developer.*
The Independent Magazine for Software Professionals

Two Principles of Conversation

By Kevin Cauble, CCP

Editor's note: *This essay originally appeared in the May, 1995 issue of* The Software Practitioner*, which is still published several times per year by software luminary and author Robert L. Glass.*

I wasn't long into my career as a programmer before I came face-to-face with the two fundamental, yet contradictory, principles of communication among software practitioners:

Principle 1 – There is much to be gained by sharing your experiences, ideas, and problems with your peers.

Principle 2 – Managers HATE to see software practitioners conversing with one another.

The benefits of Principle Number 1 encompass a wide variety of circumstances. At one end of the spectrum is the situation where you are stumped on a problem for hours. A peer walks by and asks about the scowl on your face. As luck would have it, he has had a similar problem and knows a solution, or at least the direction in which to head. In another situation, you show a screen you have designed to another programmer and she immediately sees a design flaw that you overlooked.

At the opposite end of the spectrum is my all-time favorite experience. You have struggled with a design or debugging problem for hours, going over and over it but making no progress. Finally, in desperation, you go to a cohort's office and start explaining what you have discerned about the problem and its possible causes. Halfway through the explanation, the solution comes to you like a bolt of lightning. You walk off thanking the other person profusely for their help. The interesting part is that the other person never had the vaguest idea what you were babbling about and is totally perplexed.

The formalized version of Principle 1 is the "Peer Walk-Though." Presenting your design to a group of co-workers is a great way to find flaws in your system or to get input on better ways to do things. But it also works in other phases of the software lifecycle and in much less formalized situations.

As with other professions, we are generally the last to see the flaws in our own work because we are so invested in what we created. (“It must be a hardware problem, I checked MY code!”)

Unfortunately, there is also Principle Number 2. Managers hate to see all this conversing going on. Managers seem happiest when we are furiously pecking away at our keyboards, banging out code. Or at least frowning miserably over a six-inch thick printout full of arrows, lines, and sticky notes. Or when someone comes in at 2 A.M. to fix a problem. That is when REAL programming work is being accomplished.

I guess managers assume that any time two or more programmers are conversing that they are discussing football, gardening, recipes, how to attain world peace, or any number of other trivial items. (Or perhaps plotting the manager’s overthrow?)

In the defense of managers, it should be noted that Principle Number 1 can be abused. There are programmers that seek help before they have really delved into the problem and made an effort to diagnose it on their own. However, managers should determine if the interaction is detrimental to the organization before they lower the boom on suspected abusers.

Within the first year at my first job as a computing professional, I got a good dose of Principle Number 2. A cohort and I were called into the manager’s office and given a stern lecture about spending too much time in each other’s offices. He looked at me and said, “You need to do your own digging.” He looked at my more senior cohort and said, “And he (meaning me) has nothing to offer you.”

It was a moment of humiliation I will never forget. It seemed to me our interactions were mutually beneficial. I was fresh out of Computer Science school and full of the latest concepts (relational databases, efficient screen design, etc.). He had an in-depth knowledge of our existing applications and operations. Sharing our design or debugging plans and problems made us both more productive and produced better results for the organization.

So what’s a programmer to do? Short of waiting for MIS managers to become enlightened, not much—except try to work around the problem as best we can.

I did come across a partial solution one day. I was stumped on a problem and finally went to my cohort's office to talk it over. When I got to his office, I panicked as I remembered he was gone for a week. As I sulked back to my office, I had a bright idea. I imagined my partner was sitting in my office and I began explaining the problem to "him" just as if he were there. And sure enough, the source of the problem came to me just as if he actually had been!

###

While driving a Red Cross Bloodmobile, Kevin Cauble got bored and joined a local university to take history classes. Unable to get the classes he wanted, he took a computer class as a lark. After the first minute of the first class, Kevin knew what he wanted to do. He graduated with a degree in Computer Science and is currently working as a Programmer/Analyst at a small liberal arts college in North Carolina. Kevin can be reached through the editorial staff of **developer.***.