*** Two High-Level Skills for Programming: ***
A comment on R. L. Glass' "The Importance of the Individual"

Gary Perlman

Glass (ACM SIGSOFT Software Engineering Notes, 5 3, pp. 48-50, July 1980) points out huge individual differences in programmer abilities. He also bemoans the lack of aptitude tests to identify good programmers. I think that the reason for this lack lies in methodological problems in devising aptitude tests of any kind. Despite their ubiquity, the so-called intelligence tests and scholastic aptitude tests are poor predictors of future performance in school and other areas (except of performance on similiar tests, where the success is moderate).

More plausible than the empirical approach is the use of theoretical models to describe the skills necessary for coding and debugging programs. I think cognitive psychologists can contribute to the identification of good programs and the betterment of mediocre programmers by using models of thought to explain *why* certain attributes are desirable.

I base my opinions on general human information processing limitations of people in general, and therefore programmers in particular. The major limitation of human programmers is their limited conscious memory capacity. We have the ability to keep track of only a few ideas and their interactions at one time. This implies that programmers should attack only small tasks that can easily be conceptualized. Since the tasks we are interested in are not small, they have to be made to appear small by dividing them into some number of smaller tasks, and then conquering the smaller tasks. This technique has come to be called divide and conquer. However, some software engineers have rightly suggested that the division of any module should not be greater than about six parts. The reason for this is that if a module is divided into more, then keeping track of all those parts will also overload conscious memory. These points indicate that good programmers need the ability to divide problems intelligently so that the conquering of the parts is simplified. I think testing for this trait would prove to be a good indicator of programming capability.

People are bad at hypothesis testing, which is largely a learned skill. Applied to program development, notably testing and debugging, good programmers need skills in proper experimentation. To locate and correct a bug, we make changes in input or to the program itself and note changes in output. The need for experimental control in testing and debugging implies the need for stepwise refinement of programs (adding exactly one module between tests). This restricts the source of new bugs to the new module. If more than one module is added for a test , then the source of the error cannot be determined because (experimentally speaking) more than one factor has been varied at one time. (In statistics this is called perfect confounding.)

I have suggested two skills needed for productive programming based on psychological principles.

1. The ability to code large programs requires the ability to divide (possibly recursively) a program intelligently into easily conquered parts.
2. The ability to test and debug requires (at least implicit) knowledge of the logic of experimental hypothesis testing.

Neither of these skills comes naturally to most people, but they can be taught. Detection of these skills should be useful for predicting programmer productivity.

Department of Psychology Gary Perlman
University of California, San Diego
La Jolla, CA 92093

Response from the Editor:

Programmers seem to have two qualitatively different types of skills they use in writing programs. The first type involves logical, linear, and rational skills, while the second involves creative and intuitive skills. (Note that there is some evidence to suggest that these two skill types are often associated with the activities of the left hemisphere and the right hemisphere of the brain, respectively, although that association not vital to my comment.) I once wondered whether one of these skill types was more important in creative programming endeavors, but then concluded that a balence of these skills is required. Logical/linear skills are essential to permit the orderly maintenance of a vast collection of details and their interrelationships. However, creative/intuitive skills are vital to large and complex efforts in which creative use of structure and decomposition is required when the problem becomes too big to be handled as a single entity. Someone who is strong only in logical/linear skills may be an excellent programmer in the small, but may have great difficulty in coping with large and real systems in their entirety. Someone who is strong only in creative/intuitive skills is unlikely to become a good programmer in the first place, especially if there is an intrinsic dislike of detailed mathematical thinking. However, the really disciplined imaginative designer/programmer seems to be someone with all these skills well developed and well in balance. Our educational process should consider it a challenge to help each individual to bring these sometimes opposing skill types into harmony. (If you missed it, see my somewhat related piece on the psychology of abstraction, **SEN** 4 1, p. 21, January 1979.) PGN.

Response from Gary Perlman:

Yes, people good at the design of large programs are not necessarily good at writing modules, and vice versa. This predicts a version of the Peter Principle that probably occurs in programming: a good coder is promoted to designer and becomes incompetent. GP.