4.3.3 Identification of Good Programmers

With the current shortages of application development staff in South Africa, (and apparently world-wide), management has a special need to identify, develop and retain good members of staff. Without an objective method of measuring their performance, this identification process is likely to be suspect because it is forced to be subjective, and must be based on circumstantial evidence. Being able to measure programmer productivity will help management to know who their skilled programmers are, and will provide them with a basis for determining rewards for programmer performance.

4.4 The Programmer

I have found that most opposition to the concept of measuring programmer performance tends to come from the technical staff themselves. There seems to be a fear among programmers of being measured, and sometimes even an anger at being measured. On one occasion early in my research, I was calculating the performance of a team whose system had just been implemented. The unit of measurement I was using was the number of functions produced per man-hour (while the statistics required at the time by the management of the team was the number of implemented lines of code per man-hour). Because their productivity rate in terms of the number of functions per man-hour seemed low, I asked the programmers concerned if they had experienced any particular problems during the project, which had affected their performance. The retort I received was:-

> 'But we did not know we were being measured
> THAT way.'

Eventually, because management were also angered by my question, I was instructed to assure the project team that the comments I had made did not reflect management's opinion!

A negative reaction from programmers to having their performance measured seems to deny their need to identify good and bad development methods. Perhaps, however, the reaction to being measured (it could be a fear of being categorized) is stronger than the need really to understand the results being achieved, or to monitor performance as experience and skills increase.

If this is the programmers' reasons for reacting to having their performance measured objectively, it is surprising. Because of the nature of their work, there is a sense in which programmers are being measured continually.
Brooks says it succinctly:-

> '(a programmer) must perform perfectly. The computer resembles the magic of legend in this respect, too. If one character, one pause, of the incantation is not strictly in proper form, the magic doesn't work ..... adjusting to the requirement of perfection is, I think, the most difficult part of learning to program.' (11)

Compared with this, it seems hardly more stringent to be measured in respect of levels of productivity.

CHAPTER 4 - FOOTNOTES

(1) FORMAN, J.J: 'Implementing Software Standards': The Open Channel, Computer; June 1980; pp. 67-70.

(2) QUINNAN, R. E: 'The Management of Software Engineering, Part V: Software Engineering Management Practices; IBM Systems Journal: Volume 19, Number 4 1980: pp. 466 - 477.

(3) THAYER, R. H., PYSTER, A, and WOOD, R. C: 'The Challenge of Software Engineering Project Management': Computer: August 1980, pp. 51-59.

(4) EDP ANALYZER: 'Professionalism Coming or Not?': Volume 14, Number 3, March 1976.

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

(6) JOHNSON, J. R: 'A Working Measure of Productivity': Datamation: February 1977: pp. 106-110.

(7) SILT REPORT SUMMARY: Transcript of the Presentation given by the SILT Committee at SHARE 43, Chicago, USA, August 1974.

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

(9) JONES, T. C: 'Measuring Programming Quality and Productivity': IBM Systems Journal: Volume 17, Number 1: 1978: p. 57.

(10) MARTIN, J: 'At Home with James Martin': An interview with Martin published in Computing South Africa, Volume 1, Number 17: 29 October 1981, pp. 13-19.

(11) BROOKS JR. F. P: op.cit. p. 8.

CHAPTER 4 - FOOTNOTES

(1) FORMAN, J.J: 'Implementing Software Standards': The Open Channel, Computer; June 1980; pp. 67-70.

(2) QUINNAN, R. E: 'The Management of Software Engineering, Part V: Software Engineering Management Practices; IBM Systems Journal: Volume 19, Number 4 1980: pp. 466 - 477.

(3) THAYER, R. H., PYSTER, A, and WOOD, R. C: 'The Challenge of Software Engineering Project Management': Computer: August 1980, pp. 51-59.

(4) EDP ANALYZER: 'Professionalism Coming or Not?': Volume 14, Number 3, March 1976.

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

(6) JOHNSON, J. R: 'A Working Measure of Productivity': Datamation: February 1977: pp. 106-110.

(7) SILT REPORT SUMMARY: Transcript of the Presentation given by the SILT Committee at SHARE 43, Chicago, USA, August 1974.

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

(9) JONES, T. C: 'Measuring Programming Quality and Productivity': IBM Systems Journal: Volume 17, Number 1: 1978: p. 57.

(10) MARTIN, J: 'At Home with James Martin': An interview with Martin published in Computing South Africa, Volume 1, Number 17: 29 October 1981, pp. 13-19.

(11) BROOKS JR. F. P: op.cit. p. 8.

CHAPTER 5 FACTORS WHICH INFLUENCE PROGRAMMER PRODUCTIVITY

There are some project variables which are identified by most researchers working in this field, as factors which influence programmer performance. These include:-

- programmer's experience,
- programming standards, and
- programming language.

What is surprising is how few project variables are identified as significant by more than one author, and how seldom the same variable is regarded by researchers as having the same effect on programmer performance. To illustrate, some opinions are summarized.

In an article published in 1974 Scott and Simmons (1) reported on a study they undertook to try to determine what project variables have the greatest impact on programmer productivity. They sent a list of 35 variables to a panel of 43 members, who were asked to correlate each variable with programmer productivity on a scale which indicated degrees of negative influence, no influence, or degrees of positive influence. The panel was selected because they were managing projects, or were involved in relevant research, or because of past experience. Each was considered an expert in the field of project management. A degree of consensus was reached by the panelists on the following points:-

- quality of external documentation,
- programming language,
- programmer experience in data processing,
- programmer experience in functional area,
- effect of project communications,
- independent modules for task assignment,
- well-defined programming practices.

It is interesting to compare this list with the variables suggested by Chrysler (2). In his view, programmer productivity is affected by:-

i) Computer hardware characteristics
   - memory size,
   - storage devices.

ii) Source language software characteristics
   - the instruction set,
   - level of self-documentation.

iii) Programming mode characteristics
   - on-line/off-line code entry.

iv) Organisation operations characteristics
   - programming methods,
   - standards,
   - system documentation standards.

v) Programming problem characteristics
   - input edits,
   - report formats.

vi) Programming characteristics
   - programming experience,
   - source language experience.

These points cover a more comprehensive range of influences than those suggested by the panel selected by Scott and Simmonds.

In a similar list compiled by Watson and Felix (3) there are some project variables which are not included in either article referred to earlier in this section.

Watson and Felix's list of 29 variables reads:-

- Customer interface complexity
- User participation in tne definition of requirements
- Customer orientated program design changes
- Customer experience with the application area of project
- Overall personnel experience and qualifications
- Percentage of programmers doing development who participated in the design of functional specifications
- Previous experience with programming languages
- Previous experience with application similar or greater size and complexity
- Ratio of average staff size to duration (people/month)
- Hardware under concurrent development
- Development computer access
- Classified security environment for computer
- Structured programming
- Design and code inspections
- Top down development
- Chief programmer team usage
- Overall complex ty of code development
- Complexity of application processing
- Complexity of program flow
- Overall constraints on program design
- Program design constraints on main storage
- Program design constraints on timing
- Code for real-time or interactive operation
- Percentage of code for delivery
- Code classified as non-mathematical
- Number of classes of items in the data base per 1000 lines of code
- Number of pages of delivered documentation per 1000 lines of code delivered.

In the research he conducted in 1976/77, Johnson (4) mentions a few variables which he found influenced the programmer performance on the projects which he was analyzing. He lists these as:-

- Design difficulty (innovation)
- Level of technology
- Quality of staff
- Team motivation
- Working with known technology.

Quoting work done by the (American) Air Force Electric Systems Division in 1965, Boehm (5) quotes some project variables which have not been included in the previous summaries. These include:-

- The stability of program design
- Percent mathematical instructions
- Number of sub-programs
- Concurrent hardware development.

In the survey conducted by the Department of Information Systems, University of N.S.W., Australia (6) the list of project variables on which data was required, included:-

- Programming language used
- Length of programmer's experience
- Type of program being developed (utility, update, report, edit, file extract etc.)
- Frequency of program use
- Batch or on-line environment
- Use of data base
- Programming technique used (structured programming, top down, walk-thru etc.)
- Mode of testing (batch, on-line, RJE etc.)
- Average turnaround time
- Involvement by programmers in parallel tests.

In the research he conducted in 1976/77, Johnson (4) mentions a few variables which he found influenced the programmer performance on the projects which he was analyzing. He lists these as:-

- Design difficulty (innovation)
- Level of technology
- Quality of staff
- Team motivation
- Working with known technology.

Quoting work done by the (American) Air Force Electric Systems Division in 1965, Boehm (5) quotes some project variables which have not been included in the previous summaries. These include:-

- The stability of program design
- Percent mathematical instructions
- Number of sub-programs
- Concurrent hardware development.

In the survey conducted by the Department of Information Systems, University of N.S.W., Australia (6) the list of project variables on which data was required, included:-

- Programming language used
- Length of programmer's experience
- Type of program being developed (utility, update, report, edit, file extract etc.)
- Frequency of program use
- Batch or on-line environment
- Use of data base
- Programming technique used (structured programming, top down, walk-thru etc.)
- Mode of testing (batch, on-line, RJE etc.)
- Average turnaround time
- Involvement by programmers in parallel tests.

Because of the diverse range of opinions expressed in these articles, there is a danger that, in this research, significant factors which influence programmer productivity will be overlooked. To help prevent this, project variables which I consider have significant influence on programmer performance have been grouped into two broad categories:-

- the programmer, and
- the programming environment.

These categories are then sub-divided further into factors and groups of factors.

5.1 The Programmer

The problem of significant differences in the performance levels of individual programmers has recurred repeatedly in the literature. One example is the work done by Sackman, Erikson and Grant (7) in 1967. The results of their work (see Table 1) are used to substantiate the claim that there are sometimes substantial differences in individual programmer's level of productivity. In an effort to assess the impact of on-line program development, Sackman et. al. used 24 programmers who were set the task of writing and debugging two programs. The large variances in the results almost obscure the objective of the experiment.

The researchers summarize their findings by rephrasing the nursery rhyme:-

'When a programmer is good,
He is very very good,
But when he is bad,
He is horrid'

Table 1 Some of the results of the experiment by Sackman et. al. which show differences in individual programmer performances.

Activity	Poorest Score	Best Score	Ratio
Code (Hrs) Prog. 1	111	7	16:1
Code (Hrs) Prog. 2	50	2	25:1
Program Size: Prog 1	6137	1050	6:1
Program Size: Prog 2	3287	651	5:1
Run Time (Sec) Prog 1	7,9	1,6	5:1
Run Time (Sec) Prog 2	8,0	0,6	13:1
Debug (Hrs) Prog 1	170	6	28:1
Debug (Hrs) Prog 2	26	1	26:1

Another experiment which confirms the significant differences between levels of programmer performance, was conducted by Cooke (9). In this experiment he used professional programmers to develop and complete a single successful execution of the same program, but using different programming languages and environments. The figures quoted in Table 2 are among the results. Again, the differences in individual programmer performance appear to be non-trivial. I analyse the possible reasons for these differences under the headings:-

- technical skills,
- interpersonal skills,
- motivation and
- programming objectives.

Table 2 Some of the results of the experiment by Cooke which show differences in individual programmer performance.

Activity	Programmer 1	Programmer 2	Ratio
Develop	3 hours	5 hours	1:1,6
Execute	192 seconds	53 seconds	3,6:1
(Program written in Problem Orientated Language)			
	Programmer 3	Programmer 4	Ratio
Develop	8,5 hours	29 hours	1:3,4
Execute	1,3 seconds	0,5 seconds	2,6:1
(Program written in ALGOL)			

5.1.1. Technical Skills of the Programmer

Because of the diverse and logically unrelated skills required for the programming task, proficiency in one area may not result in a high level of overall programming skill. To create logically correct algorithms requires different skills (and training) from those required to master the idiosyncracies of a particular compiler, or the power of a particular programming language.

Further, many programmers have learnt their skills by copying styles and techniques used by their peers, or the senior programmers in their development departments. Although it is possible for programmers to learn skills from universities, technicons or independent training organisations, it is significant that there is such diverse opinion on what is considered an appropriate curriculum, and that (in South Africa)

the Computer Society is concerned with the quality and standard of training at the independent organizations.

Some of this leads to uniformity in training and education. So, not only are some people technically more skilled at the programming task than others, the lack of standardized training (so symptomatic of a non-professional activity) contributes further to wide variances in programmer performance.

Programmer's Interpersonal Relationship Skills

So-called 'personality clashes' between members of staff can result in a loss of personnel's energies, and a consequent loss of productivity. However, conflict (of some description) is possibly the one common factor found in all groups of any size who are trying to complete any task. The traditional human-relations point of view was that all conflict is basically bad, and every possible step should be taken to avoid it.

There is an unprecedented step (taken for example by Kelly (10)) which suggests that conflict is not intrinsically undesirable. Kelly claims that a certain amount of conflict is both acceptable and necessary for effective performance. Management skills are required to determine the acceptable level of interpersonal conflict, in each work situation.

The goal of interpersonal relationships in the work environment is to enable the group to achieve results which are significantly beyond the results which could be expected from the same individuals working in isolation. When this synergistic effect is present, productivity can be expected to be high. If it is absent, levels of productivity can be expected to drop.

5.1.3. Programmer's Motivation

It is obvious that one of the contributing factors to variances in levels of programmer performance is the degree of motivation the programmer has at the time of developing a particular program. In this section some of the traditional approaches to the theory of individual motivation will be discussed and their inadequacies identified, then another approach to determining an individual's level of motivation will be assessed.

5.1.3.1 Some Traditional Explanations for an Individual's Level of Motivation

Factors which affect a worker's level of motivation have been identified by researchers like:-

- Maslow (Hierarchy of Needs),
- McGregor (Theory X and Theory Y) and
- Hertzberg(Hygiene Factors and Motivators).

Included in the work done specifically in the area of motivating factors for data processing staff, is research undertaken by Fitz-Enz (11). He bases his research on the premise that motivation is an inherent human trait. Fitz-Enz argues - like Hertzberg - that managers cannot motivate workers. All that they can do is to provide the environment (or climate) which allows individuals to satisfy their own internal drives.

Fitz-Enz compared the results of his research on motivating factors for data processing staff with the results of Hertzberg's research. Table 3 shows the similarities and differences in their results.

Table 3 Results of Hertzberg's 1960 Study Compared with Fitz-Enz's 1978 Study. (See (11)).

Hierarchy of Motivating Factors	
Hertzberg	Fitz-Enz
1 Achievement	1 Achievement
2 Recognition	2 Possibility for growth
3 Work itself	3 Work itself
4 Responsibility	4 Recognition
5 Advancement	5 Advancement
6 Salary	6 Supervision, Technical
7 Possiblity for growth	7 Responsibility
8 Interpersonal relations, subordinate	8 Interpersonal relations, peers
9 Status	9 Interpersonal relations, subordinate
10 Interpersonal relations, superior	10 Salary
11 Interpersonal relations, peers	11 Personal life
12 Supervision, technical	12 Interpersonal relations, superior
13 Company policy and admin	13 Job security
14 Working conditions	14 Status
15 Personal life	15 Company policy and admin
16 Job security	16 Working conditions

Fitz-Enz draws two conclusions from this comparison:-

Conclusion 1

A basic knowledge of human behaviour can help project managers perceive and cope with the attitudes, interests, needs and values of programmers. No matter how sophisticated the technological environment in which managers work, the effectiveness of their performance as managers is dependent on their ability to understand and manage people. (See Section 5.2.1.4).

Conclusion 2

The more the data with which he worked was dissected, the more it became apparent to Fitz-Enz that together with the basic similarities in motivational patterns of people, individual differences are significant. In order to allow people to be motivated, individual needs must be understood. The monolithic notion that 'people are all alike' is not supportable.

5.1.3.2 Some Inadequacies in the Traditional Approaches of explaining an Individual's Level of Motivation

The second conclusion to which Fitz-Enz came (see Section 5.1.3.1 above) is the main theme of the work done by D. A. Nadler and E.E. Lawler. (12). In their article 'Motivation: A Diagnostic Approach' they make three observations concerning the traditional approaches to motivation to which managers have been exposed during the past 20 years.

Observation 1 The traditional approaches to motivation (whether expressed in terms of 'scientific management', 'needs hierarchy', 'job enrichment' or 'self-actualisation') each seem to assume that all employees are basically similar in their make-up.

Observation 2 Most of the traditional theories on motivation seem to assume that all managerial situations are alike, so a single course of action for motivation is applicable to all situations.

Observation 3 Out of the other two observations there seems to emerge a basic principle in the traditional approaches to motivation that there is 'one best way' to motivate employees.

Nadler and Lawler concede that when these 'one best way' approaches are tried in the 'correct' situation, they will work. However, all of them are likely to fail in some situations, consequently they are suspect as adequate managerial tools.

5.1.3.3 Another Approach to Understanding an Individual's Level of Motivation

In an effort to provide an approach to understanding the causes of behaviour by individuals in an organisation, for predicting the effects of any managerial action and directing the behaviour of individuals to help improve productivity, Nadler and Lawler suggest a new approach to assessing motivation (12). The approach is based on a number of specific assumptions about the causes of behaviour in organisations:-

Assumption 1 Neither the individual nor the environment alone determines patterns of behaviour. Past experience and understanding give each individual unique sets of needs, unique ways of looking at the world, and unique expectations about how an organisation should treat him. Consequently different environments tend to produce different behaviour in similar people, just as dissimilar people tend to behave differently in similar environments.

Assumption 2 People make their own decisions about being part of an organisation (for example making application to join a Programming Department, or coming to work on a particular day). They also decide individually about the amount of effort they will direct towards performing their work. So, while an organisation may place many constraints on the individuals within the organisation, most of the behaviour which is observed, is the result of the individual's conscious decisions.

Assumption 3 Like Fitz-Enz, Nadler and Lawler base their research on the assumption that individuals have different needs, desires and goals.

Assumption 4 People tend to do those things which they believe will bring them the rewards they desire (and avoid those outcomes they do not desire).

Based on these assumptions, Nadler and Lawler claim that motivation is highest when:-

- individuals believe that their behaviour will lead to different kinds of specific outcomes (for example by bringing a project in on time will lead to User satisfaction, management approval or a sense of individual well-being).

- individuals believe that each of these outcomes has some personal value, worth or attractiveness (for example, ensuring that approval of a highly esteemed project manager can be worth all the inconvenience of working overtime to a junior programmer).

- individuals believe that they can perform the work required of them (for example, that it is possible to master the new programming language and meet the project completion date).

(This view is also held by McClelland and Atkinson - quoted in (13) who demonstrated in their research that the degree of motivation and effort rises until the probability of success reaches 50%, then begins to fall, even though the probability of success continues to increase. They suggest that people are not highly motivated if a goal is seen to be almost impossible, or virtually certain, to achieve).

So motivation is seen as the force on an individual - possibly primarily coming from within - to expend effort. To produce results, however, effort alone is not enough. The individual needs appropriate levels of skill (see Section 5.1.1). Depending on the environment in which the individual programmer is working and living, so a degree of motivation will drive the programmer to levels of performance.

Nadler and Lawler identify some steps which managers can take to help programmers to be positively motivated in their work environment:-

- identify the outcomes or rewards which have value to each individual programmer (this is not changing what people want, it is finding out what they want);

- define what levels of performance, and types of behaviour are required from programmers to ensure they know what is meant by 'good performance';

- define the required levels of performance in such a way that they are at least attainable by the individuals on the project;

- directly, clearly and explicitly link the outcome desired by the individual programmer with the specific performances desired;

- ensure that there are no conflicting expectations in the minds of the programmers (if there are, there may be the need to adjust the performance or the reward structures);

- the rewards/outcomes must be large enough to motivate significant behaviour;

- the system must be equitable, so that programmers who perform well can see that they get more desired rewards than do the poor performers;

- if possible the tasks and responsibilities within a project must be individualised to cater for individual needs.

Levels of programmer productivity will not reach their potential if managers do not ensure that levels of individual motivation are kept high (see Section 5.2.1.4).

5.1.4 The Objectives to which the Programmer is working

Weinberg (14) discusses the problems of psychological measurement. He points out there are 'literally thousands' of personal variables which an analyst might measure to establish the reasons for differences between individuals. He illustrates his point by listing two variables per letter of the alphabet - and includes:-

> 'age or agility ....
> deductive reasoning or degrees ....
> mother's tidiness or muscle tension ....
> zygosity or zodiacal sign.'

In an experiment conducted in 1974 and described in a lecture entitled 'Models of Programming Productivity' (quoted by Gilb (15)), Weinberg details the results of giving programmers clearly stated objectives to influence their performance.

Each programming group which participated in the experiment was given a single (different) objective to achieve when developing a particular program. These objectives were:-

- to minimize primary memory needed;
- to maximize program output readability;
- to maximize source program readability;
- to minimize number of program statements;
- to minimize time to complete the program.

The programs were then evaluated to determine their ranking in respect of each objective.

Table 4 below gives the results of the experiment, where 1 = best performance, and 5 = worst performance.

Table 4 The Influences of Objectives on Programmer Performances

Major Objective	Memory	Output	Source	Statements	Time
Minimize Primary Memory	1	4	4	2	5
Maximize Output Readability	5	1	1-2	5	2-3
Maximize Source Readability	3	2	1-2	3	4
Minimize Program Statements	2	5	3	1	2-3
Minimize Production Time	4	3	5	4	1

In each case the group with the specific objective was ranked as highest scorer in meeting tnat objective. In the context of this study the important observation is that when a group concentrated exclusively on one objective, they were each rated poor performers - in all but one case, POOREST performers - in meeting one other objective. (For example, the group which attempted to minimize primary memory took the longest time to complete the program, and the group which completed the program in the quickest time had the most unreadable source programs, and was ranked second poorest in both minimizing memory size and minimizing the number of source statements.)

It is interesting to speculate what results Weinberg would have achieved if he had given a sixth group of programmers the task of meeting all 5 objectives. Probably the group would not have ranked highest in any of the categories.

If we can take this experiment of Weinberg's as representative of programmers in the industry, the particular objectives to which each individual programmer is working (whether specified or assumed) will influence that programmer's level of productivity.

5.2 The Programming Environment

The actual programming environment is comprised of many factors which could influence programmer performance. This environment will be sub-divided into two sections. The first section will identify the Working Conditions of the Programmer, and the second section will analyze the influence on programmer performance, of the Task to be Done.

5.2.1 The Working Conditions

Again, it is possible to sub-divide this section. The main headings under which the influence of working conditions on programmer performance will be discussed, are:-

physical factors,
technical factors,
standards and procedures, and
competence level of management.

5.2.1.1. Physical Factors

G. M. McCue (16) makes the point that programmers normally have to work in areas that were initially intended for other uses - typically in offices designed for other business. It is seldom that the area in which programmers work is an area which has been designed and built for programmers. (The laboratory for IBM programmers at Santa Teresa is one exception). Consequently, programmers have to be content with a number of practical problems which affect their productivity. These problems include:-

i) Insufficient handling and storage place for coding sheets, listings and manuals;

ii) Inadequate lighting, air conditioning and telephone facilities;

iii) Intricate access to a library of technical manuals, or to development and testing facilities;

iv) Difficulties in communicating with each other because, either programmers are given their own offices, or (more usually) desks are separated by screens which reach part-way to the ceiling of an 'open-plan' office;

v) Noise levels and other distractions which the screen partitioning in the 'open-plan' office is incapable of preventing, which make protracted periods of high level concentration difficult;

5.2.1.1. Physical Factors

G. M. McCue (16) makes the point that programmers normally have to work in areas that were initially intended for other uses - typically in offices designed for other business. It is seldom that the area in which programmers work is an area which has been designed and built for programmers. (The laboratory for IBM programmers at Santa Teresa is one exception). Consequently, programmers have to be content with a number of practical problems which affect their productivity. These problems include:-

i) Insufficient handling and storage place for coding sheets, listings and manuals;

ii) Inadequate lighting, air conditioning and telephone facilities;

iii) Intricate access to a library of technical manuals, or to development and testing facilities;

iv) Difficulties in communicating with each other because, either programmers are given their own offices, or (more usually) desks are separated by screens which reach part-way to the ceiling of an 'open-plan' office;

v) Noise levels and other distractions which the screen partitioning in the 'open-plan' office is incapable of preventing, which make protracted periods of high level concentration difficult;

vi) Major upheavals when reorganizations and moves of staff are necessary to redeploy resources as a result of recruiting more development staff, or at the commencement of new projects.

5.2.1.2 Technical Factors

The list of technical factors in the programming environment which influence productivity include:-

- the demands of the vendor supplied software currently in use at the installation (for example, job control language, procedures for compiling programs etc.);

- the stability (reliability) of vendor supplied software;

- the types of applications being developed (batch, real-time, data-base etc.);

- the programming languages used (assembler, high-level language, query language etc.);

- the programming tools available (interactive edit facilities, traces, dumps, debugging aids, and language pre-processes etc.);

- the availability of machine-time and testing turnaround times;

- the development methodologies used (standard programming structures, Structured Design, Team Programming etc.);

- the quality of analysis and design done before programming commences (see Sections 5.2.1.3 and 7.2.3 for a more detailed discussion of this point).

5.2.1.3 Standards and Procedures

The enforcement of standards and procedures to ensure a uniform program development approach will also affect the productivity level of programmers. To make sure that programmers' performance can be at its highest, procedures are required which will:-

- limit to a minimum the administrative demands made on technical staff;

- ensure that a standard approach to programming is used by all programmers;

- ensure that technical standards will be used by the programming staff, because they are appropriate, available, up-to-date and not voluminous.

A scenario will be used to illustrate this point.

The traditional project life cycle covers the phases of business analysis, technical design, programming, testing and implementation. Unless there are standards which, for example, stipulate (and therefore give a basis for the control of) the contents of the output of the design phase, and the level of detail to which the designer must go before the design if considered complete, the next phases in the project life cycle can be significantly affected.

There are at least two possibilities:-

i) Should the design phase be inadequately completed but programming be allowed to commence, the productivity levels of the programmers probably will be limited by the need for additional design decisions to be made during the programming phase.

ii) Conversely, should the design phase continue to include the detail design of the programs, the performance of the programmers will be apparently enhanced.

One of the ways of overcoming this problem is to measure application development productivity in the context of the whole project, and not one particular phase of the project. However, this approach precludes the possibility of being able to identify where the problem lies should it be found that the productivity of those developing a project is low.

At the same time, however, I concede that to measure the productivity of programmers in isolation could be inequitable (see Section 10.6.4).

5.2.1.4. Competence Levels of Management/Supervision

In their book Hersey and Blanchard (17) claim that skills are required in at least three areas to ensure that the management process is carried out:-

Technical Skill - the ability to use knowledge, methods, techniques and equipment necessary for the performance of specific tasks. This technical skill is acquired from experience, education and training.

Human Skills - the ability and judgement in working with and through people, including the understanding of motivation (see Section 5.1.3.3) and an application of effective leadership.

Conceptual Skills - the ability to understand the complexities of the overall organization, and where one's own operation fits into the whole. This knowledge permits a manager to act according to the objectives of the total organisation rather than on the basis of the goals and needs of his own immediate group.

Proportionate mixes of these skills vary from individual to individual in managerial/supervisory positions, and as one individual advances from lower to higher levels of management within an organisation. Should any of these skills be underdeveloped, or should one or other be incorrectly emphasised in relation to the others, the result is not only a limited managerial performance, but individuals working under the manager will find their performance affected adversely.

In this context two points require further attention:

i) I am aware of a significant divergence of opinion (between technical people, data

processing management and personnel management) concerning the significance of 'Technical Skill' in managing the programming environment. There is the view that a programming manager must be able to do the job he is managing (and this is possible in development departments where the best programmer has been promoted to management level because he is best programmer, or where Chief Programmer Teams are being used). The pressure that this approach creates for the manager to keep abreast with technical developments while developing his managerial skills, may be one of the reasons why it is sometimes advocated - and Louis Allen (18) is an example - that the management function is divorced from the technical activity being managed. However, if this approach be taken there is always the possibility that the technocrats will, intentionally or unintentionally, mislead management concerning what is possible or desirable.

ii) The emphasis on human skills is crucial at all levels of management. This sometimes presents a particular problem in the programming environment. Programmers have the image of being individualists who are attracted to working with technicalities in isolation (a view supported by research done by Cougar (19)).

> Programmers appear not to need, and therefore lack, these human skills. Consequently when they are promoted to management - as mentioned in the previous point - there is a tendency for this deficiency in their human skills to result in poor managerial performance. Obviously the productivity of the programmers whom they manage is affected.

It is unrealistic to expect some standard approach to programming management. Conflicting styles are often equally effective, and individuals working under a particular manager react differently to that manager's style. However, one of the limitations of my research is that I have not attempted to measure the impact of management on the productivity levels of programmers. In fact, I have not been able to identify any research done in this area.

5.2.2 The Characteristics of the task to be done

It seems that those things which influence productivity in the context of the task that must be done, fall into two sub-sections:-

- the complexity of the task, and
- the quality of the work produced.

These two topics will be examined here in relation to the affect they have on the performance of computer programmers. However both are problem areas which receive more detailed attention in Section 7.4, while program complexity is given further attention in Section 10.8.2.

5.2.2.1 Program Complexity

People have gone to great lengths - for example at the S.A. Burroughs User Group held in May 1980 - to point out to me that the complexity of programs has a significant impact on programmer performance. ('You cannot compare unadjusted productivity figures of a programmer who has written a scientific process control program in FORTRAN', they say 'with a programmer who has written a simple COBOL extract and print program!').

The whole problem of measuring programmer productivity and the possible degrees of complexity found from program to program is covered in Sections 7.4.1, and 10.8.2. At this stage it is noted that if comparisons of programmer productivity are to be equitable, the complexity of the programs which have been developed cannot be ignored.

5.2.2.2 The Quality of the programs developed

One of the most surprising findings in the research done for this dissertation has been to discover how seldom the relationship between programmer productivity and program quality has been mentioned in the literature. (see Section 7.4.2.2 for definition of program quality). A notable exception has been Bernstein (20) who wrote in the Readers Opinion Forum of Datamation. Even the point he makes is not altogether what one would expect. He queries why the emphasis in research has been on trying to improve programmer productivity rather than trying to improve program quality.

> 'Are we putting our emphasis on productivity because the problems and issues of software quality are murkier? Or are we saying that quantity is a proper substitute for quality' (21)

He ends the article by making a plea for improving the quality of programs 'because this is what the industry needs and the user demands, and (it provides) the only rational basis for understanding and improving productivity'. (21).

The whole problem of trying to understand, define and control program quality is covered in Section 7.4.2. What one would expect to find Bernstein (and many others) saying, is that unless there is some control or standaridization of program quality, to compare the performance levels of programmers has little meaning. In fact, I make the assertion that to compare the same programmer's performance (with every other project variable being held constant) in developing two different programs, without enforcing some form of quality control, is a futile exercise. A high quality program which has required extra care in design, and extended and thorough testing, can result in the programmer's productivity figures appearing low. Ostensibly high productivity figures can be achieved by ignoring program quality altogether.

This is the point inferred by Brooks (22) in another context. He asserts that it takes nine times longer to develop a program which will:-

- interface with other programs,
- execute repeatedly in an environment other than the one in which it was developed,
- be executed by someone other than the author

than it takes to write a program which will be executed once by the programmer, in the environment in which it was developed.

Differences in program quality can account for large discrepancies in program performance figures. (see Section 5.1).

5.3 Conclusion on the factors which influence Programmer Productivity

I conclude that not only is there a broad spectrum of factors which influence the performance of programmers, but being able to identify the extent of these influences is likely to be complex and inaccurate.