Defect Mismanagement

Back in 1974, I was an application programmer at the U.S. Air Force Strategic Air Command (now STRATCOM) headquarters, Offutt Air Force Base. I worked in support of targeting. My end-user, which at the time was a Marine lieutenant colonel, needed a simple program to filter out spurious data from digitized photoreconnaissance data. Recognizing my true worth as a top-notch applications programmer (and not having anybody else to support his needs), he asked me to write the program for him.

In gathering the requirements, I noticed that there would be hundreds of data sets each night. Obviously, I couldn't run 100 different jobs, so I grouped the data together into one large nightly run. I needed a card (yes, it was that long ago – we used punched cards) to separate the data streams, so I asked the end-user what special characters would be present in the final data. The user replied, "No special characters are used at all."

Since I couldn't get real data to test (security issues), I faked up some data separating each data stream using a single card with a '!' in column one. My test worked, and I cut the program over to the operational side of the system.

The very next morning at 2 a.m., I received a call from the computer operator (yes, it was that long ago – we had real live operators to run the card decks) telling me that my job terminated with 0.001 seconds of CPU time. Now, I am a hotshot coder, but nothing ran that fast in Cobol (yes, it was that ... oh, you know). I threw on a uniform, went in, and after getting permission to access the real data, poured through my core dump (yes, it was that ... never mind) and saw that the very first data stream had multiple '!'s embedded in it. My program had seen each '!' as a data stream separator, and did not find enough data in each stream to analyze.

Being righteously indignant, I was waiting for my user to show up at 0730. I quickly pointed out to him that he had asserted that there would be no special characters, but '!' was used frequently in the data. His response was to say, "Well, '!' isn't special. We use it all the time." At that point, I became a Zen master when I realized that the user and I did not speak the same language. What I had logged as a defect was, in reality, nothing more than a simple miscommunication. Isn't it funny how that often tends to be the case?

One of the major failings in the way we currently develop software is that so many things can hide under the cover of a defect. Forgot to include a feature? It's a defect. Have bad code? It's another defect. As a Personal Software Process℠ instructor, I teach that not all defects are created equal. Some come from simple problems and are simple to fix. Others have complex causes and are complex to fix. Frequently, we have defects resulting from simple errors that are costly to fix. And, very occasionally, we hit those that have complex causes, but are simple to fix.

In BackTalk, we sometimes write columns that are cynical and sarcastic. Sometimes we write columns that are funny. And – as in this column – sometimes I get to point out the obvious. The following are "obvious things about defects you (1) probably already know, or (2) should know, or (3) wish your user/developer knew:"

1. If you have ever thought of it as something that can go wrong, it's not unexpected any more, is it? It really can't be an error anymore – it's something you thought of, made a value judgment/risk assessment about, and decided to ignore.
2. If any error can occur, it will eventually. At 3 Gigahertz, lots of instructions are executed every second. Eventually, the most bizarre timing occurrences happen.
3. There is no such thing as foolproof. Nature and genetics are producing 248 births each minute. Some of them are bound to do foolish things at an alarming rate. Don't ever think, "Nobody would be so dumb as to ..." They are. They will. It's your fault.
4. Quit hiding poor requirements as defects. Make sure you do root-cause analysis on each defect – and fix the problem, not the symptom. If you have lots of defects due to poor requirements elicitation and validation, fix your requirements process instead of just hiring additional developers to fix the so-called defects.
5. Don't settle for large numbers of defects. If your code is bad, train and educate your developers to make better code. If your design is bad, get some true designers to help with your architectural, data, and interface design. Designing code is not what design is about. Most of your errors will occur due to poor architectural design, poorly thought-out interfaces, or inefficient data. On the other hand, if your defects are from poor requirements, see No. 4.
5. Developers, you have to give the users what you agreed to give them. If you need to slip requirements or postpone deliverables, communicate that to the users. If you surprise them with incomplete or defect-ridden code, they are going to be unhappy.
6. Users, you have to explain what you need to get code that meets your needs. If you won't commit to what you really need, then you won't get what you really need.
7. Developers, before you move to the next step in your life cycle, verify and validate what you have. There is no use proceeding until you know that what you have so far is correct.
8. Users, if you want it badly, you'll get it badly. If you ask for too much, it will take too much time. You have to compromise. Understand that it takes time to create a software product. You have a choice – the more you push for fast delivery, the less quality you will get.
9. No matter how bad the defect is, it could have been worse. In fact, wait a while, and you'll remember your current problem as the good old days. (OK, I just couldn't keep the cynicism and sarcasm totally out).

– **David A. Cook**
david.cook@hill.af.mil
Software Technology Support Center/
Shim Enterprises Inc.

P.S. It's a column about defects – of course the list is misnumbered!