THE CASE FOR STRUCTURED PROGRAMMING

HENRY F. LEDGARD

Abstract.

This report is mainly a response to a paper by Henderson and Snowdon, "An Experiment in Structured Programming". The notions of structured programming, top-down programming, and stepwise refinement are compared, and some careful guidelines for the proper use of structured programming approaches are suggested.

1. Introduction.

I would like to discuss in detail a technique now known as "structured programming". Let me first discuss a related notion called "top-down programming". In the strict sense, top-down programming is a method of program development where, during the entire programming process, the programmer uses the source language at hand. The first written piece of code is the top-most level, i.e., the main procedure. The sub-procedures are then written, these are later split up, and so on until the entire program is coded. One may assume dummy procedures for those procedures that are initially left unspecified. This definition of top-down is the mirror image of the "bottom-up" method, where you usually write the lower procedures first and later write the upper levels. The two definitions are effectively symmetrical.

"Structured programming", as originally presented by Dijkstra [1], is any method of program development such that at each step the program can be logically decomposed into distinct sub-structures and such that correctness is manifest by the internal structure of the program itself. A "structured" approach need not necessarily be top-down, although often it is.

We have one other related term to contend with. What is called "successive" or "stepwise" refinement [1, 6], is a method of programming where one starts at the top level, but one is not constrained to work in a particular programming language. One generally starts with an improvised language, which in some sense is mechanical to the user. One might

The work reported herein was sponsored in part by the National Bureau of Standards, Gaithersburg, Maryland.

Received April 7, 1973.
use statements like "compute the \( n \)th prime number", "find the roots of the equation", or "process the payroll". With successive refinement we do not arrive at our target language until we are near the bottom level, and the program is almost in front of our eyes.

Unfortunately, each of these terms has been muddled in common usage. To some, "top-down programming", "structured programming", and "stepwise refinement programming" all mean the same thing. To some, structured programming is identified with the use of \texttt{if-then-else} and \texttt{do-while} constructs. Few ever give careful definitions of any of these terms.

2. A definition.

For the purposes here, I shall give my own definition of a programming approach that I believe integrates the important characteristics of each approach mentioned above. The definition is in the spirit of "structured programming", but has elements of the strict top-down and successive refinement methods. The approach is a process for writing computer programs that has the following characteristics.

1. Exact Problem Definition:
   The programmer starts with an \textit{exact} statement of the problem. It is senseless to start any program without a clear understanding of the problem.

2. Initial Language Independence:
   The programmer initially uses expressions (often in English) that are relevant to the problem solution, even though the expressions cannot be directly transliterated into the target language. From statements that are \textit{machine and language independent}, the programmer moves toward a final machine implementation in the target language.

3. Design in Levels:
   The programmer designs the program in \textit{levels}. At each level, the programmer considers alternative ways to refine some parts of the given level. The programmer may look a level or two ahead to determine the best way to design the next level.

4. Postponement of Details to Lower Levels:
   The programmer concentrates on critical broad issues at the initial levels, and postpones details (e.g., input/output, choice of identifiers, or data representation) until lower levels.