Programming process productivity measurement system for System/370

by M. J. Flaherty

Discussed in this paper are the underlying principles of a programmer productivity measuring system. The key measures (or metrics) are people and lines of code. Definitions of these metrics are refined and qualified, according to the conditions under which they are used. Presented also is a data base design for retaining and retrieving these metrics under a wide variety of applications and other circumstances. Depending on definitions, applications, and other circumstances, productivity measurements may differ widely. On the other hand, after suitable productivity metrics have been defined, consistency of application of the same metrics yields comparable results from project to project.

By the mid-1970s, many factors were contributing to a renewed interest in software productivity. Hardware costs were decreasing faster than software costs, so that software costs were becoming an increasing portion of the data processing budget even if they remained constant in absolute values. The labor-intensive nature of software development also indicated that this would be a continuing trend.[1] Adding to the concern over computer software costs was an apparent decline in overall productivity in the United States.[2]

However, a means of measuring large-system software productivity was no longer available. Products had previously been packaged as system releases, thereby providing the focal point for collecting productivity data. Improvements made in the software delivery process allowed products to be released independently of the system. With the end of system releases, the control point for collecting productivity data ceased to exist. Other changes were also occurring. Resources used to produce products were accounted for differently, and precision in counting lines of code was increasing. These changes made comparisons of current productivity with that of the past difficult if not impossible.

Productivity measurement system

In response to the need to collect and preserve consistent productivity measurements, the IBM Data Processing Product Group established a Process Productivity Department. The mission of this department was to design and develop a software process Productivity Measurement System for large-system products. This would provide the means for setting goals, tracking results, improving resource planning, and increasing awareness of the need to control software development costs. The success of the measurement system depends on the application of standard measures of productivity, or metrics. Success also requires a method of collecting and storing information in a consistent way and the use of existing metrics to minimize the effect of the collection process on the development process.

To this end, we began an extensive review of the literature on software productivity. This process led to the conclusion that lines of code (LOC) per person or per dollar were the best candidates for productivity metrics.[3]

Large-system products are developed using assembler language or a high-level system development language such as Programming Language System (PLS). A tool for counting assembler and PLS lines of code underwent a major revision to provide a higher degree of consistency and accuracy than had previously been attained. That tool has become a standard method for counting lines of code.

In parallel with the LOC counter, enhancements to a software cost-estimating system have provided a method for collecting actual and estimated product costs in the metrics *people* and *dollars* in a standard format. The cost-estimating system also contains product availability dates and numbers of lines of code, as defined by the counting tool. These parameters have become the basis for standard productivity metrics for large-system products.

The Productivity Measurement System has also required the development of a data base to preserve the data and allow for easy access for analysis.

This paper describes what we have learned through the use of the Productivity Measurement System. We discuss the fallacy of depending on numbers for comparison without using standard definitions to ensure consistency in data collection. Better than productivity numbers alone is the ability to establish a long-term trend as provided by process, tools, and productivity factors. Thus, the productivity data show such trends as the effect of varying lines of code, the effect of various staffing definitions, and the variability in rates caused by product size and class or new and modified code as a percentage of total code shipped.

We have further concluded that trends are of greater value than absolute numbers, although they are not without limitations. Trends require time to manifest themselves. There is no guarantee that as many as eight years are adequate to establish a trend. For large systems two- to three-year trends may be misleading. In our studies, we have normalized the data collected so that the lowest productivity rate is taken as one (1.0), and the other rates have been calculated relative to the lowest rate.

As stated earlier in this paper, there is a need to develop parameters for measuring large-system products. We shall see later why the metric *people* was chosen in preference to *development dollars* for measuring productivity rates. Discussed first are *lines of code* and *people*, as used by our measurement system; the data base is then described.

Lines of code

Lines of code (LOC) as a productivity metric generates much discussion. Although a search of the available literature reveals inconsistencies and problems in using LOC, it continues to be the most widely used productivity metric.[3]

There are alternatives to LOC, such as Halstead's *length*[4,5] and Albrecht's *function points*.[6] Each has been shown to have a strong correlation with LOC or with *source lines of code* (SLOC). Most of the research and studies of software engineering metrics have dealt with their application to new code only. The application of the metrics *length* and *function points* to modified code has not been adequately defined or explored for large systems. However, most large-system development effort is spent on enhancements to existing products, support for extending the System/370 architecture, and additional hardware support. Thus a significant amount of old code requires modification and maintenance.

Most large-system development effort is spent on enhancements to existing products.

For our purposes, LOC measures the change and/or new source instructions—designated as CSI—for the release of a product. CSI measures non-commentary-executable and nonexecutable source statements. In PLS, a statement ends with a semicolon (;), except for IF, THEN, SELECT, and WHEN, which are counted as instructions. Further, each unfactored element of a DECLARE is an instruction. In assembler language, each statement is an instruction.

An automated tool provides the accuracy and consistency required to measure the LOC. The tool uses a special flag on the card image for the added or modified statement. Some editors have been modified to support the automatic insertion of flags.

People

For productivity measurement, we divide the development people into three groups, designated as *di-*

rects, total development, and *location support.* Members of each group are further characterized by their function. Only the directs and total development people are used for productivity measurements; location support people are included for completeness only. Note that all activities listed under directs are included in the total development category as well.

The process productivity data base contains information on over four hundred product releases from eight development locations.

Directs
- Architecture and system design
- Design, code, and unit test
- Test
- Publications
- Performance
- Other (technical consulting, build, contract management)

Total development
- Directs
- Feasibility studies
- Abandoned effort
- Authorized program analysis report (APAR) certification
- Customer walk-through
- Second line management and higher, if not counted as direct
- APAR forward fit
- Other customer service
- Secretarial support
- Tools development
- Early support program
- Process technology
- Business planners

Location support
- Assurance
- Data processing support
- Site support
- Financial
- Personnel
- Programmer training

Data base

The process productivity data base contains information on over four hundred product releases from eight development locations and represents over one billion dollars of development effort. The primary sources of data are *software cost estimates.* These estimates are produced at the request of the product development manager at significant points in the development cycle. For productivity measurements, data from the following three checkpoints are considered relevant for analysis:

- Development funds committed to the product, including accumulated actual and projected resources
- The product as announced, including accumulated actual and projected resources
- The product after announcement, including accumulated actual resources

The following is a list of the fields maintained for each product in the productivity data base:

Identification information
- Product name
- Cost estimate number
- Development location
- Product classification
- Announcement date
- First customer ship date

Lines of code in product
- Total CSI, i.e., new plus modified change source instructions (CSI)
- New CSI
- Modified CSI
- Percent of CSI in a high-level language
- Total shipped source instructions

Resources in person-months and dollars
- System design and architecture
- Design, code, and unit test
- Test and build
- Publications
- Performance
- Other
- Data processing (dollars only)

Additional information may be collected for analysis, depending on local requirements. Also available by year for each location are the percentage of directs and the total number of development people.

Measurements

Software productivity measurements were initially stated in both dollars per CSI and CSI per person-month. Both measures were found frequently in our literature search, and they appeared to be equally useful at the time they were used. Dollars accounted for a greater portion of the resources expended than people alone, because dollars included—in addition to the cost of people—overhead and support costs. Because dollars have such disadvantages as local variations in overhead and inflation rates, productivity measurements are adversely affected during periods of higher inflation unless dollars are normalized. To avoid normalizing dollars, person-months and lines of code have been designated the preferred parameters for most of our productivity measurements.

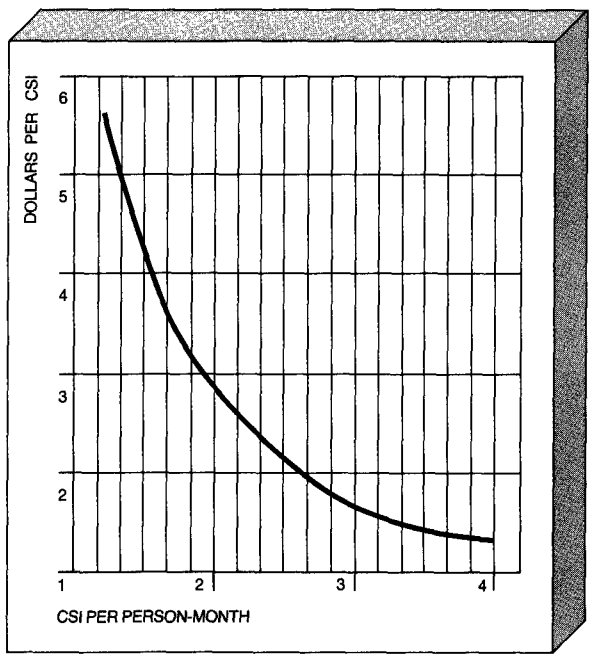
Before making that designation, however, we compared the dollar-per-CSI rate with the CSI-per-people rate, as shown in Figure 1. Usually the cost per CSI is inversely proportional to the CSI per person-month. Because the dollar and people productivity rates appeared to be so closely related, CSI per person-month became our primary measurement standard. However, dollars are still collected for each product and are used to analyze products where more than ten percent of the expense is accrued by an outside vendor. For these products, the person-month is not relevant to a study of the large-system development process.

Data were collected on the total number of development people and the percentage of direct people at each location. It became apparent after data had been collected for several years that the definition of directs was not as consistent as had originally been believed. After 1979, the trend was to increase the percentage of the total number of development people counted as direct. Although this action better established the actual expenses for a product, the productivity rate measured in terms of direct people was reduced.

Of course, there is no precise method for normalizing direct people. However, an estimate was made of the effect of moving more of the total number of development people into the direct people count. Taking this into account, the adjusted productivity rate shows a seven percent compound growth rate from 1977 to 1984.

The decision was made at the end of 1983 to use *total development people* (TDP) in place of direct people in future productivity measurements. This change essentially eliminates an inconsistency caused by counting direct people only. The expanded base used to compute productivity rates now includes most of the support resources. A count of the TDP has been tracked since 1977, making it possible to reconstruct the trends using CSI and TDP.

Figure 1 Comparison of dollars per change source instruction (CSI) and CSI per person-month



The immediate effect of such a change is to lower the absolute numbers for the productivity rates associated with large-system development. Of greater significance is the year-to-year variation in productivity rates that became more pronounced. The CSI/TDP ratio is calculated using the CSI shipped in a given year, divided by the total number of development people available that year for product development and support. Because code is not shipped at a constant rate, there are years of higher and lower productivity. In Figure 2, the year-to-year changes are shown with a nonlinear regression curve fitted to the data. This curve shows that the compound growth rate in productivity is about seven percent, which is similar to estimated productivity when the direct people were normalized. This measure is very useful in showing overall trends, and it is consistent. On the other hand, it has limited application in the analysis by product class, activity distribution, and

Figure 2 Year-to-year changes in productivity overlaid with a fitted compound growth rate curve showing a nonlinear increase in productivity

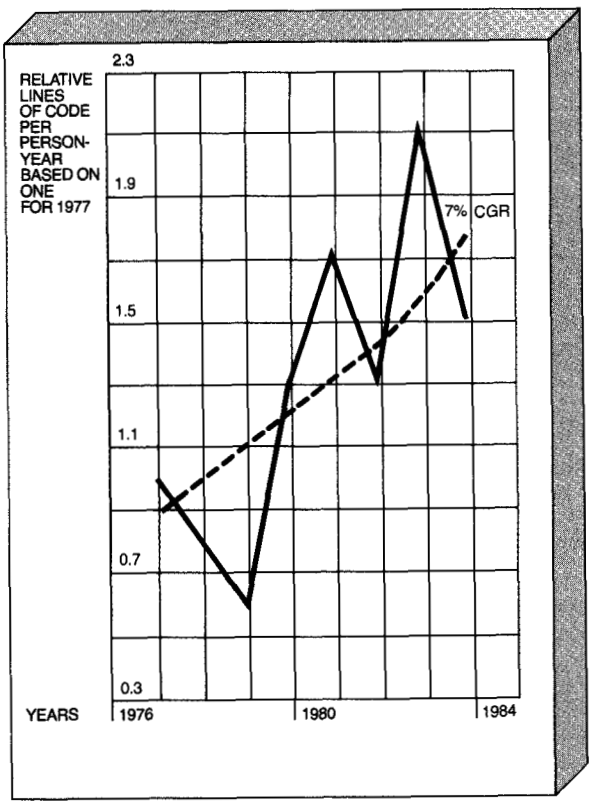
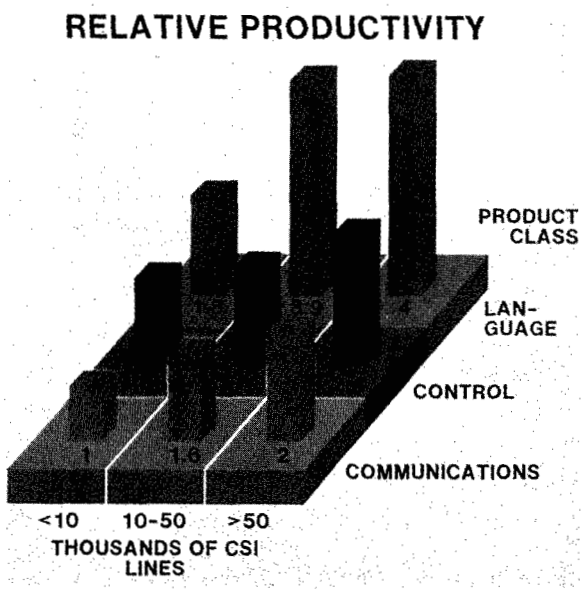


Figure 3 Relative productivity by program size and product class



size. Therefore, direct people are still tracked to provide insight into the effect of tools and process changes at the detail level.

Although the overall productivity trend for large-system products is increasing at the rate of seven percent, individual products show wide variability. To understand this variability, we investigated the characteristics of the products. Each product in the data base has been given one of the following classifications, according to the system function it supports: control program, control program support, subsystem controller, data base access, communications, and languages. At first, productivity rates were compared by class or size. We found that the products classified as communications have the lowest average productivity and that languages have the highest measured productivity in terms of CSI and direct people. Differences in functional complexity between the product classes are the most likely explanation for this changeability in productivity rates. In terms of size alone, as the CSI increased, productivity also increased. By 1982, we had sufficient data to compare productivity by class and size (Figure 3). The average productivity difference between small communications products and small language products is not significant. The average productivity for large communications products is twice that for small communications products. The difference in productivity between large language products and small communications products is about 4 to 1.

Productivity rates range from one to eleven for ninety percent of the products. The fact that the products with the smallest CSI are on the average the least productive is surprising, because many studies have shown just the opposite—the larger the product the lower the productivity.[7] Paulsen[8] and Boydston[9] compared productivity percentage using the ratio of change code (the CSI) to the total product code shipped, as shown in Figure 4. This gives an idea of the effect on productivity of modifying product code compared to producing all-new product code. It appears that modified-to-new-code productivity differences are greater than productivity differences between product classes.

Variability in definitions

The greatest range of productivity we found in the product set we studied was about eleven to one. However, productivity rates ranging as high as 56 to one have been reported in the software productivity literature.[10,11] Because there are wide variations in

measurement definitions, we used the Large-System Productivity Data Base to try to understand the effect varying the definitions for lines of code and people. We substituted assembler-equivalent code for CSI. We then used people and several alternative definitions for counting lines of code to recalculate the large-system productivity rate. The results of using these alternative definitions are discussed in the next two sections.

Assembler equivalent. Many development organizations produce products using multiple languages. In IBM, both assembler and high-level languages are used. When their respective uses remain proportional to one another over time, the distinction between languages is not significant for measurements. The trend for the large-system products has been to increase the use of high-level languages. A high-level language requires fewer source code statements for a given function than does assembler language.[12] This situation gives the impression that productivity decreases or remains constant when assembler language is replaced by a high-level language, when in fact it may have increased.

A way of comparing assembler and high-level languages is to convert the high-level language statement counts to equivalent assembler language statement counts. Figure 5 shows relative productivity based on equivalent assembler language code. The expansion rate from the PLS compiler to the assembler equivalent is an estimate based on a sample of our own data and those from other sources.[1] Whatever factor is selected, the result is to increase the productivity rate. The use of assembler-language-equivalent instructions requires that the tool used to count source statements be modified to count the new and modified assembler language statements generated by the compiler for CSI. Normalizing to assembler-equivalent instructions (for CSI) has the effect of doubling the compound growth rate for large-system products.

For the large-system products, data on productivity factors such as those in use by Walston and Felix[13] and Boehm[14] are not collected. There is general agreement, however, that people and their experience, data processing support, tools, and process control are all factors that contribute to increased productivity. There is continued focus on these factors as the means to sustain future productivity and quality improvements.

Variability in people and lines of code. Figure 6 shows the relative expansion in productivity under

Figure 4 Relative productivity by product class using the ratio of percent of change code to total product code shipped

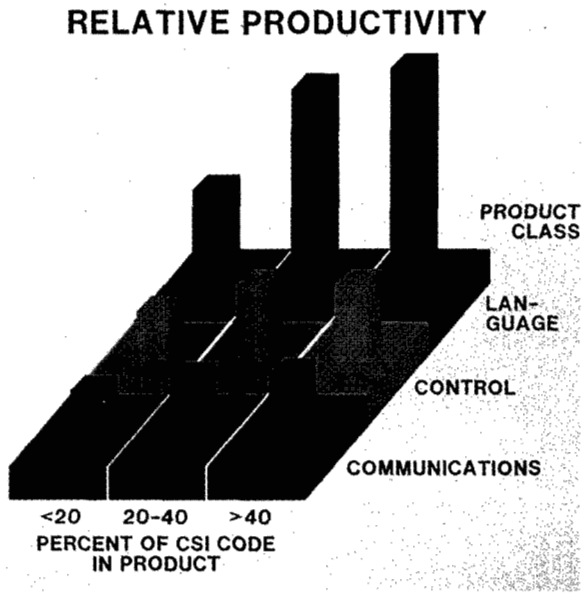


Figure 5 Relative productivity based on equivalent assembler language code overlaid with a fitted compound growth rate curve

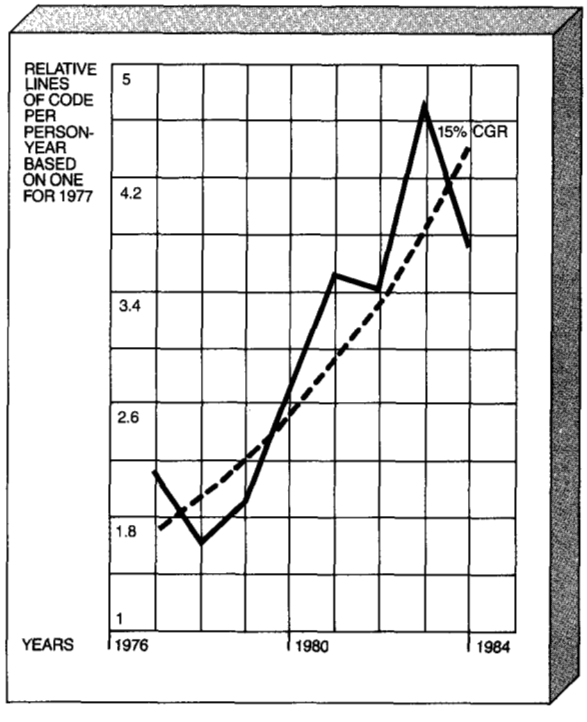
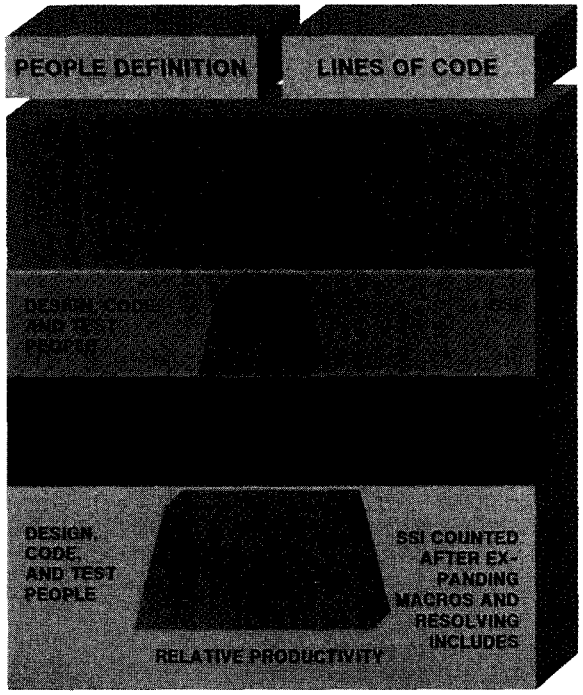


Figure 6 Relative numbers of lines of code and people compared using several definitions



several definitions of people and LOC for the same set of products. Each counting method is useful for achieving a given objective, such as estimating code development or service costs. Difficulties occur when the rates are used out of definitional context, and, as indicated previously, the resulting variability can be quite significant.

It is sometimes useful to analyze the productivity of the design, code, and unit test resources only. This productivity rate is four times that of the CSI per total number of development people. For a productivity calculation, the people may be only those related to design, code, and unit test.

Consider the effect of variability in the definition of LOC. In place of CSI, suppose the total of shipped source instructions (SSI) is used as a measure. SSI includes the new and modified code plus the base code shipped as a product. SSI is useful for estimating field service costs and represents the product size as seen in the field. This measure can be as much as thirteen times greater than CSI per total number of development people (TDP).

The reuse of code has been cited as a method for improving productivity.[15] Macros and included code are examples of reused code; these have been incorporated into large-system products for many years. When LOC is measured after macro expansion and the includes have been resolved, the estimated productivity is fifty-two times that for CSI per TDP.

There are other ways of looking at productivity, such as the previously discussed assembler-equivalent LOC, the number of unique operating systems supported by common code, and the count of card images versus source statements. The estimated productivity possible using various definitions exceeds two hundred fifty times that for our CSI per total number of development people.

Concluding remarks

Software productivity rates are not easily compared. There is significant variability in rates, depending on the standard definitions used for counting lines of code and people. Although comparisons based on different definitions may not be commensurable, a measurement system that is consistent and accurate in data collection for a specific set of products and processes can provide useful information on factors that affect long-term trends in productivity. Within a process and product set, the variability caused by functional complexity between classes of products is not as great as the differences in productivity rates between new and modified products.

While the overall trends based on information available today are very useful, future productivity measurements must focus on the tasks within the process stages in greater detail. Thus, a future requirement will be to understand the effect on productivity and quality of specific changes in processes and tools.

Increased attention to productivity and measurements is leading to a rethinking of software metrics. Today there are a number of possible alternatives to lines of code and complexity measurements. In place of lines of code there is the concept of function, or "chunks," but this metric requires further research. Function points as a metric is better developed as an alternative to lines of code, but this metric requires study of its application to modified code for large-system products. To account for complexity, there is the possibility of combining metrics of varying complexity, as proposed by Bays[16] and Jones.[17]

Cited references

1. B. W. Boehm, *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1981).
2. L. J. Arthur, *Programmer Productivity, Myths, Methods and Murthology*, John Wiley & Sons, Inc., New York (1983).
3. J. R. Johnson, "A working measure of productivity," *Datamation* **23**, No. 2, 106–112 (February 1977).
4. M. H. Halstead, *Elements of Software Science*, Elsevier Science Publishing Co., Inc., New York (1977).
5. C. P. Smith, *A Software Science Analysis of IBM Programming Products*, Technical Report TR 03.081, IBM General Products Division, Santa Teresa, CA (January 1980).
6. A. J. Albrecht and J. E. Gaffney, Jr., "Software function, source lines of code and development effort prediction: A software science validation," *IEEE Transactions on Software Engineering* **SE-9**, No. 6, 639–648 (November 1983).
7. R. W. Wolverton, "Cost of developing large scale software," *IEEE Transactions on Computers* **C-23**, No. 6, 615–636 (June 1974).
8. L. Paulsen, "The implications of program composition and size on development productivity," *Proceedings, Twenty-Third IEEE Computer Society International Conference*, Washington, DC (September 1981), pp. 149–155.
9. R. Boydston, "Programming cost estimates: Is it reasonable?", *7th International Conference on Software Engineering*, Orlando, FL (March 1984), pp. 153–159.
10. K. H. Kim, "A look at Japan's development of software engineering technology," *Computer* **16**, No. 5, 26–37 (May 1983).
11. R. C. Kendall and E. C. Lamb, "Management perspectives on programs, programming productivity," *Guide 45*, Atlanta, GA (November 1977).
12. T. C. Jones, "Measuring programmer quality and productivity," *IBM Systems Journal* **17**, No. 1, 39–63 (1978).
13. C. E. Walston and C. P. Felix, "A method of programming measurement and estimation," *IBM Systems Journal* **16**, No. 1, 54–73 (1977).
14. B. W. Boehm, "A software development environment for improving productivity," *Computer* **17**, No. 6, 30–42, 44 (June 1984).
15. P. Gillin, "T. Capers Jones on life without programmers," *Computerworld* **18**, No. 22, Special Report 3, 6 (May 28, 1984).
16. M. Bays, "Development of a programming productivity measurement system," *Journal of Information Management* **4**, No. 3, 21–34 (Spring/Summer 1983).
17. C. Jones, *Programming Productivity: Issues for the Eighties*, IEEE Catalog No. EHO 186-7, available from IEEE Service Center, 445 Hoes Lane, Piscataway, NJ 08854 (1981).

Michael J. Flaherty *IBM Information Systems and Storage Group, P.O. Box 390, Poughkeepsie, New York 12602.* Mr. Flaherty is an advisory programmer in the Programming Quality Analysis Department. He is involved primarily in the measurement of software process productivity, with special interest in the application of advanced software metrics. Mr. Flaherty received the M.Sc. degree in computer science from Union College, Schenectady, NY.

Reprint Order No. G321-5246.