The multiple meanings of a flowchart: visual representations of complexity in computer programming

Nathan Ensmenger*[1]

[1]India University, School of Informatics and Computing – United States

Abstract

A well-written computer program is, in theory at least, self-documenting; that is to say, the computer code itself contains its own complete written specification. And yet despite the computer scientist Donald Knuth's famous claim that computer programs, like literature, were meant to be read by humans as much as machines, for the most part computer programs are too arcane and idiosyncratic to serve as a useful function as a design document. From the very earliest days of electronic computing, "flow diagrams" (later "flowcharts") have been used to represent the conceptual structure of complex software systems. In much of the literature on software development, the flowchart serves as the central design document around which systems analysts, computer programmers, and end-users communicate, negotiate, and represent complexity. And yet the meaning of any particular flowchart was often highly contested, and the apparent specificity of such design documents rarely reflected reality. In fact, some of the first software "packages" (commercial applications that could be purchased off-the-shelf) were used to reverse-engineer the flowchart specification from already developed computer code. In other words, the implementation of many software systems actually preceded their own design! Drawing on the sociological concept of the boundary object, I will explore the material culture of software development, with a particular focus on the ways in which flowcharts served as political artifacts within the emerging communities of practices of computer programming.

*Speaker