Instant productivity gains

Robert M. Poston, Software Standards Editor

“I need help with software productivity now. My boss is screaming. His boss is badgering him. What can I do?” I answer this kind of call from readers several times a week now.

If I start talking about establishing a long-range productivity improvement program or implementing standards or methodologies, most callers drop out of the conversation in a hurry. That’s not what they want to hear.

Research is necessary to develop more productive software engineering methodologies, standards, metrics, workstations, and tools. Long-range planning is required for continuous productivity gains. Implementing new methodologies, standards, or productivity plans changes how an organization works. OK, but that takes time, and the practicing software engineers who call me have precious little of that.

So, here are a few quick fixes for the software productivity problem. They’ll patch enough to hold back the rising costs of software development — while you’re getting your complete productivity and quality improvement program into place (see the May issue’s Standards department).

Software engineering standards

The five software engineering standards below are available from the Computer Society’s West Coast office. Send your prepaid order to IEEE-CS Order Dept., PO Box 80452, World Way Postal Center, Los Angeles, CA 90080. Give the order numbers listed and add a $2 shipping/handling charge.

- Standard Glossary of Software Engineering Terminology (ANSI/IEEE Std 729-1983) defines more than 450 terms in general use in the software engineering field. Where a term has more than one accepted, approved definition, alternatives are listed. Examples are provided to clarify some definitions, and cross references show a term’s relationship to other terms in the glossary. (No. 930, $6.75 IEEE members, $7.50 nonmembers)
- Standard for Software Quality Assurance Plans (ANSI/IEEE Std 730-1984) provides uniform, minimum acceptable requirements for preparation and content of software quality assurance plans. Such plans can then be assessed against the standard. This document applies to the development and maintenance of critical software. (No. 945, $4.50 IEEE members, $5 nonmembers)
- Standard for Software Configuration Management Plans (ANSI/IEEE Std 828-1983) provides the minimum requirements for preparation and content of software configuration management plans. This document identifies and provides examples of the essential items that should appear in such plans. The standard is applicable to the entire life cycle of critical software. (No. 931, $5.85 IEEE members, $6.50 nonmembers)
- Standard for Software Test Documentation (ANSI/IEEE Std 829-1983) describes a set of basic test documents associated with the dynamic aspects of software testing and defines the purpose, outline, and content of each document. It is applicable to commercial, scientific, and military software. (No. 929, $7.20 IEEE members, $8 nonmembers)
- Guide to Software Requirements Specifications (ANSI/IEEE Std 830-1984) describes approaches to good practice in specifying software requirements. Using this guide will help both software customers and suppliers: Customers will be able to more accurately describe what they want, and suppliers will be able to better understand what is requested. Prototype outlines are presented. (No. 933, $6.30 IEEE members, $7 nonmembers)

These five standards are available together in a hardbound volume as Standards 729/730/828/829/830 Hardbound Edition of Software Engineering (No. 947, $22.45 IEEE members, $24.95 nonmembers).

Power to the programmer. Waiting for computer time is a big drag on productivity. There are really three different periods in software development when lack of computer time becomes a bottleneck: (1) when information (plans, requirements, designs, source code, or test input data) is being prepared, (2) when unit tests are being performed (compile/link/load/test/debug/recompile cycle), and (3) when the total system comes up for testing.

In all periods, the programmer’s product is text, but manipulation of the text is different in each period.

Text editing, formatting, and printing are required in the information preparation period. Here, the key to productivity is to provide every programmer with the right amount of text manipulation power. Remember: the kind of editor a programmer requires is much different from what a manager or secretary needs.

Pausing for computer operation every few seconds really eats into a programmer’s concentration. Subsecond response to simple editing commands such as “Delete the previous sentence” is required. (If your editor doesn’t have such a command, a smarter editor may be in order.) Response time under two seconds is necessary for complex editing commands such as global search and replace with full regular expressions (numerous wild cards).

Any IBM PC AT, or clone thereof, coupled with a powerful, commercially available editor (i.e., an excellent one) will take care of these requirements for a modest per-programmer cost.

Time sharers can get these low response times, too. However, as work loads on the time-shared system change, so will the response times. The beauty of a PC or workstation for each programmer is that each person can control his own work load and response times.

Response time is not an issue when it comes to formatting or printing. These jobs can be done when PC/workstation use is not critical.

What is critical is how much work the formatter can do for the programmer. Creating page headers and footers, section and paragraph titles, tables of contents, and indexes and resequencing...
paragraphs after moving, adding, or deleting can be handled automatically by a good formatter. Programmers' work can be reduced further if formaters are set up to work with boilerplates or templates for standard documents.

Long delays usually occur in the unit testing period (sometimes build testing is blurred in here). This is because programmers feel they cannot test the software they are responsible for (unit) without going to the target system. They think they need access to all other software in the target system (infrastructure) to test properly.

Normally, there is one target system, and all programmers want to use it. No wonder the compile/link/load/test/debug/ recompile cycle often takes days or even weeks to complete.

However, programmers do not have to go to the target system to perform unit testing. A typical unit has no more than three or four infrastructure interfaces. Each interface exercises only one or two functions.

In one morning, stubs and drivers can be written to simulate the infrastructure for an average unit under test. Then, all unit testing can be kept at the PC/workstation. A major timesaver!

Using this approach, I can set up my workstation for testing a unit of software in about three hours. My compile . . . test cycles have been reduced to two keystrokes and the time it takes to drink a cup of coffee (about five minutes).

Testing on powerful workstations is not only the most cost-effective thing to do in most cases, it is the easiest thing for the programmer to do. It also encourages programmers to build a little, test a little — a development technique (incremental testing) that has proven very successful.

Usually, by the time we reach system testing, most of the allocated time and dollars have been spent. But we've got to finish the project with what we've got left.

God love the programmer who sends error-free code to system testing! But most of the time that doesn't happen: code arrives with lots of bugs.

For system testers to discover those bugs, they have to use the target system heavily. However, once the bugs are identified, the programmer can take over at his workstation. The computing capabilities are already in place so he can do most of the debugging at his own machine. There is little need for him to tie up the target system.

Standard tools. Once programmers are on workstations, a family of standard tools is in order: standards checkers, test coverage tools, data name and type verifiers, and data-use reporters, to name a few. Of course, a standard data dictionary should be provided.

Most organizations already have at least one or two of these tools. Standardize on those you have and provide them to all programmers immediately. Managers may explain that plans are under way to supply a more complete set of tools at a later date.

If these tools are not provided, programmers will improvise workarounds or perhaps resort to bootlegging. This independent, unchallenged creativity works against team productivity.

Standard templates. In most cases, creating documentation structures is a waste of time. Nearly all information that a programmer must prepare can be arranged in a standard format. When a programmer can call up a complete document outline with one command, he doesn't even have to think about what format to use.

For most organizations, information captured in consistent formats is more accessible and usable than do-it-your-own versions anyway. But setting up standard document templates in the first place can be a bit tricky. Which standards should be used? Mine, yours, the IEEE's?

Be careful not to get side-tracked into political arguments. Remember we want quick results.

In favor of the IEEE software engineering standards, I remind you that they are the only professional, industry level, consensus-based standards available today. You can employ them right away (see accompanying box). Simply enter in your computer system the formats as they appear in the standards and make them available to everyone. Later, as you gain experience in using them, you can tailor them if you like.

I'll close with a one-liner: Learn touch-typing, please. (We should get past the two-finger entry system on those super editors.) Keep the calls and letters coming.

---

**Standards bulletin board**

**User systems interface project**

X3, the Accredited Standards Committee on Information Processing Systems, has approved a study project for user system interfaces and has assigned the project to technical committee X3V1 (text processing for office and publishing systems). X3 will also establish a liaison to the Human Factors Society, which administers the US branch of ISO's technical committee 159 on ergonomics.

Growing concern in the past few years over software ergonomics for workstations' visual displays prompted the study project. The increased medical concerns and legal questions of responsibility for the well-being of computer users also contributed to X3's decision to approve the study.

Project aims include standards for friendlier system that offer some consistency across equipment types and interface designs based on human-factors principles.

Organizations already addressing this area include CCITT, DIN, ECMA, and ISO. Their efforts address user interface devices, dialog interaction techniques, user guidance and assistance techniques, user class and user group considerations, task considerations, and system environment considerations.

---

For more information, contact L.M. Collins, X3V1 chairperson, IBM Corp., 5205 N. O'Connor Rd., Ste. 200, Irving, TX 75039-5050; (214) 556-7690.

**PL/I standard comment period**


The PL/I standard was developed jointly by ANSI and ECMA. Copies may be obtained for $24 from ANSI, Sales Dept., 1430 Broadway, New York, NY 10018; (212) 642-4900.

**Ada group looks at code reusability**

The newly established Commercial Ada Users Working Group will establish objectives and standards for technology development to support and promote a market for reusable Ada components.

People interested in joining or working with the group should contact John Pates, ACT, 16 E. Second St., Fifth Floor, New York, NY 10016.