DELIVERING UNWELCOME NEWS TO DEVELOPERS

Knowing how—and when—to report a defect

BY KAREN JOHNSON

WE SPEND CONSIDERABLE TIME learning the technical skills of our positions. We make sure we are practicing good testing techniques. We keep up to date on the latest technology. But what about the people skills of our position? What about communication? We sometimes forget that we spend much of our working time *communicating* with people about the state of the software we're testing. And quite often, the news we have to give is not welcome. Learning to deliver unfavorable news to someone gracefully and respectfully is an important skill for any professional, but is an especially important one for quality assurance. How well you present a defect to the developer can impact when a defect is resolved—or whether it is resolved at all. Deliver the information abruptly or inappropriately, and you run the risk of alienating a person or creating project hot spots that aren't needed. Deliver news too passively, and your report may be discarded. Communication is crucial to QA—and while you cannot control the reaction and response from someone, you can control the manner in which you deliver news.

I worked as a technical writer for seven years and gained an appreciation for being on the receiving side of criticism and edits. Those experiences culminated in some guidelines to follow on how to present news, especially unwelcome news, to someone else at work. Some basic rules you learned early in life apply: Don't cry wolf (that is, don't report an issue before you know the facts). Don't whine. Don't invent stories. Don't exaggerate (that is, don't make a defect sound more important than it is). Beyond that, there are three key factors to consider when you tell a developer about a defect:

- Calibration
- Timing
- Method

QUICK LOOK

- 3 keys to presenting a defect
- Handling difficult encounters

Calibration

Calibrate your conversation to the importance of the defect. Learn to prioritize. Every defect you find is not worthy of a conversation. It is not uncommon for a large or complex system to have a backlog of hundreds of bugs. You cannot and should

not turn every bug into a conversation or review session with the developer. When a bug does require a conversation, adjust your presentation and your demeanor to suit the severity of the defect.

The two most important factors about a defect are: How likely is it to occur, and what is the impact? Know the answer to these two questions when you follow up with the developer. Use the most likely scenario when describing how and under what conditions a customer will encounter the defect. At the same time, keep in mind that if you make an issue sound too far-fetched and unlikely to occur, the developer will approach the issue with a lack of enthusiasm or disregard it completely. Make it a habit to know the impact and likelihood of a defect, and include that information when reporting it.

If the defect is important (if it presents a significant impact or is highly likely to occur), try to meet with the developer in person. Be prepared. State the defect and the test case that exposed it. Give the developer time to digest the information. Answer questions. While your general mannerism does not need to be deathly serious, I recommend avoiding jokes or other light-hearted chatter when discussing a serious issue. Be willing to explore similar test conditions the developer might suggest. Think of and treat the two of you as a team on a mission to resolve the issue. Don't just drop the bug at the developer's door and walk away. The more serious the defect, the more willing you should be to invest time testing and fine-tuning your findings.

For minor bugs, email or a casual mention are more appropriate. For instance, you might tell a developer in a hallway passing or in the microwave line in the lunchroom that you found a few minor defects and recorded them in the bug system, but do not need to discuss them. Or mention you uncovered some annoyance defects while testing other conditions. This communicates the fact that you know which defects are unimportant and that you were not wasting valuable testing time; you just noticed these defects along your testing path.

At one company where I worked, we often ate lunch together in a small kitchen. We would talk about coding obstacles, defects found, and customer expectations in a less formal setting. Occasionally, we would joke about some of the defects we found. By having a casual setting to comment about nuisance bugs, I was able to convey to the team that I knew the difference between important flaws and minutiae. And by casually discussing some of the minor bugs in the system, we were able to laugh together—which is one of the best team-building activities you can participate in.

How you report a defect affects how the development staff and the team perceive you. You don't want to informally train the team to believe that instead of finding the important flaws first, you find unimportant and insignificant defects on a regular basis. Thoughts like "Is she presenting us with another nit-picking problem?" or "Has he been testing a scenario or test condition that only a testing person would run?" can be translated into team members thinking you are wasting a developer's time or you aren't testing the important conditions. This doesn't mean you shouldn't record or report nuisance defects, it just means you need to adjust your delivery of the defect in relation to its importance.

Timing

There is an old expression that timing is everything. You cannot change the timing of when you find a bug, but you can control the timing of your delivery. You will find that people are in different moods based on timing. A person's mood affects how they receive your message. Two particularly sensitive times are the end of the day and the end of a project. Make the timing work for you and your bug by choosing the best possible time to deliver your news.

It does not take much insight to see that when someone is packing up and leaving for the day, you won't receive the same patience and consideration you would have earlier in the day. Can it wait? That depends on the importance of the bug and the timing of the next release. Whenever possible, honor the person's time commitments. Do not trail someone out the door with reports of a defect.

Ideally, we'd find all the defects earlier in the project and give developers as much time to fix them as possible. But real life is much different. Bugs often appear right up until the last day (and beyond). At the end of a project, developers' tensions run high and the stakes for finding and repairing bugs are also high. If you find a bug of show-stopping quality, you have a serious and difficult message to deliver. Handle it accordingly. Know your facts. Be prepared to encounter a difficult reaction, including questions as to why the bug was found "so late."

One time, I had to report news of a serious defect late in the project. I brought the defect straight to the R&D manager to give him time to deal with the ripple effect of the defect. He demanded to know why the bug was found so "late in the game." I had no choice but to admit the truth: it was a test condition I had not covered earlier. I hadn't understood the requirements well enough to plan the test condition sooner. I took my "beating" for a late find. After all, everyone makes mistakes (quality assurance people should know this better than anyone because of all the issues we see), and we are no exception. We make mistakes when we fail to run a test condition, or when we skip a test condition, or when we fail to understand the requirements and pass a test that should fail. There are many opportunities for anyone to make a mistake. That's another reason why it's important to deliver criticism with tact, respect, and grace—because you may be on the receiving side of criticism from the same group of people someday.

Method

You have several choices when it comes to delivering your bug: personal conversation, voicemail, email, and automated notification are the most common. You should choose your method of communication based on the importance and timing of the bug.

■ **Personal Conversation** If the defect warrants taking the time, whenever possible, I opt to talk face-to-face with someone. In person, people are more likely to behave as they really are and not hide behind emails or phone tag. Personal conversations also give the receiver an opportunity to ask lots of questions—and provide the deliverer a chance to listen and learn. Since it is well known that bugs live in clusters, you can ask a developer if there are other places in the code that might pose similar issues. Together you can brainstorm. A developer can help you assess what other testing you may want to delay until the bug is resolved and the code is updated. Every conversation you have with a developer gives you an opportunity to build rapport. In person, a

developer can see not just the defect you are reporting, but that you are sincerely concerned about the product, take your role seriously, and are trying to help, not that you're gloating about finding someone's mistake.

■ **Voice mail** Use voice mail to leave a personal message to a developer regarding a bug that you normally would have discussed with them personally, but for whatever reason could not. For example, suppose you find a significant defect and enter the bug into your bug tracking software. The bug tracking system provides email notification of a bug, but the developer has already left for the day. A voice mail with some detail and reference to the bug shows you tried to talk with them in person. Let them know when you will be available to discuss it.

■ **Email** In a world of telecommuting and flexible hours, where not everyone has the luxury of talking directly, email provides another method of communication. But use it carefully. Wording can seem quite terse in email. The ability to copy people or to send the same message to multiple people is convenient and tempting, but don't send the first notification of a significant defect to anyone other than the developer. Avoid copying people on emails for political reasons. Consider how you would react if your boss was copied on an email notifying you of an error you had made. Give the developer the first notification.

■ **Automatic notification** If you work with a system that automatically sends email notifications to developers, your decision will be 1) is it necessary to contact the developer in addition to the email notification, and 2) if so, when and how?

Communications That Go Awry

What if your communication does not go smoothly? What if the developer begins to spew frustration, to argue, or to launch an attack against you, the messenger? Will they take their frustrations out on you? Perhaps, if you let them. But *quality assurance must never shrink away from difficult encounters.* The job function actually ensures you will have some, if not many, tough conversations.

One time when I delivered the news of a critical bug, the developer exploded—at himself, not me. I listened to him curse,

Delivery Dos and Don'ts

DO

Respect the other person
Respect the developer and the developer's work. Consider how it would feel to be on the receiving side of your report. Think about the Golden Rule: Do unto others as you would have them do unto you.

Consider your timing
Having a newly found and important defect reported in a team meeting can be difficult for a developer. Respect each developer's right to be informed and do not blind-side team members in group settings.

State the defect factually
Know your bug. How did you get there? What does the impact of the bug mean? How likely do you think it is that a customer will encounter the same defect? If they don't fix the bug, is there a workaround? By being able to remain factual, you will avoid accusatory behavior, gain respect that you know what you are talking about, and expedite getting the bug fixed. Another Golden Rule is think before you speak. Be prepared with bug details when delivering news of an important bug.

Reproduce the defect before reporting
A common phrase among developers is "If I can't replicate the bug, I can't fix it." I don't believe this phrase. I've worked with developers who will take an inkling of a bug and pursue it admirably. But not all do. The likelihood of getting a bug fixed that you cannot replicate is seriously diminished. Give your defect a chance of getting resolved by being able to reproduce it before you report it.

DON'T

Don't accuse
Stay clear of making bugs a personal issue. The developer may have put the bug into the code, but do not accuse the developer on a personal level. Talk about the bug in the third person: the bug does this, the bug occurs when … versus when *you* coded this or when *you* did not unit test that. Stay focused on the bug and not the developer.

Don't dramatize; bad bugs are recognizable on their own
Do not dramatize the importance of the bug. A bug's importance is generally easily understood by the developer and the team once the word gets around. If the bug prevents other testing from continuing, just state the fact clearly, and preferably only once. See if the developer has a workaround. Ask when you can expect the fix. But do not get involved in "the sky is falling" language.

Don't whine
Even though testers are paid to find bugs, sometimes finding a bug especially late in the cycle is a headache and an inconvenience. This may be the third build in two days and a bug is preventing additional testing from moving forward. Nonetheless, avoid complaining about retesting and about delayed schedules. Avoid adding to the frustration level of the developer and yourself. This conversation is about the code and about a flaw with the code. Stay focused on the bug's significance and not the impact the defect had on your day or will have on your testing schedule.

pound the desk, and slam the office door. I stayed silent and calm. I knew the anger was about his disappointment with his code and not about me. Do not take reactions personally, unless they are intended to be personal. Once his storm had passed, I reminded him of all the solid code he had built. I reminded him that he was one of the best developers I had ever worked with (which was true). This is the right time to step away from being factual and to be human. You can feel sorry that the developer is frustrated. You can offer sympathy that he may need to work late to fix the bug right away. You can empathize with her frustrations, but you cannot apologize for doing your job. We talked in detail about the test conditions we would cover when the code was ready. When I left his office, I felt we were a team and that we had a plan to resolve the issue together.

What a different result the conversation could have had if I had taken his reaction personally. What if I had reacted to his initial response? What if I left when he was frustrated and angry? What if I left him with the defect and no plan for retesting? I'm tired of the expression win-win, but not tired of how it feels when you can reach a win-win conclusion with someone. Negative situations actually provide an opportunity to bond together when handled well; and conversely, can provide an opportunity for division and finger pointing when handled poorly.

Sometimes it helps to think psychologically. Consider a person's emotional intelligence quotient (EQ). Emotional intelligence may be best described as a person's ability to react and respond appropriately to a variety of situations. Emotional intelligence is a trait that you hope the other person possesses when you critique his work; but not everyone has a high EQ. Some people rant, rave, and even swear. Again, you can't control someone else, but you can be responsible for *your* behavior in every conversation.

Do not provide an audience for the over-reactor. This might include not telling the person the news when other people are around; limiting the amount of time you spend listening to their reaction; or not starting or participating in an email sparring contest. Be patient when detailing your findings and when listening to the developer's response. You, like other professionals, do not have to listen to swearing or name-calling. If the person overreacts this poorly, excuse yourself calmly without stomping away. Be patient, but do not tolerate poor behavior. Remember that you train people informally each day in how they can or cannot treat you.

It may be a good sign when a developer becomes defensive and starts listing all the unit testing she *did* perform. The developer wants you to recognize that she put reasonable effort into her work. Remember that frustration on hearing about a bug can be a sign of pride: the person who wants to defend his work is a person who is invested in his work. I have sometimes pulled out a test plan to share how many tests the code has passed. I worked with one developer who was so intent and so concerned about his work that I knew his frustrations were with himself. I would sometimes remind him that everyone makes mistakes, and that I knew that because of the bugs I had seen. Then I would remind him that every bug found by a customer that I had not found was really a bug of mine. Quality assurance makes mistakes, too.

One of the hardest developers to deliver unwelcome news to is the developer who will dig into the new defect, corrupting her current schedule of work. Depending on your development process, developers may be required to get the go-ahead to work on a defect from a project manager or development manager. If you're working with a developer who will become distracted with the most recent bug reported, you may need to remind him that bugs found late in the project schedule may have to be authorized by someone before coding changes are allowed.

Fortify Yourself

Not every conversation goes smoothly. If you still find some encounters difficult and realize you aren't as thick-skinned as you find necessary, work toward making yourself less vulnerable to on-the-job frustrations and disagreements that come your way. Use these thoughts to strengthen yourself:

- Quality assurance does not put defects in the code; quality assurance only finds the defects.

- A customer could call with the same defect; you are giving the developer a chance to fix a defect before it is found in the field.

- When a developer blows off steam in your presence, remember she is probably frustrated by her own work, not necessarily by you or your work.

- Even if you could have found the defect sooner, you still found the defect and that is what you are paid to do.

Summing Up

Work on your communication skills with as much earnestness as your technical skills. Consider the other person. Remember to be respectful. Consider how you present your findings. Remember to have solid facts. Develop skills at handling difficult conversations. Learn not to take work criticism personally. And finally, if you deliver frustrating and oftentimes unwelcome news with respect and consideration, remember you deserve the same professional courtesy in return. STQE

Karen Johnson works as a business systems consultant for Baxter Healthcare Corporation on a rapidly expanding global Web site used internally by Baxter. She has more than seventeen years of experience in computer software; for the past ten years she has been involved in quality assurance. She lives in a suburb of Chicago and can be reached at knj_karen@yahoo.com.

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