THE IMPORTANCE OF THE INDIVIDUAL

Robert L. Glass
Boeing Aerospace Co.

Scattered through the computing literature for over 10 years now have been little clues about a matter which may be vital to the issue of programmer productivity.

These clues point to one factor which controls a probable order of magnitude difference.

It is not an easy factor to convert into meaningful results. And yet with that kind of promise, it deserves more than the cursory attention it has gotten over the years.

The factor is simple: Individual differences between programmers are immense.

For instance, way back in 1968 three SDC researchers (Sackman, Erikson and Grant) stumbled across individual differences of up to 28-1 while doing a study comparing batch and timesharing methodologies. "The most important practical finding involves the striking individual differences in programmer performance," they said (SACK68).

For instance, also in 1968 Jules Schwartz, a computing pioneer, said "Within a group of programmers, there may be an order of magnitude difference in capability" (SCHW68). Schwartz was studying the problems of developing large scale software systems, and he found human factors at the heart of the problems - and their solutions.

For instance, in 1973 Barry Boehm shook up the whole software world with his "High Cost of Software" articles (BOEH73). Nearly buried under the avalanche of other provocative data was the statement "Productivity variations of 5:1 between individuals are common."

For instance, in 1975 D. E. W. Bucher, writing one of the rare papers on software maintenance (BUCH75), said "The prime factor affecting the reliability of software is the selection, motivation and management of the personnel who design and maintain it."

For instance, in 1978 Glenford Myers, in a definitive experiment on software reliability methodologies, (MYER78), found "There is a tremendous amount of variability in the individual results. For instance, two people...found only one error, but five people found seven errors. The variability among student programmers is generally well known, but the high variability among these highly experienced subjects was somewhat surprising...The detection of individual types of errors varies widely from individual to individual."

For instance, exploring a different discipline, the impact of higher order languages on avionics software (RUBE78), Raymond Rubey wrote "a programmer having no prior experience...wrote a program that was 100% inefficient, while an experienced programmer wrote a version of the same program that was 20% inefficient. Another study reported a 25% improvement in efficiency with greater programmer experience. Clearly, the programmer's experience is a major factor in achieving high efficiency."

And as the ultimate for instance, Werner L. Frank, discussing "The New Software Economics" (FRAN79), put it on the line: "When all is said and done, the ultimate factor in software productivity is the capability of the individual software practitioner."

The message is clear. If you want to have the maximum impact on the quality and quantity of the software you build and maintain, you must concentrate heavily on getting the right people, and getting rid of the wrong people.

And there lies the problem. How do you recognize the right people?

It is not the purpose of this article to answer that question. It is, rather, the purpose of this article to motivate people to seek the answer to that question. The question then is "What do we know about recognizing the right people?" The correct answer is, "Not much."

Remember the era of programmer aptitude tests? There wasn't much correlation between test scores and job performance. Have you heard of any good aptitude tests lately?

Now we have an era of programmer certification. There's the Certified Data Processor exam and the ACM self-assessment program. Do we know how the results correlate with on-the-job performance across a variety of applications? It sure seems like we ought to, but there don't seem to be any study results.

Not much else has been tried. Evaluating programmers has been sort of like judging paintings - "I may not know much about art, but I know what I like." The result may be personally esthetically satisfying in the art world - and then again, it may not - but in the programmer world it leads to a kind of educated cronyism. There is the nagging question, "Are our "Best" programmers really our best programmers?"

The stakes are high. If we are misjudging our best people, then we are probably misusing our best people. But to make the stakes even higher, the whole issue is enlivened with some methodological questions. Does Weinberg's "egoless programming" (WEIN71) really make sense? Should programmers work in chief programmer teams, or as collections of individuals, or in a more traditional hierarchic organizational harness? If Kraft's fear of the "routinization of computer programming" (KRAF77) valid? Are the craftsman-like capabilities of the software artisan destined for the assembly-line human potential scrap heap? Does the return-to-craftsmanship trend in manufacturing, being tried at Volvo and Caterpillar, for example, have a message for software routinization?

Have we ever measured the difference between the craftsman and the routinized approach, as (GLAS78) facetiously suggests?

And what of quality? Software quality, already a difficult to measure but vitally important entity, may be inextricably tied to individual capabilities. What happens to a software quality assurance program if that is true?

What happens on a 400-programmer project if that is true? Must large projects function to a level of common mediocrity in order to succeed? And if so, can a creative, hard-charging individual function (either as a technologist or as a person) in that environment?

That's a lot of questions. The ultimate question is, what's being done about answers? Fortunately, some work is being done. At the 1979 National Computer Conference, there was a session on "Software Psychology: Exploring the Human Factor." The papers were interesting, with titles like "First-year results from a research program on human factors in software engineering." But most of them were methodology-first explorations. Like, "the effects of modern programming practices on programmer efficiency" and "prediction of programmer performance from software complexity metrics." These are laudable and vital studies. But they dodge what is beginning to look like the more important human factors issues:

- We need to know why some programmers are an order of magnitude better than others
- We need to know what those exceptional programmers do differently from others
- We need to know if the results of the why and what explorations can undergo a human technology transfer from the exceptional to the needy

What is discouraging in this area is that the really definitive work, and the fundamental underlying questions, were all done in that first 1968 SDC paper:

- Capability to debug differed by factors up to 28-1
- Capability to code differed by up to 25-1
- Timing efficiency of the resulting program differed by up to 11-1
- Sizing efficiency differed by up to 6-1

The authors found "no consistent correlation between performance measures and the various (aptitude) grades and test scores...Class grades and test scores showed substantial intercorrelations." The meaning? Neither aptitude tests nor academic performance appear to be predictors of on-the-job performance.

The authors concluded "It is apparent...that very substantial savings can be effected by successfully detecting low performers. Techniques measuring individual skills should be vigorously pursued..."

There, in 1968, was the challenge.

It is now 1980. We have not responded to that challenge.

(BROOSO) has raised the challenge again: "Specification of the...behaviors and processes which differ between better and poorer programmers...are research problems that are...intriguing."

Perhaps in an era where "productivity" is the key word, the challenge will be taken up again.

(BOEH78) "The High Cost of Software," Practical Strategies for Developing Large Software Systems, Addison-Wesley, 1975; Boehm

(BROOSO) "Studying Programmer Behavior Experimentally: The Problems of Proper Methodology," Communications of the ACM, April 1980; Brooks

(BUCH75) "Maintenance of the Computer Sciences Teleprocessing System," Proceedings of the International Conference on Reliable Software, 1975, Bucher

(FRAN79) "The New Software Economics," Computerworld, 1979; Frank

(GLAS78) "Will the Real Henry Ford of Software Please Stand Up," Tales of Computing Folk: Hot Dogs and Mixed Nuts, Computing Trends, 1978; Glass

(MYER78) "A Controlled Experiment in Program Testing and Code Walkthroughs/Inspections," Communications of the ACM, Sept. 1978; Myers

(RUBE78) "Higher Order Languages for Avionics Software - A Survey, Summary and Critique." NAECON 1978; Rubey

(SACK68) "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," Communications of the ACM, Jan. 1968; Sackman, Erikson and Grant

(SCHW68) "Analyzing Large-Scale System Development," Software Engineering Concepts and Techniques, Proceedings of the 1968 NATO Conference; Schwartz