IS IT NOT TIME TO DEFINE "STRUCTURED PROGRAMMING"?

Peter J. Denning

Department of Computer Science
Purdue University
Lafayette, Indiana 47906

I was amused by a statement in Mike Schroeder's report on the SIGPLAN/SIGOPS Interface Meeting [OSR 7, 3(July 1973), 4-9.] Mike reports that the participants seemed to agree that "it is not yet clear precisely what structured programming is, [but] there is general agreement that such a thing is possible and would be extremely useful in constructing large, complex systems." (I think Mike meant to say that it would be extremely useful for avoiding the construction of unnecessarily large and complex systems.) If no one knows what it is, how is it possible rationally to agree that it would be "extremely useful"? Do the participants at this meeting mean to imply that, whatever turns out to be "extremely useful" for simplifying the construction of large systems will by general agreement be considered as "structured programming"? If it is not known what "structured programming" is, how is it possible to conclude with certainty that it is "possible"?

I have read and heard statements of this type frequently in the last year. No one knows what the words "structured programing" mean, exactly. Yet we find those arguing strongly in its favor and those arguing strongly against it: How does one argue for or against something when one is not even sure what it is? How can one be sure it is useful or useless without being able to describe it? Progress has been made in operating systems in recent years because there has been a general desire for precision of definition and clarity of thought. Were a project manager today, on being asked what type of operating system he is constructing, to utter the words so often heard a few years ago, "Well, we're not sure exactly, but we do know the system is possible and will be extremely useful," -- he would not long have his job. It is incongruous for us to be striving for precision and clarity in every aspect of operating system design, and yet seemingly to permit imprecision and unclarity to be propagated with respect to one of our most important tools, programming.

But what is "structured programming"? The following seem to encompass most of the impressions people seem to have of it:

1. It is a return to common sense, an awakening to the realization that we are about to choke on the myriad of "features" and "options" we have been building into languages and systems.

2. It is the general method by which our leading programmers program.

3. It is programming without the use of goto statements.

4. It is the process of controlling the number of interactions between a given local task or block and its environment, so that the number of interactions is some linear function of some parameter or parameters of the task or block.

5. It is top-down programming.

There seems to be truth in all of these impressions. (I use the word "impression" because it does not imply a definition.) It is apparently true that many of our languages and systems are too flexible, having so many features and options that their misuse is more likely than their efficient use. It is true that our better programmers have realized this and have limited themselves to a small number of simple constructions and have avoided the use of every possible feature in the languages or systems they use. It is true that there exist languages in which intelligent and comprehensible programming without goto statements is possible; however, the absence of goto statements cannot of itself be taken as an indication that "structured programming" has been used. It is true that limiting the number of interactions between a task or block and its environment generally leads to more comprehensible programs; however, a limited number of interactions cannot of itself be taken as an indication that "structured programming" has been used. I am not sure what "top down programming" means; I have seen the phrase used in at least two ways:

a. It is a method of decomposition into modules by stepwise refinement.

b. It is the goal of evolving a program so that its final structure can be presented or described as a hierarchy of tasks or blocks.

I hope most of those who use the phrase mean b) rather than a), as all the evidence I have seen suggests that a) if applied literally will fail when applied to large programming projects.

If we are going to be so enthusiastic in our pursuit of happiness by the path of "structured programming", should we not at least preface our discussions of it with a definition? Should we not apply our (newfound) criteria of precision and clarity to our discussions of programming too?