Editor: Diomidis Spinellis ■ Athens University of Economics and Business ■ dds@aueb.gr

Basic Etiquette of Technical Communication

Diomidis Spinellis

Parents spend years trying to teach their children to be polite, and some of us had to learn at school how to properly address an archbishop. Yet, it seems that advice on courteousness and politeness in technical communication is in short supply; most of us learn these skills through what's euphemistically called "on the job training." With enough bruises on my back to demonstrate the amount and variety of my experience in this area (though not my skill), here are some of the things I've learned.

Talking to Humans

We developers spend most of our time issuing instructions for computers to execute. This type of command-oriented work can easily lead to *déformation professionnelle* (see also J. Bigler's alternative interpretation at www.mit.edu/~jcb/tact.html); I can still remember, years ago, a Navy officer who was talking to his son as if he was ordering a sailor. When we compose a mail message or open a chat window, our keystrokes are directed to another human, not to a shell's command-line interface. Therefore, we should switch our tone to courteousness, kindness, and consideration. "Please" and "thank you" aren't part of SQL (or even Cobol; but interestingly "please" is an important part of Intercal), but they should be sprinkled liberally in every discussion between humans. Are you asking a colleague to do something for you at the end of the business day? This isn't a batch job that a computer will run in the background. Think of how your request may affect your colleague's family life. Ask him whether he can do it without too much hardship, and at the very least apologize for the urgency of your request.

Starting your exchange with some (sincere) flattery can work wonders. This is especially important if harsh criticism is to follow; it will help you express yourself in a more compassionate way and lift the spirits of the unfortunate soul who will read your words. Imagine the feelings of your email's recipient by reading your message again through his eyes; according to human-communication theory, he will interpret the email more negatively than it was intended. Therefore, aim to encourage rather than complain. If your email is especially harsh, don't send it immediately. Put it aside and sleep on it or ask other, more experienced colleagues for advice. Although Google is experimenting with a feature that lets you revoke an email within a very small grace period, in general there's no way to undo a sent message—you can only regret the damage it made.

In technical discussions, focus on technology issues, not personal weaknesses. Read the message, "You indent with horrible inconsistency like a loser" as "the code in Foo.java can be better indented"—this has to do with the code, not you. Similarly, instead of shouting, "Your choices of method names for the class Foo are awful," phrase your concern as, "The methods of class Foo would be easier to remember if they were verbs." More concretely, Linda Rising, author of *Design Patterns in Communications*, recommends the following format: "Deliver an Oreo cookie by saying something nice, then present your suggestion of improvement, then close with an appreciation."

Email Smarts

Every email should tackle one topic and that topic

0740-7459/09/$26.00 © 2009 IEEE

should be the subject line. When composing a new message, don't start a new subject by replying to an old email, this will confuse colleagues reading email in threads. Most developers dislike the overhead of attachments. Instead, use simple text or point to the URL containing what you wanted to attach. Avoid using the reply-all command and when you do, trim away copied recipients who are no longer relevant. When you add recipients outside your organization, ensure that the message's contents are fit for general distribution and that it isn't addressed to a private list. Also, blind carbon copies (bcc) in general are a bad idea. If the blind-copied recipient replies to all, everybody will know you're sending copies behind their back. To notify your reports without having them appear in the recipient list, simply forward them the original message with a small explanatory note. And before pressing that "send" button always reread your message. It takes little time but delivers a significant reward.

Be careful how you manage an email discussion. When you reply, have your email client quote the preceding message; don't waste time repeating it with your own words. However, quote with care, trimming the replied-to message only to the pertinent parts. Most technical people prefer the response to be below each corresponding part of the original message so that they can read the entire exchange in serial order from top to bottom. If, however, someone is using a different convention, follow it when replying; there's nothing more puzzling than replies stacked both on top and bottom of a message.

Our globalized profession creates a big potential for communications problems based on differences in personality, language, and culture. Robert Watson, a member of the FreeBSD Core Team who's spent countless hours mediating disputes between developers, says he's always amazed by how much comes down to simple communication problems. Whether you speak the same language natively and are taking all the wrong social cues, or speak different languages and perceive terseness and briskness as giving offense, failing to realize that the issue is communication and not content gives rise to a remarkable amount of pain! Robert adds that he often benefits from stepping away from the computer for a few minutes, then coming back and rereading email, often to find that the offensive tone he thought he'd read was a product of either his imagination or language issues.

Homework

Computing professionals with ample time to kill are a rare species. Avoid wasting your colleagues' time by doing your homework before communicating with them. If you think you've found a software bug or have a proposal for an improvement, consult the system's issue database to see if it's already there. If it isn't, filing your issue can help you organize your case by documenting it through a specific scenario, a test case, and perhaps even a proposal for a fix.

There are also other sources you should reference for related work. How do competing systems implement your proposal? Is it affected by a specific standard, regulation, or formal specification? What's the history of your idea? Often, digging through the software's version control system revision log can provide valuable insights.

If your proposal involves a code or algorithmic improvement that's supposed to increase efficiency, arm yourself with hard data. Having your sophisticated code rejected because you can't prove that it will actually improve the system's operation or, worse, because it's a regression compared to the code it replaces, is disheartening and humiliating. Demonstrate the superiority of your approach with benchmark results and a measure of their standard deviation—Poul-Henning Kamp's ministat program is expressly designed for this task (www.freebsd.org/cgi/cvsweb.cgi/src/usr.bin/ministat). Thus, you can easily convince your colleagues that your code is worth integrating into a code base. If your contribution comes with test cases that demonstrate its correctness, even better, especially if the code you aim to replace lacked such testing infrastructure.

Every email should tackle one topic and that topic should be the subject line.

In Rome Do ...

There are many more things that will reduce friction in your technical communication. Some are particular to specific organizations. For this, you have to learn and adopt the local practices. In your first weeks in a new group, observe carefully how others communicate and follow their example; don't send a message to a large email list until you're confident you know its conventions. For instance, some communities have an intense dislike for the waste of time associated with so-called bike shed discussions of trivial technical matters (www.bikeshed.org). In technical groups, there are often local rules associated with specific tools: how you write commit messages in the version control system, what you include in the bug database, who can change a page on a wiki. Sometimes, as is the case in Wikipedia, these are spelled out in detail; in other cases, you have to learn them by watching and asking around.

Although a code review with an archbishop is unlikely, you'll sometimes communicate with your organization's big shots. There's no need to be servile in such situations. If you're sincere, avoid technical jargon, and appreciate the priorities and constraints of the higher ups, you'll do fine. Remember! You can't go wrong when you're considerate, polite, and respect other people.

Diomidis Spinellis is a professor in the Department of Management Science and Technology at the Athens University of Economics and Business and the author of *Code Quality: The Open Source Perspective* (Addison-Wesley, 2006). Contact him at dds@aueb.gr.

Post your comments online by visiting the column's blog: **www.spinellis.gr/tools**