Cognitive Psychology and Programming Language Design

Ben Shneiderman
Indiana University
Computer Science Department
101 Lindley Hall
Bloomington, IN 47401

Researchers and language developers appear to have finally begun to accept the importance of the human element in the programming process. However, at the same time, language designers are not studying the relevant disciplines to gain their background, are not using the appropriate techniques to assess the validity of their work and are still focused on computing rather than programming.

As early as 1965 Dijkstra had acknowledged that programming was a purely human activity and that languages had to be designed for ease of use by people. The utility of this decomposition of human-centered concerns from the machine-centered efficiency issues should be obvious to any student of "structured programming", but the orientation of language designers is still often geared to ease of implementation and efficiency of object code. Weinberg's text, The Psychology of Computer Programming was a step in the right direction, but those doing research in programming have only recently begun to take the hint that they should be studying the psychological literature. The psychologists quickly recognized the advantages of using computers to conduct their experiments and do their statistics, but computer scientists are only beginning to grasp the importance of applying psychology to the study of programming.

The cognitive psychologists have been analyzing human thought processes, learning, memory, perception and problem solving for several decades and have developed the experimental techniques necessary for such research. Much of this work is directly applicable and relevant to the study of programming. After all, programming and programming language design is the study of problem solving, but at a level unexplored by psychologists. Program composition, comprehension, debugging and modification are complex tasks which are more sophisticated than the relatively simple task studied by psychologists. Still, controlled psychological experiments which minimize the number of variables and focus on precise issues can be extremely helpful in guiding the programming language development process[1].

The participants in Tony Wasserman's session on "Issues in Programming Language Design" each independently had some notion of the programmer as the intermediary between the problem and the program (as expressed in the syntax of a particular language). They properly focused on the transformation that the programmer effects from the problem to the program and generally avoided questions of parsing ease, efficiency of execution, or character sets. Charles T. Zahn coined the phrase "conceptual distance" to describe the remoteness of the problem from the program, but gave us no clues on how to measure the distance or even a suggestion of whether we needed a ruler, laser rangefinder, thermometer, or analytic balance. Jack Dennis commented that "the closeness of the language to

the problem domain affects the ease with which the programmer may express a problem solution", but offered no ideas on how to gauge "closeness". Tim Standish remarked that paraphrase was an "easy" extension to a language for users, while orthophrase and metaphrase were "hard" extensions, but his scales were equally intuitionistic and vague. Tony Wasserman got closest to acknowledging the human factors issues by indicating that the goal of a good programming language was to minimize the programmer's "cognitive stress" during the transformation from problem to program.

These remarks indicate that the speakers recognize the importance of the cognitive issues in program development but programming language researchers have much to do in this direction. A first step would be to develop a practical cognitive model of the programming process, which would be verified by a program of experimental testing. The immediate goal of such research would be the improvement of programming languages, the development of sound standards for stylistic issues (such as mnemonic variable names, commenting, indentation, etc.) and the validation or refutation of the multiplicity of proposed design techniques. These studies could be conducted as controlled comparative experiments, by introspective protocol analysis or by the non-interfering case study approach. A number of researchers, including myself, have begun such studies. Longer range goals would include the development of accurate, reliable programmer aptitude tests, programmer ability measures, the improvement of the teaching/learning or programming, problem and program complexity measures and recommendations for entirely new programming languages (for example, facilities for the casual user of computer utilities). These experimental techniques become particularly important when developing facilities for the naive user whose experience and background are radically different from that of the programming language designer.

In summary, programming language designers can no longer be content with a thorough knowledge of computer science, but must become familiar with the ideas and techniques of the cognitive psychologist. Communication between computer scientists and cognitive psychologists will be helpful in the development of the next generation of programming languages. It will also facilitate more widespread computer literacy.

Reference

[1] Shneiderman, B. "Experimental Testing in Programming Languages, Stylistic Considerations and Design Techniques," Proceedings of the 1975 NCC, pp. 653-656.