On being one's own programming self

by PETER J. DENNING

Purdue University
West Lafayette, Indiana

“Structured programming” has become the name of a popular movement toward better programming in the past two years. Among the popular interpretations of the advice advanced by the masters, are many strong proclama-
tions that anyone, by following a few simple prescribed rules, can be transformed into a programmer of top grade, and the productivity of programming shops everywhere can be multiplied. Experience, particularly with students in programming courses, leads to the tentative conclusion that some (but not all) will be helped by the “rules of structured programming”; and that the quality of their products will rise, though not to the extent one might expect from the inflated claims of some structuralists.

There is a striking similarity between writing good programs and writing good English. When translated into the context of English Style, the counterparts of some structured programming rules border on the absurd. As examples, consider two specific directives one hears:

1. Construct all programs by combining simpler ones using only the control structures of “sequencing,” “if-then-else,” and “do-while.”
2. Program top-down, successively refining “stubs” (unimplemented lower-level modules) until all modules are completed.

The problem with the first directive is that it has lost sight of the objective: to confine oneself deliberately to a small set of esthetically pleasing forms that permit clear expression without loss of generality. Just as many more than three prose forms are in good taste, many more than three programming forms are in good taste. The great novelists and journalists have often achieved their statures precisely because they created a new form or a new combination of old forms. Even so in programming.

The second directive is more deceptive than the first. Once again, it loses sight of the objective: to structure a complex work so that its important elements can quickly be grasped in relation to the whole, and that it may easily and convincingly be presented. A hierarchical structure is a pleasing and effective way of achieving this objective. But the structure of the product need not resemble that of the process that created it. Even as the successful authors of large books have deviated from the advice of grammar school composition teachers—write first all the chapter titles, then all the section headings, then all the subsection headings, then all the topic sentences of paragraphs, then the paragraphs themselves—so the successful authors of large programs do not follow a particular, structured pattern of creativity: however quintessential in hierarchical design be their products. That is, in fact, the crucial point: a work of skill is judged by the beauty of its structure, though no one really believes that the craftsman’s creative impulses conformed to any predetermined pattern. Each writer has his own style of creating his work and will achieve success only to the extent that he is able to find and cultivate his own style.

As you can see, the problem with both directives is their inflexibility. By saying, “Do it this way,” they confine the individuality of the programmer and disallow the expression of his own creativity. Each writer—whether of prose or of program—has his own style of creating his works and will succeed only to the extent he is able to find and cultivate it.

Let it be clearly understood that I am not downgrading the movement toward better style in programming. I am one of its strongest supporters. By imitating the style of the masters I am able to instill more pleasing forms into my own writing and programming, and certain of the forms suggested by the masters are unquestionably effective. I am suggesting only that you beware dogmatic insistence on a fixed set of rules about programming. It is like insisting that students of composition write only in one style: if heedless, this advice would lead to a dreadfully dull world. Let us translate our experience with prose style into the context of programming. It may help us approach programming with the same realism, maturity, and flexibility as we approach prose.

REFERENCES
