Literate Programming

Donald E. Knuth
Computer Science Department, Stanford University, Stanford, CA 94305, USA

The author and his associates have been experimenting for the past several years with a programming language and documentation system called WEB. This paper presents WEB by example, and discusses why the new system appears to be an improvement over previous ones.

A. INTRODUCTION

The past ten years have witnessed substantial improvements in programming methodology. This advance, carried out under the banner of "structured programming," has led to programs that are more reliable and easier to comprehend; yet the results are not entirely satisfactory. My purpose in the present paper is to propose another motto that may be appropriate for the next decade, as we attempt to make further progress in the state of the art. I believe that the time is ripe for significantly better documentation of programs, and that we can best achieve this by considering programs to be *works of literature*. Hence, my title: "Literate Programming."

Let us change our traditional attitude to the construction of programs: Instead of imagining that our main task is to instruct a *computer* what to do, let us concentrate rather on explaining to *human beings* what we want a computer to do.

The practitioner of literate programming can be regarded as an essayist, whose main concern is with exposition and excellence of style. Such an author, with thesaurus in hand, chooses the names of variables carefully and explains what each variable means. He or she strives for a program that is comprehensible because its concepts have been introduced in an order that is best for human understanding, using a mixture of formal and informal methods that reïnforce each other.

I dare to suggest that such advances in documentation are possible because of the experiences I've had during the past several years while working intensively on software development. By making use of several ideas that have existed for a long time, and by applying them systematically in a slightly new way, I've stumbled across a method of composing programs that excites me very much. In fact, my enthusiasm is so great that I must warn the reader to discount much of what I shall say as the ravings of a fanatic who thinks he has just seen a great light.

Programming is a very personal activity, so I can't be certain that what has worked for me will work for everybody. Yet the impact of this new approach on my own style has been profound, and my excitement has continued unabated for more than two years. I enjoy the new methodology so much that it is hard for me to refrain from going back to every program that I've ever written and recasting it in "literate" form. I find myself unable to resist working on programming tasks that I would ordinarily have assigned to student research assistants; and why? Because it seems to me that at last I'm able to write programs as they should be written. My programs are not only explained better than ever before; they also are better programs, because the new methodology encourages me to do a better job. For these reasons I am compelled to write this paper, in hopes that my experiences will prove to be relevant to others.

I must confess that there may also be a bit of malice in my choice of a title. During the 1970s I was coerced like everybody else into adopting the ideas of structured programming, because I couldn't bear to be found guilty of writing *unstructured* programs. Now I have a chance to get even. By coining the phrase "literate programming," I am imposing a moral commitment on everyone who hears the term; surely nobody wants to admit writing an *illiterate* program.

B. THE WEB SYSTEM

I hope, however, to demonstrate in this paper that the title is not merely wordplay. The ideas of literate programming have been embodied in a language and a suite of computer programs that have been developed at Stanford University during the past few years as part of my research on algorithms and on digital typography. This language and its associated programs have come to be known as the WEB system. My goal in what follows is to describe the philosophy that underlies WEB, to present examples of programs in the WEB language, and to discuss what may be the future implications of this work.

I chose the name WEB partly because it was one of the few three-letter words of English that hadn't already been applied to computers. But as time went on, I've become extremely pleased with the name, because I think that a complex piece of software is, indeed, best regarded as a *web* that has been delicately pieced together from simple materials. We understand a complicated system by understanding its simple parts, and by understanding the simple relations between those parts and their immediate neighbors. If we express a program as a web of ideas, we can emphasize its structural properties in a natural and satisfying way.

WEB itself is chiefly a combination of two other languages: (1) a document formatting language and (2) a programming language. My prototype WEB system uses