

https://dspace.org/

On the possibility of measuring programmer productivity

Crossman, Trevor Douglas

2015-02-17
http://hdl.handle.net/10539/17007

Downloaded from a DSpace repository. https://dspace.org/

[redacted]

A DISSERTATION PRESENTED TO

THE UNIVERSITY OF THE WITWATERSRAND,

JOHANNESBURG

MASTER OF COMMERCE

by

T D CROSSMAN

OCTOBER 1981

ON THE POSSIBILITY OF MEASURING PROGRAMMER PRODUCTIVITY

A DISSERTATION PRESENTED TO

THE UNIVERSITY OF THE WITWATERSRAND,

JOHANNESBURG

In fulfilment of the
Requirements for the
degree of

MASTER OF COMMERCE

by

T D CROSSMAN

OCTOBER 1981

ACKNOWLEDGEMENTS

I wish to thank the following people for the assistance and advice given towards the completion of this research:

PROF. J. T. STEELE who acted as supervisor.

MR P. FRIJOHN for assistance with the chapters on statistical processing.

MR K. GILL for assistance with the chapters on cost and management accounting.

MR G. RUSSELL of The South African National Productivity Institute.

MRS C. MEYRICK for typing the script.

I certify that, except as noted above, this dissertation is my own work and all references are accurately reported.

T. D. CROSSMAN

CONTENTS

		PAGE
CHAPTER 1:	THE SOFTWARE CRISIS.	1
CHAPTER 2:	BOUNDARIES OF THE RESEARCH.	12
CHAPTER 3:	DEFINITIONS.	14
CHAPTER 4:	WHY PROGRAMMER PRODUCTIVITY SHOULD BE MEASURED.	24
CHAPTER 5:	FACTORS WHICH INFLUENCE PROGRAMMER PRODUCTIVITY.	33
CHAPTER 6:	SOME ATTEMPTS AT MEASURING PROGRAMMER PRODUCTIVITY.	63
CHAPTER 7:	SOME SIGNIFICANT PROBLEM AREAS.	103
CHAPTER 8:	REQUIREMENTS FOR A METHOD OF MEASURING PROGRAMMER PRODUCTIVITY.	198
CHAPTER 9:	THE LIMITATIONS OF THREE COMMON APPROACHES TO MEASURING PROGRAMMER PRODUCTIVITY.	210
CHAPTER 10:	A POSSIBLE ALTERNATIVE APPROACH.	234

CONTENTS

			PAGE
CHAPTER 11:	FURTHER RESEARCH.		335
CHAPTER 12:	CONCLUSION.		337
APPENDICES:			
	'A'	QUESTIONNAIRE 1	346
	'B'	QUESTIONNAIRE 2	349
	'C'	PROGRAM HISTORY SHEET	350
	'D'	INSPECTION CHECK LIST	341
BIBLIOGRAPHY			355

SYNOPSIS

This research concentrates on some of the symptoms of the 'software crisis' - the difficulties of planning, controlling and evaluating the application development processes. It is an empirical study of some of the known methods of measuring programmer performance. Identified approaches are analysed critically and categorized. A hypothesis that programmer productivity can be measured in terms of the number of program functions implemented is tested and evaluated. Specific areas for further research are identified.

CHAPTER 1 THE SOFTWARE CRISIS

In his largely autobiographical Turing Award Lecture at the annual conference of the Association of Computing Machinery in August 1972, E. W. Dijkstra draws some pictures to illustrate his understanding of the evolution of computer programming. (1)

He explains that the whole approach to programming in the early 1950s was based on concepts which thirty years of technological developments have proved fallacious. Dijkstra claims that, at the time, programmers believed very little of their work would have lasting value. The whole idea of computer systems being executed regularly for a period of years, was rare. More important was the phenomenon that, because the machines tended to be unreliable and their memory small, clever programmers derived immense intellectual satisfaction from using cunning tricks to squeeze the impossible into the constraints of the equipment.

> One often encountered the naive expectation that once more powerful machines were available, programming would no longer be a problem ..... But in the next decade something completely different happened: more powerful machines became available ..... but instead of finding ourselves in a state of eternal bliss ..... we found ourselves ..... in the software crisis.' (2)

1.1. What is the Software Crisis?

Because of the difficulties which face application system developers, it is possible to examine the problem of the 'software crisis' from three points of view:-

- as it affects the end User (see Chapter 3 for definition);

- as seen by General Management;

- from within the data processing industry.

1.1.1 The End User

If the end User's requirements and expectations are not kept central in all application systems which are developed, data processing soon becomes little more than an expensive game. In a paper presented at the International Federation for Information Processing (IFIP) in Toronto, Canada in 1977, F. Brooks (3) compared application systems developers to toolmakers, whose work enables the Users to perform their work more effectively:-

> 'A toolmaker succeeds as, and only as the users of his tool succeeds'
>
> (4)

1.1.1.1 The Application Back-Log

Unfortunately, however, two factors have contributed to the User's request for data processing services not being met timeously and consequently have impeded the effectiveness of their work:-

i) The dramatic decrease in cost/performance of general purpose computers has broadened the base of applications which are viable to computerize. This had led to an increase in the demand for computer applications.

ii) An increase in User expectations, and an appreciation of the possibilities of data processing, has resulted in a greater demand for more sophisticated application systems.

These increased demands for programming resources have exceeded the programmer's ability to implement programs. A back-log of User requests for computerization continues to grow industry-wide. At the IBM Executive Seminar in London in November 1979, Bryan Ward, IBM Business Management Consultant, estimated that this back-log is between three and four years work. The User therefore could wait for up to five years to have a computerized system installed.

The same increasing discrepancy between supply and demand was forecast by the SILT report of SHARE, August 1974 (5). The graph presented at that conference is reproduced in Figure 1.

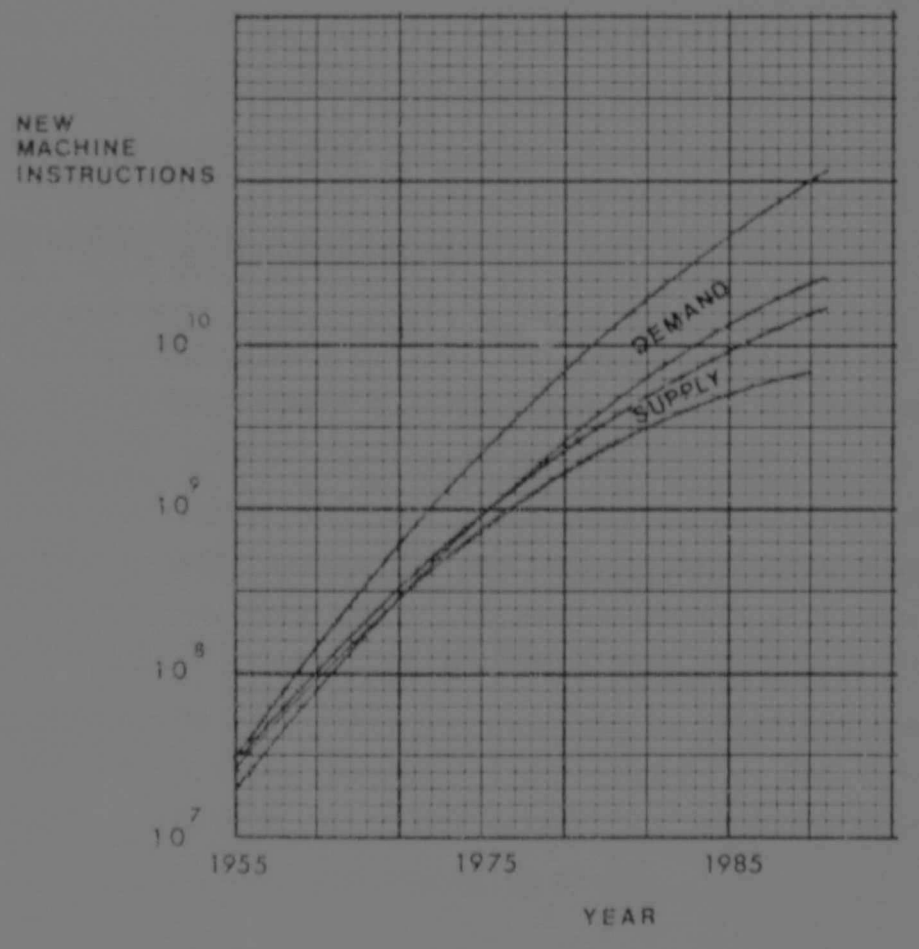


Figure 1 The predicted increase in demand for data processing services.

1.1.1.2 Further Contribution to the Credibility Gap

The Users whose requests for data processing services are being met, face a different set of frustrations. In this area problems are caused by:-

- long development cycles;
- inaccurate estimating;
- missed dead-lines;
- revisions or enhancements required immediately after implementation;
- unreliable systems demanding high incidence of maintenance.

So, looking at the 'software crisis' from the User's point of view, the symptoms are an increasing inability of data processing personnel to produce what is required, when it is required, within the expected cost parameters.

1.1.2 Senior Management

The May 1975 issue of the EDP Analyzer (6) suggests that there is also a growing credibility gap between general management and the data processing activity. The article suggests that the main cause of this attitude is the continually escalating budgets of the data processing departments - with annual increases of between 15% and 20%. This phenomenon is likely to be so for all data processing departments which are passing through Nolan's Stages Two and Three (7), both of which are categorized by rapid increases in capital and operating costs.

The SILT report estimates that by 1985 some 13% of the United States of America's Gross National Product will be used in data processing costs (8), with budgets forecast to double every 5 years to 1990.

Two other factors contribute to general management's attitude to the data processing activity:-

i) As predicted by Dr B. W. Bohm of TRW (9) there has been a move of cost emphasis away from hardware costs to the cost of developing and maintaining software.

ii) The fantastic cost-performance improvement in hardware has not been complemented by a corresponding increase in the cost-performance of the software developers.

It is not always clear to general management whether an equitable return on investment in data processing is being achieved. This concern is aggravated by a lack of effective direction and control of the data processing facility by general management (6).

1.1.3 The Data Processing Industry

Within the data processing industry the symptoms of the software crisis are apparent in at least three areas:-

- the lack of adequate people resources;
- the need for on-going training and education;
- the difficulty of managing development projects.

The Lack of Adequate People Resources

The increased demand for data processing personnel has been dramatic, and indications are that it will continue to be so.

Commenting on the results of the annual survey done by Computer Personnel Limited, Johannesburg, of some 8000 data processing people working at over 300 South African organisations, the researchers wrote:-

> 'Our annual survey ..... confirmed yet again that the shortage of computer people had worsened over the past year'
>
> 'We believe there are vacancies for some 2500 people in the systems development field, representing some 30% of the available resources.' (10)

The increase in programmer numbers in the United States was from approximately 30 000 in 1960 to some 175 000 in 1970, according to SHARE's SILT report, with predictions indicating that numbers could reach about 480 000 in 1980 (11).

This prediction seems to be supported by an article published by the American Federation for Information Processing (quoted by (12)). The authors of the article, entitled 'Information Processing, in the United States: a Quantitative Summary', predict that between 1979 and 1989 the expected percentage increase in the number of systems analysts in the United States is about 65%, while the expected increase in the number of programmers over the same period will be about 49%. Such high increases are likely to lead to further staff shortages.

There are at least two negative results from the explosion in the need for application development staff:-

i) High staff turnover becomes common, and salaries are easily inflated - a problem which further concerns general management.

ii) Often people are given responsibilities - either technical or managerial - which are beyond their capabilities. Consequently people who do not have the necessary skills are expected to perform the complex tasks of systems design and programming, or they are expected to manage people who are performing these tasks. If either the management or the execution of those tasks is inadequate, more problems are created (see Section 5.3) and the software crisis is worsened.

1.1.3.2 The Need for Further Training and Education

Because of the advances being made in technological fields, new operating systems, programming languages and design methodologies are developed and marketed by machine companies and software houses. The capabilities, and in some cases the complexity, of these new products seem to be matched only by the rapidity with which new releases are made available.

Training and education become part of the software crisis because technical staff need to keep abreast of technological advances to avoid becoming redundant. This is sometimes only possible at the expenses of project development time.

The options are either a risk of staff not being able to cope with the unfamiliar jargon, capabilities and difficulties of the changing technology, or added overhead being built into project completion dates and costs, to provide the possibility of continuing education.

1.1.3.3 Difficulty in Managing the Application Development Project

Bernstein (13) makes the point that the single most critical aspect of the application development is not the complexity of either the software or the particular application, but the management of the development process. This process presents managerial problems because it is difficult to:-

- estimate the size of a project without some industry-wide accepted technique;

- control a project of unknown size and complexity, which is being developed by people whose performance cannot be predicted accurately or measured objectively;

- know at any point in time how much of the project is actually completed, or still has to be completed;

- assess the performance of the development staff at the end of the project, irrespective of whether the time and cost parameters are being met, or not;

- concentrate on development when continual interruptions are necessary so that existing systems, which must remain operational, can be maintained or enhanced.

These factors characterize the software crisis. As will be anticipated, the data processing community has reacted to these problems to try to cope with the crisis. Some of these reactions will be identified in the next section.

1.2. Some Reactions of the Data Processing Industry to The Software Crisis

R. W. Floyd of Stanford University, in his Turin Award lecture of 1978 (14), makes the point that the data processing industry's reaction to the software crisis (he goes so far as to suggest that 'software depression' is a more apt term) has been earnest, but not always effective. One of the ways of trying to cope with a situation that both projects an unprofessional image and distracts from high levels of job satisfaction, has been the rise of 'communities' within the industry. Each community advocates that a particular programming language or approach to design, or development methodology will help to solve the software crisis. The choice of possible approaches to analysis, design and programming, or estimating, resource allocation and project monitoring can be bewildering. New approaches seem to be suggested in the proceedings of each conference.

Meanwhile, all the symptoms of the software crisis remain. Floyd quotes Robert Balzer of the Information Sciences Institute, who wrote:-

> 'It is well known that software is in a depressed state. It is unreliable, delivered late, unresponsive to change, inefficient and expensive ....'
>
> (15)

Still, approximately half of the programming work-force are involved in maintaining existing systems, depleting the already scarce development resources even further (see Section 7.2.6).

Neither the Users, nor general management can be blamed for remaining sceptical when they hear that the situation in their data processing environment is expected to improve because a new approach has been taken in the method of analysing, designing or programming systems, or managing the application development process. They are affected by the symptoms of the software crisis, and the symptoms tend to remain in spite of changes made.

The contribution that being able to measure programmer productivity will make towards helping to resolve these problems is covered in detail in Chapter 4. In summary, it is a step towards being able to control, assess and understand the program development activity. This will give the data processing industry an objective basis to explain its performance to both the end Users and general management.

CHAPTER 1 - FOOTNOTES

(1) DIJKSTRA, E. W.: 'The Humble Programmer': Communications of ACM: October 1972: pp. 859-866.

(2) Ibid.

(3) BROOKS, Jr. F. P.: 'The Computer Scientist' as Toolsmith - Studies in Interactive Computer Graphics': Proceeding of the IFIP Conference 1977: pp. 625-634.

(4) Ibid., p. 625.

(5) SILT REPORT SUMMARY: Transcript of the Presentation given by the SILT Committee of SHARE 43, Chicago, U. S.A., August 1974: Slide 28.

(6) EDP ANALYZER: 'Are we Doing the Right Things?: Volume 13, Number 5, May 1975.

(7) NOLAN, R. L.: 'Managing the Crisis in Data Processing': Harvard Business Review: March/April 1979: pp. 117-118.

(8) SILT REPORT SUMMARY: op.cit. p. 20.

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

(10) COMPUTER PERSONNEL LIMITED (CPL): 'Comment': January 1980, Johannesburg: p. 2.

(11) SILT REPORT SUMMARY: op.cit. p. 20.

(12) 'SHONE, L.: 'Computer Profession Faces Staggering Challenge and Opportunity': System/Stelsels, September 1979: pp. 22-25.

(13) BERNSTEIN, M. I.: 'Hardware is Easy: Software that's Hard': Datamation: November 1978 pp. 32-36.

(14) FLOYD, R. W.: 'The Paradigm of Programming': Communications of the ACM, Volume 22, Number 8, August 1979: pp. 455ff.

(15) Ibid., p. 456.

CHAPTER 2 BOUNDARIES OF THIS RESEARCH

The application development activity can be divided into:-

- analysis (determining what system is required)
- design (determining how it should be developed)
- programming (developing and testing the system)
- implementation (installing the system)
- maintenance (ensuring the system keeps functioning).

In Chapter 11 it is suggested that research be done to identify possible methods of measuring performance in the other phases of application development, but this research is confined to measuring programmer productivity.

The programming activity, however, falls into three broad categories:-

- development programming,
- maintenance programming,
- system (or software maintenance) programming.

No effort has been made to measure the performance of either the maintenance programmer or the systems programmer (see Chapter 11). However, development programming also covers the broad categories of:-

- development of commercial applications,
- development of scientific applications,
- development of process control applications,
- development of systems software.

This research is confined to the identifiable subset of the development of commercial applications - although the concepts discussed may be applicable in the other application areas.

The type of environment in which programmers work, can range through:-

- machine vendor company,
- software house,
- service bureau,
- contract programming,
- in-house installation.

Again it was necessary to limit the areas of research. This study is primarily focused on programmers working in an in-house installation although, again, the concepts may be applicable in other environments.

The objective of this research, then, is an attempt to identify an empirical method of measuring the productivity of programmers who work in in-house installations and are involved in the development of commercial applications.

CHAPTER 3 DEFINITION OF TERMS

Some of the terms used in this research are defined in the context in which they are used (for example, 'Acceptable Percentage Deviation' is defined in Section 10.3.2.). Other terms are defined in detail in a particular section of the text (for example, 'Program Quality' in Section 7.4.2.2.). The definitions in this chapter are of terms used in multiple places in the text, and which may not have consistent definitions in the literature.

Balance Period (statistical procedures)

This is the period over which the factors which influence performance - both positively and negatively - will compensate each other, to give an 'average' level of productivity. (See Section 10.3.2.2)

Break-thru Technology

Any technology which is being used for the first-time by application developers (for example, a new programming language, operating system, or data base software etc.) is regarded as break-thru technology. Consequently, the same technology can be break-thru technology in one environment, but not in another. (See definition of Input Factor in this Chapter, and Sections 10.2.1 and 10.8.3).

Chief-Programmer Team

A concept of team programming used by Mills and Baker on the New York Times project - (1) - where the nucleus of the team is the Chief Programmer, a Back-up Programmer and a Librarian. There is an interesting contrast between Mills and Baker's implementation of the concept, and that of the 'Surgical Team' suggested by Brooks (2). (See also definition of Programming Team in this Chapter).

Close-Knit Programming Team

A concept of team programming used in the Standard Bank Data Processing Division (and based on the concept taught at Van Zyl and Pritchard, Johannesburg) where:-

- the team comprises between 3 and 5 members working under an appointed leader;

- the team leader is responsible for the technical design of the system, contributes to the programming for the system, and manages the whole programming effort;

- all group activites are governed by an agreed set of 'ground-rules' which identify areas of responsibility, hours of work, and method of working etc.

- all program design is reviewed by, and agreed to by the whole team;

- program comments are coded before non-comment source statements;

- all code is reviewed by someone in the team other than the coder;

- the librarian concept (as in the Chief Programmer Team) is used.

Data Block (statistical procedures)

A small measurable unit of work which describes the task being performed to that level of accuracy that it can be regarded as an element of the size of the whole task. (See definition of Output Factor in this Chapter).

Data-Pairs (statistical procedures)

This is another term used to reference each of the observations which were processed through a series of statistical procedures. The two units of data which make up the pair, in the context of this research, are the number of Functions in the programs, and the Program Development Time. (See definitions of each of these in this Chapter).

Democratic Team

A concept of team programming suggested by Weinberg in (3), where the leadership of the team is not imposed from outside the team.

Development Time

In the context of the research, Development Time is the total number of man-hours (see definitions of Man-Hours in this Chapter) required to develop a program. For each program, this will include all the time spent by all the people involved in ensuring that the program is:-

- designed,
- coded,
- reviewed,
- unit tested.

This does not include any time spent on systems testing, user acceptance testing, etc.

Egoless Programming

This term was coined by Weinberg in (4). He felt that some programmers tend to become psychologically attached to their programs - to the extent that they tend to think of the programs they write as an extension of themselves. This attitude results in programmers regarding errors found in their programs as damaging to their self-image. This leads to a tendency to ignore or hide errors.

Enhancement/enhancing

Changing or adding processes to a program or system (see definition of System in this Chapter) to provide facilities to the User which were not previously required. (Compare Maintenance defined in this chapter).

Error (in program or system)

The definition of an error in this context has three dimensions:-

1 A defect which prevents the User from doing what he is permitted to do, or leads to erroneous or unpredictable results.

2 A violation of the standards specified for the particular environment in which the program/system is developed.

3 A violation of the specifications for the particular program/system (for example, maintainability, efficiency, portability etc.).

Function

The number of Functions in any program is determined at program coding stage.

At program coding stage each Function becomes a block of code with the following characteristics:-

The code has a single goal which justifies or necessitates coding grouped together in one paragraph (for example, initialise fields, compute values, validate input);

Control is passed to the block of code at one entry point and is returned to the calling paragraph from one exit point;

The block of code will usually comprise between about 5 and about 60 lines of non-comment source code.

In the unstructured programming environment where the process of functional decomposition has been followed, identifying these Functions is a trival matter because they tend to be the actual program paragraphs or sections.

Others etc.

Because of the lack of a single pronoun appropriate to both gender, the use of 'his' implies 'his and hers' (just as the use of 'he' implies 'he and she' etc.)

Input Factors

In the context of this research this is defined as the effort contribution which is put into program development to get the job done. This includes factors such as:-

- methodologies
- management styles
- standards
- people - their skills and motivation
- quality of specification.

Each of these factors could be different from those any other Programming Department could introduce into the task to produce the required Output (See definition of Output Factor in this Chapter, and the factors which influence programmer productivity in Chapter 5).

Man-Hour (also Man-Day, Man-Year)

A unit for measuring work - equals the amount of work done by one man in one hour (day, year etc.). Sometimes referred to as Work-Hours or Busy-Time. (Compare elapsed, or 'clock-on the wall' time).

Maintenance/maintaining

The correcting and preserving of a program or system (see the definition of a System in this Chapter) in a particular environment to ensure it consistently meets User requirements. (Compare Enhancement defined in this Chapter).

Output Factors

In general terms, this is defined as actual quantity of work produced during the actual hours worked. (See definition of Function, and Input Factor in this Chapter).

Program Quality - see Section 7.4.2.2.

Quality Assured Data

Data subjected to the quality control procedures outlined in Section 10.2.2.2. (Compare Raw Data defined in this Chapter).

Quality - Program - see Section 7.4.2.2.

Raw Data

Data as it is received from its source (for example, from Questionnaire 2) - compare Quality Assured Data.

System

Brooks (5) starts his definition of a system by defining what he calls a program. He defines a program as a product which is complete in itself, ready to be run by the author on the configuration on which it was developed. There are two ways that a program can be converted into a more useful (but more costly) product.

i) The first is what Brooks calls a Programming Product which he defines as a program which can be run, tested, repaired and extended by anyone. This product is usable in many environments, so it is written in a more generalized fashion, comprehensibly tested, and thoroughly documented.

ii) Secondly, a program can become a component of a programming system by ensuring that every input and output conform to precisely defined interfaces. This enables the program to interface with other programs whose co-ordinated processes provide an entire facility. Programs which execute as one of a group of interacting programs must be designed to use only prescribed resources - for example, memory, input - output devices, processing time. Testing must also be comprehensive because all expected combinations of input data and processing requirements must be covered.

A System (called by Brooks, a 'programming systems product') (5) differs from a program in all the above ways.

Team Programming

Program development in which more than one person is involved in the development process. It is not just a group of people who happen to be working on the same project. There are a wide variety of team approaches, however each approach has the following common factors:-

i) being a co-ordinated working unit with each individual contributing towards meeting a common goal;

ii) having a common goal which is identified and understood by each individual working in the team;

iii) achieving results beyond those which could be expected from the same team members working relatively independently.

(See also definitions of Chief-Programmer Team, Close-Knit Team and Democratic Team in this Chapter).

User (sometimes End-User)

Anyone who requires the services of a computer system (see definition of System in this Chapter) to perform his business activities.

CHAPTER 3 - FOOTNOTES

(1) BAKER, F. T. and MILLS, H. D: 'Chief Programmer Teams': Datamation, December 1973: pp. 57 ff.

(2) BROOKS JR. F. P: 'The Mythical Man-Month': Addison-Wesley: Reading, Massachusetts: 1975: pp. 29 ff.

(3) WEINBERG, G. M: 'The Psychology of Computer Programming': Van Nostrand Reinhold, New York, 1971: pp. 67 ff.

(4) Ibid., pp. 72 and 146.

(5) BROOKS JR. F. P: op.cit. pp. 4 ff.

CHAPTER 4 WHY PROGRAMMER PRODUCTIVITY SHOULD BE MEASURED

There does not appear to be complete agreement in the data processing industry concerning the value of measuring programmer productivity. Therefore, as a basis for the remainder of my research, the value of measuring programmer performance will be established for:-

- the data processing industry;
- top management of any company using a data processing facility;
- data processing, and project management;
- programmers.

4.1 The Data Processing Industry

Although there is not complete agreement on the meaning of the word, it has become popular to refer to aspects of data processing as 'Software Engineering'. (For example (1), (2) and (3)).

It seems that the data processing industry has a strong need to be accepted as one of the professions (see (4)) and a step towards achieving this is to regard, in particular, the application development process as an engineering discipline.

I have a problem with this concept. In the Collins English Dictionary (1979) Engineering is defined as:-

> 'the profession of applying scientific principles to the design, construction and maintenance of .... machines etc.'

Unfortunately there are few (if any) scientific principles on which the design, construction and maintenance of computer systems are based.

Consequently, I agree with researchers (like Bernstein (5)), who maintain that computer systems development is not a disciplined engineering process, and cannot become so until it can be quantitatively measured and controlled.

It is possibly premature to imply that the application development process is a professional activity. Perhaps, too, calling application development staff 'software engineers' (eg, South African Computer Society's Software Engineering Special Interest Group) should be delayed until such time as their work can be measured to the extent that:-

- the results achieved can be understood;
- past mistakes are not repeated;
- it is known how to attain reproducable results;
- the application development process moves away from being an 'art' towards being a more scientific and credible discipline.

Being able to measure the productivity of programmers objectively is one step in this direction. It is only after more progress is made along this road that the title of 'Software Engineering' may be appropriate.

4.2 Senior Management

James R. Johnson writes:-

> 'Productivity is only productivity if it is measurable. Knowing, feeling or believing (that) productivity is high is not, or soon will not be, acceptable in the programming environment. Faith is no longer an attribute of higher management. Proof of performance

> is the central issue. As maintenance and overheads increases, justification of the growing data processing budget becomes more difficult. First and second level managers are comfortable with existing management techniques, but higher level management is making the message clear: SHOW US YOUR PRODUCTIVITY INCREASES.' (6)

Already it has been noted (see Section 1.1.1.1) that this view is held by the authors of the SILT Report. Contrasted with the increase in budgets, the increase in software development is predicted to be a meagre 3% per annum (7). Not only is this much less than the increase in budgets, it will be hardly perceptible in contrast with the cost-effectiveness of hardware (see Section 1.1.1.1). Programmer productivity, therefore, is the potential bottle-neck in the future growth of the data processing facility on which many companies have come to rely.

It was significant that the work in which I was involved at the Standard Bank on measuring programming productivity was initiated by the Senior General Manager through the Costing Department and not directly from the Data Processing Division. Concern about programmer performance came from that level of management. It is evident that top management needs to be certain that:-

- budgeted productive and non-productive time in the data processing departments fall within acceptable limits;

- all productive time is actually being booked to projects currently being developed by the departments;

- the performance of technical staff during application development is in accordance with specified (or industry acceptable) standards.

Without being able to measure programmer productivity, these assurances cannot be given.

4.3 Data Processing (or Project) Management

The value of measuring programmer productivity to the levels of management directly involved in application development, falls into the categories of:-

- improving estimating,
- providing the possibility of actually managing a project,
- providing the possibility of being able to identify good application development methods,
- providing the possibility of identifying good performers.

4.3.1 Improved Estimating

Project estimating is not a well developed or exact science. Brooks (8) has some strong views on the subject. He feels the lack of substantiated estimates is one of the reasons for poor quality systems. He claims poor quality systems are likely to be implemented if the project manager lacks the courteous stubborness required to make the User wait for a good product. Brooks points out, however, that it is difficult to make a vigorous, plausible, job risking defence of an estimate, if that estimate is not derived from a quantitative method, nor certified by little other than the hunches of the manager.

This problem has a cycle of three clearly identifiable steps:-

i) Without realistic estimates, project planning is likely to be unrealistic.

ii) Unrealistic planning inevitably leads to ineffectual control of the development process.

iii) Without tnis control, meeting the specificed completion dates tends to become possible only by working excessive overtime, or skimping on some phase of the project life cycle. This skimping is, unfortunately, usually either in the area of careful design, or thorough testing. This results in what Capers Jones calls 'psychopathetic' application development where undetected defects early in systems development lead to even longer development times. (9).

This cycle will invariably begin if the original estimating is poor. Now, any estimating presupposes that productivity can be measured. If data on the actual performance of the programmers can be provided, estimates will be supported and estimating can improve. In turn this will lead to improved project planning, project control and the quality of the products produced.

In fact being able to measure programmer performance makes it possible to set realistic targets and objectives for application development staff. Until this can be done, the ability to manage the application development activity must be questioned.

4.3.2 Identifying Good Application Development Methods

As has been noted in Section 1.2 one of the data processing industry's reactions to the 'software crisis' is a proliferation of application development methods. The proponents of each method claim their approach brings with it high improvements in programmer performance. It is, however, not sufficient to accept their unsubstantiated claims. The results of using any method to improve programmer productivity need to be analyzed and good techniques incorporated in other projects.

In an interview published in Computing South Africa, James Martin said:-

> 'I have decided to swing my next round of seminars towards the subject of productivity .... there are .... forms of management of data processing that can make enormous .... changes in productivity. We need to be asking the questions:
>
> What are they?
> Can we use them now? ..... (10)

Being able to measure productivity objectively will make it possible to identify those methods and innovations in programming which bring about real improvements in programmer performance.

Author Crossman T D
Name of thesis On the possibility of measuring programmer productivity 1981

PUBLISHER:
University of the Witwatersrand, Johannesburg
©2013

LEGAL NOTICES:

Copyright Notice: All materials on the University of the Witwatersrand, Johannesburg Library website are protected by South African copyright law and may not be distributed, transmitted, displayed, or otherwise published in any format, without the prior written permission of the copyright owner.

Disclaimer and Terms of Use: Provided that you maintain all copyright and other notices contained therein, you may download material (one machine readable copy and one print copy per page) for your personal and/or educational non-commercial use only.

The University of the Witwatersrand, Johannesburg, is not responsible for any errors or omissions and excludes any and all liability for any errors in or omissions from the information on the Library website.