Improve Tester-Developer Relationships with Helpful Feedback

By **Michael Stahl** - October 28, 2019
Share URL
Bookmark

Summary:
Testers and developers often have a strained relationship. Each side has a certain level of expectations as to what the other side should know and do, while there is little understanding of the constraints, conditions, and requirements that the other team has to work within. But it does not have to be this way. A little effort in giving more specific and helpful feedback can go a long way toward improving attitudes.

Everyone knows that cats and dogs don't get along. One of the less scientific explanations I got for this is that it's all about bad communication. When the dog wags its tail, it's saying, "Everything is cool; let's play." But as far as the cat is concerned, what the dog said is, "Look out! I'm really irritated!"

Not that I am calling anyone a dog (or cat), but relationships between developers and testers often are similarly strained. It may just be the natural "us vs. them" mentality that teams develop as a means of group identity. Sometimes, though, it reaches an inappropriate level of disrespect to the work and efforts that the other team puts in.

I think that this situation, at least in part, is a result of bad communication. Each side has a certain level of expectations as to what the other side should know and do, while there is little understanding of the constraints, conditions, and requirements that the other team has to work within. Lack of understanding about how the work of one team impacts that of the other adds to the already existing tension.

This can result in one team getting impatient with the other team and feeling that they are a lost cause. Once this sentiment sets in, it's a short way to losing any chance to learn from each other.

For example, let's say that our team of testers arrived at the conclusion that the reason the developers write unclear design documents is that they just don't care about anyone (namely, us) who needs to read and use these documents. As a result, when we get yet another badly written document, we just keep complaining to each other instead of attempting to work with the authors to improve their writing skills. Similarly, when a developer rejects a bug with the claim that we did not provide enough details, we will just tag that person as someone who's lazy and plays for time instead of just doing the job and fixing the bug.

But it does not have to be this way. A little effort can go a long way toward changing this attitude. Here are two stories from my experience that can teach us something.

A developer sent me a long letter complaining about the low quality of a bug report submitted by my team. The basic claim was that the provided details are not enough, that the reproduction steps are lacking, and that our poor reporting skills caused him to lose much time. So far, nothing

new; if you've worked long enough in testing, it's guaranteed you too have gotten such a letter at least once.

What made this letter unique was the rest of it. The developer explained in detail exactly what caused him to lose time and what changes in the bug report would have saved this time. He pointed to some basic investigation steps he expected the tester to do in order to isolate the problem. He showed how a small change in the provided details would have saved him from going in the wrong direction, why the test scripts we referenced don't work on his machine (and how to fix this), and why it was difficult to reproduce the bug using the steps in the bug report. The tone was not one of "Look how bad you do your job"; rather, it was "Please understand the impact of what you do on what I do, and use this knowledge to do a better job in the future."

Apart from the fact that the details in the letter were insightful and could be used in bug-reporting training, it also increased our appreciation of this specific developer. Instead of writing a nasty "If you had any idea how to write a proper bug report, we would all be better off," he put effort into explaining in detail the weaknesses of our report, how it impacted his work, and what could prevent or mitigate the problem. He also was extra careful to make sure that not only were the details correct, but also the tone was one that we would be willing to listen to. Naturally, we continued to listen to comments from this developer going forward.

Taking this to our side of the divide: If something the developers do disturbs us, we can get annoyed, complain, and perpetuate the image of the developers as people who do not really care about quality; or we could reflect on what exactly the problem is and how it negatively impacts our work. It does not call for a big operation. Each of us can just do it without much ceremony. All it calls for is the willingness to give up the immediate gratification gained by sending an acrimonious message. Instead, invest some time and thought into pointing out things that can be fixed or improved.

As part of my job, I get to read quite a lot of architecture, design, and test documents. Many of them are hard to understand, not so much because of the technical content, but because of the way they are written. They abound with complicated phrases, sentences that can be understood in more than one way, and a lack of consistency in naming objects.

One option is to sigh and accept that this is just how it is. A different approach is to write a cryptic comment: "Unclear." The third option, one that I use often, is to explain what bothers me in the text and why I think it needs modifications.

This takes a lot of time, primarily because it means that I write many more comments: not only content-related comments, but also editorial comments. I explain how the existing phrasing can be understood to say different things or is impossible to understand at all for lack of some critical information. If the text contradicts something written elsewhere in the document, I give a reference to that section or provide a quote. When something is written in a complex and convoluted way, I try to provide an alternative way to say the same thing. This usually means writing, editing, and rewriting a few times until I am satisfied.

My goal in all this is to help the authors improve their writing instead of only providing negative critiques. Sometime the result is that the authors do indeed improve the document. Hopefully it has some lasting impact on their writing skills. Of course, there are cases where all the editorial comments are just ignored, and there are those rare and satisfying cases when someone actually

says thank you. Conversely, when I happen to get a well-written document, I make sure to thank the authors and to commend them to their manager.

When you take this constructive route in any aspect of your work, the feedback needs to be very specific in order to be effective, since the person receiving the feedback likely has skills and knowledge that are different from yours. When other developers read a design document, it may be totally understandable to them. To the testers and other stakeholders, it may very well be obscure. Therefore, when providing feedback, we need to help the authors see their text with our eyes by being verbose in our comments.

Similarly, for the tester who writes a bug report that a developer has a hard time following, certain details about the test setup and conditions probably seem so trivial to the tester that there is no point mentioning them in the report. When a developer asks for additional details for something that the tester sees as self-evident, it is easy to interpret the request as annoying or to mark the developer as lazy or just below par. However, when the comment or request is accompanied with a proper explanation, there is a much higher chance that the tester will see the developer's side, will accept the comment as constructive, and will provide the missing information.

The tension between testers and developers is, to some degree, natural and unavoidable. However, there is much that can be done to use this tension not as a destructive force, but for the mutual improvement of all involved.