Improvements in programming technology have paralleled improvements in computing system architecture and materials. Along with increasing knowledge of the system and program development processes, there has been some notable research into programming project measurement, estimation, and planning. Discussed is a method of programming project productivity estimation. Also presented are preliminary results of research into methods of measuring and estimating programming project duration, staff size, and computer cost.

A method of programming measurement and estimation

by C. E. Walston and C. P. Felix

New materials technologies and architectures are significantly affecting computing system hardware. While the cost per bit of storage and the execution cost per instruction have both been decreasing, the same trend has not been true for software. Since software development has continued to be a people-oriented activity, a higher percentage of the cost to acquire a computing system is accruing to software development.[1]

New management and programming techniques have been developed to improve programming efficiency. Among the improved programming technologies are the following:

- *Chief programmer team*, a programming organization built around a chief programmer, a backup chief programmer, and a librarian that effectively produces code in a disciplined and open environment.[2]
- *Hierarchy plus Input-Process-Output* (HIPO), a graphic design and documentation method that is used to describe program functions from the topmost level to great detail.[3]
- *Development support library*, a tool that provides current information about project programs, data, and status.[4]
- *Structured programming*, a programming method based on the mathematical Structure Theorem[5] that enables programmers to understand and enhance programs that have been written by others,[6] as well as one's own programs.
- *Design and code inspections*, a review of program design, code, and documentation to detect errors prior to program execution.[7]
- *Top-down development*, an ordering of program development and testing that begins at the topmost functional level and proceeds decrementally to the lowest functional level.

This paper discusses research into programming measurements, with emphasis on one phase of that research: a search for a method of estimating programming productivity. The method we present is aimed at measuring the rate of production of lines of code by projects, as influenced by a number of project conditions and requirements. We do not, however, measure the performance of individual programming project members. As we continue our research, we are continuing to learn more about the attributes of the programming process, about programming itself, and about better ways of analyzing the data.

Before starting the programming measurements research reported here we analyzed the literature on programming measurements and productivity that includes the work of Aron,[8] Weinwurm and Zagorski,[9] and Nelson.[10] We also wanted to isolate the effects of improved programming technologies from the effects of monetary inflation, variations in computing cost, and ambiguities in the definition of computing quantities (such as that of the size of the delivered program product).

The software measurements project began in 1972 when we decided to assess the effects of structured programming on the software development process. To do that, a rigorous program was established to measure the then current software methodologies so that changes that result from the introduction of new methodologies could be measured. The initial phase of the measurements program was to identify variables to be measured, to design a questionnaire (called the Programming Project Summary questionnaire), and to develop a system for processing the data to be collected. The first report entered the system on January 23, 1973, and data have continued to be received and entered into the system from that time on. Since the establishment of the Programming Project Summary, two new reporting formats have been designed: the Software Development Report and the Software Service Report.

objectives

We understood at the outset that the established objectives might change as the project matured and as data were accumulated. Current objectives of the project discussed in this paper are the following:

- Provide data for the evaluation of improved programming technologies.
- Provide support for proposals and contract performance.
- Gather and preserve historical records of the software development work performed.
- Provide programming data to management.
- Foster a common programming terminology.

Table 1 Software project measurements reports

Report Name	*Nature of the Report*
Programming Project Summary	Detailed report on the software development environment, the product (including errors), resources, and schedule.
Software Development Report	Detailed report on the software development environment, the product (including changes and errors), resources and schedule.
Monthly Software Development Report	One report on product, resource, and schedule status. Changes in software development environment are noted.
Software Service Report	Report on project size and on the software service environment.
Quarterly Software Service Report	Detailed data on product being serviced, resource status, and changes in software service environment.

Key to a software measurement program are the analysis of measurements and feedback of results to the suppliers of the raw data and to management. This paper discusses the data reporting and analysis in the software measurements program of the IBM Federal Systems Division. Described first are the measurements data base, the services that are available to users of the data, and descriptive statistics from the data base. The next section covers the analysis of programming productivity and describes a productivity estimation technique. Following this, the results of other analyses that have been performed with respect to factors such as documentation, staffing, and computer costs are presented. The last section briefly describes analysis efforts that are being contemplated for future research. In the measurements project discussed here, a sufficiently large quantity of data is being collected so that a programming project measurement system is used to provide the necessary flexibility and capability of storage control and analysis. The Programming Project Measurement System is briefly described in the Appendix.

Measurements data base

Data contained in questionnaires submitted by line projects at prescribed reporting periods are stored in a computer data base where they are accessible for answering queries, preparing reports, or for analytical studies.

The basic measurements data base is structured so that it contains all the reports received, as described in Table 1. Each Pro-

gramming Project Summary milestone report is given a unique number when received and is assigned a separate record in the file. Project milestones are the following: start of work; preliminary design review; midway through software development; acceptance test completion; and every three months during the maintenance or service phase. Each initial and final Software Development and Software Service report is also filed as a separate record.

The monthly Software Development Reports plus any changes to the initial submission constitute separate records, as do the quarterly Software Service Reports and any changes to the initial Software Service Report. Each record is structured into fields, or variables, that correspond to the question response fields in the questionnaires.

Sixty completed software development projects are now in the data base. These completed projects, which represent a wide variety of programming technology, are summarized in Table 2. Their delivered source lines of code range from 4000 to 467000, and their effort ranges from 12 to 11758 man months. These programs include real-time process control, interactive, report generators, data-base control, and message switching programs. Some of the programs have severe timing or storage constraints and other programs have both types of constraints. Twenty-eight different high-level languages and 66 different computers are represented in the listing in Table 2.

services

With the previously described data base and with the capabilities provided by the Programming Project Measurement System, a wide variety of services can be provided to line project personnel. Queries can be answered, analyses performed, and programming project productivity estimation provided for new or on-going projects. Examples of services are discussed throughout the remainder of this paper.

Queries that can be answered by direct retrieval of data from the data base are of two general types. First is a request for data about a specific project. In this case, the Programming Project Measurement System generates a report that lists the answers to the questions submitted by personnel of the specified project. The second type of query is a more general request for information, an example of which is, "What has been the utilization of improved programming technologies by projects in the data base and what was the effect on project productivity?" The answer is provided in the form of a listing of productivity data sorted by project. The following breakdown indicates the types of reported and derived data that can be provided in response to a general inquiry:

Table 2 Characterization of programs in the programming project data base

A. Small less-complex systems
- Batch storage and retrieval
- Batch inventory
- Batch information management
- Batch languages preprocessor and information management
- Batch reporting
- Batch financial information
- Batch scientific processing simulation
- Batch utility
- Batch operating system exerciser

B. Medium less-complex systems
- Special-purpose data management (2)
- Batch storage and retrieval (2)
- Process control simulation
- Batch reporting
- Batch data-base utility (2)
- On-line scientific processing simulation
- Batch on-line scientific information management
- On-line business information management
- On-line storage and retrieval
- Batch hardware test support
- Batch scientific algorithm feasibility (3)
- Interactive scientific processing (2)
- System test support
- Batch planning (3)
- Batch military information management
- Special-purpose operating system

C. Medium complex systems
- Real-time, special-purpose system exerciser
- Special-purpose operating system (2)
- Batch information modification
- Batch information conversion
- Data management
- Sensor-based mission control
- On-line scheduling
- Sensor-based mission simulation
- Interactive scientific processing
- Process control (3)
- On-line graphics
- System performance monitoring and measurement (3)
- Terminal data management
- Interactive information conversion
- Operating system extensions

D. Large complex systems
- Sensor-based mission monitoring and control
- Interactive information acquisition
- Process control
- Sensor-based system exerciser (3)
- Sensor-based mission processing and communication (2)

Programming project data:

- Number of lines of delivered source code ordered by project. (Source lines are 80-character source records provided as input to a language processor. Job control languages, data definitions, link edit language, and comment lines are included. Reused code is not included.)

Table 3 Programming project descriptive data report for completed projects

	Median 50%	*Quartiles* 25–75%
Productivity		
Source lines per man month of total effort	274	150–440
Product		
Source lines (thousands)	20	10–59
Percentage of code lines not to be delivered	5	0–11
Documentation per thousand lines of source code (pages)	69	27–167
Resource		
Computer cost (as a percentage of project cost)	18	10–34
Total effort (man months)	67	37–186
Average manning	6	3.8–14.5
Effort distribution of preliminary design review (percent)	18	11–25
Distribution of effort (percent)		
Management and administration	18	12–20
Analysis	18	6–27
Programming and design	60	50–70
Other	4	0–6
Duration (months)	11	8–19
Development error detection		
Errors per thousand source lines	3.1	0.8–8.0
Incorrect function	76	50–86
Omitted function (percent)	8	0.22
Misinterpreted function (percent)	17	7–25
Errors per programming man month	0.9	0.3–2.4

- Pages of documentation (including program source listings) delivered.
- Source languages used to develop code.

Resource data:

- Total effort (in man-months, including management, administration, analysis, operational support, documentation, design, coding, and testing) required to produce the lines of source code by project.
- Duration of project in months.

Use of improved programming technologies (expressed as a percentage of code developed using each technique):

- Structured programming.
- Top-down development.
- Chief programmer team.
- Design and code inspections.

Table 4 Data for completed service projects

	Median 50%	*Quartiles* 25–75%
Product		
Lines of maintained source code (thousands)	103	56–474
Ratio of developed to maintained code	0.04	0–0.19
Resources		
Average manning per project	6	4–11
Maintained source lines per man (thousands)	15	5–24
Distribution of effort		
Management and administration (percent)	15	10–20
Analysis (percent)	10	4–30
Programming (percent)	72	40–80
Other (percent)	3	0–7
Duration (months)	18	11–31
Errors detected		
Errors per thousand lines of maintained code	1.4	0.2–2.9
Incorrect function (percent)	73	
Omitted function (percent)	11	
Misinterpreted function (percent)	13	

Table 5 Descriptive data from on-going projects

Programming	*Median* 50%	*Quartiles* 25–75%
Source lines (thousands)	12.5	5.4–30
Percentage of code lines not to be delivered	2.6	0–11
Effort (man months)	72	30–205
Distribution of effort:		
System analysis (percent)	15	10–20
System design (percent)	20	15–25
Code and unit test (percent)	30	20–35
Integration and test (percent)	20	15–30
Other (percent)	5	0–10
Duration (months)	12	9–18

Service	*Median* 50%	*Quartiles* 25–75%
Maintained source lines (thousands)	148	55–340
Ratio of developed to maintained code	0.05	0–0.09
Effort (man months)	88	27–185
Average manning (persons)	5	3–19
Duration (months)	10.5	6.5–12

Derived data:

- Productivity achieved, ordered by project. *Programming productivity* is defined as the ratio of the delivered source lines of code to the total effort (in man-months) required to produce the lines of code, and is computed from product and resource data.
- Average number of people required to work on the project, computed by dividing the total effort in man-months by the duration of the project in calendar months.

More complex and extensive search and analysis questions can also be answered. These are supported by a question analysis subsystem of the Programming Project Measurement System, which incorporates a statistical package for the manipulation and statistical analysis of many types of data. The more complex requests can be grouped into two types, descriptive and analytical. A descriptive request requires searching the data base for specific variables or derived variables and computing characteristics about their distributions such as the mean, mode, and standard deviation, in order to prepare the report. Table 3 illustrates one such report for the completed projects in the data base.

Because of the variability in the measurement data, the statistics in Table 3 are presented in terms of medians and quartiles. The median for the size of the delivered software product is 20 thousand lines and fifty percent of the projects reported that the sizes of their delivered source code ranged from 10 thousand to 59 thousand lines. The effort for software development reported was distributed into the major categories as shown. The error detection section of the table shows the distribution of errors reported during the development phase.

Table 4 provides descriptive statistical data for completed service projects. Most service activity is not purely maintenance, but includes development efforts as well. The ratio of developed source lines of code to maintained lines of code was 4 percent at the median.

Table 5 presents data on on-going programming and service projects. Since only a small percentage of these projects have been completed at this time, the statistics represent a mixture of actual measurements and estimates at various stages of completion.

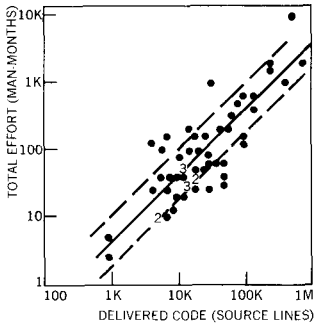
Productivity analysis

We have identified five major parameters that can help programming project personnel make estimates. These parameters, pro-

ductivity, schedule, cost, quality, and size, are listed in order of increasing difficulty and complexity of analysis. Some of the difficulties arise from a lack of detailed data in the data base, as in the case of schedule data. Complexity of the quantitative data can create other difficulties. One significant difficulty is in identifying and measuring independent variables that can be used to estimate the desired variable, as is the case in estimating the size of the product to be delivered.

Productivity, which can be defined in terms of the quantitative measures that are in the data base, is a vital factor in all software estimating processes and, therefore, is of immediate value to project personnel. For this reason, the analysis performed to date has focused on productivity estimation. Productivity has often been defined as the ratio of output to input. *Programming productivity* is defined here as the ratio of the delivered source lines of code (DSL) to the total effort in man-months (MM) required to produce that delivered product.

Figure 1 Relationship between delivered lines of code and effort



The basic relationship between delivered lines of code and effort is shown in Figure 1. Each plotted point represents the data reported by a completed project in the data base. A number on the chart indicates a position where a number of data points are grouped sufficiently close together that they cannot be individually identified on the plot. The data have been plotted in the log-log domain so that they become approximately linear. The linear coefficients become power relationships when transformed back to the original domain of the data. The least squares fit to the data as plotted in Figure 1, yields the result:

$$E = 5.2L^{0.91}$$

where

E = total effort in man months

and

L = thousands of lines of delivered source code.

productivity index

This relationship is nearly a first-power (or linear) relationship between effort and product size. The dashed lines indicate the standard error of the estimate on either side of the least squares fit. To identify the sources of scatter or variation of Figure 1, those variables that are related to productivity have been investigated. Preliminary findings have led to the development of a productivity estimator that provides an on-line capability to support proposed as well as on-going projects. A set of sixty-eight variables was selected from the data base, and those variables were analyzed to determine which were significantly related to productivity. Twenty-nine of the variables showed a significantly high correlation with productivity and have there-

fore been retained for use in estimating. Table 6 lists these variables and the reponses associated with them. To illustrate the meaning of Table 6, consider the first entry, which is derived from a multiple-choice question that asks the information supplier to circle his response to the following statement: Customer interface is (less than, equal to, greater than) normal complexity.

When the mean productivity was computed for all the completed reports in the data base that indicated less than normal customer interface complexity, the result obtained was 500 delivered source lines of code per man-month of effort (DSL/MM). By a similar computation, the mean productivity for all projects that reported normal complexity was 295 DSL/MM, and the mean productivity for those reporting greater than normal complexity experience was 124 DSL/MM. The change in productivity between less-than-normal and greater-than-normal customer interface is 376 DSL/MM, as noted in the final column in Table 6. Three variables in the table (overall personnel experience, code complexity, and design constraints) were formed by combining the answers to several questions in the questionnaire. It should be noted that this analysis was performed on each variable independently and does not take into account either the possibility that these variables may be correlated, or that there may be interrelated effects associated with them.

The twenty-nine variables were then combined into an index, based on the effect of each variable on productivity, as indicated by the above analysis, to form a productivity index. The productivity index is computed as follows:

$$I = \sum_{i=1}^{29} W_i X_i,$$

where

I = productivity index for a project

W_i = question weight, calculated as one-half $\log_{10}$ of the ratio of total productivity change indicated for a given question i

X_i = question response (+1, 0 or −1), depending on whether the response indicates increased, nominal, or decreased productivity

An index was computed for fifty-one projects, and a plot of actual productivity for each project versus the computed productivity index and the least squares fit to this relationship is shown in Figure 2. The standard error of the estimate (standard deviation of the residuals) is shown as dashed lines.

rapid estimates

To support project estimates, a shortened version of the data collection form is used that contains excerpted questions associated with the twenty-nine variables used in the index. A

Table 6 Variables that correlate significantly with programming productivity

Question or Variable	Response Group / Mean Productivity (DSL/MM)			Productivity Change (DSL/MM)
Customer interface complexity	Normal 500	Normal 295	> Normal 124	376
User participation in the definition of requirements	None 491	Some 267	Much 205	286
Customer originated program design changes	Few 297		Many 196	101
Customer experience with the application area of the project	None 318	Some 340	Much 206	112
Overall personnel experience and qualifications	Low 132	Average 257	High 410	278
Percentage of programmers doing development who participated in design of functional specifications	< 25% 153	25–50% 242	> 50% 391	238
Previous experience with operational computer	Minimal 146	Average 270	Extensive 312	166
Previous experience with programming languages	Minimal 122	Average 225	Extensive 385	263
Previous experience with application of similar or greater size and complexity	Minimal 146	Average 221	Extensive 410	264
Ratio of average staff size to duration (people/month)	< 0.5 305	0.5–0.9 310	> 0.9 173	132
Hardware under concurrent development	No 297		Yes 177	120
Development computer access, open under special request	0% 226	1–25% 274	> 25% 357	131
Development computer access, closed	0–10% 303	11–85% 251	> 85% 170	133
Classified security environment for computer and 25% of programs and data	No 289		Yes 156	133

program, running in the Time Sharing Option of OS/VS (TSO) was developed to compute and list the index estimates. This terminal-based program allows rapid response to project requests for information. The estimate of expected productivity is returned to

Question or Variable	*Response Group* / *Mean Productivity (DSL/MM)*			*Productivity Change (DSL/MM)*
Structured programming	0–33% 169	34–66% –	66% 301	132
Design and code inspections	0–33% 220	34–66% 300	> 66% 339	119
Top down development	0–33% 196	34–66% 237	> 66% 321	125
Chief programmer team usage	0–33% 219	34–66% –	> 66% 408	189
Overall complexity of code developed	< Average 314		> Average 185	129
Complexity of application processing	< Average 349	Average 345	> Average 168	181
Complexity of program flow	< Average 289	Average 299	> Average 209	80
Overall constraints on program design	Minimal 293	Average 286	Severe 166	107
Program design constraints on main storage	Minimal 391	Average 277	Severe 193	198
Program design constraints on timing	Minimal 303	Average 317	Severe 171	132
Code for real-time or interactive operation, or executing under severe timing constraint	< 10% 279	10–40% 337	> 40% 203	76
Percentage of code for delivery	0–90% 159	91–99% 327	100% 265	106
Code classified as non-mathematical application and I/O formatting programs	0–33% 188	34–66% 311	67–100% 267	79
Number of classes of items in the data base per 1000 lines of code	0–15 334	16–80 243	> 80 193	141
Number of pages of delivered documentation per 1000 lines of delivered code	0–32 320	33–88 252	> 88 195	125

the requester in the form of a report that contains a comparison between the project estimate and the one derived from the data base. Also included is a list of the reported attributes or variables that had a significant influence on the estimate. Where possible, detailed discussions are held on special factors associated with a project that may not be properly handled in the present algorithm.

Figure 2 Relationship between productivity and productivity index for twenty-nine variables

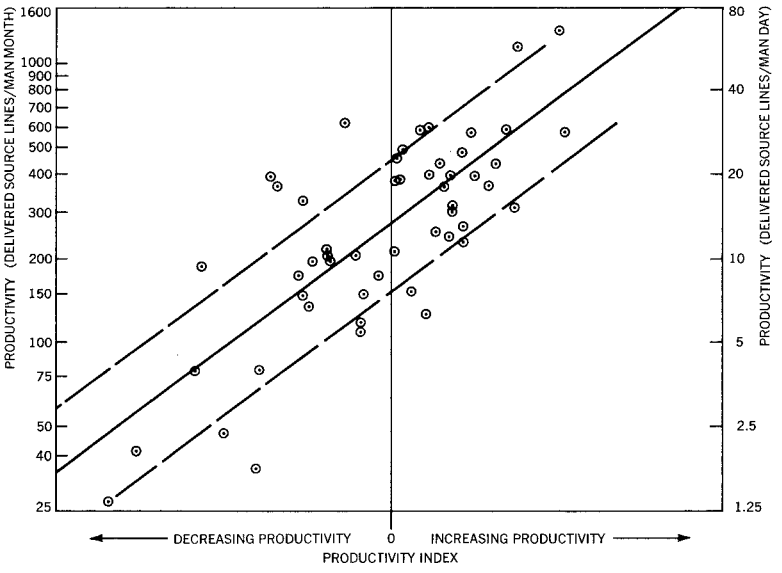


Figure 3 Estimated productivity for a hypothetical project

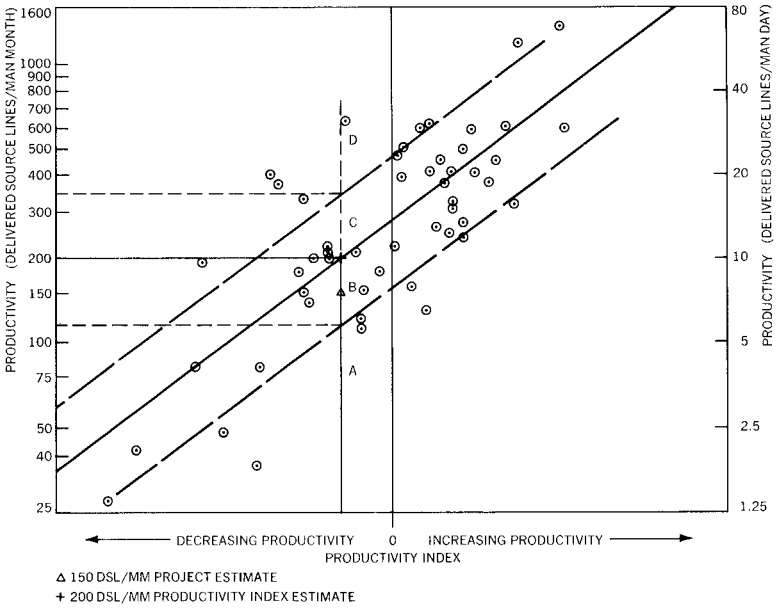


Figure 3 is a plot similar to the one shown in Figure 2, which is presented for a hypothetical project. The productivity index is computed for the project from responses to the proposal questionnaire and yields the expected productivity to be attained, as determined by the measurements data base. In the case shown in Figure 3, the estimated productivity is seen to be two hundred

Table 7 Additional productivity related variables

Variable	*Response Change (low to high percent)*	*Productivity Change* (percent)*
Ratio of developed to original plus developed code	0 to 100	705
Effort at primary development location	50 to 100	215
Remote job entry computer access	0 to 100	205

*Based on least squares fit to data in Figure 4.

delivered source lines of code per man-month (DSL/MM), with one standard error range of 115 to 340 DSL/MM. The project team's independently developed productivity estimate for the same conditions was 150 DSL/MM. Thus, in this case, the project estimate is a more conservative estimate than that given by the productivity index.

other estimates

Consider additional conclusions that can be drawn from Figure 3. If we assume a normal distribution for the observations, when they are plotted as a log of the productivity versus a log of the productivity index in Figure 3, the probability P_A that the project would have a productivity estimate in region A (i.e., less than 2.06 or the $\log_{10}$ of 115 DSL/MM) is about 0.17. The probability that the productivity estimate would be in region B (i.e., between 2.06 and 2.30, the log of 115 and 200 DSL/MM) is about 0.33. Similarly, P_C is 0.33 and P_D is 0.17. This is the probability distribution of productivity estimates, not the cumulative probability that a project will (or will not) achieve or exceed the productivity that was estimated.

Investigation is continuing into other variables from the data base that may also be related to productivity. Figure 4 shows several distributions that appear to have a significant relationship to productivity, although in two of these cases they are based on a limited number of observations. Table 7 expresses the net effect of the data plotted in Figure 4 in tabular form. Figure 4(A) shows productivity (in source lines of code per man month) plotted against the ratio of developed source code to the sum of any original (or reused) code plus the developed code. The plot in Figure 4(A) suggests that productivity is highest when there is no original or reused code, that is, when all the code is developed from the inception of the project. As the percentage of reused code grows, the expected productivity decreases.

Figure 4 Additional productivity relationships

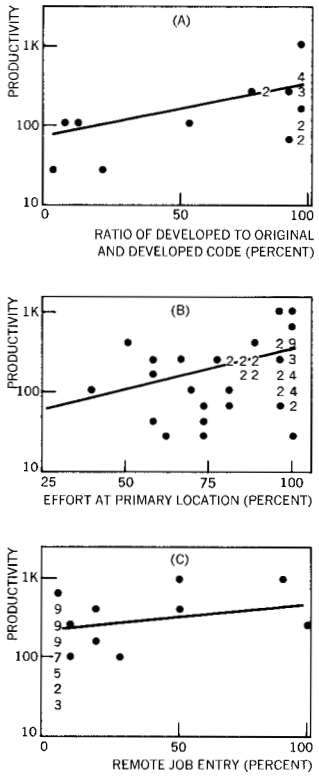


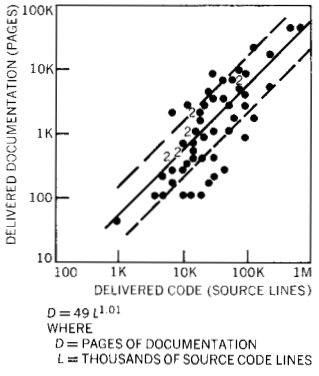
Figure 4(B), although it contains a large amount of scatter, suggests that when the development effort is spread across more than one location, i.e., as the percentage of effort at the primary location becomes less than 100 percent, the productivity decreases. Another question currently of interest is the impact of remote job entry on productivity. Most of the completed projects in the data base were developed without the use of terminals, as Figure 4(C) shows. On the basis of a least squares fit, however, those projects that use remote job entry do appear to have an increase in productivity.

Other results of programming analysis

Although the primary effort has been directed toward productivity analysis, other analyses have been performed on the data base. Results of these efforts to the present time are presented here. The data can be used to check productivity estimates, and to check current project parameters against past experience, as reflected by the data base. Such results provide a multidimensional approach to crosschecking a number of the factors that enter into estimates of effort: productivity, duration, documentation, and computer costs. These results also indicate the nature of the analyses that can be performed against the data base.

Documentation is a critical product of every software project, and documentation costs are an important component of the estimation process. A useful parameter for measuring documentation is number of pages. Figure 5 is a plot of delivered documentation in number of pages versus delivered source lines of code. Documentation is defined here as program functional specifications and descriptions, users' guides, test specifications and results, flow charts, and program source listings that are delivered as part of the documentation. As a first approximation, the least squares fit indicates that a linear or first-order relationship exists; that is, the number of pages of delivered documentation varies directly as the number of lines of source code.

Figure 5 Relationship between documentation and delivered code



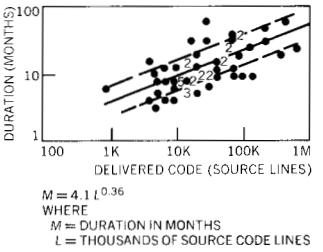
After programming project estimates have been completed, those estimates can be checked against the data base by using the plots in Figures 5–10. If, for example, the size of the delivered software product is estimated as ten thousand lines of source code (as shown in Figure 5) it can be seen from past experience that the expected number of pages of documentation to be delivered is five hundred. The range for one standard error for this given value is one hundred eighty to thirteen hundred pages. This provides an independent calibration point that the manager can use to compare his estimate against the experience of past projects. A significant difference between the two does

not necessarily imply an error on the part of the manager, but it does suggest that the assumptions and estimates might be re-examined.

project duration

The question of how much time to allow for the development of software is always difficult to assess. The relationship between duration (expressed in months) and delivered source lines of code is shown in Figure 6. Project duration as a function of total effort in man months is shown in Figure 7. Initial analysis indicates that a cubic relationship fits the data in both of these figures. This implies that the duration of effort increases by the cube root of the number of source lines of code delivered or by the cube root of the total effort applied to the development of the code. For example, for a project that is developing a software product of 10 thousand lines of source code, the expected duration of the effort is $4.0 \times 10.0^{0.38}$ or 9.6 months. Figure 7 does not imply that simply reducing total effort automatically permits a reduction in project duration. Such a reduction would more likely make it impossible to produce and test the required volume of code.

Figure 6 Relationship between project duration and delivered code



$M = 4.1\,L^{0.36}$
WHERE
M = DURATION IN MONTHS
L = THOUSANDS OF SOURCE CODE LINES

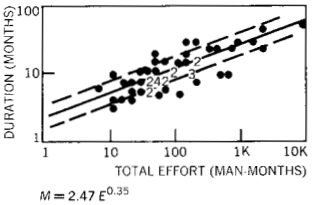
staff size

The staff size utilized to develop a given software product is influenced by a number of factors, including the time allowed for development, the amount of code to be developed, and the staffing rates that can be achieved. After a project has been estimated, one convenient measure used to describe the size of the project is the average number of people required. Figure 8 shows a relationship that can be used as another check on the estimating process. It shows the relationship between the staff size—expressed in terms of the average number of people (defined as total man-months of effort divided by the duration)—and the total effort applied.

computer cost

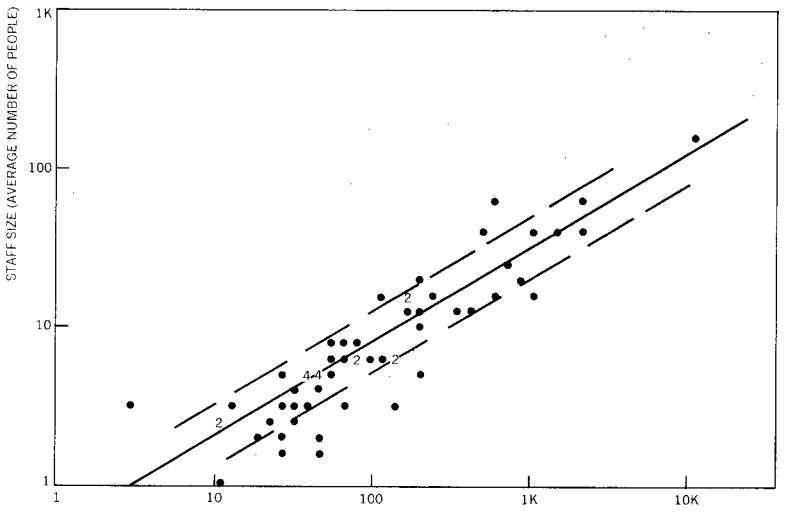
Estimating computer costs is very difficult, but at the same time it can also be a very significant fraction of the total cost. Although only eighteen of the completed projects in the data base had computer costs reported, some interesting relationships are indicated when computer costs are compared with the amount of delivered code and the total effort, as is shown in Figures 9 and 10. In Figure 9, two observations (circled) are evidently out of bounds when plotted against delivered code. These same two observations, however, fit well with total effort, as shown by the plot in Figure 10. Based on this limited evidence, it appears that computer costs are closely related to effort, and they appear to have nearly a first power (or linear) relationship. Note that in Figure 9, the two out-of-bounds points are not included in determining the least-square fit.

Figure 7 Relationship between project duration and total effort



$M = 2.47\,E^{0.35}$
WHERE
M = DURATION IN MONTHS
E = EFFORT IN MAN MONTHS

Figure 8 Relationship between average staff size and total effort



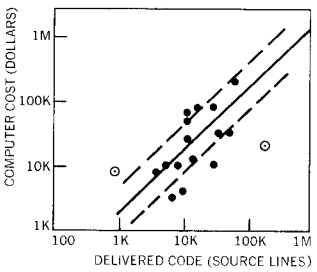
$S = 0.54\,E^{0.6}$
WHERE
S = AVERAGE NUMBER OF PEOPLE ON STAFF
E = TOTAL EFFORT IN MAN MONTHS

Concluding remarks

regression method

The present approach to productivity estimation, although useful, is far from being optimized. Based on the results of the variable analysis described in this paper, and supplemented by the results of the continued investigation of additional variables related to productivity, an experimental regression model has been developed. Preliminary results indicate that the model reduces the scatter. Further work is being done to determine the potential of regression as an estimating tool, as well as to extend the analyses of the areas of computer usage, documentation volume, duration, and staffing.

Figure 9 Relationship between computer cost and delivered code



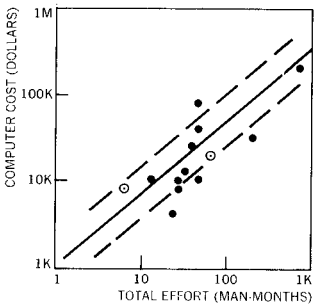
$C = 1.84\,L^{0.96}$
WHERE
C = COMPUTER COST IN THOUSANDS OF DOLLARS
L = THOUSANDS OF SOURCE CODE LINES (TWO OUTLIERS EXCLUDED FROM LEAST SQUARES FIT.)

Appendix

The effective utilization of programming measurements data requires the ability to store, retrieve, process, and report data. Specialized capabilities to do various types of statistical analyses are also required. These capabilities are provided by a Programming Project Measurement System. This system is composed of two subsystems, the question processing subsystem and the question analysis subsystem. The basic functions provided by the question processing subsystem are the maintenance of the data base (which contains the information submitted in response to the questionnaire), the retrieval and listing of data from the data base in various report formats, and the extraction

of data for transfer to the question analysis subsystem for statistical analysis. Figure 11 shows the overall flow of information in the programming project measurement system. The question analysis subsystem uses the Statistical Package for the Social Sciences (a product of SPSS, Inc.), which is an integrated system of computer programs for the analysis of data, and provides the user with a large set of procedures for data selection, transformation, and file manipulation, and offers a large number of commonly used statistical routines.

Figure 10 Relationship between computer cost and total effort



$C = 1.1\,E^{0.81}$
WHERE
C = COMPUTER COST IN THOUSANDS OF DOLLARS
E = TOTAL EFFORT IN MAN MONTHS

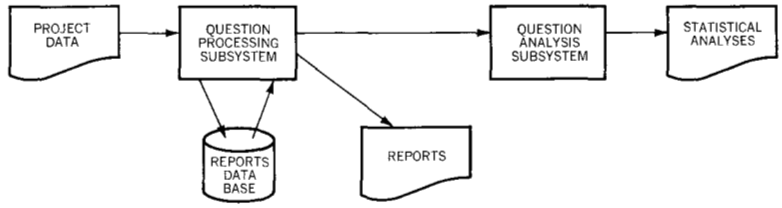
Statistical routines include descriptive statistics, frequency distributions, cross tabulations, correlation, partial correlation, multiple regression, and factor analysis. The package has its own internal data management facilities that can be used to modify analysis files of data and can be used in conjunction with any of the statistical procedures. These facilities enable the user to generate variable transformations, recode variables, sample, select or weight specified cases, and add to or alter the data or the analysis files.

Project data enter the Programming Project Measurement System by way of questionnaires that are answered by project personnel. At the inception of the measurement program discussed in this paper, one questionnaire was used for both development and service (maintenance) contracts. On service contracts, questionnaires were to be submitted quarterly. For development projects, four questionnaires were to be prepared by the project at major milestones during the life of the project. Identical questionnaires were to be submitted, but not every item required an answer at each submission. The four reporting milestones were the following:

- Start of work.
- Preliminary design review or equivalent.
- Top-down programming—completion of integration of one-half of the program units, or
  Bottom-up programming—completion of unit test of three quarters of the program units.
- Acceptance test completion.

Problems arose when project personnel tried to use the same questionnaire form for both development and service contracts; differences between those two types of activities made it difficult to fit all the necessary questions into one questionnaire format. A further problem was the reporting frequency of development contracts. The four milestones might often be six months to two years apart, and many changes could occur in project organization, in project specifications, and in the definitions of products to be delivered, so that it was difficult to correlate questionnaire responses from milestone to milestone.

Figure 11 Programming project measurement system



For these reasons changes were made in the questionnaires and the frequency of reporting. Separate questionnaires were created for development projects (Software Development Reports) and for service efforts (Software Service Reports). Development reports, which cover detailed qualitative items as well as quantitative data, are submitted at the start of work and again at acceptance test completion. Between these two submittals, a Monthly Software Development Report is submitted. This is a one-page summary of the status of a product, cost, and effort that is submitted each month. The Software Service Report is an overview of a product that is being serviced and is submitted at the start and end of service. The Quarterly Software Service Report is a summary of the product, cost, and effort status, plus a detailed reporting of errors and their impact. Reporting is done by programming projects that are developing or servicing products in the form of lines of code and that employ two or more programmers with an expenditure of twelve or more man-months of effort.

CITED REFERENCES

1. B. W. Boehm, "Software and its impact: a quantitative assessment," *Datamation* **19,** No. 5, 48–59 (May 1973).
2. F. T. Baker, "Chief programmer team management of production programming," *IBM Systems Journal* **11,** No. 1, 56–73 (1972).
3. *HIPO–A Design Aid and Documentation Technique*, Order No. GC20-1851, IBM Corporation, Data Processing Division, White Plains, New York 10504.
4. F. M. Luppino and R. L. Smith, *Programming Support Library Functional Requirements*, U.S. Air Force, Headquarters, Rome Air Development Center, Griffis Air Force Base, New York (July 1974). See also Rome Air Development Center, Structured Programming Series, Vol. V.
5. E. W. Dijkstra, "Notes on structured programming," pp. 1–82, O. J. Dahl, E. W. Dijkstra, and C. A. R. Hoare, *Structured Programming*, Academic Press, New York, New York (1972).
6. F. T. Baker, "System quality through structured programming," *AFIPS Conference Proceedings* **41,** Part I, 339–343 (1972).
7. M. E. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal* **15,** No. 3, 182–211 (1976).
8. J. D. Aron, "Information systems in perspective," *Computing Surveys* **1,** No. 4 (December 1969).

9. G. F. Weinwurm and H. J. Zagorski, *Research Into the Management of Computer Programming: A Transition Analysis of Cost Estimation Techniques*, SDC Report TM-2712, System Development Corporation, Santa Monica, California (1965).
10. E. A. Nelson, *Research into the Management of Computer Programming: Some Characteristics of Programming Cost Data from Government and Industry*, System Development Corporation, Santa Monica, California (November 1965).

GENERAL REFERENCES

1. *Improved Programming Technologies—An Overview*, IBM Systems Reference Library, Order No. GC20-1850, IBM Corporation, Data Processing Division, White Plains, New York 10604.

2. P. Van Leer, "Top-down development using a program design language," *IBM Systems Journal* **15,** No. 2, 155-170 (1976).

3. G. J. Myers, "Characteristics of composite design," *Datamation* **19,** No. 9, 100-102 (September 1973).

4. W. P. Stevens, G. J. Myers, and I. L. Constantine, "Structured design," *IBM Systems Journal* **13,** No. 2, 115-139 (1974).

5. E. W. Dijkstra, "Notes on Structured Programming," T. H. Report WSK-03, Second Edition, Technical University Eindhoven, The Netherlands (April 1970).

6. E. W. Dijkstra, "GOTO statement considered harmful," *Communications of the ACM* **11,** No. 3, 147-148 (March 1968).

7. H. D. Mills, *Mathematical Foundations for Structured Programming*, FSC 72-6012, IBM Corporation, Gaithersburg, Maryland 20760 (February 1972).

8. H. D. Mills, *Structured Programming*, FSC 70-1070, IBM Corporation, Gaithersburg, Maryland 20760 (October 1970).

9. J. G. Rogers, "Structured programming for virtual storage systems," *IBM Systems Journal* **14,** No. 4, 385-406 (1975).

10. G. J. Myers, *Software Reliability: Principles and Practices*, Wiley-Interscience, New York, to be published.

11. G. J. Myers, *Reliable Software Through Composite Design*, New York: Petrocelli/Charter (1975). Also see W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured Design," *IBM Systems Journal* **13,** No. 2, 115-139 (1974).

12. D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM* **15,** No. 12, 1053-1058 (1972).

13. N. Wirth, *Systematic Programming: An Introduction*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey (1973).

14. J. F. Stay, "HIPO and integrated program design," *IBM Systems Journal* 15, No. 2, 143-154 (1976).

15. M. F. Fagan, "Design and code inspections to reduce errors in program development," *IBM Systems Journal* **15,** No. 3, 182-211 (1976).

16. G. J. Myers, "Composite design facilities of six programming languages," *IBM Systems Journal* **15,** No. 3, 212-224 (1976).

Reprint Order No. G321-5045.