Process programming: A seductive danger?

Galen Gruman, Assistant Editor

"We now have the second software crisis: maintenance," said Bob Balzer of the University of Southern California as he opened the March 31-April 2 Software Engineering Conference in Monterey, Calif. Software has undergone a major shift from a product in an assembly line to the recognition that software is a process, the conference cochairman argued. "We need to tackle maintenance not only in the maintenance phase but also in design," he said.

"We're moving from a mathematical approach to recognition that software engineering is a design process — an activity of many people over a long time," Balzer said. "The waterfall model is dead. What will replace it?" Balzer asked the 1000 people attending the opening session.

Process programming might replace it, and that proposition caused a strong debate between the conference's plenary speakers and evoked strong reaction from the audience. The opinions were strongly divided. Recognizing the divisions, the conference chairmen set the opening session up as a two-speaker presentation.

One speaker described the promise of process programming, while the other warned that the idea was too seductive for students to pursue because it might lead them astray. A member of the audience decried the focus on process programming, calling the approach dehumanizing.

Software design as software. "Software processes are software, too," argued Leon Osterweil of the University of Colorado at Boulder. The process approach seeks to automate and formalize the software process — treating the software design process as if it were a software program as yet unwritten, he said. The task is to "start with a process description and end up with a problem solver," Osterweil said.

"Software is an aggregate of information and objects tied together in complicated ways. The software product is a complex, messy thing. [But] we have little choice. So how do we do it?" Osterweil asked. "At best, this process is very informally specified," he said.

"Think of it as a data aggregate. You're trying to build a product that is the process of software development. The application domain is the domain of solving software problems," Osterweil said. "Why not program software development? And, dare I say it, with a programming language?" he asked.

"I do not believe you can replace software engineers. But we will change to program portions being executed by programs, and humans — managers — must decide who does what," Osterweil predicted. "We're creating a description that creates another description that goes off and solves the problem," he said.

To be successful, "we must make the process explicit. All too often the process can never be made tangible — it's not reusable. When that person [the designer] dies or is promoted, the effect is the same: The process is lost," Osterweil argued, so the description must be made rigorous.

His early process programming work was an attempt to model the software life cycle, Osterweil said. The waterfall-paradigm code developed had no control flow, no concurrency, and no parallelism, "but going through the effort of formalizing makes the technology apparent and the side effects more obvious," he explained.

But process programming requires more than rigorous descriptions, Osterweil warned. "You have to address not only the procedural aspects but also be very specific about the data. These are the stuff and things of the product," he said.

"Everyone in this room deals with these problems between their ears all the time — but not with a formal process description," Osterweil argued. "Maintenance is a process, too," he added. "A major point of maintenance is analyzing what may be reanalyzed. But usually it is all specified inside the head," Osterweil said.

To make process programming a reality, researchers must develop and use several tools:

- A programming language. "I suspect we'll come up with a specification for a language that doesn't exist," Osterweil said.
- Databases. "Software engineering can be greatly helped by the right database," he said.
- Artificial intelligence techniques. "We need to encapsulate what we do in modules. Perhaps we could use incremental sets of rules to handle problems and keep design as an activity bound to a human with rule-based help," Osterweil said.
- Interface management. "Binding to humans means it will be handled like debugging: tools to help users," he said.

"Software engineering is *not* the intersection of these disciplines, *not* a manifestation of these fields," Osterweil argued. "It is the software process itself. It is the study of problem-solving," he said.

Dangers seen. "The very act of pursuing the process influences the process. There is no such thing as a static program. The process is a function of itself," argued Meir "Manny" Lehman of the Imperial College of Science and Technology in London. "The interac-

tions are totally unpredictable. *People* have to decide what is correct — correctness is not an absolute concept even in mathematics," he said. It is a human interpretation that varies from person to person and situation to situation, he explained.

"The essence of a programming language is that underlying every verb is an interpretive mechanism that completely defines what the process is. Using verbs, descriptions of this type suggest a determinism that does not exist — that cannot exist," Lehman said. "We can never build machines that act like human minds," he said.

"The programs that we can make deal with the easy parts. The problems that arise are the ones that cannot be described," Lehman asserted.

Process programming is a good modeling technique, he conceded. If a step is algorithmic, it can be mechanized, Lehman said. Process programming "is one component," he said, an important part of formal specifications.

"[But] it represents a very real threat," Lehman warned. For example, "if you use natural language, you think you've defined the spec. But you really haven't since you cannot have a total definition," he said.

While process programming is "good for well-modeled, well-understood parts — the ones we can mechanize," it misses a significant point, Lehman argued: "Fundamental to programming is not knowledge, it is understanding." Process programming "is likely to convey a false sense of understanding," he said.

"It has a seductive value that will lead to a waste of efforts," Lehman warned. It requires self-control before use, he said. "Because it is so seductive, process programming is not appropriate for student research or PhD activity," Lehman argued.

Debate. "The truth is somewhere in between" Osterweil's and Lehman's views, Balzer said. "The essence of the technique now is to separate the straightforward pieces [of the programming process] and put it in a mechanical realm. This will focus human activity on the hard things," Osterweil responded to Lehman's critique.

Harlan Mills of IBM said from the floor that "I regard software engineering as an intellectual activity. I see here [in process programming] an anti-intellectual revolution." The audience clapped in support. Mills decried the absence of computer science giants like David Gries, Fred Brooks, Anthony Hoare, and Edsgar Dijkstra: "We're freezing out the intellectual forces out of this conference. What is the alternative to making software engineering an intellectual activity?"

"I worry when it is too mathematical *and* too humanistic. You must balance them. That is the essence of our discipline," Osterweil responded. "We can get more insight on the intellectual questions by segregating out what is not intellectual," he argued.

"Some process programming is based on a very naive view of human processes. Twenty VAXs will return the same answer; 20 programmers with the same degree of experience will not," said MCC's Bill Curtis from the audience. "We need to give more than a process instantiation. We need an empirical approach," he argued.

"You can take unmasked input from the real world and . . . apply the process to that data and decide what to do with it." Osterweil responded.