SOLOWAY84

Soloway, E. "A Cognitively-Based Methodology for Designing Languages/ Environments/Methodologies", Proceedings of the ACM SIGSOFT/SIGPLAN Software Engineering Symposium on Practical Software Development Environments ACM Software Engineering Notes Vol.9 No.3 May 1984 ACM SIGPLAN Notices Vol.19 No.5 May 1984 (joint issue) The issue discussed in this paper is the basis for the design of Languages/environments/methodologies. It is not clear that current developments in these areas actually result in more productive programming or that the implicit and explicit "shoulds" such as strong typing can really be described as good. The typical approach to design of a proposal based on the experience of those offering it is challenged by a cognitive methodology. The design of a language/environment/methodology can be based in part on psychological grounds by developing a theory of the work habits of people, empirically testing the hypotheses and then deriving design implications from this foundation. It involves a more formal modelling of the situation in which the tool is to be used. A cycle exhibiting this approach is to first observe the work habits of programmers or designers; identify the behaviours of the group particularly those areas where difficulties arise; and then try to explain these behaviours in psychological terms with reference to the knowledge structures and procedural strategies involved. Although this seems like a very involved process a great deal of work in cognitive psychology and AI can be used to build a theory rather than redoing the work already done. The example cited in the paper is that a theory of how programmers read and understand programs was developed from research in text processing and problem solving. The general aim of the methodology is to improve the cognitive fit of the tools in use. The hope is that understanding the processing used by individuals will mean that problems can be avoided by having the tools correspond to the needs of the individual. One of the examples involves programmers. In general there are two areas which distinguish expert programmers from novices - knowledge of programming plans and knowledge of rules of programming discourse. Plans are simply sets of stereotypical problems and their solutions which programmers use to solve other problems. Rules of discourse are conventions such as variable names reflecting the function of the variable. These provide the programmer with certain expectations and correspond to conventions used in speech. These rules of discourse provide the medium for expressing the programming plans known. The results of a study are mentioned which indicated that plan-like programs are much easier to comprehend than ones which are unplan-like. This is largely due to the expectations in the minds of the programmers. It makes sense therefore to design languages/environments/methodologies which are based upon the presence of these two behaviours in programmers. Criticisms levied against this approach were addressed. The basis of the study of experts in the current world was challenged as not rendering the technique useful should new technology arise. This was answered, rightfully, as not being particularly relevant in the sense that the work will be of use in the extension for development within the new technology. A point not mentioned which occurred to me in this regard is that the current approach certainly would cause more problems with new technology because no set of proposers would have the necessary experience. The standard criticism of methods involving study and testing, that they are slow and costly was also answered well by pointing to corporations which have seen results from this approach. It was also carefully affirmed that this is one approach and not the only one. The paper gives a brief look at the cognitive approach to design. It is obviously meant to be an introduction to spark interest in the approach. This is achieved through the use of a discussion at a high level and good references to the supporting studies. In this way the reader is not lost in a mire of details but can readily understand the flavour of the approach. It presents an approach which is not new for problems in general and suggests that it is applicable in useful ways to the design of programming tools. The use of prior research is well acknowledged.