your company who work directly with the customers.

GOOD TESTERS
Model user behavior
Focus on what can go wrong
Focus on severity of problem

GOOD DEVELOPERS
Model system design
Focus on how it can work
Focus on interest in problem

PERSPECTIVE
FROM A DEVELOPER

TOM FLAHERTY admits it: If software development were left solely in the hands of the developers, applications would be pretty much as they were twenty-five years ago.

"Not in terms of what functions they perform," he's quick to clarify, "but in the sense that only people whose vocation is computer technology would be able to use our products." And that, he says, is just part of the way he and his fellow developers naturally look at software. "We sit there all day thinking about the insides of an application," says Flaherty. "Unless it's a piece of software for which we'll end up being the primary users, we think more about what the inner mechanics of function will make possible—not about the million different ways an end user on the outside might try to extract that functionality." Developers depend on testers' abilities to look at the product the way the real world will see it, manipulating it from a user's point of view.

Throughout his eighteen years of programming, Flaherty has relied on his testers' unique perspectives to balance his own approach. "I don't have hours to play with each part of the product," he says, "trying to pretend like I don't know how it works, trying to overcome my biases about how things *should* work or how users *should* interact with my code." You can make a valiant effort, he says, but sometimes developers' efforts to empathize with the unknown key tapper can only go so far. You *do* know the product, and your brain travels in those familiar paths. "The testers can find things in a product you weren't even aware of."

Flaherty depends on his QA team members for other reasons, too. "Sometimes," he says, "we developers walk a very fine line in how much time and attention we can spend thinking about bugs." Do only a brief look for functionality showstoppers before handing code off to the testers and the testers grumble. Spend a week of your development time investigating one bug and your manager is unhappy. "I work out where that line is and rely on a good testing team to do what needs to be done," Flaherty says. "Better to let the experts spend time on bug-finding because testers are usually faster, more efficient, and better at finding those elusive bugs that, for us, are not reproducible, no matter how many times we try." Testers' abilities to find those bugs through repetitive testing earn them Flaherty's admiration. "Thank goodness a tester's mind works like that; to do that, to focus after the thirty-ninth run of the same test…would drive most developers out of their trees." *—A.W.*

Thinking Empirically

Good testing is governed by the scientific model. The "theory" being tested is that the software works. Testers design experiments, as Kaner says in *Testing Computer Software*, to see if they can falsify the "theory." Good testers know how to design experiments, and they often benefit from previous study of the sciences. Good testers think empirically, in terms of observed behavior.

Developing software, on the other hand, is much like *creating* theories. Laws are specified and rules are applied. Many developers have benefited from studying mathematics. Good developers think theoretically.

Developers who are focusing on their *theory* of how the software works might dismiss a bug report that describes behavior not allowed by their software theory. "That can't happen; it's not possible." Good testers focus on what actually happens, and like other experimenters, keep detailed logs. "Well, it did happen; here are the circumstances." People who are used to thinking theoretically have a hard time accepting aberrant behavior without some kind of explanation. Good testers are skeptics, whereas good developers are believers.

In my project, I had trouble justifying to some developers why we considered certain tests necessary. In these cases, they wanted some reason for doubting that the software worked. They wanted me to provide some kind of explanation for why this code might fail. I didn't have much to say. Mistakes happen, so we test. Indeed, the purpose of testing is to observe whether the software actually works as its designers imagine it will. But my developers sometimes behaved as if this attitude was a challenge to their honor.

Don't presume that your testers need to have degrees in computer science. Many excellent testers have training in *experimental* sciences. Look for and appreciate this background in your recruiting.

GOOD TESTERS

- Empirical
- What's observed
- Skeptics

GOOD DEVELOPERS

- Theoretical
- How it's designed
- Believers

Tolerating Tedium

Good testers realize that software testing can often be repetitive. They learn to accept this. Many developers hate repetition and are often very good at finding ways to automate it. After all, one major purpose of computers is to perform mental drudgery.

On my project, the developers initially expected to automate and improve much of the test process. I had a hard time discouraging this and getting them to focus on executing tests and looking for bugs. It is a challenge to do repetitive work; but it's important and often necessary. Good testers must be able to remain alert and attentive, even when applying the same test for the fourteenth time.

Living with Conflict

Good testers must not shy away from argument. It's their job to report bad news—and this is not always taken well. Sometimes it's hard to pinpoint the source of a bug. Is it a design bug, a coding bug, or maybe a documentation bug? Or perhaps the tester is just confused, and it's not a real bug at all. Regardless, the tester's job is to report the problem. Some testers may go overboard, thinking it's their job to pass judgment on the developers or force them to do better work. This is not helpful. But good testers need to be the kind of people who are not afraid of other people's reactions to bad news.

On the other hand, developers often need to avoid the emotional intensity that makes concentration difficult. It is not easy concentrating on complicated technical material hour after hour. I saw developers on my project take much more time diagnosing problems than I would have. It is important to report problems as soon as they've been confirmed, and good testers will report a problem once it is reproducible and narrowed down. I saw developers, however, move right into debugging, identifying the faulty code and the build that introduced it. Part of this is attributable to the fact that they had the skills and interest in debugging the code. But it also seemed as if they were reluctant to report that they had found a problem until they knew exactly what they had found. They wanted to avoid a situation where people started speculating as to who or what may have a caused a problem.

When I interview potential testers, this is one of my favorite questions: "What would you do if a developer rejected your defect report as 'works-as-designed'?" There are lots of good answers to this question. They might try to get more information from the developers about why they think it isn't a defect. They might try to find a more dramatic instance of the defect that can't be so easily dismissed. Or they might contact customer support regarding their opinion. Here, the wrong answer is to not *have* an answer. Testers need to demonstrate that they can continue to think and work in the face of conflict. I want to see that they will be tenacious, and that they have some faith in their own perspective. Some people have strong technical skills, but are very uncomfortable in these kinds of situations; these candidates rarely make good testers.

GOOD TESTERS

- Tolerate tedium
- Comfortable with conflict
- Report problems

GOOD DEVELOPERS

- Automate tedium
- Avoid conflict
- Understand problems

In Summary

Appreciating differences is critical for productive teams. Different approaches aid in finding solutions, and mutual respect dramatically improves group problem solving. Testers should not be judged according to developer criteria. Empirical thinking is an asset rather than an inability to think theoretically. A jack-of-all-trades should be appreciated rather than criticized for being a master of none. Many of the skills and attitudes that good testers

demonstrate contrast with what we often look for in developers. When hiring testers, look for, and develop, these skills. Don't just settle for junior developers. Productive teams with both developers and testers need both skill sets. Just as developers have defined career paths, so should testers. To remain competitive in this industry, nurture both the skills of developers and the different but equally important skills of testers. STQE

Bret Pettichord is a test automation engineer at Tivoli Systems. He edits the Software Testing Hotlist (www.io.com/~wazmo/qa) *and frequently speaks at software testing conferences. Bret can be reached at* b.pettichord@computer.org.

STQE magazine is produced by STQE Publishing, a division of Software Quality Engineering.