CHAPTER 5 - FOOTNOTES

(1) SCOTT R, F. and SIMMONS D. B.: 'Programmer Productivity and the Delphi Technique': Datamation May 1974: pp. 71-73.

(2) CHRYSLER, E: 'Some Basic Determinants of Computer Programming Productivity': Communications of ACM, Volume 21, Number 6, June 1978: p. 474.

(3) WALSTON, C. E. and FELIX C. P.: ' A Method of Programming Measurement and Estimation': IBM Systems Journal: Volume 16, Number 1, 1977: p. 64.

(4) JOHNSON, J. R.: 'A Working Measure of Productivity': Datamation: February 1977: p. 109.

(5) BOEHM, B. W.: 'Software and its Impact: A Quantitative Assessment: Datamation, May 1973: pp. 49-59.

(6) JEFFERY, D. R. and LAWRENCE, M, J; 'An Interorganizational Comparison of Programming Productivity': Information Systems Forum, University of New South Wales, Australia: March 1979.

(7) SACKMAN, H., ERIKSON, W. J., GRANT, E. E.: 'Exploratory Experimental Studies Comparing Online and Offline Programming Performance': Communications of ACM, Volume 11, Number 1: January 1968: pp. 3-11.

(8) Ibid., p. 6.

(9) COOKE, L. H. JR.: 'Programming Time versus Running Time': Datamation: December 1974: pp. 56-58.

(10) KELLY, J; 'Make Conflict Work for You': SBL BSP/06E, University of South Africa, 1975.

(11) FITZ-ENZ, J.: 'Who is the DP Professional?': Datamation, September 1978: pp. 124-128.

(12) LAWLER, E.E.: 'Motivation in Work Organizations': Brooks/Cole Publishing Co., Monterey, California:

(13) HERSEY, P. and BLANCHARD K. H; 'Management of Organizational Behaviour'; Prentice-Hall Inc., New Jersey, 1977. p. 25.

(14) WEINBERG, G. M.: 'The Psychology of Computer Programming': Van Nostrand Reinhold, New York, 1971. pp. 35 ff.

(15) GILB, T: Lecture Notes from the Course 'Advanced Programmer Training': In-House to DPD of the Standard Bank of South Africa, June 1975: p. 83.

CHAPTER 5 - FOOTNOTES (Cont'd).

(16) McCUE, C. M: 'IBM's Santa Teresa Laboratory - Architectural Design for Program Development': IBM Systems Journal Volume 17 - Number 1: 1978; p. 4.

(17) HERSEY, P. and BLANCHARD, K. H: 'Management of Organizational Behaviour'; Prentice-Hall Inc., New Jersey, 1977.

(18) ALLEN, L. A: 'Professional Manager's Guide' Louis A. Allen Associates, Inc: Chicago: 1975, p. 20.

(19) COUGER, J. D. and ZAWACKI, R. A: 'What Motivates DP Professionals?' Datamation, September 1978: pp. 116-123.

(20) BERNSTEIN, M. I: 'Software: Productivity or Quality?': Readers Forum: Datamation: April 1979: pp. 227-229.

(21) Ibid., p. 227.

(22) BROOKS, Jr. F. P: 'The Mythical Man-Month': Addison-Wesley: Reading, Massachusetts: 1975 pp. 4 ff.

CHAPTER 6 SOME ATTEMPTS AT MEASURING PROGRAMMER PRODUCTIVITY

Information on the methods which have been used to attempt to measure the productivity of programmers was gathered from three sources:-

- the answers to Questionnaire 1 (see Appendix 'A');
- surveying the literature;
- discussions with individuals from various South African companies.

In this Chapter no attempt is made to evaluate the identified methods of measuring programmer performance. (This is done in Chapter 7, where some problems associated with measuring programmer productivity are identified and in Chapter 8, where the suggested requirements for a method of measuring programmer productivity are identified).

6.1 An Analysis of the replies to the Questionnaire 1

The distribution of the Questionnaires will be discussed in Chapter 10 (see Section 10.2.1.1.). Replies to Questionnaire 1 were received from 42 companies. This represents 34,4% of the population to which the Questionnaires were distributed.

Table 5 below gives a summary of these responses.

Table 5 Summary of replies received to Questionnaire 1.

	YES	NO
Number of companies replied	42	80
Number who try to measure productivity	30	12

CHAPTER 6 SOME ATTEMPTS AT MEASURING PROGRAMMER PRODUCTIVITY

Information on the methods which have been used to attempt to measure the productivity of programmers was gathered from three sources:-

- the answers to Questionnaire 1 (see Appendix 'A');
- surveying the literature;
- discussions with individuals from various South African companies.

In this Chapter no attempt is made to evaluate the identified methods of measuring programmer performance. (This is done in Chapter 7, where some problems associated with measuring programmer productivity are identified and in Chapter 8, where the suggested requirements for a method of measuring programmer productivity are identified).

6.1 An Analysis of the replies to the Questionnaire 1

The distribution of the Questionnaires will be discussed in Chapter 10 (see Section 10.2.1.1.). Replies to Questionnaire 1 were received from 42 companies. This represents 34,4% of the population to which the Questionnaires were distributed.

Table 5 below gives a summary of these responses.

Table 5 Summary of replies received to Questionnaire 1.

	YES	NO
Number of companies replied	42	80
Number who try to measure productivity	30	12

Table 5 (Cont'd).

	YES
Method of Measuring Productivity:	
Lines of Code (source) only	1
Meeting estimates only	22
Lines of Code (source) and meeting estimates	2
Lines of Code (source) and Lines of Code (executable)	1
Lines of Code (source) and meeting estimates and other	2
Lines of Code (executable) and otner	1
Meeting estimates and other	1
Why do not measure productivity:	
Replied to question	10
Never thought about it	1
Tried, does not work	1
Cannot choose method	4
Too few programmers	3
Impossible and too few programmers	1
Try to measure productivity, but ...	
Unsure of what it achieves	1
Impossible and cannot choose method	1

The following observations are made:-

i) A relatively high percentage of companies which responded to the Questionnaire try to measure the productivity of their programmers (71,4%).

ii) Of those Companies which do try to measure productivity, a surprisingly high percentage (73,3%) measure programmers' performance in terms of their meeting estimates. This is a source of concern, because it was established later in the Questionnaire that 74,2% (23 out of 31 replies) of the sample population determine estimates on past experience ONLY. (see further discussion on this point in Section 9.2).

iii) The use of lines of code, either on its own, or together with some other factor, was the method used by only 23,3% of those who responded to the Questionnaire. Considering the prominent place this method of measuring programmer productivity enjoys in the literature, this was an unexpected result, from even such a small sample population.

iv) Of those who were not trying to measure programmer performance, some seemed sceptical about the whole concept. Comments like:-

'Impossible ...'
'..... it does not work'
'Unsure of what it achieves'

were not expected, but seem to reflect a mood of disenchantment with the problem. One mood which was expected, but which was not apparent in these replies was a concern that little progress had been made by the industry in providing a solution to the problem.

Because of the limited size of the sample population, no generalization for the industry as a whole can be made from these replies.

6.2 Reviewing the Literature

To help classify the diverse points of view found in the literature on measuring programming performance, the model in Figure 2 was used. Each category identified in the model will be illustrated with details of approaches suggested in the literature.

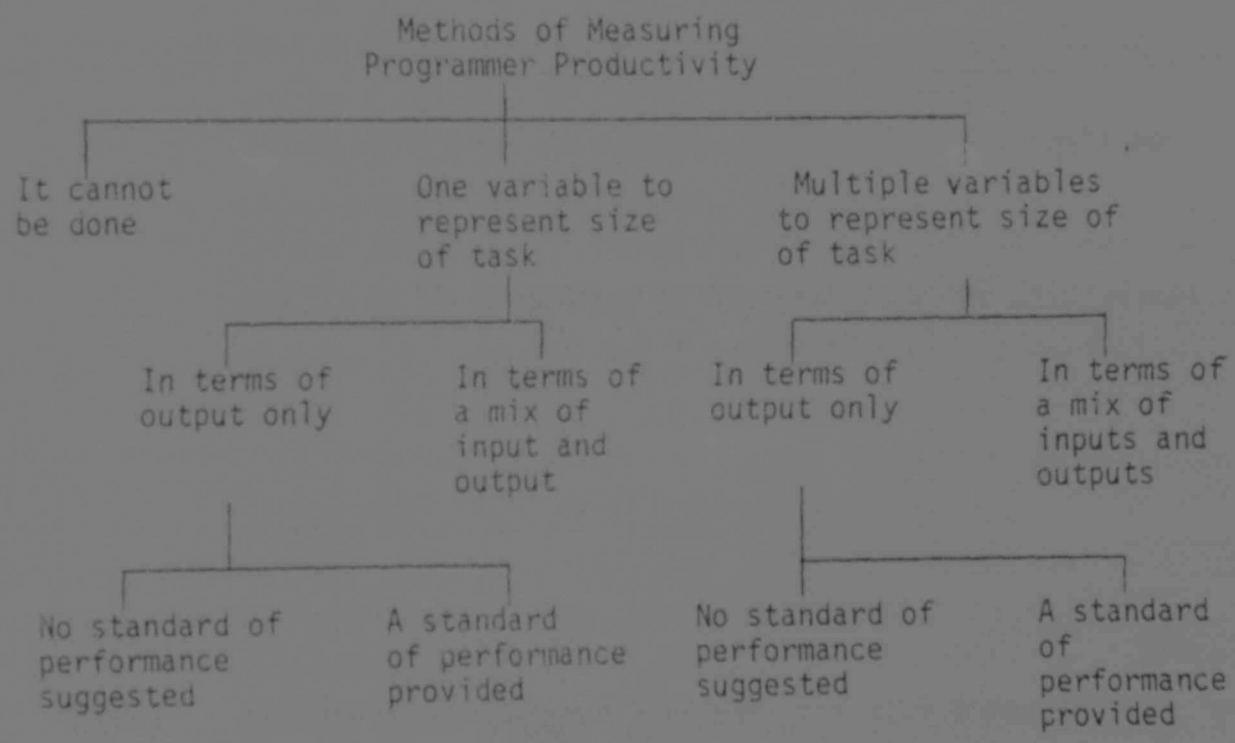

Figure 2 The model used to classify methods of measuring programmer productivity

6.2.1 Cannot Be Done

There is a group of people who feel that it is just not possible to measure programmer performance. Various reasons are given, but primarily the view is held that programming is too like an art (or a craft) for any measure of programmer productivity to be meaningful.

Bernstein appears to be one of the stronger supporters of this view. He writes:-

> 'We have no adequate way of quantifying the state of our software development effort .... some measurements are being made, but they provide information that is "meager and unsatisfactory".' (1)

Bernstein claims that we are forced to the conclusion that software development is a craft we cannot control and as a result, can barely manage. In his view programming will not reach that point where it will be totally controllable and predictable. He feels it is unlike the established engineering disciplines. He concludes programming will not fall into the disciplined engineering category until some time in the 'indefinite future' (2).

6.2.2 Measurement in Terms of only one Variable

Although programming entails performing a range of unrelated activities (e.g.algorithm creation, mastering job control language and debugging) the majority of suggested methods of measuring programmer productivity advocate that this measurement be done in terms of one variable. Examples are:-

- the number of source statements per unit of time,
- the number of executable statements per unit of cost,
- the number of modules per unit of time.

Bernstein appears to be one of the stronger supporters of this view. He writes:-

> 'We have no adequate way of quantifying the state of our software development effort .... some measurements are being made, but they provide information that is "meager and unsatisfactory".' (1)

Bernstein claims that we are forced to the conclusion that software development is a craft we cannot control and as a result, can barely manage. In his view programming will not reach that point where it will be totally controllable and predictable. He feels it is unlike the established engineering disciplines. He concludes programming will not fall into the disciplined engineering category until some time in the 'indefinite future' (2).

6.2.2 Measurement in Terms of only one Variable

Although programming entails performing a range of unrelated activities (e.g.algorithm creation, mastering job control language and debugging) the majority of suggested methods of measuring programmer productivity advocate that this measurement be done in terms of one variable. Examples are:-

- the number of source statements per unit of time,
- the number of executable statements per unit of cost,
- the number of modules per unit of time.

However, within the group of researchers which suggest that measurement be in terms of one programming activity, there are clearly two schools of thought:-

i) Those who measure performance purely in terms of the size of the task to be done (that is, those who measure OUTPUT only - see definition in Chapter 3);

ii) Those who also take cognisance of the circumstances in which the task was performed (that is, those who attempt to measure both INPUT and OUTPUT - see definitions in Chapter 3).

6.2.2.1 Measurement in terms of one Variable which attempts to reflect the size of the task

There is a number of suggested methods of measuring programmer performance in terms of output factors only. These methods can be sub-divided into those which just claim that programmer productivity can be measured, and those that also provide some standard of performance.

6.2.2.1.1 Measurement in terms of one Variable which reflects the size of the task, but does not provide a standard of performance

Although a number of methods of measuring programmer productivity identified in the literature fall into this category, there is no consensus among researchers concerning what should be measured to reflect the size of the task.

i) A Whole Program

Two authors whose work falls into this category compared performance in terms of the length of time specified programs took to develop. While this may have the semblance of providing a standard of performance, in neither case was an indication given of the size of the programs.

Cooke (3) compared the length of time and the cost of developing and successfully completing a single execution of a program to calculated interest rate for the United States Federal Reserve Bank. Although this experiment using professional programmers provided other insights, (see Section 5.1), it gives no indication of how programmer productivity can be measured.

Sackman, Erikson and Grant (4) also measured the length of time programmers took to complete specified programs. In their experiment they compared the time taken to write (and debug) programs in either batch or on-line mode. Again this does not give an indication of how the performance of programmers who did not write those programs can be measured.

ii) Lines of Code

Those who suggest that programmer performance should be measured in terms of the number of lines of code produced do not agree on whether the lines of code should be measured in terms of cost or time, nor do they agree on whether the lines of code should be source, object or executed code.

Comper (5) suggests measuring lines of executable code produced per man-day. Although he does not provide a standard of performance, he does suggest historical trends should be noted.

Dijkstra (6) suggests that the number of lines of code written is not as significant as the number of lines of code which the machine actually uses when the program is executing. He does not give an indication of whether this should be measured in terms of time or cost.

Jones (7) has written a number of articles on the subject of programmer productivity and program quality. He recommends that programmer performance be measured in terms of cost per 1000 lines of source code. This concept is interesting because it is an effort to get away from Bernstein's fear (8) that programmers should not be measured as if they turned out lines like bottle-tops off a production line.

Reynolds (9) is not completely happy with the use of lines of code as a measure of programmer performance. He feels, however, that the number of lines of code produced and tested is the generally used unit of measure. Unfortunately he does not suggest a measure which he finds more acceptable.

Scott and Simmons (10) try to determine what project variables have the greatest impact on programmer productivity. People participating in their research were asked to correlate each of a list of variables with programmer performance expressed in terms of object instructions generated per unit of time.

None of these approaches is sufficiently developed to be of value to managers or programmers who want to measure levels of programmer productivity.

6.2.2.1.2 Measurement in terms of one Variable which reflects the size of the task, and which provides some standard of performance

The majority of suggested methods of measuring programmer productivity fall into this category. Although many suggest the same elements be used to measure programmer performance levels, all have been included here because of the lack of consensus of what constitutes standard performance. (see Section 6.2.2.1.3) In the light of assertions I make later (see Section 9.3) perhaps the most significant results are those which exclude the use of lines of code as an element of measurement.

Baker (11)

In an article which describes the functioning of the so-called Chief Programmer Team, Baker gives details of a system developed for the New York Times. The project was significant for at least the following reasons:-

- This is one of the first documented accounts of the use of concepts such as structured programming, librarians, top-down system development etc;

- The speed at which the work was done, and the quality of the programs, were considered high;

- The increased manageability of the project by using the Chief Programming Team structure was significant.

The published productivity figures are:-

Stages	Source Lines Per Man-Day
Unit design, programming, debugging and testing	65
All professional staff	47
Librarian support included	43
The entire team	35

At the time, the approach taken was regarded as unique, however as more installations use these methodologies tnese figures could have industry-wide relevance.

Boeing Computer Services

In one of the answers to Questionnaire 1 sent from Boeing Computer Services in Seattle, United States of America, it was indicated that the measure of programmer performance in that environment was expressed in terms of producing one module per hour.

Unfortunately I have not been able to establish what their definitions are of either 'module' or 'hour'. Whatever their definitions, this method of measuring programmer productivity approaches the method I test in this research (see Section 10.1).

Brooks (12)

From his experience on the project to develop the IBM OS/360, Brooks found that reasonable levels of productivity from programmers will depend on the type of application being developed.

As a rule of thumb, he suggests that the writing of translators takes about three times as long as writing batch application programs and developing operating systems takes about three times as long as developir translators. (He includes in his development time, time spent planning, coding and component testing, systems testing and some support activities).
He concludes:-

Product	Debugged Lines of Code/ Man-Year
Operating Systems	600 - 800
Translators	2000 - 3000
Batch Programs	6000 - 9000

If it is assumed that there are 2000 man-hours in a man-year, the expected performance figures for programmers developing batch programs can be expressed as:-

Low	22,5 Lines of Code/man-day
High	33,75 Lines of Code/man-day

Brooks also quotes data published in a survey prepared by Morin (1974) as part of a Masters dissertation at the University of North Carolina (USA).

Aron's Data (13)

This data was gathered from IBM programmers working on large systems. Aron classifies systems in terms of the number of interactions necessary between programmers (and parts of the system) and he suggests the following:-

Very few interactions	10 000 instructions man-year
Some interactions	5 000 instructions man-year
Many interactions	1 500 instructions man-year

If these figures were converted to expected performance per man-day, they would read:-

Low	5,6 Lines of Code/man-day
Average	18,75 Lines of Code/man-day
High	37,5 Lines of Code/man-day

(It is noted that Aron's development time only included design and programming).

Camp and Jensen (14)

In a study which attempts to justify the use of structured programming rather than modular programming, the authors compare the development efforts of five similar projects. Unfortunately, what seems like significant experiment does not provide results relevant to this study. All the figures quoted in the article have been normalized, 'based on the operational (on-line) program only'. Because the exact meaning of the statement is not clear and because the results reveal productivity levels ranging from 0,75 to 3,1 'normalized' instructions per man-month, it is difficult to relate these figures to any others found in the literature.

Frank (15)

Frank quotes figures from the IBM technical report TR 02.764 of January 1977 of expected development times for programs of varying sizes.

Program Size In Lines of Code	Manpower Time In Months
Less than 2 000	0,5 to 3
2 000 to 10 000	0,8 to 5
10 000 to 64 000	1 to 8
64 000 to 512 000	2 to 16
Over 512 000	4 to 28

These figures were not presented as if they were a panacea to the programming industry, but rather to guide people in determining the possible development costs (of assembler programming) in their particular environment. Again it is not possible to use these figures in attempting to establish an industry-wide standard of performance for application programmers.

Jeffery and Lawrence (16)

In an interesting study of three organizations in Australia, all of which used COBOL as a programming language, conducted by the Department of Information Systems, University of New South Wales, an effort was made to identify the best determinant of program development time. Programming time was defined as the total time expended by the programmer from the time he receives the programming specification until he delivers the working tested program. (There was an inconsistency because not all the organizations included test data development in this calculation).

The results of their research revealed that development time (T) could be expressed in terms of PROCEDURE DIVISION lines of code (PL) in the formats below:-

Organization 1 T = 0,09 PL + 4
Organization 2 T = 0,14 PL + 9,5
Organization 3 T = 0,08 PL + 43

The following steps were taken to try to convert these formulae into the expected number of lines of code per man-day. It was assumed that T = 2 000 man-hours in each case. The number of Procedure Division Lines of Code which could be coded in a man-year was calculated. This figure was converted to an approximate performance figure per man-day (assuming a man-day = 7,5 man-hours).

The results can be expressed as:

Organization 1	83 Lines of Code/man-day
Organization 2	53 Lines of Code/man-day
Organization 3	92 Lines of Code/man-day

It is noted that in each case the Lines of Code figure represents only Procedure Division Lines of Code, and consequently the conversion to Lines of Code/man-day may not be meaningful.

Johnson (17)

Based on data collected from 16 projects - the smallest of which took 40 man-days, and the largest 79 200 man-days to develop - Johnson found significant differences in the level of programmer performance on the different sized projects. He tabulates his findings as follows:-

	Projects	
	Small	Large
Average Lines of Code/hour	8,7	2,9
Range	6,7 to 12,5	1,2 to 5,1

Johnson goes to some length to define his unit of time. He includes productive and non-productive time in this measure (that is, the number of people assigned to the project multiplied by the duration of the assignment). The duration of the assignment covers all phases of the project. He warns that the man-day totals could vary by as much as ± 15% due to historical inaccuracies.

These results could be expressed as:-

Average (large Projects)	21,75 Lines of Code/ Day
Average (Small Projects)	62,25 Lines of Code/ Day

Were these figures to exclude the non-productive time (that is, be expressed in terms of man-days) they would be significantly higher.

Lehman (18)

Lehman conducted a survey to attempt to discover how projects are managed in the United States of America Aerospace industry. Although details were gathered from 57 projects, Lehman concedes that these are not representative of the data processing community at large. He published two figures as a result of the survey, neither of which is particularly nelpful in the context of this research. He discovered from answers to his questionnaire that the average cost of producing a single line of code in the companies surveyed ranged from a low of $5 per line to a high of $330 with an average of $49,11 per line of code. He was also able to report that the average production rate of programmers in the companies questioned was 17 lines of code/day (unfortunately there is no clear definition of what is meant by either line of code or day).

Menard (19)

In a paper presented at the Application Development Seminar in Monterey, California, Menard described the steps taken by the Exxon Corporation to reduce the cost of developing computer systems. By combining Jackson's program design techniques, structured walk-thrus and top-down development into a standard methodology, significant programmer performance improvements were observed. The productivity levels experienced on five projects were above 7000 lines per work-year, with programmers on one project producing 8 500 lines per work-year. When program analysis time was deducted from the calculations, the productivity figures increases to between 10 000 and 12 000 lines of code per work-year. An interesting, unsubstantiated comment in Menard's paper is that these figures compare favourably with an Industry Average of 2 000 to 4 000 lines of code per work-year.

Normalized, these figures can be expressed as:-

Average	26,2	Lines of Code/man-day
High	31,8	Lines of Code/man-day
'Industry' Low	7,5	Lines of Code/man-day
'Industry' High	15,0	Lines of Code/man-day

Walston and Felix (20)

The authors gathered data from 60 projects completed in the IBM Federal Systems Division. These projects varied considerably in size and complexity (process control, message switching, report generators, data-base control etc.). The objective of their research was not primarily to provide a measure of individual programmer performance, but rather to:-

'- provide data for evaluating 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.'

(21)

To meet these objectives, however, it was necessary for Walston and Felix to calculate a standard of programming performance. This they did in terms of lines of source code per total effort involved in developing the code. (This 'total effort' is defined as a measure of time, expressed in man-months, used in managing, administering, analysing, designing, coding, testing and documenting the system). This relationship between delivered lines of source code, and total effort is expressed by the formula:-

$$E = 5{,}2L^{0{,}91}$$

where E = total effort
and L = tnousands of lines of delivered source code (22)

By calculating a 'productivity index' from 29 variables (23) the suggested standard of performance was adapted to assist in estimating:-

- project duration
- size of documentation
- staffing requirements
- computer costs.

South African Post Office - Data Processing Department.

In private correspondence from tne Data Processing Department of the South African Post Office dated 18 April 1979, I was told that their development department has a standard which requires a programmer to write 35 procedural division instructions per 8 hour day. Development time is calculated from the time the program specification is handed to the programmer until the completion of program tests. Because of other responsibilities which programmers are given (e.g. working on more than one program during any one day, and maintenance responsibilities) tne standard is seldom enforced. It is clear that time is measured as 'Clock-on-the-Wall' time and not in Man-Hours.

Standard Bank Data Processing Quarterly Statistics.

Programmer performance at the Standard Bank was (during the period 1976 - 1979) measured in terms of the number of lines of code produced. Two figures were calculated - one was the number of lines of code produced per Programmer Man-Day (tnat is, only programming time was measured) and the other was the number of lines of code produced per Project Man-Day (which took into account the time taken for analysis and design, as well as programming).

The standard set the programmers was 50 Lines of Code/Project Man-Day. This figure was consistently beaten - with performance usually reaching 60 to 65 Lines of Code/Project Man-Day.

During 1978/1979 an interesting result was achieved by the team working on a Personnel System. They misunderstood the objectives given to them by management to achieve 50 Lines of Code/Project Man-Day. At the end of their project they were concerned about what they regarded as their productivity figure. They apologized for achieving 97 Lines of Code/ Project Man-Day, because they thought they were supposed to achieve 50 Lines of Code/ Project Man-Hour!

In an effort to guide the programmers on one project, tne Project Designer published approximate times to be spent on each phase of program development. These times, expressed in hours, were regarded as representative for a program of approximately 500 lines of non-comment source code.

It was estimated that such a program should be developed to a stage where systems testing could commence in 28 to 30 man-hours. The designer, therefore, expected a productivity rate during those phases of development of approximately 125 Lines of Code/Man-Day.

6.2.2.1.3 Summary

While some of these approaches appear to be practical, their contribution to the industry is limited by the lack of agreement concerning an acceptable standard of performance and the fact that they have based their measurements on the number of lines of code produced. The use of lines of code as a productivity measure is regarded as suspect (see Section 9.3).

6.2.2.2 Measurement in terms of one variable which reflects the size of the task, but which requires modifications to accommodate particular circumstances.

These methods of measuring productivity, which mix Input and Output (see definitions in Chapter 3), are commonly found as various estimating techniques. In any estimating it is necessary to calculate expected levels of performance in particular circumstances. Each method of measuring productivity which attempts to assess the impact of project variables on programmer performance automatically falls into this category. There can be no standard of performance possible within this category because the input factors introduce levels of subjectivity, and by definition a standard cannot be based on a subjective assessment.

Albrecht (24)

In his work at IBM's Data Processing Service Organization, Albrecht found tnat a weighted count of the number of inputs, outputs and master files, and the number of enquiries processed by the program gave a useful measure of the size of the program. These weightings are:-

- number of inputs x 4
- number of outputs x 5
- number of enquiries x 4
- number of master-files x 10

This result should then be adjusted to reflect program complexity to produce what Albrecht calls the number of 'function points'. The number of these 'function points' represents the size of the job.

He suggests that the cost of developing the system should be expressed in work-hours (man-hours) and should include the whole application development cycle from systems design to systems test.

This approach is categorized here because the adjustment to the weighted total is based on a subjective assessment of program complexity (which is similar to estimating) and constitutes a mix of input and output factors.

Albrecht (24)

In his work at IBM' Data Processing Service Organization, Albrecht found that a weighted count of the number of inputs, outputs and master files, and the number of enquiries processed by the program gave a useful measure of the size of the program. These weightings are:-

-	number of inputs	x	4
-	number of outputs	x	5
-	number of enquiries	x	4
-	number of master-files	x	10

This result should then be adjusted to reflect program complexity to produce what Albrecht calls the number of 'function points'. The number of these 'function points' represents the size of the job.

He suggests that the cost of developing the system should be expressed in work-hours (man-hours) and should include the whole application development cycle from systems design to systems test.

This approach is categorized here because the adjustment to the weighted total is based on a subjective assessment of program complexity (which is similar to estimating) and constitutes a mix of input and output factors.

Chrysler (25).

The title of this paper is unfortunately misleading. Instead of a dissertation on programmer productivity measurement, the objective of Chrysler's research is stated as:-

> '....to determine the impact of program and programmer characteristics on program development time' (26)
>
> '....to develop a viable technique for predicting the amount of time necessary to create a computer program.' (27)

Chrysler found that the smallest standard error of estimate in the multiple linear regression was achieved when a combination of program and programmer characteristics were correlated with time. (Unfortunately it is not clear if this time is expressed in terms of man-hours or elapsed time.) These variables are:-

- number of months of experience the programmer has had at a particular installation;
- number of input files in his programs;
- number of control breaks and totals in the program;
- number of input edits required;
- number of input fields.

A predictive (or evaluation) formula which resulted from the multiple regression (28) enables the total programming effort expressed in hours, to be calculated.

Because the primary interest is to predict program development time, Chrysler must introduce input-factors into his productivity index. This means his work cannot fall into the category of providing a standard of performance for programmers, or an objective measure of productivity.

Hall (29)
Although he believes that the 'standard' unit of work measurement in the data processing industry is the number of lines of executable code over a given time frame (30), Hall criticizes this view and suggests that productivity of programmers should be assessed in terms of:-

- the project being developed within budget and schedule;

- programs operating efficiently without failures;

- programs meeting the user and design specifications exactly;

- programs being modifiable without rewrite;

- programs being adequately and effectively documented.

(31)

In Hall's view, the multi-dimensional result of a high level of programmer productivity suggests that it is more accurate to refer to 'project' productivity than 'programmer' productivity. He feels his case is strengthened because operating and maintenance costs (approximately 70% of total costs) are more significant than development costs. (32)

In the final analysis, then, Hall believes that improved profits within the organization using the data processing facility constitutes effective productivity. It is this point which creates the major problem with Hall's approach. It is not meaningful to assess the programmer's performance in developing an application only when the enterprise using the application can determine its increase in profits. This is an assessment of the work done by the systems analyst.

Halstead (33)

The method used to analyze the performance of programmers postulated by Halstead is based on the ability to count, for any program, the number of unique operators (n1) (e.g. Read, Write, If, Greater Than), the number of unique operands (n2) (e.g. variables and constants), the total usage of operators (N1) and the total usage of operands (N2).

Halstead defines the vocabulary as:-

$$n = n1 + n2 \qquad (34)$$

and the implementation length as:-

$$N = N1 + N2 \qquad (35)$$

Program size is measured by the metric V (volume) which is independent of the language used to represent the algorithm. The formula for volume is:-

$$V = N (\log 2n) \qquad (36)$$

The potential volume (V^*) of a program does not change if it were translated from one language to another. The relation between the actual and the potential volume is expressed by the formula:-

$$V^* = \frac{L}{V} \qquad (37)$$

where L is the level of the program abstraction. This level of abstraction is close to 1 for high-level compact programs.

The removal of 'impurities' from a program (see Section 7.4.2.2 for Halstead's definition of a quality program) improves the agreement between V^* and VL.

The total programming effort is defined by Halstead as:-

$$E = \frac{V}{L} \qquad (38)$$

This number represents the number of mental discriminations (decisions) that a programmer would make when implementing the algorithm.

Halstead suggests that the time a program will take to develop is directly proportional to the effort E.

This time can be calculated by the formula:-

$$T = \frac{E}{S} \qquad (39)$$

where S is the number of mental discriminations per second of which the particular programmer is capable. It is this introduction of the characteristics of a particular programmer's skill into the calculation which forces Halstead's work to be categorized as a mixture of input and output factors.

Peeples (40)

In a Datamation article Peeples describes how a company-wide programme to measure productivity was introduced into the 14 data centres of the enterprise for which he works. One of the specific objectives set for this programme was 'to measure and compare performance'. Among the measures of performance in Project Development was to implement 95% to 100% of all projects within time budget.

This idea constitutes an interesting departure from the common approach of measuring productivity in terms of the amount of work done. Peeples defines 'Within Time' as a window of + 5% and -20% of approved estimate. The concept of a window was introduced to try to avoid the problem of misjudging actual performance because of inaccurate estimating (see Section 9.2).

6.2.3 Measurement in Terms of Multiple Variables

One group of methods of measuring programmer productivity takes into account the diverse activities which a programmer must perform. A measurement is recommended for each activity, e.g. orientation, documenting, coding, reviewing, testing etc.

Again there are categories within this group. These correspond exactly to those described in Section 6.2.2 above.

6.2.3.1 Measurement in terms of Multiple Variables whose summation reflect the size of the task

This category also sub-divides into methods which provide a standard of performance for each category, and methods which purely indicate the elements which can be used to measure programmer performance.

6.2.3.1.1 Measurement in Terms of Multiple Variables whose summation reflect the size of the task, but which provide no standard of performance for any of the activities

Belford and Broglio (41)

One of the assertions which these researchers attempted to prove was that the number of 'decision-to-decision' paths through a system is a better method of calculating development costs than basing these costs on the estimated number of lines of code in the system.

Data was collected from 3 sub-systems, comprising a total of more than 13000 lines of code. The number of man-hours expended during detailed design of the system, coding of the programs and testing (although it is not clear what phases of testing were included in this section) for each sub-system was also measured.

Their conclusion was that a different measure for each of three phases of development would give the best predictions:-

i) Detailed design - the number of man-hours per decision-to-decision path.

ii) Coding - the number of man-hours per executable source statement.

iii) Testing - the number of man-hours per test case.

(42)

I was not able to find any other reference to the concept of measuring program development performance in terms of the decision-to-decision paths through the algorithm. The idea has appeal. It is surprising that the statistical correlation was not sufficiently diverse to support the hypothesis that this is a better measure for the prediction of development costs than lines of code.