◊ ◊ ◊

Literate Programming
not just another pretty face

M.A. Guravage *(NLR)*

ABSTRACT. The structure of a software program may be thought of as a "web" that is made of many interconnected pieces. To document such a program, we want to explain each individual part of the web and how it relates to its neighbors. – D.E.K.

My association with literate programming was love at first sight. I admired the crisp form and clear content of the programs I read. I simply became jealous; I wanted my programs to look and work like that. When I first read Professor Knuth's description of a program as a "web" of interconnections, I became curious whether the pattern of connections particular to literate programs could be modeled, analyzed, and quantified.

The most obvious model to represent information about the relationship between the pieces of a web is a graph. Graphs have been used extensively to model all sorts of relationships, and the theory of graph metrics is well developed. The trick is to interpret the various standard graph metrics, e.g. flow, width, depth, size, and edge-to-node ratio, in the context of a web's literate structure. If the same can be done for a traditional program's call-graph structure, it might be possible to compare objectively literate and traditional programming styles.

The two graphs below represent two different metrics applied to the same literate program. The left graph was created using an 'activation' metric; that correctly identified, in thick blue, the main conceptual threads through the program. The right graph was created using an 'upward flow' metric; that colored yellow the paths to nodes most often reused.

Using graph visualization software developed at CWI in Amsterdam, I hope to show how various metrics can shed light onto the structure of both literate and traditional programs. For example, a measure called 'tree impurity' can tell us how far a graph deviates from being a tree, might allow us to compare literate and traditional programming styles.