



Are We Headed Toward a Disciplined World?

Forrest Brown
Managing Editor



Software developers often view software as a stand-alone entity, rather than as a piece of a larger system that includes hardware and users. This can lead to many problems in system development. However, this problem can affect other people in the systems development loop, as demonstrated by the following tongue-in-cheek E-mail exchange:

Project X team lead: To the senior management team: Since last week's update, System X's hardware has remained on schedule, but the software problems are looking even more serious. We're averaging nearly an error per function point on the System X software, and we now project that the software is 18 weeks behind schedule, plus it's already \$220,000 over budget with an additional \$27,000-per-week burn rate. We're still waiting for your input on these and the other software problems we've been telling you about.

Senior management: We're glad the hardware is on schedule. Please stop bothering us with updates on software glitches. Just do whatever it takes to ensure System X meets the delivery schedule in eight weeks.

Project X team lead: Okay, we won't bring up software problems with you anymore. By the way, we have refigured the system schedule: The *system* will be delivered at least 18 weeks late, and

the *system* is already \$220,000 over budget and counting. Please comment.

Senior management: We need an emergency meeting right away. Why didn't you tell us earlier that the system was in trouble?

Efforts to implement better practices and more mature processes have been going on for years now in the Department of Defense (DoD). The Capability Maturity Model (CMM) for Software was developed to "provide software organizations with guidance on how to gain control of their processes for developing and maintaining software and how to evolve toward a culture of software engineering and management excellence" (CMM for Software, Version 1.1, CMU/SEI-93-TR-24, p. 5). In the DoD, it has been the model of choice for software and has improved cost, quality, and schedule performance in organizations where it has been skillfully applied.

Unfortunately, most software organizations still have a long way to go before they will be mature and disciplined enough to avoid the self-inflicted problems that are wrongly assumed to be a natural part of software development. The same applies to the problems caused by the widespread lack of discipline in systems development.

Many improvement efforts are now ongoing in the various engineering disciplines related to systems development. Early indicators from organizations that use the resulting models reveal that the application of the practices defined in these models leads to more discipline, which results in improved schedule, cost, and quality.

Each of the past and present models and standards attempts to move system development toward more disciplined, mature behavior. To help you keep track of all these developments, Randall Wright's article (p. 7) eliminates some of the confusion regarding different systems engineering models that have evolved since process improvement efforts began.

CROSSTALK has long discussed processes and process models for software. Most of the articles in this issue discuss software in the context of the larger system. We hope it will help you better understand DoD's and industry's game plan for future success in systems development. Slowly but surely, we should hope to see the day when most organizations produce systems and systems software with greater discipline and maturity. ♦



Software Metrics Hazards

Elizabeth Starrett's article, "Measurement 101," *CROSSTALK*, August 1998, was interesting and well written, but it left out a critical point. Metrics based on "source lines of code" move backward when comparing software applications written in different programming languages. The version in the low-level language will look better than the version in the high-level language.

In an article aimed at metrics novices, it is very important to point out

some of the known hazards of software metrics. The fact that lines of code can't be used to measure economic productivity is definitely a known hazard that should be stressed.

In a comparative study of 10 versions of the same period using 10 different programming languages (Ada 83, Ada95, C, C++, Objective C, PL/I, Assembler, CHILL, Pascal, and Smalltalk), the lines of code metric failed to show either the highest pro-

ductivity or best quality. Overall, the lowest cost and fewest defects were found in Smalltalk and Ada95, but the lines of code metric favored assembler. Function points correctly identified Smalltalk and Ada95 as being superior, but lines of code failed to do this.

Capers Jones
Software Productivity Research