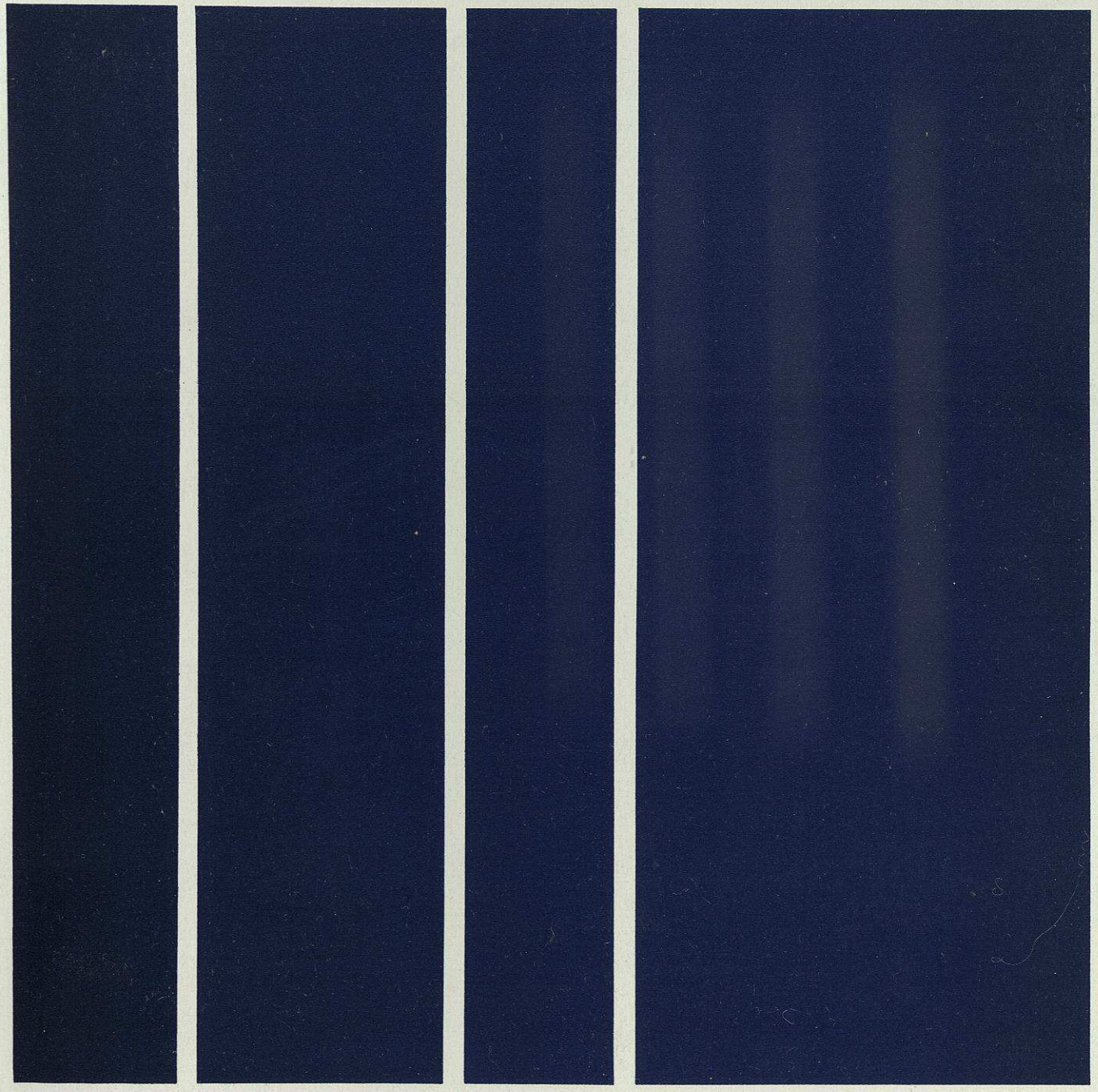
Report Series
No 11

Improving Systems' Productivity

February 1979



The Butler Cox Foundation

Abstract

Report Series
No 11

Improving Systems' Productivity

by Tony Brewer
February 1979

Both the users and the providers of information systems are already faced with the problem that the demand for systems is greater than the supply. There is already a backlog of applications waiting to be developed. This backlog will grow, due partly to a widening of the base of users and partly to an increase in the scope of information systems brought about by convergence of computing, communications and office automation.

Supply will not be increased to satisfy demand merely by increasing the resources. The productivity of those resources must be dramatically increased. This Report is concerned with the ways in which this increase in productivity might be achieved.

The main theme of the Report is that all attempts to increase systems productivity made so far have focussed too narrowly on individual aspects of system development work. We believe that the productivity problem can only be solved by taking a much broader view of the whole system life-cycle.

We discuss various approaches to improving productivity, including project management, environmental factors, data management, application packages, non-traditional system development methods and the role of the system user. Section V of the report contains a critical review of fourteen methods or tools that are claimed to improve productivity.

Finally, the Report recommends action that should be taken now to improve productivity and identifies issues that need to be considered from a strategic point of view.

The Butler Cox Foundation is a research group which examines major developments in its field – computers, telecommunications, and office automation – on behalf of subscribing members. It provides a set of 'eyes and ears' on the world for the systems departments of some of Europe's largest concerns.

The Foundation collects its information in Europe and the US, where it has offices through its associated company. It transmits its findings to members in three main ways:

- As regular *written reports,* giving detailed findings and substantiating evidence.

- Through *management conferences,* stressing the policy implications of the subjects studied for management services directors and their senior colleagues.

- Through *professional and technical seminars,* where the members' own specialist managers and technicians can meet with the Foundation research teams to review their findings in depth.

The Foundation is controlled by a Management Board upon which the members are represented. Its responsibilities include the selection of topics for research, and approval of the Foundation's annual report and accounts, showing how the subscribed research funds have been employed.

Report Series No. 11

IMPROVING SYSTEMS' PRODUCTIVITY

by Tony Brewer

February 1979

Butler Cox & Partners Limited Morley House 26 Holborn Viaduct London EC1A 2BP

TABLE OF CONTENTS

I INTRODUCTION 1

II THE PRODUCTIVITY PROBLEM 3

A Meanings of System Productivity 3
B Macro and Micro Productivity 3
C The Significance of System Maintenance 4
D User Problems 8
E Development Problems 10

III THE SYSTEM LIFE CYCLE 12

A Introduction 12
B Definition of Requirements 14
C System Design 15
D System Construction 16
E System Maintenance 16
F Alternative Approaches 17

IV. THE PRODUCTIVITY SOLUTION 19

A Terotechnology 19
B Approaches to Improving System Development Productivity 23
C Project Management 25
D Environmental Factors 28
E Data Management 29
F Application Packages 31
G Empirical System Design 32
H Role of the User 33

V SYSTEM DEVELOPMENT METHODOLOGIES 35

A Definition of Requirements 35

1. The Lancaster Approach 35
2. Structured Analysis and Design Technique 37
3. The ISDOS Project 38
4. Software Requirements Engineering Methodology (SREM) 40
5. Higher Order Software 41

B System Design 41

1. Structured Design 42
2. Hierarchy Plus Input-Process-Output (HIPO) 43
3. System Optimisation and Design Algorithm (SODA) 44

C System Construction 45

1. Jackson Structured Programming 45
2. Delta 47
3. Pseudo Code 48
4. IBM Improved Programming Technologies 48
5. AUTOGEN 49
6. ADAM 50

VI. FINDINGS AND RECOMMENDATIONS 51

A Principal Findings 51
B Action Required Now 52
C Strategic Issues 53

BIBLIOGRAPHY 56

I. INTRODUCTION

"Productivity" has been one of the most overused words in the English language during the last twelve months. For politicians and businessmen it explains the British nation's poor economic performance but suggests the means of recovery. For trades unionists it presents the magic ingredient by which every pay claim can justifiably exceed the Government's guidelines. For sociologists and media men it contains the threat of a twenty first century scenario, with the work being done by silicon chips while the bored nation sits in front of its viewdata screens.

We believe that productivity is an important concept in the context of developing information systems. For many years management services managers have been used to the idea that their productivity was improving, that they were able to supply their users with more for less. As evidence, they pointed to the dramatic fall in the cost of equipment and to the provision of better ways of doing the work. We do not dispute that this improvement has been achieved, but there are aspects of system development productivity that cause serious concern.

The first problem is the relationship between the numbers of data processing staff and the quantity of their output. The growth of the DP industry to date has been due more to an increase in the number of staff employed than to an increase in productivity. For example, in the United States the number of programmers increased by over seven times between 1960 and 1975. Their output per person increased at only 3% per annum over the same period. Between 1975 and 1985 the DP industry is expected to grow by four times. But the output from the expected number of programmers will only double if the present rate of productivity improvement is not increased. Clearly, either more programmers must be found, which seems unlikely, or a significant improvement in their productivity must be achieved.

A second problem concerns the nature and the scope of computing applications. Historically, the earliest applications were concerned with processing business data in well-structured systems, e.g. payroll, and sales analysis. Routine decision-making systems were then tackled so that computers were introduced into business operating systems, e.g. sales order processing, stock control, and production control. Now, computers are being applied to management information systems. However, one effect of the convergence of computing, communications and office automation will be to open up a whole new range of applications, and this will increase still further the demand on management services. At the same time, improvements in management education and increased sales activity by suppliers will reinforce this effect. The emphasis will change from the situation where management services staff seek to persuade reluctant users, to one where frustrated users make continual demands on over-stretched management services staff. Again, the only way to satisfy this demand will be to improve the methods of systems development.

A third problem concerns the maintenance of existing systems. Later in this Report we provide evidence that well over half the effort devoted to typical systems during their life is incurred in maintaining them after they have been implemented. It needs no mathematical skill to appreciate that the more new systems are developed the smaller will be the proportion of staff available to develop additional systems. This presents the alarming prospect that, unless a radical change occurs, eventually all systems staff will be occupied on maintenance.

A fourth problem concerns changes in unit costs. Just at the time that equipment costs are falling rapidly and offering the prize of cheap computing for all, the balance of cost is shifting from hardware to software, and the unit cost of the people who provide the software is increasing rapidity. There is a real risk that the prize may be snatched from the user's grasp. Virtually every study we have researched indicates that by 1985, 90% of the cost of data processing will be people costs, rather than equipment costs.

The final problem associated with productivity concerns not the internal problems of output or cost, but the user's point of view. This Report argues later that the provision of systems that are more effective in business terms, and more reliable in operation, is an important aspect of productivity.

The purposes of this Report are to look deeply at the concept of productivity as it applies to the development and use of information systems, to identify the potential sources of productivity improvement, to review critically the most important new methodologies that are available for developing systems, and to make practical recommendations for our readers.

Section II discusses the productivity problem in greater detail. It provides a definition of system productivity that focusses attention on the major cause for concern, namely system maintenance.

Section III discusses the various stages in the life-cycle of an information system from a productivity point of view. It suggests that even the most recent new methodologies may already have been overtaken by events.

Section IV stands back from the systems context to see whether other industries have already solved an analogous problem, and discusses the available approaches to improving productivity.

Section V contains a critical review of the most important system development methodologies. It indicates their origin, the aspect of the development process to which they are applied, their strengths and weaknesses, and their importance to the typical management services department.

Section VI draws the conclusions from this research, indicates what can be done now to improve productivity, and suggests the areas of interest for improvements in the future.

II. THE PRODUCTIVITY PROBLEM

A Meanings of System Productivity

The conventional industrial (and now political) meaning of productivity is output per person. Thus, when comparisons are made between the productivity of the British Ford worker and his German counterpart, the measure is vehicle units per man-day. While this simple definition may be easily understood, and thus may be useful in a production environment or, in our context, programming shop environment, it suffers from the basic weakness that it does not include any cost factor. It may be possible to increase productivity, in this sense, either by using higher level languages or by providing on-line programming facilities. If the result is that the job is done with fewer people but at higher cost, the productivity increase may well be worthless.

An alternative definition of productivity is output per unit cost of system development. This is the economists' definition and it overcomes the weakness of the first alternative. In a production environment, where the cost of labour is directly comparable with, say, the cost of fixed assets, this definition is useful. But, in the system development environment much of the output is not cost related. Activities like carrying out a system survey or designing a program suite depend far more on political or creative skills than upon the cost incurred. It is not possible to reduce the cost of system development while holding the quality of the product constant, because it is virtually impossible to assess the quality.

Both of the definitions discussed above concentrate on the productivity of the production unit and ignore the user. The productivity of the user is highly significant in the context of overall system productivity. To say this is more than merely a reflection of the current fashion to consider the consumer.

We believe that the only useful definition of system productivity is output per unit cost of the system life-cycle. Productivity means satisfying the requirements of the system, throughout its life, at minimum cost. This definition recognises that productivity can be increased either by increasing the development cost, if this results, for example, in greater reliability of a computer system, or by decreasing the development cost, if what the user really needs is a simple system that satisfies only 80% of his requirements.

B Macro and Micro Productivity

The three alternative definitions given above illustrate the difference between macro- and micro-productivity. Micro-productivity is concerned with the utilisation of individual system components, whereas macro-productivity is concerned with the system as a whole. During the history of computing there has been a gradual change of focus from micro-productivity, expressed first as machine utilisation and more recently as programmer productivity, to macro-productivity, where machine efficiency or programmer productivity are of much less importance than the effectiveness and the reliability of the system from the user's point of view.

It is not difficult to appreciate the reasons for this trend. Initially, when the fixed cost of a computer installation was high but the marginal cost of developing and running an additional program was relatively low, it was essential to achieve high machine utilisation if computing was to be cost justified. These circumstances lead directly to the validity of Grosch's Law that the efficiency of computing was proportional to the square of the size of the computer. The user was truly grateful for whatever he was able to receive.

Gradually, as equipment costs fell and the user found his voice, quality became important and the spotlight shone on the supposed source of all the errors, programming. Consequently, the emphasis over the last ten years has been on improving programming methods. But this attention has merely repeated the search for micro-productivity while ignoring the much greater benefits to be gained from improvements at the macro-level. We are now concerned with a trade-off between the various micro-productivities of the technical aspects of the system and the macro-productivity of the whole system. The key is to realise that most of the costs and most of the procedures are associated with operation and maintenance of the system rather than with its development. It is likely that operations and maintenance requirements will increase as new applications emerge and as more complex systems are developed. Thus, greater attention must be paid to user requirements and the method of operation, so extending the view of system development staff beyond completion of the development project.

C The Significance of System Maintenance

Typically, systems managers have been concerned with system development as an end in itself. A development project ends when the system is implemented and responsibility passes to the user manager. Subsequent attention to the system falls into the realm of correction or enhancement. While these are recognised as functions to be performed they are not generally regarded as significant in the context of the development project. This view is supported by the analyses that are frequently published, which show the breakdown of development cost between the various development stages.

However, as soon as system productivity is defined with respect to the total life-cycle cost, the significance of system maintenance becomes very obvious. Figure 1 shows the results of the Hoskyns survey of 905 British installations. Already about one quarter of these installations are devoting more than 50% of their software effort to system maintenance. Figure 2 shows an analysis of DP costs from 1955 to 1985. This indicates that by 1985 over 60% of DP costs will be incurred on system maintenance. In his paper on software engineering, Boehm (9) claims that 70% of software life-cycle costs are devoted to maintenance and Mills (49) claims that 75% of programmers are already concerned with maintenance.

Whatever the definition and whatever the percentage claimed, it is clear from these figures that system maintenance is not a cost that can be ignored as someone else's responsibility. It is true that the percentage of an installation's cost devoted to system maintenance represents the summation of maintenance on individual systems developed over a period, and that the ratio of maintenance to development cost will depend on the complexity and longevity of the system. There is, however, sufficient evidence to support the claim that maintenance is responsible for over 50% of the cost of a typical system.

Figure 3 shows a typical breakdown of system development costs, together with the same figures expressed as percentages of the total life cycle cost. This illustrates the dramatic change in the cost profile when maintenance cost is taken into account. It also shows the very low expenditure on getting the system requirements correctly identified at the beginning, compared with the very high cost of putting them right at the end.

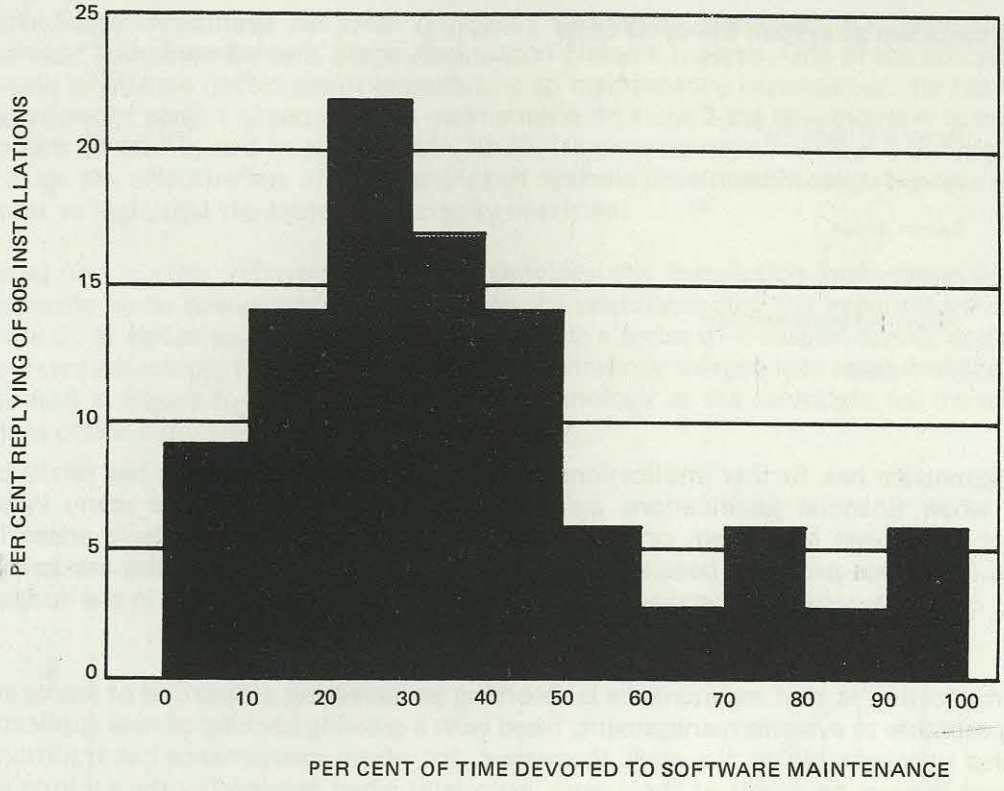


Figure 1 Hoskyns Survey of 905 British installations
Source: Tom Gilb, Software Metrics, Winthrop, 1976

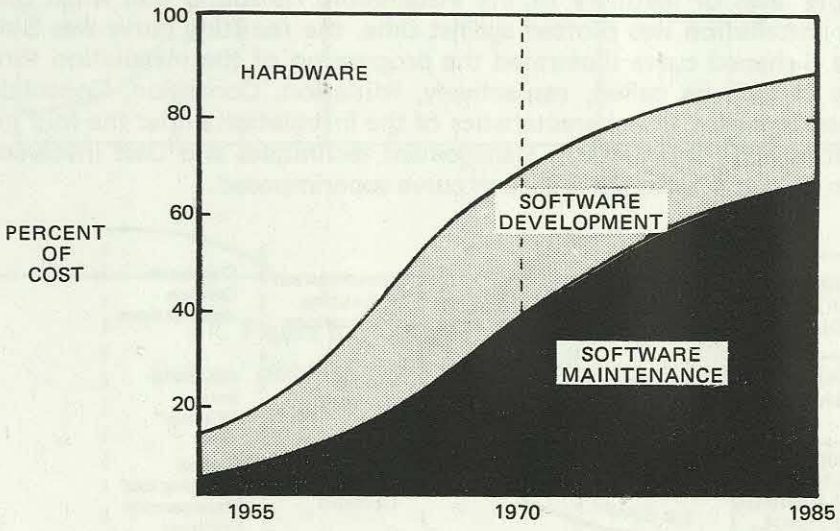


Figure 2. Analysis of DP costs, 1955-1985
Source: Boehm, 'Software Engineering'
IEEE Transactions on Computers, December 1976

Figure 3 Breakdown of system life-cycle costs

	Without maintenance %	With maintenance %
Survey and feasibility	5	2
Analysis and specification	10	4
System design	10	4
Programming	40	16
Testing and implementation	35	14
Maintenance	–	60
	100	100

System maintenance has further implications. First, the cost of maintenance has rarely been considered when financial justifications are prepared. If it had been, either many systems would never have been developed, or a demand for higher quality would have arisen long before now. There has probably been a tacit understanding between systems and user managements that, once the system is installed, the costs of maintenance can be lost in the budgeting procedures!

A second implication is that maintenance is absorbing an increasing proportion of scarce staff. This is unacceptable to systems management, faced with a growing backlog of new applications and it is also unacceptable to the staff themselves, for whom maintenance has traditionally been the least interesting aspect of their work, especially when tempted by the exciting alternatives of work on databases, communications systems, minis, or other glamorous projects.

A third implication is that maintenance work is very expensive. Boehm (9) quotes figures for an aircraft computer of $75 per instruction for software development and $4000 per instruction for maintenance. He also claims that the cost of correcting an error rises exponentially with the system development stage in which it is detected.

A fourth implication is the relationship, suggested by Nolan (51), between the level of DP expenditure and the level of maturity of the installation. He found that when the total DP expenditure in an installation was plotted against time, the resulting curve was S-shaped. He suggested that the S-shaped curve illustrated the progression of the installation through four stages of maturity, which he called, respectively, Initiation, Contagion, Consolidation and Integration. He then analysed the characteristics of the installation under the four headings of Applications portfolio, DP organisation, Management techniques and User involvement. This analysis is shown in Figure 4, with the S-shaped curve superimposed.

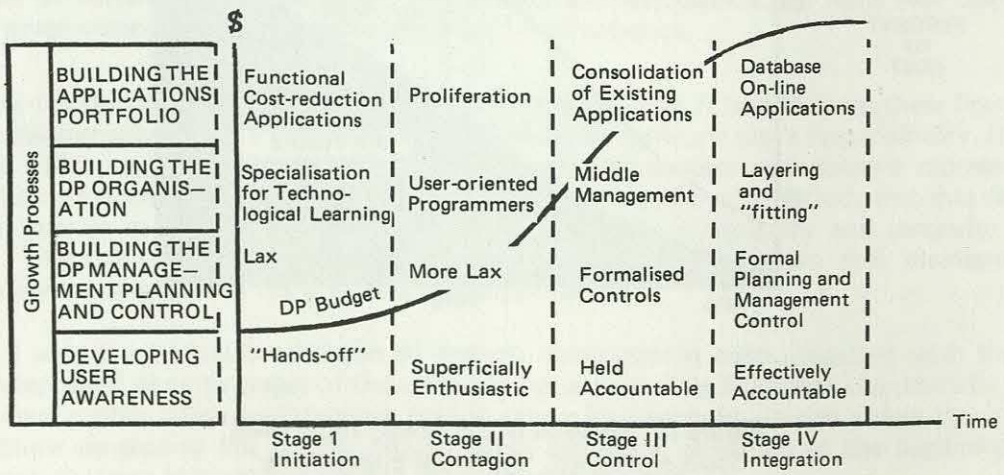


Figure 4 Nolan's Stage Hypothesis

Source: Nolan, 'Managing the Computer Resource: A Stage Hypothesis'

Nolan's Stage Hypothesis has been developed further by Robinson (62), who has based an assessment technique he calls Stage Audit upon Nolan's analysis. One of the criteria he uses is the ratio of system development expenditure to maintenance expenditure. He has found that installations in stage 1 spend little on maintenance. In stage 2 the development to maintenance ratio falls to 60:40, and in stage 3 it is 30:70. He gives no ratio for stage 4 but argues that by this stage the effectiveness of maintenance of systems developed in stage 1 is now so low, and the cost so high, that the system is normally rewritten.

Because this system redevelopment also provides the installation with the opportunity to incorporate some new aspects of technology, he postulates that the expenditure curve is not a single S, as Nolan suggested. He claims that it is a series of S-shaped curves, each one based on a new technology, with stage 4 of the first technology merged into stage 1 of the second, as illustrated in Figure 5. He suggests database technology as the candidate for the second, with perhaps office automation as the third technology.

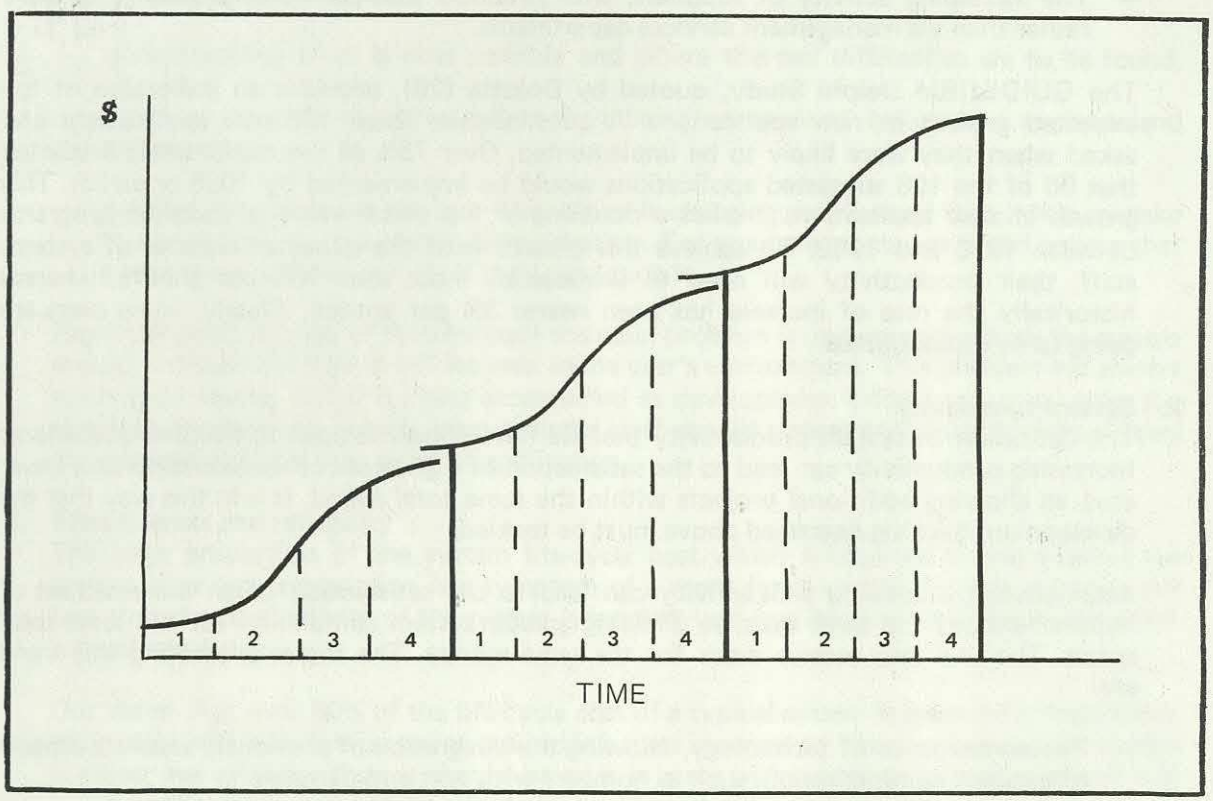


Figure 5 Repetitive Stages of Growth

Source: Robinson, 'Beyond the four stages: what next?'

It is clear from this analysis that systems management has not solved the problem of system maintenance. Because it is not possible to satisfy the changing requirements of the business with existing systems, and because of the temptation of new technologies, it is generally easier to rewrite the system than to keep it going. This behaviour obviously limits the life of the system, so that the productivity of the original development is not as great as it might have been.

D User Problems

1. Applications backlog

In the Introduction we made the general point that the demand for new systems will be much greater than the expected supply. We said that this demand will comprise not only traditional data processing and business operating systems but also new applications associated with non-routine planning and decision making plus administrative functions based on office automation.

The forces creating this demand are:

- The falling cost of equipment, making new applications more easily justified.
- The convergence of technology, opening up new application areas.
- The increasing acceptance by users of the value and simplicity of automated systems.
- The increasing activity of suppliers, with products that can be sold directly to users rather than via management services departments.

The GUIDE/IBM Delphi Study, quoted by Dolotta (26), provides an indication of the expected growth in new applications. A questionnaire listed 108 new applications and asked when they were likely to be implemented. Over 75% of the respondents predicted that 96 of the 108 suggested applications would be implemented by 1985 or earlier. This growth in new applications implies a doubling in the dollar value of installed programs between 1975 and 1985. To achieve this growth with the expected number of systems staff, their productivity will need to increase by more than 10% per annum, whereas historically the rate of increase has been nearer 3% per annum. Clearly, some users are going to be disappointed.

2. System complexity

The definition of system productivity that we have chosen is open to two interpretations. Increasing productivity can lead to the satisfaction of a given set of requirements at a lower cost, so allowing additional projects within the same total spend. It is in this way that the development backlog described above must be tackled.

Alternatively, increasing productivity can lead to the satisfaction of an increased set of requirements at the same cost, so allowing greater system complexity for the same total spend. The user can receive more for the same money. The forces promoting this trend are:

- The convergence of technology, allowing the integration of previously separate aspects of business administration, such as numbers with text and pictures.
- The move away from the standard data processing and routine decision-making systems into the areas of higher-level business systems concerned with non-routine decision-making and information storage and retrieval.

The implications of convergence are fully discussed in Report No. 5, The Convergence of Technologies. One conclusion is that the managing of system complexity will be a major feature of system development over the next few years. Until now, suppliers of computers have attempted to make their products more easy to use by providing massive and highly-complex internal operating systems. It is easy to imagine the opportunities there are for further complexity. For example, with a distributed processing system that had multi-function terminals scattered over a large organisation, the operating system to control the network might be even larger and more complex than today. Over the next

decade one of the main criteria for assessing computer suppliers will be their ability to provide the facilities that users need without constraining their business activities unduly.

Users are faced with the dilemma that, just at the time that improvements in productivity are opening up new application areas, inability to control the technical complexity will deny them the benefits.

The solving of this problem may itself present a further trap. Suppliers will be seeking to make their products so simple to use that the users themselves can construct their own solutions without calling on the services of systems staff. The result could be that, at the same time as it becomes technically possible to take a "systems" view across the whole organisation, responsibility for systems development may be passing back to those with a vested interest in maintaining the status quo.

3. **Communicating with users**
One of the problems that arises from the increase in system complexity is the increased difficulty of communicating between users and system staff. From the user's point of view the problems include:

- Understanding what is now possible and where the real difficulties are to be found.
- Thinking in systems terms rather than being constrained by current methods and organisation.
- Appreciating that information technology is changing rapidly and that it is a mistake to cling too long to a particular generation. For example, some users still believe that a record can only be correct if they can touch it.

From the point of view of systems staff the main problem is understanding how the system should function and how it will be used in the user's environment. This problem has always existed, of course, but it is being accentuated as development work moves away from the standard applications (which most systems staff should understand) into the higher level and probably unique new types of application.

4. **Effectiveness and reliability**
The large proportion of the system life-cycle cost which is incurred on maintaining the system after implementation is a symptom of a more fundamental problem, namely, the fact that the productivity of the system is seriously reduced because the system is ineffective and unreliable.

Our claim that over 50% of the life-cycle cost of a typical system is spent after implementation does not take into account the hidden costs incurred by the user because the system is either less effective than it should be or is in error without the user being aware of it or is inoperative while it is being corrected.

It is surely significant that most of the data on the occurrence of errors and the costs of maintenance have been collected from large defence or military applications where the reliability and effectiveness of the system is of crucial importance. By contrast, for most commercial systems the general levels of control of quality and performance are much lower.

If our definition of system productivity is extended – as it should be – to include user costs as well as software and operating costs, the task of improving effectiveness and reliability is very much a productivity problem.

5. **Standard solutions**
One reaction to the problem of controlling system complexity, referred to above, will be

for suppliers to offer standard, packaged solutions to an increasingly wide range of standard applications. The advantages to the user are that he can know exactly what he is implementing, he can obtain it off the shelf at a known cost. The disadvantage is that the package may not provide an exact solution to the user's problem. He must decide whether, and for how long, the benefits of increased control over the system are greater than the shortcomings of a sub-optimal solution. Bearing in mind Nolan's theory of the stages of installation maturity, described in II.C, above, packaged solutions probably have the greatest attraction for first-time, stage 1, users.

E Development Problems

1. Polarisation

Until now the typical systems analyst has been a generalist who has neither possessed, nor needed, strong technical skills. Employers have looked more for experience in the specific applications areas and political skill with handling the systems/user relationship. This picture is likely to change. As databases and distributed networks assume greater importance, some systems staff will require high levels of specific technical skills. The remaining staff, who might be called applications analysts, will need to know how to use the technical facilities, but they will not need to know how they work. However, when they set out to tackle higher level business systems and problems of office automation, they will need a much greater appreciation of business management, sociology, industrial relations and ergonomics than at present. Also, their experience in the standard application areas will be of little value in the new territory.

This polarisation of skills is already taking place amongst programming staff, with functional specialisation by systems programmers and applications programmers. At management level, there may not be an actual division of duties, but the range of skills that has to be controlled will be much wider than is typical today. The systems manager will not only be hampered by his lack of experience in the new skills, he will also be misled by his obsolete experience in the old skills. We referred in Section II.D, above, to the problem of users clinging to out-of-date methods of computing, because those methods are something they understand and have experience of. The same problems can be observed frequently, but with more serious consequences, amongst many of the systems managers and management services managers of today.

2. The gestation period

One consequence of the low productivity of the present methods of system development is that the elapsed time from user request to successful operation is very long. This elapsed time is obviously a function of the size of the system, but many months is typical and several years is not unusual. This elapsed time may be reduced by carrying out some substages in parallel, but this must be planned and controlled very carefully or it may result in inconsistencies and require more time rather than less.

In order to reduce the risk associated with the development time and cost, project control methods are used. They often appear to increase this development time rather than shorten it.

Attempts to shorten the time by drafting in more staff are also doomed to failure. Putnam (58) has suggested that systems have a natural development time which is directly linked with their inherent difficulty. Reducing the development time by increasing the number of staff causes the difficulty factor to increase rapidly and is almost always counter-productive. Thus, it is much more than twice as difficult to complete a 12 man-month project with four staff in three months than it is with two staff in six months.

A further consequence of Putnam's theory is that the traditional device that systems

managers resort to, of rewriting the system when they can no longer control its maintenance, may no longer be available. Systems that have been in use for several years may have been enhanced and modified to such a degree of complexity that the natural development time for a rewrite would be unacceptable to the users. In these circumstances the rate of change of the business has become greater than the rate of development of the system. System managers can no longer offer a solution to the business problem. The only escape from this impasse is to replace the development methods upon which Putnam's theory is based by others that have much greater productivity.

3. **Higher level languages**
   One of the main reasons for increases in system development productivity in the past has been the provision of new, procedure-orientated languages, such as COBOL, FORTRAN and PL/I. Dolotta (26) argues that the provision of new languages is no longer likely to make a significant impact on programmer productivity. Improvements in existing languages can be achieved which might improve programming productivity by a factor of two, while claims are made for new languages of improvements over COBOL of a factor of five, but at least a ten-fold improvement is required.

   However, the real issue is not the availability of new languages but rather the reluctance of installations to make use of them. For example, COBOL became available in the early 1960s, but it took eight years or so to gain widespread acceptance. No fundamentally new language has emerged and received widespread approval in the last 15 years.

Clearly, any solution to these productivity problems, as they are perceived from both a user's and a system development point of view, will not be achieved by improvements in micro-productivity. A major improvement at the macro-productivity level is required.

A Introduction

The discussion and recommendations in this Report are based on the belief that improvements in system productivity will not be obtained by increasing the output per person at particular stages of the development process, what we have termed micro-productivity, but only by considering the whole system life-cycle and seeking improvements in macro-productivity.

It is therefore appropriate at this stage to take a detailed look at a typical system life-cycle from a productivity point of view.

In recent years attention has been focussed on the nature of the life-cycle. Studies have shown that there is a distinct pattern in the life of a piece of software, from its conception in terms of user requirements, progressing to its routine operational use, through a period of enhancement which ultimately leads to the end of the viability of the system.

Many writers have analysed the life-cycle and have given various names to the stages that they have identified. Boehm (9) describes the software life-cycle as consisting of seven sequential stages, each one of which feeds back to the previous one in an interactive process of correction and refinement, as shown in Figure 6. In Report No. 8, Project Management, the method of project management was based on the natural life-cycle. The Report recognised seven stages in the system development process.

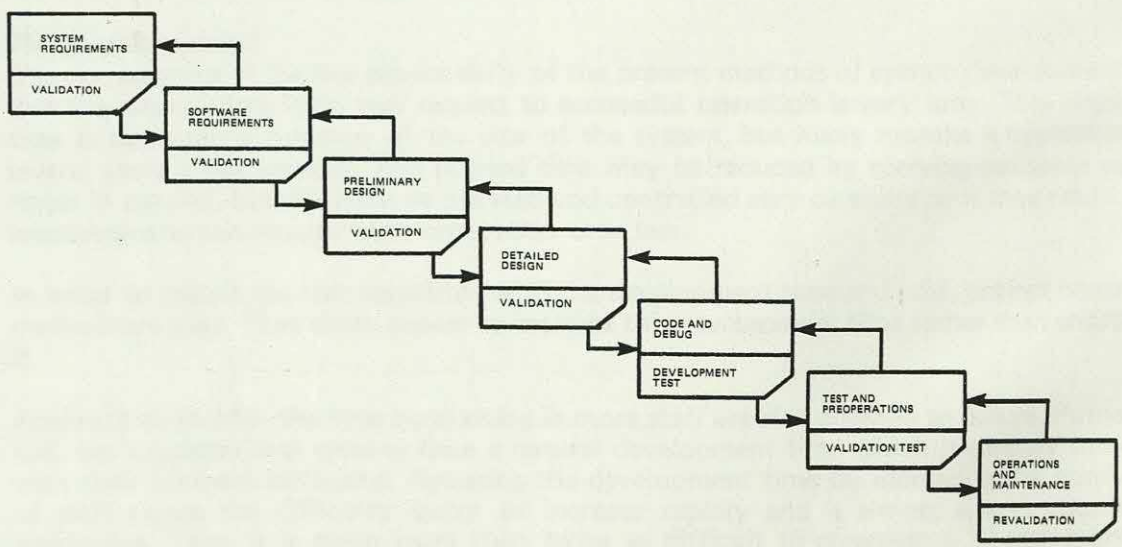


Figure 6 Boehm's Seven Stages of the Software Life-Cycle
Source: Boehm, 'Software engineering'.

For the purpose of this discussion we shall take a simplified view of the life-cycle and consider it in four stages:

1. Definition of requirements
   The user's problem or opportunity is identified and the requirements of the new system are defined.

2. System design
   The means by which the system requirements will be satisfied are designed in a logical, as opposed to a physical, way.

3. System construction
   The logical design is realised in a physical way, by writing programs for a specified computer and by designing clerical procedures for 'real' people. The new system is implemented.

4. System maintenance
   The new system is maintained for the remainder of its operational life, errors being corrected and new requirements being incorporated.

It should be emphasised that the process of developing a system involves building a series of models of the system at increasing levels of abstraction, from English language description through to object code. The problems of defining, constructing, testing and maintaining the final operational system also apply to the previous models in the process. If, for example, the system specification cannot be constructed, tested and maintained correctly, there is little chance that the final system will be.

Two distinct, but often confused, types of activity are involved in developing a system, and the term "structure" is often applied to both of them. System structure is concerned with the logical design of the system, that is, with the way in which the requirements are satisfied. Project structure is concerned with the management of the project, that is, with the way in which it is broken down into separately planned and controlled stages. There is a definite interaction between the two, but to ignore the distinction is equivalent to assuming that practising one will automatically achieve the other. In fact, both are essential.

We referred in Section II.E, above, to Putnam's analysis of system life-cycles. In particular, we referred to his conclusion that systems have a natural development time, and that attempts to shorten this by adding staff merely increase the inherent difficulty. At this point, it is worth describing his work in greater detail.

From an analysis of more than 100 large systems, with development times of two to five years and development efforts of 25 to 1000 man-years, he found that the typical man-power against time curve was as shown in Figure 7. He claims that:

1. The high point on the manpower curve corresponds closely to the development time.

2. The overall manpower curve is the summation of a series of sub-cycles which are non-symmetrical, and all have long tails. This explains the "90% complete" syndrome, 90% of the work being completed in 66% of the time.

3. The manpower curve can be represented by an equation with three parameters:

   - Total life-cycle effort (man years).
   - Development time.
   - Difficulty factor.

4. The difficulty factor = $\frac{\text{life-cycle effort}}{(\text{development time})^2}$

   Thus, the difficulty factor for a 12 man-year project which is completed in six years is 0.3. If, by contrast, the project is completed in three years the difficulty factor is 1.3, more than four times as great.

5. The development cost is approximately equal to 40% of the life-cycle cost.

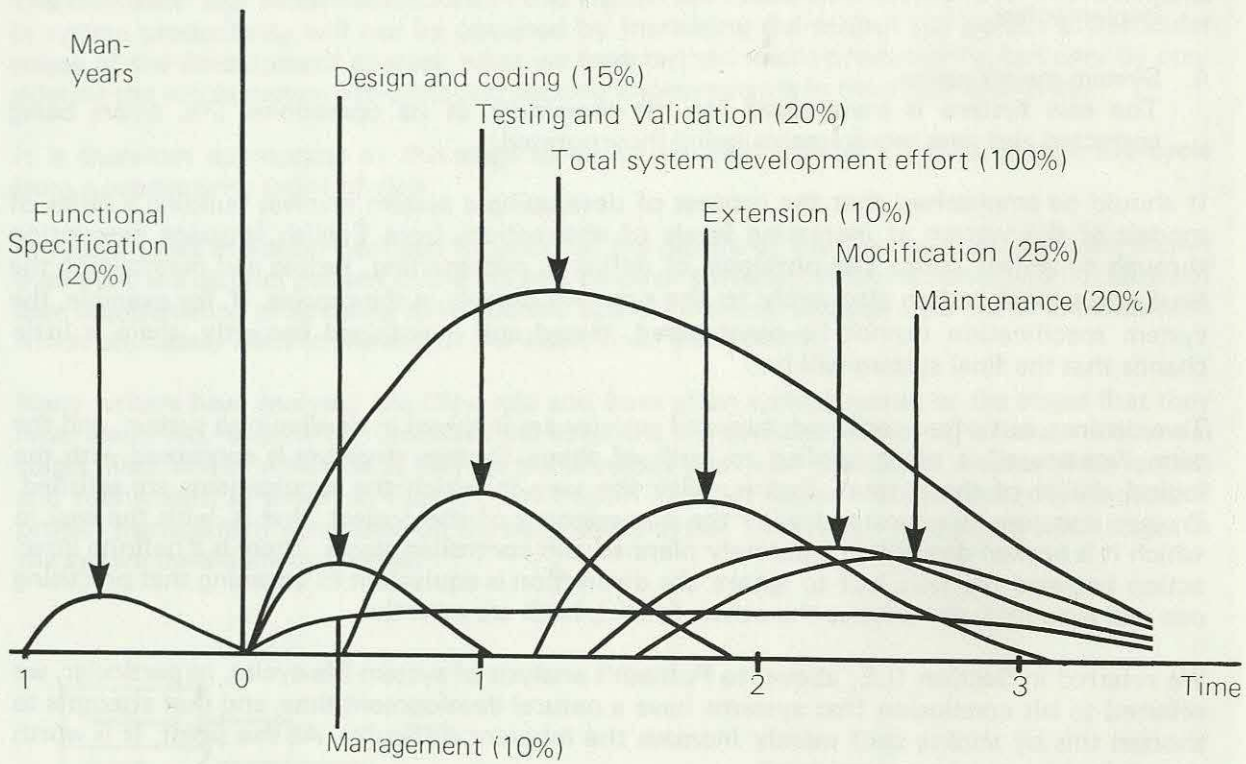


Figure 7 Putnam's Life-cycle Model
Source: Putnam, The Influence of the time-difficulty factor in large-scale software development

B Definition of Requirements

From the point of view of achieving an improvement in macro-productivity, the requirements definition stage is probably the most important one in the system life-cycle. This is because most of the avoidable problems that arise in the system maintenance stage can be traced back to this first stage.

To achieve a correct definition of requirements is difficult and so this aspect is frequently neglected. There are several reasons for this:

1. Most projects start badly, with imprecise terms of reference, poor understanding of the responsibilities of both user and systems staff, and preconceptions about what is required.

2. Most so-called "system surveys" generally collect opinions, rather than reliable facts. Thorough fact-finding is difficult, and it should not be left to inexperienced staff.

3. System problems are frequently not hard and precise but soft and vague, varying with the point of view. This is especially true of higher-level business systems. Defining system requirements is then necessarily a matter of opinion, and opinions change with time.

4. To a large extent the definition of the problem depends on the alternative solutions available. Different solutions solve different problems, so feedback between discussion of feasible solutions and problem definition is essential.

5. Even if reliable facts can be obtained, there are no methods of analysis in general use that can help reveal the system requirements.

It is against the background of these problems that we discuss the relevant new methodologies in Section V.

C System Design

Our definition of the system design stage, as concerned with the logical rather than the physical solution of the user's problem, is to some extent artificial and arbitrary. We have already indicated that there needs to be feedback between the logical solution and the definition of the problem. Similarly, no system designer can exist in logical purity — he must always be aware of the real physical world of programming languages and processing methods. Consequently, the interface between design and construction is also indistinct.

From a productivity point of view the two main problems in the design stage are consistency and quality, and traditionally both are very difficult to assess.

Consistency is important because in the series of activities that are required to convert a definition of requirements into a system specification, it is easy to omit individual requirements, and any omissions are difficult to detect. The result is a working system which, although apparently free from errors, does not fully satisfy the user's requirements. The work of rectifying errors of omission is often very difficult and costly. Methods are needed that help ensure that the models of the system that are created during the development process are all consistent.

Quality of design is important because, where there are two error-free designs, the good one is preferable to the bad. Equally important is the need to achieve an error-free specification. Analyses of errors in implemented systems show that design errors (those errors that require a change to the system specification) account for a high proportion of the total.

A further aspect of quality of design is the extent to which the system is designed for easy maintenance. This means being able to find errors quickly and to rectify them without inducing additional errors. It also means being able to enhance the system, as the business requirements change, without distorting the basic structure of the system. In this way, both aspects of maintenance can be made more easy, with a consequent increase in productivity.

One of the fundamental problems of the design stage is to decide on what basis the design is to be conceived, given that, in addition to the traditional approach of process-orientated design, methods based on data structures have now been introduced. Process-orientated design usually relies on functional decomposition to reveal the key activities in the system and their inter-relationships. The weakness of this method is that it is difficult to judge the consistency and quality of the result. Data-orientated design, of which the Jackson design method is probably the best known example, is based on designing the system structure to mirror the

data structure. The functional activities are then handled once the design strategy has been determined. The advocates of this type of method claim that it is not dependent on creative inspiration, and that the result is testable against the original data structures. The weakness of this method is that it is more appropriate for highly-structured systems, using batch processing, than for transaction-processing systems. However, data-orientated methods do indicate a move in the right direction – towards higher quality and productivity.

D System Construction

Most of the effort that has been directed at increasing productivity has been concentrated in the system construction stage, and particularly at programming. One reason is that programmers form the largest group of staff in a typical system development department, they are difficult to recruit, they are expensive to train, and they are easily lost, so it makes sense to extract the maximum value from them. A second reason is the belief that most system errors are in fact programming errors. Analyses by IBM and TRW suggest that this may not be true, but even if it was the case the cost of correcting programming errors is far lower than the cost of design errors.

We believe that the emphasis on programmer productivity is misplaced and represents a classic example of "missing the wood for the trees". It is true that some progress has been achieved, at the micro-productivity level in increasing the output per programmer and in de-skilling some of the programming tasks. However, there is evidence from both the UK (50) and the USA (35) that the improved methods are not widely used, and that, in any case, they do not address the macro-productivity problem.

Methods are required that:

- Significantly reduce the elapsed time that is required for the system construction stage.
- Reduce the probability of errors in the finished system.
- Improve the maintainability of the finished system.
- Do not detract from the job satisfaction of the staff who do the work.

As Boehm has said (36), "the key to good software design lies in getting the best out of good people and in structuring the job so that less-good people can still make a positive contribution".

In Report No.8, Project Management, we quoted examples of reducing the elapsed programming times for the most frequent standard jobs (such as file updates, input edits and report printing) by up to 50%. This is the kind of benefit that is required.

E System Maintenance

One of the themes of this Report is our belief that the large proportion of the life-cycle cost spent on system maintenance is the main symptom of low productivity. Looking closely at the maintenance stage may therefore reveal the causes.

There are three types of activity normally associated with system maintenance:

1. System errors, which arise because the system as implemented does not satisfy the definition of user requirements.

2. System errors, which arise because the definition of user requirements is not complete or is not correct.

3. System enhancements, which arise because the user's requirements change.

Ideally, maintenance should be concerned only with system enhancements. Therefore, the objectives for improving productivity at the maintenance stage should be, first, as far as possible to remove the need for it and, second, to ensure that maintenance activity does not make the subsequent operation or maintenance of the system more difficult and more costly.

Progress towards the first objective can be achieved only by directing attention at earlier stages in the life-cycle. If methods can be found for correctly identifying the true user's requirements – and not merely the analyst's perception of them – and for ensuring that no errors are introduced at succeeding stages of the development process, then the need to put the system right after it is implemented will be greatly reduced. This will require a change in the shape of the system cost profile, but additional cost earlier in the cycle will be handsomely repaid by reduced cost later.

The difficulty with the second objective, of reducing the damage caused by maintenance activity, is that repeated corrections and modifications tend to increase the complexity of the system and to obscure its original structure. Consequently, further maintenance is made more difficult, and the risk of unreliable operation is increased. How many systems managers wince when their systems analyst incidentally announces that he has to write a quickie for the end-of-year stock run? No doubt the quickie will be correct but what will follow in its wake?

This problem has been quantified by Davis (38), who found that a modification to a small portion of operating system software can be expected to induce ten new errors. Given that correcting errors in operational software costs ten times more than correcting them during design or coding, the consequences are serious.

It is clear that, although system development and maintenance are generally regarded as separate functions, they are necessarily closely linked. To achieve better maintenance requires a change from optimising development around cost and schedule criteria to optimising around maintenance criteria. Factors such as good structure, testability, and good documentation require greater emphasis.

F Alternative Approaches

In this Section we have been discussing the traditional stages in the system life-cycle from a productivity point of view, in order to establish some criteria against which the methodologies described in Section V may be judged. However, there is the possibility that the traditional system development process may be replaced by alternative methods. If this occurs, research directed at the traditional process may prove to be of little more than academic interest.

We have already mentioned the growth of structured design methods, based on data-orientated rather than process-orientated principles. These are discussed in detail in Section V.B. The use of data analysis at the requirements definition stage, of structured design methods at the system design stage, and of database management systems at the system construction stage could radically change the pattern of system development. A great deal of effort and technical

skill would be required to create the database and the communications network, but the provision of applications programs might become so easy and cheap that no maintenance would be carried out. It would be simpler to discard the inadequate programs and develop new ones.

It is significant that, in describing the facility of externally described data in their recently announced System 38, IBM states "the field definitions are contained and maintained in the database. This function eliminates the requirement to define the data fields in each program, provides more consistent definitions, reduces the database maintenance effort and reduces the amount of changes required to programs when changes are made to the database".

IV. THE PRODUCTIVITY SOLUTION

We have stressed earlier the theme that improvements in productivity, in the amounts needed to handle the likely volume and complexity of systems work in the future, are unlikely to be achieved by a narrowly focussed examination of how individual workers in the system development process carry out their tasks. The desired improvement will be achieved only by cutting away large volumes of non-productive work so that those involved can be re-allocated to productive work. No doubt this theme has a familiar ring. Therefore, before we discuss approaches to improving productivity in the systems field, we believe it is instructive to look outside this field.

A Terotechnology

As part of its drive to improve the productivity of British industry the UK Department of Industry has promoted the study of terotechnology to focus attention on the productivity of physical assets.

Terotechnology is defined as "a combination of management, engineering, financial and other practices, applied to physical assets in pursuit of economic life-cycle costs" (24, 25). The technique is already well established for military procurement, and, like operations research, it is now spreading into the civil, general management field.

The approach recognises that profits are maximised by minimising the life-cycle cost of assets, given that the life-cycle cost is often many times the initial purchase cost. Typically it includes the following cost elements:

- Specification and design costs.
- Acquisition costs.
- Installation and commissioning costs.
- Operating costs (labour, indirect materials, tools, fixtures, overheads, lost production, low utilisation, poor quality, poor reliability).
- Disposal costs.

Terotechnology postulates a four-step method of life-cycle cost analysis.

1. Defining the cost structure to be used. This cost structure is normally based on the phases in the life-cycle of the asset (e.g. research and development, investment, operation and support, disposal) but it is also influenced by the required breadth and depth of the analysis. Good definition allows the trade-offs between the various phases to be recognised early.

2. Defining the cost elements. Within the overall cost structure the various individual cost elements are then identified, e.g. operation and support might include the costs of spare parts, preventive maintenance, overhauls, training, fuel, power services, etc.

3. Developing cost estimating relationships. Means must then be found of estimating the likely costs of the cost elements for the options that are being evaluated. Frequently this involves forecasting on the basis of historical data, and so emphasises the importance of collecting the raw operating data in the first place.

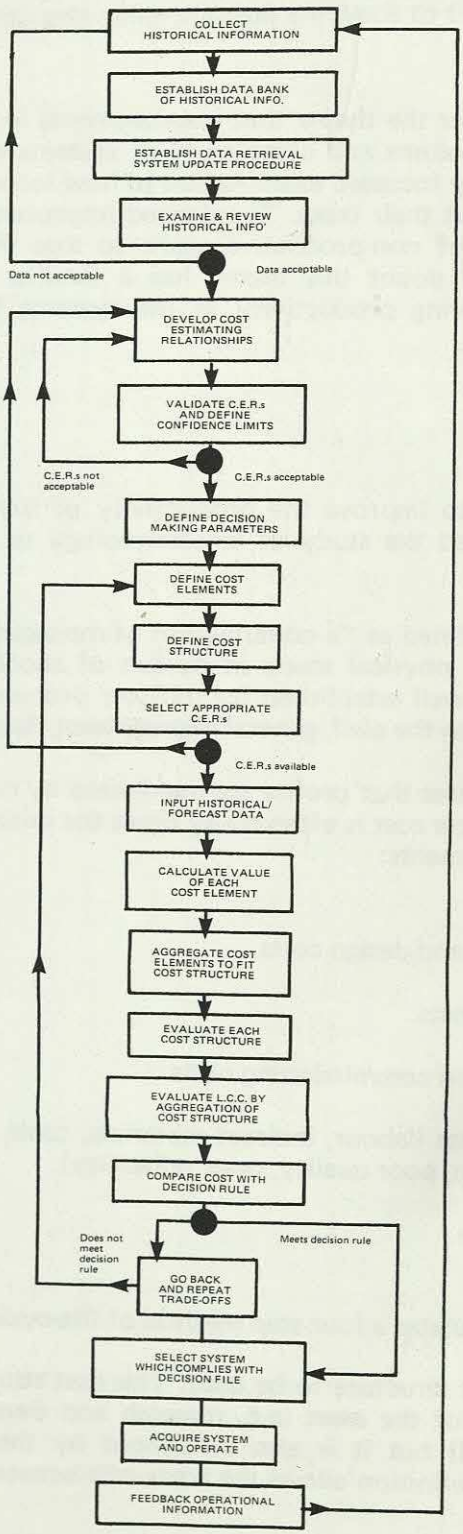


Figure 8 Terotechnology life-cycle cost analysis methodology

Source: Harvey, 'Life-cycle costing, a review of the technique', Management Accounting, October 1976.

4. **Carrying out the evaluation. Depending upon the number of options and the complexity of the cost estimating relationships, either computer or manual methods may be used at this stage. Inflation factors should be applied to the cost elements, and the cash flow should be discounted.**

PHYSICAL ASSET	INFORMATION SYSTEM
Research & Development costs	Analysis & Design costs
specification	system survey
design	system analysis
development of prototypes	system specification
testing	
Investment costs	Construction costs
installation	program writing and testing
commissioning	clerical procedures
tools and fixtures	system testing
spares stocks	operator and user manuals
operator training	operator and user training
site costs	file conversion
Operation & support costs	Operation & support costs
replacement spares	computer time
preventive maintenance	operator and user time
breakdowns	consumables
overhauls	communications
training	environmental expenses
fuel and power	erroneous output
data collection	error correction (user and systems staff)
labour	enhancements
lost production	
low utilisation	
poor quality	
poor reliability	

Figure 9. Life-cycle cost structures

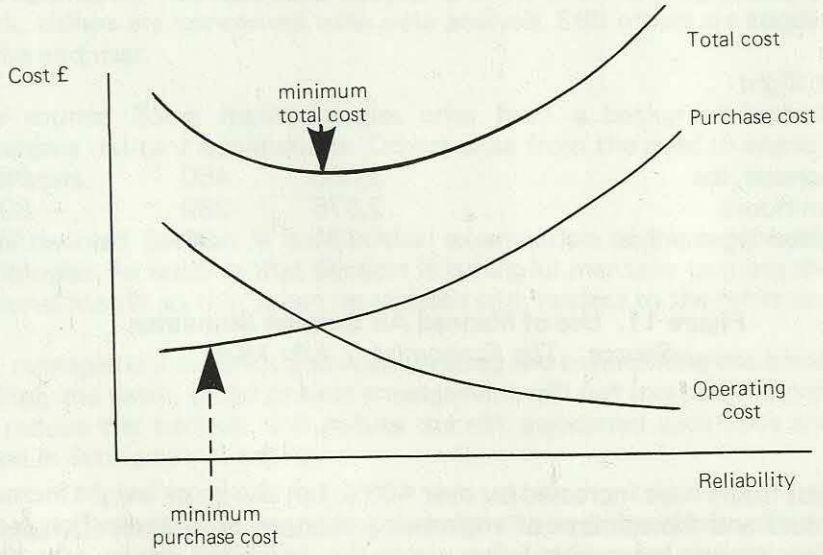


Figure 10. The cost of asset ownership
Source: "Life-cycle costing in the management of assets", HMSO, London

Figure 8 shows one of several methodologies that have been proposed for terotechnology analysis. Figure 9 shows a typical life-cycle cost structure for a physical asset, alongside the corresponding cost structure for an information system. Figure 10 shows the way in which the graphs for purchase costs and operating costs vary with increasing asset reliability, and illustrates how the minimum total cost does not coincide with the minimum asset cost.

Examples of the application of terotechnology include:

1. The Greater London Council (GLC) 'Island Block' of offices, situated at the south end of Westminster Bridge. From the early stages of its design the life-cycle cost of the building was considered. The result was that, with little increase in capital cost, the operation and maintenance costs of the building are greatly reduced.

   e.g. Mechanical and electrical maintenance:

Island Block	1.37% of capital cost p.a.
Similar GLC Buildings	3.00% p.a.

   Consumption of energy:

Island Block	£4.63 per sq. metre p.a.
Similar Buildings:	£6.51 per sq. metre p.a.

   Increased design cost has been traded-off against reduced operating cost.

2. The use of the Manned Air Combat Simulator (MACS) by the McDonnel Douglas Company, St. Louis, Missouri. Figure 11 shows some of the data comparing the design of the F.4 aircraft, without MACS, with the F.15 aircraft, with MACS.

	F.4 without MACS	F.15 with MACS	% change
Wind tunnel test hours prior to first test flight	4,300	22,400	+ 421%
Between first test flight and first production model			
– gross weight increase, lbs	3,050	460	– 85%
– engineering man-hours	2,576	289	– 89%
– number of engineering changes	134	36	– 73%

Figure 11. Use of Manned Air Combat Simulator
Source: The Economist, 1 July 1978

The wind tunnel test hours have increased by over 400%, but the gross weight increase, the engineering man-hours and the number of engineering changes, all between first test flight and first production model, have each fallen markedly. Increased design cost has been traded-off against decreased development cost.

3. The development of computer disc drives, where by greatly simplifying the engineering, and by reducing the number of moving parts and enclosing the read-write heads within the disc cartridge, the reliability under non-ideal user conditions has been greatly increased and the servicing made much simpler.

   In this case, increased design cost and production cost (subsequently offset by increased production volume) have been traded-off against increased user value and decreased service cost.

The lessons for system development productivity are clear:

1. Consider the whole system life-cycle, not just the development stage.

2. Collect more data about life-cycle costs, particularly operating and maintenance costs, so that relationships between the stages in the cycle can be understood better.

3. Trade-off a small increase in development cost and effort for a large decrease in operations and maintenance cost and effort.

B Approaches to Improving System Development Productivity

Given that a widely focussed view of the whole system life-cycle is required, rather than a narrowly focussed examination of individual tasks, several approaches have been taken that could result in productivity improvement.

We mentioned earlier that it was important to recognise that two types of activity take place, side by side, during the development of a system, namely project management and system development. Each offers an opportunity for improving productivity.

System development methodologies are concerned with the way in which the work is done. They may be regarded as existing in a three-dimensional matrix, as represented in Figure 12, with the following dimensions:

1. The system development stage of application. Some methodologies are directed at the requirements specification stage, others at the programming stage.

2. The approach. Some methodologies are based on a programming-like approach to the work, others are concerned with data analysis. Still others are concerned with the attitude of the end user.

3. The source. Some methodologies arise from a background of massive, complex and expensive military applications. Others arise from the need to appeal to non-technical first time users.

We have devoted Section V to a critical examination of the most useful or most interesting methodologies. In reading that Section it is helpful mentally to place the subject in this three-dimensional matrix so that it can be assessed with respect to the other examples.

Project management is concerned with creating and maintaining the conditions for successfully controlling the work. Good project management will not increase the goodness of a system but it will reduce the badness and reduce the risk associated with time and cost. This subject is discussed in Section IV.C below.

Environmental factors are concerned not with the work itself but with the worker's perception of it. There is considerable evidence that systems staff regard project management methods and system development methodologies as acting against their own interests, implying that clumsy

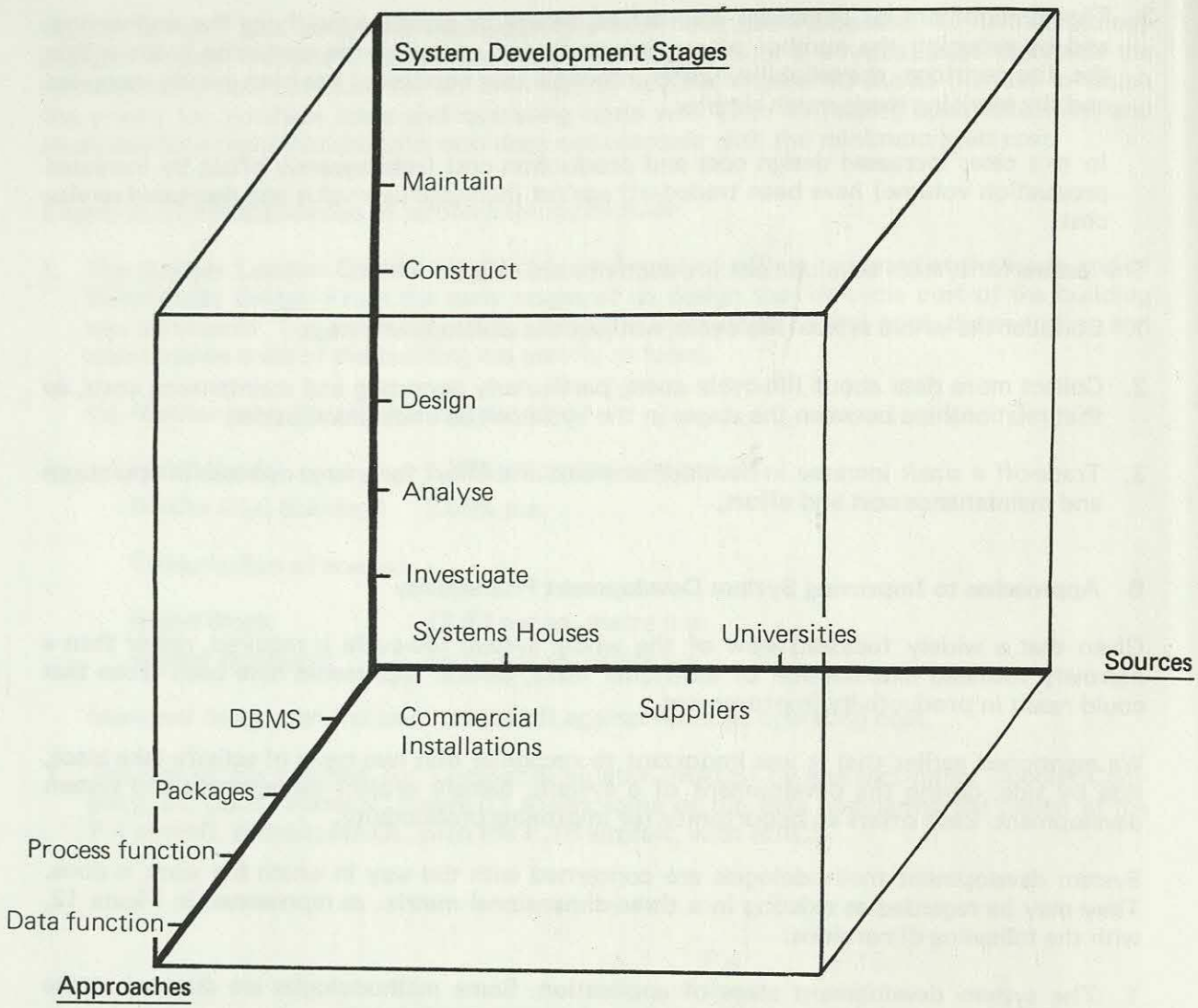


Figure 12. Three dimensional matrix of system development methodologies

introduction of new methods can be counterproductive. Environmental factors are discussed in Section IV.D below.

Data management is concerned with creating and maintaining a data facility to provide a base for system development and an information resource for the organisation. The growth in the use and success of database management systems is already improving systems productivity and is also changing the nature of system development work. Data management is discussed in Section IV.E below.

Perhaps the most obvious approach to improving productivity is to share out the cost and effort amongst many users. Application packages are, therefore, discussed in Section IV.F.

Most system development methodologies are based on the assumption that requirements must be defined before construction can begin. The alternative approach of empirical system design, or "trial-and-error" is discussed in Section IV.G.

Finally, the role of the user in system development and operation is discussed in Section IV.H.

C Project Management

In Report No. 8, Project Management, we argued that system development is best conducted in an organised and stable atmosphere where the ground rules are clearly established in advance and do not change to suit the whims of managers. We emphasised that if a high quality product is to be delivered on time and within budget, there are not many ways of doing it right.

Project management has two aspects:

- Planning the project.
- Controlling the work.

Too often project management ends with the preparation of the project plan, so that it is little more than a way of finding out how badly the project is going. Project management should be a way of identifying threats to success and of ensuring that they are avoided.

There are many ways of dividing the development cycle into controllable stages. The stages are given a variety of names and they depend upon the methods used, the conventions, the standards and the facilities available in the particular environment.

The five stages that we suggested in our Report No. 8 are detailed in Figure 13.

1. Survey	The problem area is studied and possible solutions identified.
2. Evaluation	The proposed solution is examined in depth, from both a business and a technical point of view.
3. Specification	The solution is specified in detail, providing the link between the user and the programmer.
4. Programming	The programs are written and tested.
5. Implementation	The files are converted, staff are trained, final testing is completed and the new system is introduced.

Figure 13. Five Stages of System Development
Source: Butler Cox Foundation, Report No. 8, Project Management

The most important principles of good project management are:

1. The boundaries of the project, defining and limiting its area and scope, must be clearly defined. If, as the project progresses, they need to be redefined then this must be a conscious decision which takes account of all the implications.

2. The responsibilities of all those involved with the project must be clearly defined and understood. This is particularly true of user management. They must accept responsibility for the successful completion of clearly stated tasks. Projects fail more frequently because of poor definition or poor execution of user tasks than for technical reasons.

3. The risk of failure increases dramatically with both the technical complexity and the number of staff involved. It is much more than twice as difficult to complete a 12 man-month project with four staff in three months than with two staff in six months. So the size and scope of the project must be within the capability of the resources.

4. The project should be broken down into natural stages, with each stage managed as a project in itself, with its own objectives, target dates, resources, and reviews. In this way, risks can be recognised and avoided early.

5. The costs to be incurred and the resources required must be estimated realistically, and agreed in advance. No project should require either a giant leap in the dark or implied authorisation for unlimited expenditure.

6. Target dates must be treated as personal commitments rather than best endeavours. A person cannot, however, be expected to make a commitment if either:

   - The target is an externally imposed deadline. In this case, costs and resources may need to be greatly increased to avoid the risk of failure.

   - Achievement of the target depends upon factors clearly outside the person's control.

   - The person has an inadequate understanding of the work involved.

7. The documentation of the project must be complete, and it must be compiled as part of the work, not as an after-thought. Documentation is used to consolidate understanding at every level, as a means of communication between people with different skills, as evidence upon which to assess progress and effect control, as a platform for future support and development of the project, and as a means of learning from one project to another. Without adequate documentation effective management is impossible.

8. Any changes to the requirements or to the system design must be formally agreed and properly communicated to those affected.

9. Standards for working procedures and quality control must be realistic, and they must be enforced.

10. The best planning tool is hindsight. Much can be learned from reviewing previous mistakes.

Although computer-based information processing systems are a type of general system, few applications of general systems theory have been widely described. One interesting example is the attempt to apply system control theory to information systems by Belady and Lehmann (4, 44). Their models of the life-cycle of systems include the concepts of "releases" and "versions".

They postulate that the process of large-scale program development and maintenance appears to be unpredictable – its costs are high and its output is fragile. They are concerned with studying the process, with the immediate goal of achieving an organised quantified record of observations. The hope is to identify areas of the process that are major causes of concern.

They contend that the organisation, the project, and the product form a system whose behavioural characteristics may be described by a series of Laws of Large Program Evolution. These laws suggest rules and tools for software development and maintenance. They show where and how management can take effective decisions and where and how the dynamics are the main determinant. Thus, they provide a natural framework for developing insight into the nature and the intrinsic properties of the programming process and of current software engineering practice, and into the potential for, and the possible directions of, process improvement.

The Laws of Large Program Evolution are shown in Figure 14. The results of Lehmann's work lead to conclusions which are outlined below:

1. Self-contained programming entities must not be permitted to grow. On the contrary, the largest programs that are maintained as entities should be considerably smaller than at present. They should be limited in functional content and complexity, so that each implementation can be fully tested and validated.

THE LAW OF CONTINUING CHANGE

A large program that is used undergoes continuing change or becomes progressively less useful. The change or decay process continues until it is judged more cost-effective to replace the system with a recreated version.

THE LAW OF INCREASING COMPLEXITY

As a large program is continuously changed its complexity, reflecting deteriorating structure, increases unless work is done to maintain or reduce it.

THE FUNDAMENTAL LAW OF SOFTWARE ENGINEERING

There exists a dynamic of large program evolution which causes measures of global project and system attributes to be cyclically self-regulating with statistically determinable trends and variances.

THE LAW OF CONSERVATION OF ORGANISATION STABILITY

The global activity rate in a large programming project is statistically invariant (for example: normally distributed in time with constant mean and variance).

THE LAW OF CONSERVATION OF FAMILIARITY
(PERCEIVED COMPLEXITY)

For reliable, planned, evolution, a large program undergoing change must be made available for regular user execution (released) at intervals determined by a safe maximum release content (changed or new) which, if exceeded, causes integration, quality and usage problems with time and cost over-runs whose consequences maintain the average increment of growth invariant.

Figure 14 Five Laws of Large Program Evolution
Source: M.M. Lehmann, Laws on conservation in large program evolution, 1978

2. Larger functional entities can then be created by the interconnection of such programs. With the rapidly growing technology of microprocessors, one would envisage each program entity running on its own processor. The large systems of today could be replaced by distributed systems of microprocessors.

3. The interfaces between program entities must be completely defined and controlled. This might be achieved through standard hardware interfaces implemented as, say, microprocessor devices.

4. It will be necessary to specify and control the program entities separately.

These concepts represent a significant step forward in the development of constructive proof techniques and of large but maintainable and long-lived software.

D Environmental Factors

In the search for increased productivity the effect on system development workers has been the same as in all other industries since the Industrial Revolution. The work has become increasingly controlled, fragmented and de-skilled. The reaction of the workers, not surprisingly, has been to resist change, to lose motivation or even to change profession. Due to the shortage of staff the most skilful are required for the most difficult and most interesting work. Consequently, the largest proportion of the work, applications design and programming and particularly system maintenance, is left to the less gifted staff. Simpler methods are then introduced, removing interest and the scope for initiative and so the vicious circle is given another twist.

In reviewing experience with IBM's Improved Programming Technologies in the USA, Holton (35) found that, although the techniques could contribute to increased productivity, they were not widely used. In the UK, although structural programming, for example, is recommended by the Central Computer Agency for use within Government Departments, it is only used in about 30% of departments. In both cases the reason is thought to be resistance by experienced staff to the new methods.

Improved pay and working conditions are unlikely to solve this problem. They may increase the numbers of people but they are unlikely to increase their output.

Improved facilities, especially the increased use of computer-aided methods (such as problem statement languages, specification test processors or on-line programming and debugging) may partly solve the problem, by removing some of the tedious tasks and increasing the worker's control over his own output.

Management structures will have a very significant influence on productivity. The hierarchical management structure is largely to blame for the disillusion of skilled staff who have been promoted out of their craft, for the loss of communication between staff and management, for the loss of technical skills by managers and for the growth of work fragmentation.

Alternative working structures have been used and these offer ways to increase motivation, output and quality. Two examples are:

- Adaptive teams (15, 75), where the team's load is assigned according to the strengths of its members, with a collective responsibility for output and quality and no formal leader.

- Chief programmer teams (2, 3), where a chief programmer is responsible for all aspects of the project, supported by a librarian and back-up programmers.

The most significant environmental factor is the worker's perception of his work objective. Due to the way that the profession has developed, the tacit assumption has been made that the objective is to design neat systems or to write clever code. Management has then imposed upon these internal objectives the need to achieve them on time and within budget. The real, external objective of assisting the user to manage his activities more effectively either has been missed altogether or has been accepted without its significance being appreciated.

As developments in facilities and methodologies remove more and more of the traditional tasks, the only way to achieve job satisfaction is by concentrating on service to the user, irrespective of the way it is provided.

One installation in the UK has transformed both its development and its operations departments by using a simple bonus scheme to focus attention on achieving benefit for the user. Interestingly, the bonus is funded by contributions from the user's own bonus earnings.

Another installation solved the problem of the unwillingness of programmers to work on maintenance by setting up a new system maintenance section as a career staging post between programming and systems work. The section was charged with keeping its users 'up and running'. The members of the section, initially sceptical, rapidly realised they had one of the most interesting jobs in the department and, because of the short time between problem and solution, one of the most rewarding in terms of feedback of user satisfaction.

E Data Management

In the past decade, interest in the management of data as opposed to the structuring of procedures and programs has steadily increased. It is clear that this trend will continue. In this Section of the Report, we look at the aspects of data management which have an impact upon the productivity of systems development. It is not our intention to summarise the whole field of database systems, since that is not the purpose of the report. Instead, this discussion is intended to assess the value of data management solely as an advance in system building methods.

In traditional processing systems, the equipment, the data, and the programs were all part of a single, central facility. Gradually, each of these components is achieving its freedom. Minicomputers and microprocessors have already allowed the equipment to be distributed, but the processors have normally taken the data and the programs with them. Now interest is being focussed on the independence of data from its processor. This issue will be explored in Report No.12, The Future of Database Management Systems.

The present trend is towards the amalgamation of files. One of the aims of such amalgamation is to reduce the level of redundancy in data, but this aim can be achieved only by relatively complex linking facilities.

If the linkage facilities are fixed in advance of usage, then a detailed knowledge of future data retrieval requirements is required. If, on the other hand, a wide range of possibly useful links is created, the system overhead may be unacceptable. Today, users of data management systems find the best compromise solution to this problem by trial and error. If service from the system declines, then the user is likely to intervene to simplify the data linkages. While service is maintained, even complex linkages will be maintained too, despite the increased cost of entering and maintaining data.

This in outline is the dilemma that confronts the user of a data management system. The impact of this problem upon the system designer is serious, because the parameters of the design are uncertain, also because they may be subject to rapid change during the running-in of the system.

In spite of these comments, the next major phase of growth in the industry lies in the use of database management systems. In recent years an important and continuing debate has taken place on what kind of facilities users will require in the future. At present there is something of a hiatus as far as the average user is concerned. Relatively straightforward file management systems are freely available, but they do not cover the wide range of facilities for the expression of relationships between data elements which are typical of a genuine DBMS. Those DBMSs that do provide such facilities, for example, IBM's IMS, have not yet reached the stage where an average user can exploit them with full confidence that the performance of the system will be as expected and that a high rate of efficiency will be secured.

Most users of data processing equipment are, therefore, keeping a wary eye on the develop-

ment of DBMS and are leaving it to larger companies, who have greater resources, to do the necessary pioneering work.

In 1974, the National Computing Centre conducted a survey of 21 users of database management systems. Their major findings were:

1. Development costs and timescales
   In just under half the cases, reduced costs and lead-times had been experienced in developing new applications, and three organisations claimed substantial reductions. The majority of the other organisations were expecting to achieve benefits when they had become more familiar with the new software.

2. Maintenance effort
   Many organisations claimed that maintenance had been significantly reduced, one even quoting a figure of 50%. In four cases, it was stated that although program maintenance had not been reduced, it had been made easier.

   Almost all the organisations interviewed had found that it was easier to expand systems than with the conventional approach to file management.

3. Integration
   There were few highly-integrated databases. Files within an application were integrated, but there was not a great deal of integration between applications. Most organisations were intending to move towards more integration, but some felt there might be problems of reliability and security if there was too much integration.

Factor	Number who experienced better or much better benefit than expected	Number who experienced worse or much worse benefit than expected	Total	Percentage who experienced better or much better than expected
Reduced programming effort in development	48	6	54	89%
Reduced programming effort in maintenance	42	5	47	89%
Reduced data duplication	55	2	57	96%
Better data consistency	57	0	57	100%
Faster response to new user requests	47	8	55	85%

Figure 15 The experience of respondents who expected strong or very strong benefits for each of the factors shown from use of a DBMS

Source: Butler Cox Foundation DBMS Survey 1979

A similar survey was conducted in 1978 by the Butler Cox Foundation in preparing the report on database management systems. The results of a preliminary analysis are shown in Figure 15. This shows the main reasons that the respondents were using a DBMS, plus the percentages of those respondents who had expected either good or very good improvements in these factors whose expectations were exceeded. Clearly, some DBMS users are highly satisfied with the results.

The claims that the database approach can improve productivity, by reducing the cost and effort of development and maintenance programming and by allowing a faster response to user requirements, are being confirmed.

As a consequence of the growth in the use of database management systems there has been a corresponding growth of interest in data analysis and data structures. This has added a second dimension to the system development process. Alongside the specification of the business problem and the design of its solution in terms of processing functions, the analysis of the data and the development of a conceptual data schema are recognised as equally important system development tasks.

A new generation of DP disaster stories is already being told about installations that leapt onto the DBMS bandwagon, built databases without first designing them, and have now had to write off their investment and start again. The risk is that unless the theory of data structures and how it should be applied is understood, inflexible databases will be constructed. These, at best, will be difficult to use, and, at worst, will actually give the wrong results.

As the technology develops and makes it possible to consider independently the location of the equipment, the data and the programs, so the importance of good data structuring will increase.

F Application Packages

The use of a suitable ready-made application software package can greatly reduce the time and the cost of implementing a computer system. A wide variety of packages (some having dedicated hardware) is marketed, but their adoption in this country is low when compared with practice in the USA. Indeed, a survey in 1973 (76) indicated that of the most popular packages marketed, none was orientated towards solving a specific business problem. The management and teleprocessing monitors were the most common type of package supplied in terms of dollar sales. As the cost of hardware comes down, the number and the scope of available application packages linked to low cost dedicated hardware can be expected to increase. For example, computer-driven word processing systems which take advantage of microprocessor technology, low cost floppy disc stores, and high quality serial printers represent a trend towards the 'electronic office'. In that role, the computer may appear to the user not as a programmable device but as a black box, performing extensive but specific functions on a trusted basis. The chief benefit provided from the use of a package is usually a reduction of programming costs, both for implementation and maintenance. The use of a package will contribute to a reduction in the time taken to implement a system. Apart from the fact that cost savings may result from realising benefits early, the use of a package is a valid way of achieving tight deadlines.

There are two main disadvantages with the use of application packages. The first is that the package may not provide an exact solution to the user's problem. Since we have been arguing that the mismatch between the user's requirements and the system solution is the major reason for low productivity, this disadvantage may seem to cancel out the benefits of reduced development and maintenance costs.

However, if a good survey and analysis of the problem reveals the full set of the user's require-

ments, and if he then consciously decides to adopt a package as a solution to a sub-set of these requirements, this disadvantage is minimised. The user makes a trade-off between a sub-optional solution and the benefits of reduced cost and increased control of the system.

A further compensation is that the use of a package reduces the number of variables that need to be controlled. This is particularly important for first-time users. While the staff are learning to live with a computer-based system, and while the management are learning to control a new type of operational activity, it is helpful to remove the problems of controlling the activities and the staff involved in a system development project.

One prospect is the use of throwaway packages, especially where these are associated with dedicated equipment. The first level would be a very simple and basic system that could be installed within a few weeks and would be used for a few months. This would be replaced by a second level system with more facilities, that was used for a few more months. More sophisticated levels would be used later, but none would be maintained.

The second main disadvantage of application packages is the "not invented here" problem. Given the traditional objectives and motivations of systems staff, discussed in Section IV.D above, reasons can always be found why packages are either not suitable or require extensive in-house customisation. This trap must be avoided. The proper role of systems staff is to identify the requirements and to select the best product on the basis of commercial, systems and technical criteria.

G Empirical System Design

Throughout this Report we have been stressing the importance of giving more attention to the early stages in the system development process, in order to reduce the need for system maintenance. The philosophy is to identify all the requirements before designing or constructing the solution.

There is an alternative philosophy. If methods of construction can be provided which are cheap, quick and easy, build the thing and then decide whether it is suitable. If not, build another.

We have already described, in Section II.E, the polarisation of system development work with, on the one hand, complex technical work required to provide basic resources and, on the other hand, little effort required to develop applications of these resources. So we believe that the trend is already in the direction of empirical design. In addition, other approaches, discussed below, are already being used.

1. Prototype development
   Byron Pumps Inc., of Vernon, California, uses what is called a prototype development process. An initial system is developed which is not intended to be the final product. It is a basic system with which the user can work with the computer during the design process. The basic philosophy is that the solution to a problem is most often a function of how the problem is defined.

   The major restriction to this philosophy is the formalisation of conventional file design. As a result the System 2000 database management system was adopted to provide the required design flexibility. System 2000 contains a very powerful report generation capability called 'Immediate Access' which allows parametric searches of files. For example, it allows a considerable amount of analytical work to be done on databases, essentially without any programming.

   The approach does not recognise a clear distinction between testing and the production use of a system. System testing is taken to start from the outset of the project. A quality

assurance function is constantly auditing the system, talking to users and operators and evaluating the procedures, auditing the database to make sure that the database elements are being used. The aim is to get the system running, and then to optimise during its operation.

The project team in this environment consists of one systems analyst together with a number of user personnel. They are required through the development process to identify those data elements that are going to be involved. The systems analyst then sets up the computer file and information is loaded into it. Then, through a process of trial and error, questions are asked of the database. The analyst handles the reports, the users evaluate them. The approach requires a total commitment and involvement from the user in the development process. It is claimed that developments costs are considerably less, because development timescales are shorter and better requirement specifications are produced. Also, maintenance costs are low because of a swift response to errors. A major problem is the learning curve associated with the methodology, particularly on the part of the more experienced data processing personnel. No staff reduction has been made but there has been a distinct change in emphasis. The previous ratio of four programmers to two analysts has changed to six analysts to three programmers.

A similar approach can be taken by any user by taking advantage of the facilities that most timesharing companies offer. Both in the United States and in Europe a considerable number of timesharing companies have concentrated upon the small to medium-sized computer user as their most advantageous potential market. It is ironic that, in providing what might be termed elementary services to the small user, some of these timesharing companies have produced system building facilities which are of a degree of sophistication higher than those sometimes found in more ambitious installations. Particularly significant are those packages which provide, if not the full range of facilities of a DBMS, at least an impressive imitation of them. Some packages provide the first-time user with the ability to construct a database, admittedly of limited complexity, using natural language as the principal medium of construction. All the elements of data within the array are named and classified directly by the user in an on-line mode. Once the database has been constructed the user has the ability to interrogate the data using a wide range of facilities triggered off by simple macro commands.

The process of system building has been rendered relatively painless and modifications to the database can be implemented with considerable ease.

2. Systematics
The concept of Systematics was originally developed in 1966 by Grindley (31, 32), as a system definition language. Since then he has added a program generator so that, using a sub-routine library, defined systems can be converted into working programs. The intention is to provide a kit of parts so that systems can be developed quickly and easily on a trial and error basis.

H Role of the User

It is a truism to say that the only good systems are those that are used. Nevertheless, good systems work is frequently wasted by lack of attention to user aspects such as:

- Design of clerical procedures.
- Provision of user documentation.
- Training.
- Ergonomics, in the design of machines and the layout of working areas.

— Industrial relations, in the introduction of new systems.

Opinions on how to design and implement clerical procedures vary from those who claim that no person has the right to design another person's working methods, implying that design must be a collective, social activity, to those who state that the fewer people who are involved the better, provided that the resulting procedures are well designed.

The best approach will depend upon the particular circumstances. The development of distributed processing and interactive systems is making systems more "user friendly". But social trends are making systems more difficult to implement. Some factors to consider are:

1. What type of staff are involved? Are they educated, well motivated with some personal initiative, or are they poorly educated with limited verbal and numerical skills?

2. How stable are the staff? Is there a high turnover in the job? Is the job often done by temporary, rather than permanent staff?

3. How should the work be structured? Should one person do all the tasks for specific groups, such as customers or products, or should one person do one task for all the groups?

4. What does the operator need to do the job? Is a telephone, a code book, or filing essential?

5. What is the work pattern? Can the peaks be spread into the troughs to give a steady load, or is there high activity followed by boredom?

6. How responsible is the work? What is the effect of getting it wrong?

7. How should the new procedures be implemented? Will they be imposed, will the staff be involved in their design, are union negotiations required?

V. SYSTEM DEVELOPMENT METHODOLOGIES

The purpose of this section is to provide a critical review of some of the latest methodologies for developing information systems. The examples are chosen either because they have been used successfully or because they provide a conceptual breakthrough in our way of thinking about system development.

To provide some structure for the Section we have grouped the examples within the same three main stages of system development, as we used in Section III, namely:

- Definition of requirements.
- System design.
- System construction.

The allocation of examples under these headings is necessarily arbitrary as each one emphasises a different aspect of the development process and they each span a different range of activities.

A Definition of Requirements

In this stage the user's problem or opportunity is identified and the requirements of the system are defined. In Section III we stressed the importance of this stage, since most of the problems that arise after the system has been implemented can be traced back to inadequate definition of requirements. This problem is compounded because, unlike programming errors, requirements errors are generally difficult and very expensive to correct. Until recently this stage has received little attention, in terms of thinking about the work involved and developing methods and tools to assist with it.

1. The Lancaster Approach

The Department of Systems at the University of Lancaster has played a significant role in pioneering an approach to the analysis of information systems. They point out that the theoretical and the actual stages in a systems project are generally as shown below:

Theoretical	Actual
—	Investigation
Analysis	—
Design	Design
Construction	Construction
Implementation	Implementation
—	Rectification

Investigation may be carried out, but analysis is usually omitted because no-one knows quite how to do it.

Their general methodology for investigation and analysis, described by Checkland (20) and Collins (22), falls into seven steps, illustrated in Figure 16.

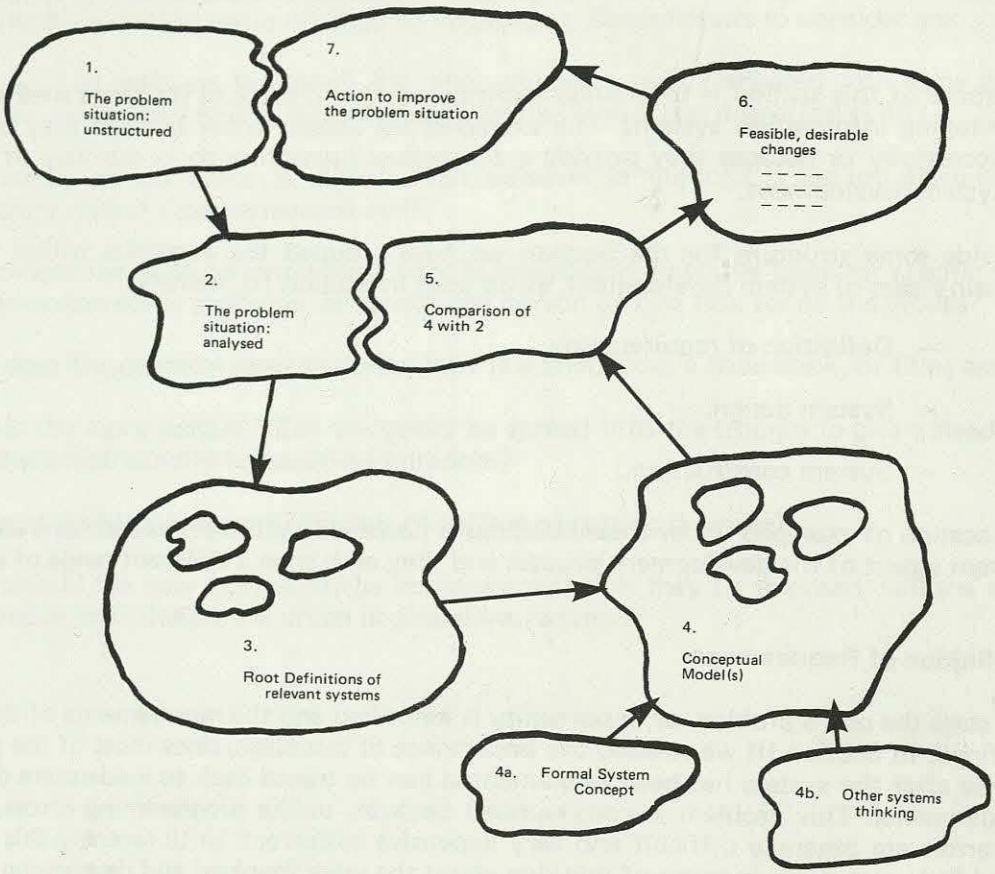


Figure 16 The Seven Step Lancaster Approach

Source: 'The Development of Systems Thinking by Systems Practice — a Methodology from an Action Research Program'.
P.B. Checkland, 'Progress in Cybernetics and Systems Research', Vol.2., Hemisphere Publishing Corporation, Washington, 1975

a. Recognise that a problem exists.

b. Analyse the problem in a neutral way that does not distort it into any particular form, investigate the existing outlooks, objectives, procedures and constraints.

c. Select some viewpoints that seem potentially relevant to bringing about a solution to the problem and formulate 'root definitions' of the problem from these viewpoints.

d. Develop conceptual methods of the minimum necessary activities of the system based upon the root definitions.

e. Make formal comparisons between the actual problem, as defined in step b. above, and each of the conceptual models postulated in step d. above, to identify potential changes.

f. Identify feasible, desirable changes.

g. Take action to make the changes and improve the problem.

Steps a, b, e, f, and g. above take place in the problem area, in collaboration with the staff concerned. Steps c. and d. above are explicit systems thinking.

The crux of the methodology is step c, which involves selecting ways of viewing the problem and formulating a root definition. The components of a root definition are:

– Ownership	Who controls or sponsors the system.
– Actors	Who is involved with the system.
– Transformation	What the system achieves.
– Customer	Who the client, beneficiary, or victim of the system is.
– Environmental constraints	How the system interacts with wider systems.
– World view	The viewpoint that gives meaning to this root definition.

Checkland quotes, as an example, the following root definition of the visiting arrangements of a mental hospital:

> "A regular, volunteer student-manned, medically approved mental hospital patient comforting system".

He points out that the weakness of this root definition, and thus the strength of the method, is that the ownership is not clear. It is not possible to decide from the definition whether the system exists for the benefit of the doctors, or the visitors, or the patients.

The methodology is significant for two reasons. First, it emphasises that the world view must be recognised. In well-structured problems it is usually obvious. However, in higher-level, unstructured, soft problems many world views may be possible, and this leads to many conceptual models of the system. Second, it provides a formal method and a set of criteria for comparing system models.

The methodology was evolved in the course of an action research programme of live systems studies in 1969-71, covering more than 100 projects.

2. Structured Analysis and Design Technique

The Structured Analysis and Design Technique (SADT) was developed by Douglas T. Ross and is marketed by Softech Inc. (63).

It is based on the functional decomposition of a system. The SADT philosophy is that the functional specification of the system should be analysed in as much detail as possible without letting design decisions intrude. When it is no longer possible to analyse the system unless a design decision is made, that decision is made and documented and analysis then continues.

Each decision has an impact on the system, and this impact should be analysed before the next decision is taken. In this way, constraining decisions are taken at as late a stage as possible in the development of the system, and this means that the system design is not committed to a particular implementation until the last possible moment. Iteration is a formalised process in the procedure.

The method is built around a diagramming technique, utilising a few simple rules. It is a way of analysing problems top-down, and of refining the specification in more and more levels of detail until it is a complete and unambiguous statement of the work to be done. The most important thing about this process is that the parent diagram for each decomposition provides a context for the new diagram, and this must not be extended or limited by the process of decomposition. SADT contains a number of ways whereby inadvertent violation of this rule can be trapped and corrected. In this way, it is possible to ensure that the system represented by the sum of the lowest-level diagrams is the same system, with exactly the same external interfaces, as the one represented by the top level diagram.

A full, hierarchical description of a system is called a model, and models may be constructed from different viewpoints. The most common viewpoint is that of the system designer and this model is the one which shows the functional inter-relationships of the various parts of the system. A complete model will contain two separate hierarchies — one for activities and the other for data. The two aspects will be modelled from the same viewpoint, but independently. The resulting models will then be used for the additional activity of cross checking each other.

To apply the method it is first necessary to learn the syntax and the semantic rules, and this is not difficult. They provide a structured way of thinking which can be applied to the task of analysing any kind of problem.

The SADT approach can be used at all stages of development, from problem analysis through to system specification. The lowest, most detailed level of a functional hierarchy provides the input to system design and the approach is compatible with other techniques.

The main drawback is that although the rules are easy to learn, skill and experience are required in applying them. Since the method contains no computer assistance, the analysis of complex systems can involve preparing hundreds of pages of diagrams, all of which must be checked for accuracy and consistency. Also, it shares the danger that is inherent in all top-down approaches, namely that if the system contains a large number of basic mechanisms the system cannot be appreciated until the mechanisms are understood. However, SADT has been used on a large number of both large and small projects.

3. The ISDOS Project

The basic concept behind the Information Systems Design and Optimisation System (ISDOS) project being developed by Professor Teichroew (67, 68, 70) at the University of Michigan, is that the various models of a system that are created during system development are all manifestations of the same set of data. Therefore it should be possible to use a database with a DBMS to define, analyse, implement and maintain this data. Basing system development on the use of a database gives the advantages that data describing the system (both user functions and user data) can be held centrally, can be checked for duplication and consistency and can be analysed in various ways to meet the requirements of the system designer.

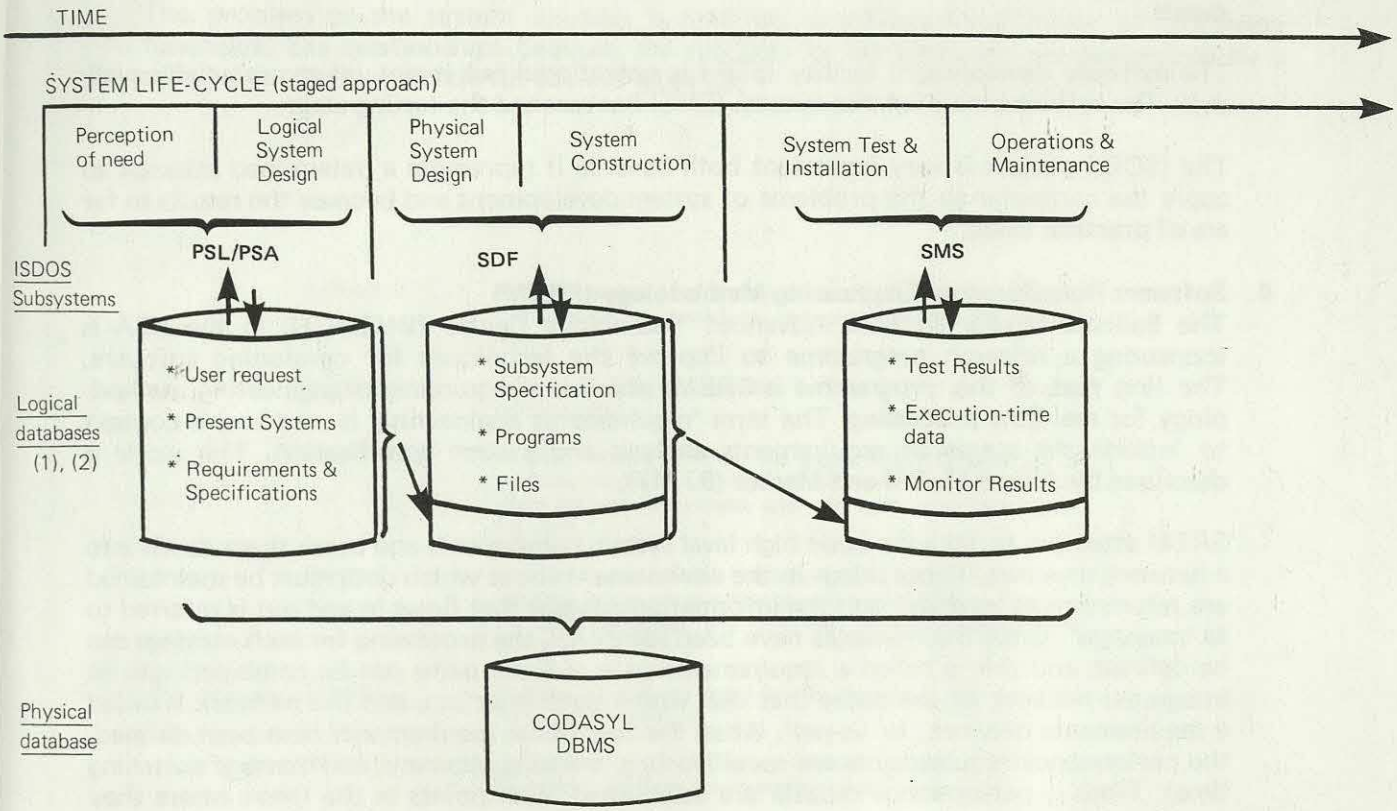
The project is envisaged in three stages; each one uses the same physical data but handles it through logical subsets using different pairs of definition and analysis languages. The stages are:

- Problem definition.

– Software development.

– Software maintenance.

Their relationship to the system database is illustrated in Figure 17.



Notes: (1) Arrows indicate flow of information to and from logical databases

(2) Access to a logical database includes access to the databases in the previous stages, thus enabling proper traceability across the entire system life-cycle. For example, if a program needs to be modified during the Maintenance stage, access to the SDF database will show how the modification will affect all programs and files with which the chosen program interacts. At the same time, access to the PSL/PSA database traces back the modifications to the systems specifications and user requirements.

Figure 17 An ISDOS Database

The problem definition stage is already operational and provides a Problem Statement Language (PSL) and a Problem Statement Analyser (PSA). PSL is used to specify a problem in terms of entities, classes, relationships, timings, volumes, synonyms, attributes etc. PSA is then used to check and analyse the data and provides a number of useful summaries such as:

– Formatted problem statements.

– Directories and key-word indices.

– Graphical summaries of flows and relationships.

– Statistical summaries.

Teichroew argues that, since PSA can be used to produce all of the documentation required for either a requirements specification or a system specification, which would

need to be done anyway, all of the additional features of error checking and analysis are free.

PSL/PSA is being used in a number of, generally large, commercial, aerospace and government installations. Its main weakness is that, having been developed to handle batch business systems, it is difficult to express real-time and man-machine interaction requirements.

The software development facility (SDF) is operational but is not yet commercially available. The software maintenance system (SMS) has reached the testing stage.

The ISDOS project is very important both because it represents a determined attempt to apply the computer to the problems of system development and because the results so far are of practical value.

4. Software Requirements Engineering Methodology (SREM)

The Ballistic Missile Defence Advanced Technology Centre (BMDATC) in the USA is sponsoring a research programme to improve the techniques for developing software. The first part of this programme is SREM, which is a requirements engineering methodology for real-time processing. The term 'requirements engineering' is used in this context to include the stages of requirements analysis and system specification. The work is described by Alford (1, 36) and Marker (37, 47).

SREM attempts to take the basic high level system components and break them down into a network structure. Those things in the environment about which data must be maintained are referred to as 'entities', and the information content that flows in and out is referred to as 'messages'. Once the messages have been identified, the processing for each message can be defined, and this is called a requirements path. All the paths can be combined into an integrated network of the paths that deal with a given interface, and this network is called a requirements network, or 'R-net'. When the functional requirements have been defined, the performance requirements are specified (e.g. tracking accuracy, and message switching time). Finally, performance criteria are established, with points in the R-net where they can be monitored.

The process of creating and validating this network is clearly an immensely complex one. A set of computerised tools has been developed to support it through the Software Requirements Engineering Program (SREP).

SREP is probably the most powerful and most extensive tool yet devised for requirements engineering. It is based on the ISDOS concept, it uses the ISDOS data management system, it has a requirements statement language (RSL), and an analyser (the Requirements Evaluation and Validation System – REVS).

One of the most important features of the SREP approach is the use of simulation models to test the specification.

SREP requires extensive and sophisticated hardware and software so that it has not been used outside the Ballistic Missile Defence Advanced Technology Centre.

The limitation of the SREM methodology with the current SREP tools is that it is designed specifically to handle ballistic missile type problems, with the emphasis on autonomous, real-time, extremely reliable applications. The ability to handle large file processing and man-machine interaction is absent.

Nevertheless, SREM is important because it has demonstrated that it is possible to develop a methodology with objective yardsticks. It has also defined what a software requirements specification must contain if it is to be complete, consistent and traceable.

5. High Order Software

The Higher Order Software (HOS) methodology was developed by Hamilton and Zeldin at Draper Laboratories, Inc. (33). It was designed to provide a formal means of defining reliable, large scale, multi-programmed, multiprocessor systems such as are required for NASA projects.

The problem or the system solution is specified as a hierarchical model of processing functions. The relationships between the modules in the hierarchy are determined by a set of six axioms which are set out in Figure 18.

Axiom 1	A given module controls the invocation of the set of valid functions on its immediate, and only its immediate, lower level.
Axiom 2	A given module is responsible for elements of only its own output space.
Axiom 3	A given module controls the access rights to each set of variables whose values define the elements of the input space for each immediate, and only each immediate, lower function.
Axiom 4	A given module controls the access rights to each set of those variables whose values define the elements of the input space for each immediate, and only each immediate, lower level function.
Axiom 5	A given module can reject invalid elements of its own, and only its own, input set.
Axiom 6	A given module controls the ordering of each tree for the immediate, and only the immediate, lower levels.

Figure 18 HOS Axioms

Source: Datamation, November 1977

To simplify the work of checking that the relationships satisfy the axioms a specification language called AXES is provided, in which the axioms are embedded. This is used to set up the hierarchy. A Design Analyser and a Structuring Executive Analyser are also provided to process the AXES specification and check that it satisfies the axioms. These are not essential and the model can be checked against the axioms manually.

The HOS approach can be applied to the specification of problem definitions and system requirements and it can also be used in the design of software. It has the merit of providing a rigorous theoretical basis for specification and software design. However, it does not assist with data structuring.

B System Design

The design stage starts from a complete definition of the system requirements. In this stage alternative systems solutions that satisfy some or all of the requirements are considered, and the best one is selected. The system is designed from a logical, as opposed to a physical, point of view up to the stage where it can be passed to the system construction stage, where it is realised in a physical way.

The criteria for good design are:

- The designed system must satisfy the agreed set of requirements. None must get lost in the process.
- The system must be conducive to use. It should fit easily into the user's work pattern.
- The system structure must be as simple as possible, in order to assist both development and maintenance.
- The system structure must be as flexible as possible, so that additional requirements can be incorporated easily.
- The system should be efficient in its use of resources, both computer and the user's.

We have already indicated that, although their main application is in the requirements definition stage, ISDOS, SADT and HOS can be applied in the system design stage also.

1. **Structured Design**
   The Structured Design methodology was developed over a number of years by a group of researchers mainly from IBM. The orignator was Larry Constantine and one of his fellow researchers was Glen J. Myers, who developed the related techniques of Composite Design. The first formal description of the methodology appeared in 1974, and further descriptions have appeared more recently (13, 66).

   The methodology is based on the principle of functional decomposition and it takes the concept of modular programming as one of its basic assumptions. It makes extensive use of problem data in forming the design, although it uses data flow rather than data structure in the design process. The data flow is depicted through a special notation which identifies each data transformation, and each transforming process, and the sequence in which they occur.

   The sequence of steps in the process is:

   - Constructing an outline chart of data flows.
   - Constructing a chart of the structure of the functions involved in the data flows.
   - Iterating the structure chart until it is compatible with the data flow chart.

   In this way, a top-level design is derived, and the design process consists of further decomposition of each level of the design until the design is complete. At each stage, criteria for assessing the quality of the design with respect to the component modules are applied.

   Data flow graphs (or bubble charts) are used to express requirements in a graphical form. The bubble chart is particularly useful in helping the designer to recognise points where significant changes to the data occur.

   The chief concern of Structured Design is with the choice of program modules. In evaluating a program module Structured Design considers its connections to other modules (i.e. 'coupling') and its strength or intramodule unity (i.e. 'cohesion'). The technique rests upon two classification spectra: one for coupling and one for cohesion.

Cohesion levels range from 'functional' (the module performs a single specific function) to 'coincidental' (there is no real relationship between the module elements which are grouped for packaging considerations). Coupling levels range from 'data' (all communication is via data elements) to 'content' (one level references the content of another). Using these criteria, a given design can be assessed and modified to improve its quality. The assessment process is obviously one of trade-offs, but such decisions and their implications are consciously realised.

A structure chart is used to show, in a precise manner, the structure of a program in terms of the modules and their interactions. The design process, then, is an iteration of bubble charts and structure charts in a process of increasing levels of detail.

The advantages of Structured Design are that it leads to designs that are both clear and simple, and also easy to code, debug and maintain. Also, it provides a basis for managing software development.

Its disadvantages are that it seems to be difficult to learn and to become skilled in, it leads to larger, slower programs, and it does not provide assistance with designing data structures. The methodology does not include computer assistance.

Structured Design has been widely used, and has proved to be of value in a variety of applications. However, there are some considerations regarding the size of problem for which the methodology is appropriate. A 300-line program probably does not need the techniques, and a 30,000 - line program may be too big for it. General principles, including structured analysis and data flows, can be used on a large program to decompose it into smaller chunks of 5,000 lines or fewer, where the design method is most applicable.

2. **Hierarchy Plus Input-Process-Output (HIPO)**
HIPO is a design and documentation technique developed by IBM (65). It is a hybrid graphic and narrative means of representing a design. The purpose is to supplement and precede the traditional tools of system design and program design, and in this way to improve the quality of the finished product.

HIPO uses two types of diagram (see Figure 19). The H chart is a hierarchy chart of those functions of a system which state 'what' is to be done, rather than 'how' it is to be done.

The functions in the chart go from general (at the top) to specific (at the bottom). The connecting lines between the function boxes do not show the flow of control, but rather the decomposition of the functions into subfunctions.

The IPO chart describes each function by its inputs, processes and outputs. A function box on the H chart is represented by one IPO chart which lists the various inputs, the processes on those inputs, and the resulting outputs for that function. This IPO information is then used to create the next lower, more detailed level on the H chart.

HIPO charts are useful for defining major program functions, but they give a limited and disjointed view of what a program is doing as a whole. The technique tends to ignore the sequential nature of programming, and its condensed format makes it difficult to estimate the degree of complexity and the amount of coding involved. The communication aspect relates to the degree of acceptance of HIPO charts on the part of the user. The technique is highly regarded as a documentation technique, and it is in this context that its major contribution to maintenance lies.

It is claimed that HIPO can be used as a design tool, as a means to improve communication with users, as a means to provide documentation, and as an aid for maintenance. The technique has not yet met widespread acceptance. In several cases it has been tried and subsequently discarded.

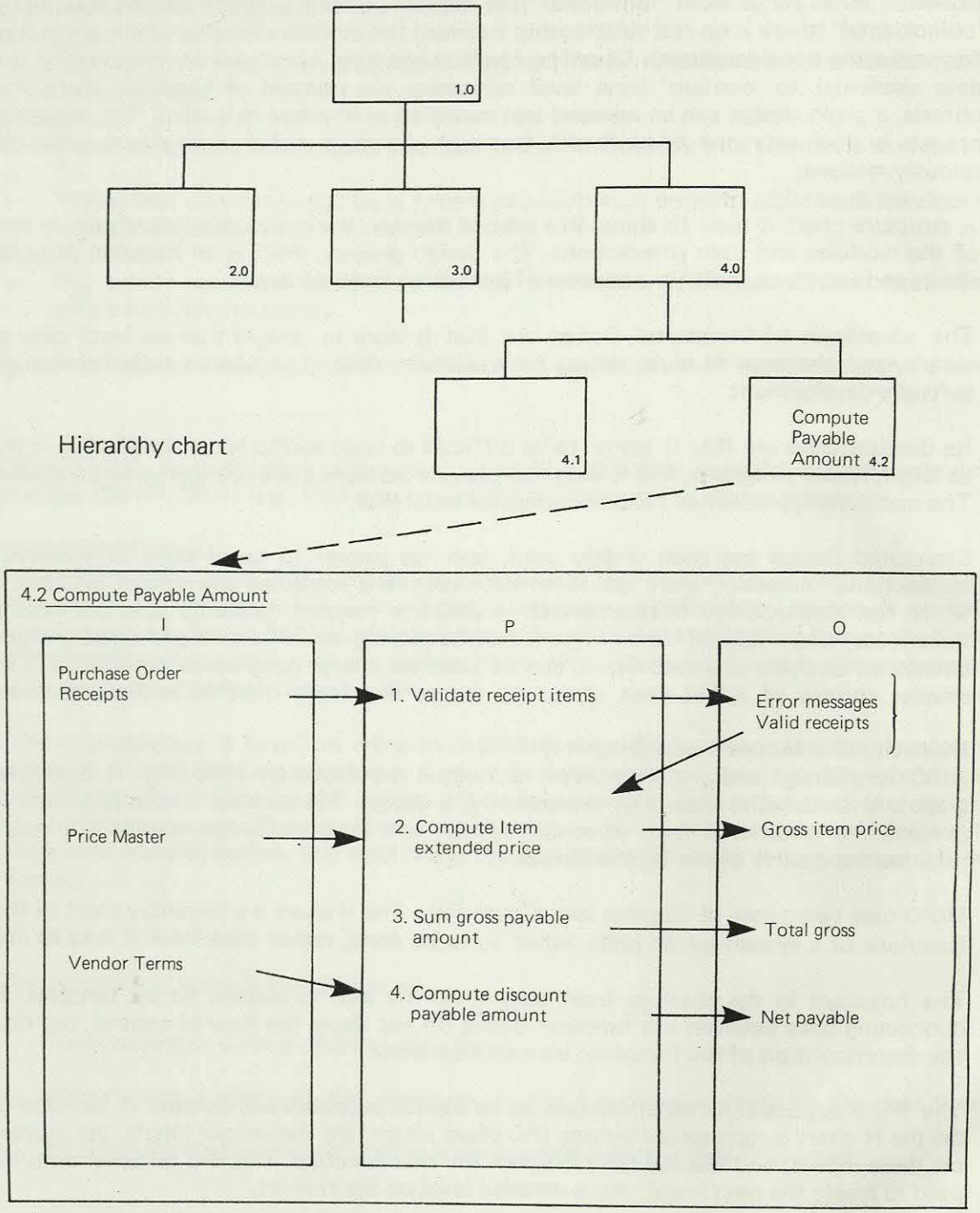


Figure 19 HIPO Charts

3. System Optimisation and Design Algorithm (SODA)

SODA originated in the ISDOS project and has been developed through work at several American universities (52, 53). The purpose of the method is to provide a computer facility with which the system designer can:

- Specify his design, with performance criteria.

- Evaluate alternative designs that satisfy the system and the performance requirements.

System requirements are specified using a problem statement language, which caters for data descriptions, processing requirements, and operational requirements. Processing requirements specify the formulas for transforming input to output. Operational requirements specify volumes, timings, and frequencies.

SODA also includes a problem statement analysis language, a design generator, and a performance optimiser and evaluater. When evaluating alternative designs, the designer can consider any one of the following:

1. Computer-generated designs only, or

2. Manually-generated designs only, or

3. Both computer-generated and manually-generated designs.

The value of SODA is not so much that it assists with the system design process, but that it provides a tool for generating and comparing alternatives. Its value depends upon the accuracy with which the evaluation algorithm is specified. It does allow complex alternatives to be compared.

C. System Construction

In the system construction stage the logical system design (which was prepared in the previous stage of development) is realised in a physical way. This is done by writing programs for a specified computer and by designing clerical procedures for real people.

Methodologies described earlier, such as the software development stage of ISDOS, HOS, Structured Design, and HIPO, can all be applied in the system construction stage also.

In recent years a great deal of attention has been directed at this stage of systems work. IBM's work on Improved Programming Technologies (IPT) and the controversy surrounding the Jackson design method is evidence of this. The methodologies have usually been directed at improving micro-productivity, and have tended to ignore the more general problem of macro-productivity.

Important criteria for a methology in the system construction stage include the following:

- The resulting system should be simple both to understand and to maintain.

- The resulting system should be simple to use (for both the end user and the computer operations staff).

- The methodology should be simple to learn and apply.

- The methodology should reveal all errors and inconsistencies.

- The methodology should be cheap in its use of resources.

1. Jackson Structured Programming

The Jackson method was developed over a number of years by Michael Jackson (40) and has been widely used in commerce and government since the mid-1970s. The methodology represents a radical departure from many of the widely-accepted concepts of functional

decomposition. A fierce debate has centred around its basic philosophy, which is that the program structure should reflect the data structure.

The method has the following characteristics:

- It is non-inspirational (i.e. it does not depend on creativity or experience).
- It is rational (i.e. it is based on reasoned principles).
- It is teachable.
- It is practical.

With the method, the input and the output data structures are first defined, using context-free grammars. Then, a general program structure is created to match both the input and the output data structures. Finally, the elementary operations are listed and each is allocated to an appropriate program component. No rules are given to guarantee completeness or correctness.

The essential feature of the methodology is the breaking down of the complexity of design into a series of steps leading from requirements to code, as follows:

a. Drawing data structures. By drawing out the logical structure of the problem data, a clear, unambiguous picture is obtained of the nature of the problem in terms of the 'outside world' requirements – the input and the output data.

b. Finding correspondences. If the program is not to be unnecessarily complex, the data structures must have entities which correspond in terms of being available for processing at the same time. These corresponding entities are identified to show where the data structures can be combined into a single program structure. 'Clashing' structures are eliminated at this stage.

c. Forming the program structure. Combining the data structures into a single program structure shows precisely the processing sequence, whilst still retaining the 'shape' of the problem inherent in the data structure.

d. Listing and allocating program operations. When the structure of the program has been defined, it is a simple and logical step to allocate to it the detailed operations the program needs in order to carry out the requirements specified. Operational detail is added only when the program structure has been established.

e. Producing schematic logic (pseudo code). Converting detailed program structure to a text listing is achieved in a simple way by following its implicit top-to-bottom, left-to-right sequence, and writing the operations in the form of indented pseudo code. Conversion of the pseudo code to program code is a straightforward and 'mechanical' task of translation into the specific programming language used.

The most important of the steps above are the first three, since it is in those steps that all the critical design problems are identified and resolved.

A considerable number of DP problems can be handled by the technique described above. However, the following two difficulties can arise:

a. A 'structure clash'. This occurs when a single program structure will not handle simultaneously both the input and the output data structures. The problem can be overcome:

   - By using an intermediate file.

- By writing the programs as co-routines.

- By writing one program as a subroutine of the other.

b. 'Back-tracking'. This is the need to make run-time decisions before the evidence is available, e.g. handling erroneous data.

These difficulties can be overcome by adaptations available within the method. Of more fundamental concern is the fact that the method is limited in practice to serial files, and there may be no direct link between the data structure and the program quality.

The Jackson method has a considerable number of points in its favour and a few against it. Its advantages largely centre around the clarity with which the method approaches the design task. The limitations concern the scope of problems which the method can handle.

The significance of the approach lies in the discipline it brings to the design process. It can be applied consistently by different users, and this leads to increased standardisation and consequent improvements in maintainability.

2. Delta

The Delta system was developed by Sodecon A.G. of Switzerland and it uses the principles of an engineering assembly shop to build software. The three basic components of the system are management, tools, and a stock of semi-finished program routines (71).

Management aspects of the system include:

- Organisation structure. System construction staff work in teams in which each member has his own speciality. (See the description of adaptive teams in Section IV.D.)

- Standard procedures. The work load is subdivided into elements in a standard way, so that both management and the team members understand what is required. The elements are allocated to the teams as pieces of discrete work, each of which has a precisely described endpoint, so that the work flow can be controlled.

- Standard system development methods. System design and programming is carried out using standard methods, so that standard, high-quality products are delivered.

The system development tools include:

- Interactive programming. The clerical work that is usually associated with design and programming has been replaced by computer-based methods, using a Prime 350 linked via RJE to a large host system.

- Macro language. Program elements can be assembled into a program using a macro language.

- Function processors. A variety of processors is provided to support the description of program structures. These include, for example, a structured programming processor, a decision table processor, a Jackson method processor, and a control break processor.

- Report writer. Reports can be developed using a report writer, which first allows the report to be defined and then prints out a pseudo list which is similar to a spacing chart. A documentation record is then generated, and finally the COBOL or PL/I code is produced.

A system development library is stored on the host computer. It contains function descriptions, data descriptions, program structure descriptions, macros, and program text.

All of the tools and methods can be used simultaneously, and new system development methods can be introduced if required.

Delta has been used to develop systems, ranging in size from very small to very large, for thirty clients. It is a very good example of the way in which standard methods and computer assistance can be combined to improve throughput and quality.

3. Pseudo code

A pseudo code is a semi-rigorous language that can be used to describe computing procedures. In general pseudo codes contain a mixture of programming concepts, such as "IF–THEN", "DO–WHILE", "GO TO", with English language statements and algebra.

They are most often used to communicate the details of a top-down design into a program specification. They vary in formality from home-made versions, that are orientated towards the compiler language and the standard procedures used in a particular installation, to machine-processable versions, which are more rigidly defined.

The output from a machine-processable pseudo code is one of the following:

- Standard program documentation, or
- Program statements that can be compiled by a high-level language, or
- Executable object code.

Program Design Language (PDL), for example, enables the user to prepare an outline of a program structure using rules that broadly correspond to sequence, selection, and iteration of structured programming (72).

Input to the PDL processor consists of control information plus defined procedures. The output is a design document, printed in a standard layout with cross referencing, which can be included in the development documentation. The usefulness of this type of approach to design derives partially from the use of computer processing, but mainly from the discipline that is imposed on the design by the rules of the language.

4. IBM Improved Programming Technologies

IBM has developed a number of programming aids under the general title of Improved Programming Technologies (IPT).

They are:

a. For designing software:
   - Top-down design.
   - HIPO (Hierarchy plus Input-Process-Output).
   - Pseudo code.

b. For building software:
   - Development support library.
   - Structured programming.
   - Top-down programming.

- Chief programmer teams.

c. For testing software:

- Structured walk-throughs.
- Interactive debugging and testing.

The technologies that we have already discussed are:

- Top-down design, as structured design in Section V.B.
- HIPO, in Section V.B.
- Pseudo code, in Section V.C.
- Chief programmer teams, in Section IV.D.

The other programming aids above are briefly discussed in the following paragraphs.

Development Support Library is a formalised procedure for maintaining the documentation that is generated during a system development project.

One example of structured programming was discussed when describing the Jackson method. IBM's methodology uses the same three constructs, namely sequence, selection and iteration, but it places more emphasis on the modular structure of the program than on the program structure as a reflection of the data structure. Top-down programming is the use in the programming process of the same top-down approach that was a characteristic of structured design.

Structured walk-through is a formalisation of the process of desk checking a system at regular intervals during its development. It fits in well with the top-down approach since, at any particular level of detail, the check team can take a complete view of the system, checking it for overall consistency as well as for internal errors. The check team is usually composed of four or five people. These are normally members of the project team, but independent quality assurance staff or senior managers are sometimes used instead. Two rules seem to be important for success. First, the purpose should be to detect errors, not to correct them. Second, the end result should be better performance by the project team, in terms of an error-free system, rather than the exposure of a poor programmer. Interactive debugging and testing involves the use of the computer to generate test data and to allow the programmer to test his work rapidly and effectively. The significance of IPT is that it represents an attempt by IBM to improve all the various aspects of programming. They provide a cheap and risk-free way to improve discipline in programming work. In 1976, Holton (35) conducted a survey of the use of IPT in American installations. He found that the techniques were not widely used. Where they had been tried, both top-down programming and structured programming were mentioned as being moderately effective in increasing programmer productivity. However, structured programming was not regarded as being effective in raising programmer morale.

5. AUTOGEN

AUTOGEN is a COBOL generator developed by the Co-operative Insurance Society, Manchester. The concept grew out of the system specification method that they were using, which required all output data fields to be specified in terms of file data fields or input data fields. They believed that this specification was sufficiently precise to provide the input to a COBOL generator, and so they developed one for their own use.

System procedures are specified in AUTOGEN by systems analysts, and are compiled into

program modules. Once written, these modules are held in a module library for subsequent linking into programs. Program structures are designed and written in Assembler language by programmers.

The method is claimed to give faster system development, with programs that are better structured and easier to maintain. The generated programs generally contain more code, but often they run faster because their basic structure is better. When system amendments are required, the amendments are written by systems analysts in AUTOGEN, and so they are self documenting.

The method has been used successfully for the last six or seven years. During that time the balance of staff has changed. There are now more analysts and fewer programmers, but the present programmers are more skilful than were the previous ones.

6. ADAM

ADAM is a small business computing system offered by the Logical Machine Co. It is marketed as a machine that requires no programming, and it has been developed specifically for the needs of the small business computer user.

ADAM consists of a central processor, 32K bytes of main memory, a 10-megabyte disc, a single VDU, and a 165 cps. matrix printer. A larger disc drive and faster printer may be used with the system, but multi-VDUs are not currently supported.

The most important feature of ADAM is its claimed 'programmerless' environment. The system incorporates a built-in set of approximately forty 'verbs' and 'nouns', and these allow the user to define his own processes and data. ADAM 'learns' the meaning of new words via previously-learned words. A verb is a character string or a subroutine to be executed by the system. A noun is a data field or an information block on which the verbs act. The simplification of the user interface has required a number of trade-offs that can affect system performance. For instance, all verbs and nouns (the dictionary) reside on disc, and the execution of a verb can result in a significant amount of disc access. Frequent disc accesses plus the fairly high processor overhead associated with interpretive list processing can slow up the processing capabilities of the system. Also, the simplicity of the ADAM language is such that instructions to the system can become very verbose. For example, to add two fields and place the result in a third, the 'ADD' verb must be executed, leaving the result in a noun called 'SUM', which must then be moved (by executing the 'MOVE' verb) to the destination field. Additionally, the user has to be capable of performing the logical analysis required to use ADAM.

The prospective ADAM purchaser has to consider two distinct factors:

1. Whether he can write his own ADAM instructions and, if so, whether to do this would be more effective than to use a conventional language (e.g. either BASIC or a specialised language offered by the vendor). If the user's application problem is similar to those of others in his industry, a standard package might be feasible.

2. Whether the hardware will be adequate to meet his needs. The processor will generally be satisfactory, but the available disc space might be restrictive, and interactive facilities are limited.

VI. FINDINGS AND RECOMMENDATIONS

A Principal Findings

The principal findings of our research are:

1. Demand for the development of new systems will increase rapidly over the next five to ten years, due both to growth in the scope of systems and to a widening in the base of users.

   This demand will not be satisfied merely by increasing the resources. Their productivity will also need to be greatly improved.

   The problem will be exacerbated by the need to maintain existing systems, which will absorb an increasing proportion of systems staff.

2. Over 50% of the software cost of a typical system is incurred after the system has been implemented, on correction of errors and on enhancement of the system to meet changing requirements.

3. Several of the new methodologies will improve the productivity of individual aspects of the development process (what we have termed micro-productivity). None of them, however, are of value in improving the overall life-cycle productivity (macro-productivity) because either

   - They affect only a small portion of the life-cycle, or
   - They are not yet sufficiently well developed to be of general use, or
   - They are too complex or too expensive to be of general use.

4. The current emphasis is on shortening the timescale of system development and on reducing its cost. The new methodologies reflect this. The emphasis must be changed so that the whole system life-cycle is considered, not just the development portion.

5. Most new methodologies are based on the traditional staged approach to system development.

   It is likely that the greatest improvements in productivity will be gained by the use of different approaches which will make the new methodologies obsolete.

The implications of these findings are bleak. There are no magical solutions available off the shelf and so the emphasis today must be on making the best use of what is already available. In the remainder of this Section we recommend what should be done now to improve systems productivity and we then discuss the strategic issues that could lead to a solution of the productivity problem in the future.

B Action Required Now

As part of the research upon which this Report is based we carried out interviews with systems managers in both large and small installations. The purpose was to test the claims that were made in the literature in order to verify that they were valid now, in Europe. In many installations the evidence of success was very strong and, in a few, the evidence of failure was equally obvious. Most of the successful installations had several features in common.

These ingredients of success are described below, and no systems manager would dispute their validity. In successful installations they are actually made to work. In unsuccessful installations they are merely paid lip service.

1. Improve the political environment
   Probably the most noticeable difference between successful and unsuccessful systems departments is the political environment in which they operate. Successful departments have a good working relationship with their users, based on a good understanding of their respective roles and a proper definition of and acceptance of their respective objectives. Successful departments exercise strong management control over project work. They identify system requirements clearly and they stick to them. Their users appreciate the strengths and limitations of information technology and they take appropriate account of this in their general management planning. These factors apply irrespective of the nature, extent, complexity or methods of systems work.

   In unsuccessful departments most or all of these factors are missing. As a result, a high premium is placed on political, rather than systems, skills and systems managers spend their time grappling with political problems, rather than with managing systems work.

   Quite clearly, systems managers have the responsibility for creating the right political climate for their work. Any improvement in this climate will improve the motivation of their staff and will help liberate their talents.

2. Concentrate on the real objective
   In Section IV.D we discussed the environmental factors that affect systems staff. The main recommendation is that the underlying objective of their work must be changed from the present internal technical orientation to the need to help their users increase their effectiveness.

   Many systems staff would protest that this is already their objective. We suspect that, all too often, they are merely justifying their true objective of exploring computers. Not that this situation is surprising. If systems staff are prevented from achieving the proper objective because the political environment is poor, they are likely to turn to any alternatives that provide job satisfaction.

3. Manage the projects
   Good project management will ensure that the best results are achieved from the available resources and methods. One effect of introducing formal project management is usually to increase the system development time. Systems managers must ensure that both their staff and their users appreciate the reason. This increase in time and possibly cost will be more than repaid by a reduction in the time required to test and implement the system and by its increased reliability and effectiveness.

4. Emphasise the requirements definition
   The traditional method of developing a system relies for success on a good requirements definition. Usually insufficient attention is given to defining the requirements fully and correctly. This is partly due to the urge to create a finished product, but also to the absence of formal methods with which to do this stage of the work.

Systems analysts, if they are to deserve this title, need to appreciate much more deeply the nature of the requirements definition stage. They also should learn to use the tools that are becoming available, described in Section V.A. It is well established that, unless errors are trapped at this stage, the work of correcting them later is very expensive and may even destroy the basic structure of the system.

5. Design for the user
   Designing the system so that it can be used effectively is just as important as file or program design. Good systems are frequently wasted because insufficient attention is given to the user's aspects.

   Increased trade union involvement in system development may force the profession, belatedly, to give proper attention to these issues.

   This subject will be discussed in Report No. 20, Improving The Interface Between People And Equipment.

6. Design for maintenance
   In our discussion of terotechnology in Section IV.A we described how other industries have reduced their maintenance problem by designing the product for easy maintenance.

   In systems terms this means that the system must have a simple, clear structure, must be built up in discrete modules and must be well documented. The design emphasis must be transferred from questions of run-time or programmer efficiency to maintenance efficiency.

7. Standardise programming
   The emphasis in programming productivity should not be in finding more effective methods of re-inventing the wheel but in using standard wheels. At best this means building up a library of pre-coded modules that can be linked into the finished product. A technique for achieving this is described in Section V.C.

   If pre-coded modules cannot be used then standard program structures should be defined for common tasks, in order to speed up programming and simplify maintenance.

8. Measure performance
   Several of the installations that we visited claimed that they had been able to improve productivity. However, few of them were able to substantiate this claim with quantified data.

   Project management is impossible unless realistic plans are first prepared and performance is then monitored against the plan. The incentive to reduce maintenance will not be apparent unless the cause of maintenance is identified and the cost is measured. Realistic trade-offs cannot be made between increased development and reduced maintenance unless they can be timed and costed. In short, productivity is a worthless concept unless it can be expressed in figures.

C Strategic Issues

There are no magical solutions to the productivity problem available now, so that we are forced to apply the more obvious, prosaic approaches described in VI.B. Nonetheless, the systems world is changing fast, and systems managers should be taking account of these trends in their strategic planning. They should also be demanding that more attention be given to these issues at national, supplier, user group and individual installation levels. Our main recommendations are:

1. Raise the level of expertise
   We suggested in Section II.E that system development work is likely to polarise into more technical activities on the one hand, and more user-orientated activities on the other. To control and exploit this polarisation, systems managers must raise the level of expertise in their departments in both types of activities.

   They need to extend their understanding of the factors to be considered in selecting a DBMS, the basic architectures of databases and the implications of designing database systems. Any competent manager should have this level of understanding even though he has no intention, at present, of setting up a database. Without such an understanding he cannot decide rationally not to set up a database. Also, he cannot decide when to start emphasising data analysis and data structure design in the current system development work. These issues will be discussed in the forthcoming Report No. 12, The Future of Database Management Systems.

   Developments in networking facilities and software provide a second example. They were discussed in Report No. 1, Developments in Data Networks.

   The social, ergonomic and industrial relations aspects of system development will assume much greater importance as computing penetrates right into the user's work area.

   Managers need to be realistic, of course, and they need to remember Nolan's Stage Hypothesis, described in Section II.C. They should not attempt to apply stage 4 techniques if they are still in either stage 1 or stage 2. But they must have sufficient knowledge to understand the new techniques.

   At the staff level, a typical systems analyst or programmer will need to learn a continually changing range of skills during his career. It is always difficult to keep up to date in a busy installation, but it is in his manager's and his own interest that he should stay intellectually alive and able to absorb new concepts.

2. Plan for organisational change
   One effect of the polarisation of system development work will be that the work of developing new applications will tend to pass back to the users, with the systems department being responsible for creating and maintaining the data resource and the communication network.

   This change will not happen without the proper training and preparation of both user and systems staff. It should not be allowed to happen in a piecemeal, unplanned way. Again, it is a factor to consider in the preparation of strategic plans.

3. Promote the use of computer aided development tools
   The concepts upon which, for example, the ISDOS project is based (see Section V.A), need to be turned into reality, into tools that typical installations can easily learn to use without unjustified expense. Computers are used to help design cars, they should be used to help design systems. Word processors are used to help manipulate legal specifications, they should be used to help manipulate system specifications. A systems department employs people and generates cost just like an accounts department. The suppliers of equipment should be urged to offer hardware and software solutions to system development problems, as well as to 'user' problems.

4. Promote the development of a market for program packages
   It clearly makes sense to share the development and maintenance cost of a system, whenever possible, by the use of a program package. At present in the UK and Europe it is not easy to take advantage of this approach. This is partly because the range of packages available is restricted to the most common applications, and partly because of the attitude of systems managers and staff to the use of packages, as discussed in Section IV.F. Systems

managers should appraise the use of packages more realistically and they should make their requirements known to potential suppliers so that a more mature market can develop.

The idea of using a library of standard modules should be extended from the installation to the national level. An attempt to set up a national program library in the USA has largely failed, but this was due more to lack of commitment by the library's users than to technical reasons. The Japanese are planning to create a national program library. The possibility of a British program library should be explored.

5. Promote the development of analysis methods
Of all the activities in the craft of systems work, the most primitive, the most casual and the least understood is the analysis of requirements. The importance of getting the system requirements right is quite clear. What we now need is methods to achieve this.

Installations should encourage and support research projects directed at this subject, such as the Lancaster Approach, ISDOS and the work of Lehmann at Imperial College.

In addition, there is a need to ensure that the work being done within the UK and Europe is, as far as possible, co-ordinated and relevant to end users. Probably only Government bodies would be able to do this.

6. Increase the provision of staff
Systems work must be the only activity that claims to be a profession but yet accepts entrants with no qualifications and allows them to call themselves experts after two years experience.

A much greater flow of entrants is required with post-graduate qualifications in computing. Only in this way will the systems worker's self-perception as a gifted amateur be overcome.

Systems work will never become more than a craft if every entrant has to start from scratch, with no basic understanding of the subject.

7. Promote system life-cycle concepts
The UK Department of Industry has already created a terotechnology group to promote improvement in the productivity of physical assets. We suggest that this initiative should be extended to include the productivity of information systems.

8. Focus on the fundamental issues
The field of information technology is developing very rapidly. Much that is happening is irrelevant to the main issues that have been discussed in this Report. Nevertheless, we are sure that significant developments will occur, within the next few years, that will lead to a rapid increase in systems productivity.

Given the extent of the productivity problem, it is essential that any new methods and tools that do provide a solution to the problem should be put to use rapidly. If this is to occur, system managers must be aware of them as soon as their value has been demonstrated. An increased level of awareness is therefore required.

The increased awareness will be difficult to achieve because of the great volume of information that is generated. No-one, even with a starting qualification and continuous training, will be able to keep up with all the changes and, at the same time, do the job that they are paid for. There is a need for a filter that removes the irrelevant matter and exposes those issues that are of fundamental importance. The Butler Cox Foundation will continue to focus on these critical issues.

BIBLIOGRAPHY

1. Alford M. — 'A requirements engineering methodology for real-time processing requirements', TRW-55-76-07, 1976.

2. Baker F.T. — 'Chief programmer team management of production programming', IBM Systems Journal Vol.11, No.1, 1972.

3. Baker F.T. and Mills H.D. — 'Chief programmer teams', Datamation, Dec. 1973.

4. Belady L.A. and Lehmann M.M. — 'A model of large program development', IBM Systems Journal Vol.15, No.3, 1976.

5. Bell T.E. and Thayer T.A. — 'Software Requirements: are they really a problem?', TRW-55-76-04, 1976.

6. Benjamin R.I. — 'Control of the information system development cycle', Wiley and Sons, 1971.

7. Blosser P.A. — 'An automatic system for application software generation and portability, Aug. 1976, Purdue University.

8. Boehm B.W. — 'Software and its impact: a quantitative assessment', Datamation, May 1973.

9. Boehm B.W. — 'Software Engineering', IEEE Transactions on Computers, Vol.25, No.12, Dec.1976.

10. Boehm B.W. — 'Some steps toward formal and automated aids to software requirements analysis and design', Proc. IFIP Cong., 1974.

11. Boehm B.W., Brown J.R. and Lipow M. — 'Quantitative evaluation of software quality', Proc. 2nd Int. Conf. on Software Engineering, 1976.

12. Brooks F.P. — 'The mythical man month: essays on software engineering', Addison-Wesley, 1975.

13. Brown R.R. — 'The technique and practice of a structured design a la Constantine', Proc. Infotech Conf. on 'Structured Design', 1977.

14. Canning G. — 'Improving the System Building Process' EDP Analyser Vol.12, No.12, Dec.1974.

15. Canning G. 'Structuring of EDP Projects', Data Processing, July-Aug. 1975.

16. Canning G. 'Are we doing the right things?', EDP Analyser, Vol.13, No.5, May 1975.

17. Central Computer Agency 'Estimation, planning and control of programming activities', Guide No. 3, HMSO 1974.

18. Central Computer Agency 'Program Validation', Guide No. 9, HMSO 1978.

19. Charman J. 'Effective project costing and killing techniques', AMAS Conference, 1971.

20. Checkland P.B. 'Towards a system-based methodology for real-world problem solving', Journal of System Eng., Vol. 3, No. 2, 1972.

21. Chrysler E. 'Some basic determinants of computer programming productivity', CACM, Vol. 21, No. 6, June 1978.

22. Collins J.H. 'The application of the systems approach to the design of computer based data processing systems', Journal of Systems Eng., Vol. 4, No. 2, 1976.

23. Cougar J.D. and Knapp R.W. 'Systems Analysis Techniques', Wiley and Sons, 1974.

24. Department of Industry 'Life cycle costing in the management of assets: a practical guide', HMSO 1976.

25. Department of Industry 'Terotechnology handbook', HMSO, 1977.

26. Dolotta T.A. et al 'Data Processing in 1980-85', Wiley-Interscience, 1976.

27. Duke M.O. 'Testing in a complex systems environment' IBM Systems Journal, Vol. 14, No. 4, 1975.

28. Fagan M.E. 'Design and Code Inspection to reduce errors in program development', IBM Systems Journal, Vol. 15, No. 3, 1976.

29. Frank W.N. 'The ten great software myths', Computerworld, March 1978.

30. Gildersleeve T.R. 'Optimum Program structure documentation tool', Journal of Systems Management, March 1978.

31 Grindley C.B.B. 'Systematics – a non-programming language for designing and specifying commercial systems for computers', Computer Journal, Aug 1966.

32. Grindley C.B.B. 'The Role of the Trigger in Systematics' to be presented IFIPS Conference, Oxford, April 1979.

33. Hamilton M. and Zeldin S. 'Higher Order Software — a methodology for defining software', IEEE Transactions on Software Engineering, Vol. 2, No. 1, March 1976.

34. Hansen P. and Penney G. 'Job Trends in Data Processing', NCC, 1978.

35. Holton J.B. 'Are the new programming techniques being used?' Datamation Vol. 23, No. 7, July 1977.

36. Infotech 'Software Engineering Techniques', State of the Art Report, 1977.

37. Infotech 'Software Reliability', State of the Art Report, 1977.

38. Infotech 'Structured Analysis and Design', State of the Art Report, 1978.

39. Inman W. 'An example of structured design', Datamation, Vol. 22, No. 3, March 1976.

40. Jackson M.A. 'Principles of program design', Academic Press, 1975.

41. Johnson J.R. 'A working measure of productivity', Datamation, February 1977.

42. Jones C. 'Optimising Program Quality and Programmer Productivity', IBM TR 02.764, Jan. 1977.

43. Leavenworth B.M. 'Non-procedural data processing', Computer Journal, Vol. 20, No. 1, Feb. 1977.

44. Lehmann M.M. 'Laws of program evolution — rules and tools for programming management', Infotech State of the Art Conf. 'Why Software Projects Fail', April 1978.

45. Liu C.C. 'A look at software maintenance', Datamation, Nov. 1976.

46. Love T. 'Software Psychology: shrinking life-cycle costs'.

47. Marker L.R. 'Software Requirements Engineering', Infotech Conference, 'Structured Design', 1976.

48. Martin G.N. 'Managing Systems Maintenance', Journal of Systems Management, July 1978.

49. Mills H.D. IEEE Trans. Software Eng., Dec. 1976.

50. National Computing Centre 'Approaches to Systems Design', 1974.

51. Nolan R.L. 'Managing the Computer Resource: a stage hypothesis', CACM, Vol. 16, No. 7, July 1973.

52. Nunamaker J.F. Jnr. and Konsynski B.R. 'From Problem Statement to automatic code generation', Systemeering, March 1975.

53. Nunamaker J.F. Jnr. and Konsynski B.R. 'Computer-aided analysis and design of information systems', CACM, Vol. 19, No. 12. Dec. 1976.

54. Peeples D.E. 'Measure for Productivity', Datamation, May 1978.

55. Peters L. 'Managing the transition to structured programming', Datamation, May 1975.

56. Peters L.J. and Tripp L.L. 'Is software design wicked?', Datamation, May 1976.

57. Peters L.J. and Tripp L.L. 'Comparing Software Design Methodologies', Datamation, Nov. 1977.

58. Putnam L.H. 'The influence of the time-difficulty factor in large-scale software development'.

59. Ramanoorthy C.V. and Ho S.B.F. 'Testing large software with automated software evaluation systems', IEEE Trans. Software Eng., March 1975.

60. Randell B. 'System structure for software fault-tolerance', IEEE Trans. Software Eng., June 1975.

61. Remus H. 'Directions for the applications of structured methodologies', AIIE Conference on Structured Systems, July 1978.

62. Robinson D.G. 'Beyond the Four Stages: What Next?' DP Management Co., Aug. 1977.

63. Ross D.T. and Schaman Jnr. 'Structured Analysis for requirements definition', Proceedings of 2nd Int. Conference on Software Engineering, Oct. 1976.

64. Scharer L.L. 'Improving System Testing Techniques', Datamation, Sept. 1977.

65. Stay J.F. 'HIPO and integrated program design', IBM Systems Journal, Vol 15, No. 2, 1976.

66. Stevens W.P., Myers G.J. and Constantine L.L. 'Structured Design', IBM Systems Journal, Vol. 13, 1974.

67. Teichroew D. and Sayani H. 'Automation of System Building', Datamation, August 1971.

68. Teichroew D. and Hershey E.A. 'Computer-aided structured documentation and analysis of information processing system requirements', Share Conference, Aug. 1976.

69. Teichroew D., Hershey E.A. and Yomonoto Y. 'Computer Aided software development', Feb. 1977.

70. Teichroew D. and Hershey E.A. 'PSL/PSA: a computer aided technique for structured documentation and analysis of information processing systems', IEEE trans

Software Eng. Proc., Jan. 1977.

71. Thurner R. Dr. 'System Development Technology', Proc. BCP Management Conference, Oct. 1978.

72. Van Leer P. 'Top-down development using a program design language', IBM Systems Journal, Vol. 15, No. 2, 1976.

73. Walsh D.A. 'Structured Testing', Datamation, Vol. 23, No. 7, July 1977.

74. Walston C.E. and Felix C.P. 'A method of programming measurement and estimation', IBM Systems Journal, Vol. 16, No. 1, 1977.

75. Weinberg G.M. 'The psychology of improved programming performance', Datamation, Nov. 1972.

76. Welke L. 'The million dollar sale packages', Computing, July 19, 1973.

77. Wolverton R.W. 'The cost of developing large-scale software', IEEE Transactions on Computers, Vol. 23, No. 6, 1974.

Abstract

Report Series
No 11

Improving Systems' Productivity

by Tony Brewer
February 1979

Both the users and the providers of information systems are already faced with the problem that the demand for systems is greater than the supply. There is already a backlog of applications waiting to be developed. This backlog will grow, due partly to a widening of the base of users and partly to an increase in the scope of information systems brought about by convergence of computing, communications and office automation.

Supply will not be increased to satisfy demand merely by increasing the resources. The productivity of those resources must be dramatically increased. This Report is concerned with the ways in which this increase in productivity might be achieved.

The main theme of the Report is that all attempts to increase systems productivity made so far have focussed too narrowly on individual aspects of system development work. We believe that the productivity problem can only be solved by taking a much broader view of the whole system life-cycle.

We discuss various approaches to improving productivity, including project management, environmental factors, data management, application packages, non-traditional system development methods and the role of the system user. Section V of the report contains a critical review of fourteen methods or tools that are claimed to improve productivity.

Finally, the Report recommends action that should be taken now to improve productivity and identifies issues that need to be considered from a strategic point of view.

The Butler Cox Foundation is a research group which examines major developments in its field – computers, telecommunications, and office automation – on behalf of subscribing members. It provides a set of 'eyes and ears' on the world for the systems departments of some of Europe's largest concerns.

The Foundation collects its information in Europe and the US, where it has offices through its associated company. It transmits its findings to members in three main ways:

- As regular *written reports,* giving detailed findings and substantiating evidence.

- Through *management conferences,* stressing the policy implications of the subjects studied for management services directors and their senior colleagues.

- Through *professional and technical seminars,* where the members' own specialist managers and technicians can meet with the Foundation research teams to review their findings in depth.

The Foundation is controlled by a Management Board upon which the members are represented. Its responsibilities include the selection of topics for research, and approval of the Foundation's annual report and accounts, showing how the subscribed research funds have been employed.