Report Series
No. 36

Cost-effective Systems Development and Maintenance

August 1983

The Butler Cox Foundation

THE BUTLER COX FOUNDATION

REPORT SERIES NO. 36

COST-EFFECTIVE SYSTEMS DEVELOPMENT AND MAINTENANCE

ISSUED AUGUST 1983

Abstract

Both the nature and the scope of most organisations' computer applications will expand dramatically over the next three to five years. The pressure on management services staff is already here: in many organisations, users are demanding that flexible new systems be developed urgently.

In working towards cost-effective systems there is a bewildering choice of approaches, methods and tools. There are too many alternatives for most organisations to investigate, let alone install and use.

The purpose of this report is to identify the key factors that influence systems cost-effectiveness, and to offer guidance to Foundation members on how 'cost-effective' systems can be achieved.

The report concludes that there are opportunities to significantly improve the cost-effectiveness of systems development and maintenance, if a co-ordinated programme of action is undertaken.

Research team

The team that researched and wrote this report was:

David Flint, a consultant with Butler Cox who has considerable experience of commercial systems development. He was the author of Foundation Report No. 12 — Trends in Database Management Systems — and has carried out numerous systems studies.

Rob Moreton, a consultant with Butler Cox specialising in systems development and data processing management. He has contributed to the research programmes of several Foundation reports, and was responsible for arranging the programme for two recent Foundation conferences that dealt with systems productivity. He has also lectured extensively on these topics.

Chris Woodward, a consultant with Butler Cox specialising in information systems and the management of computing projects. He has extensive experience of systems development and maintenance and has carried our numerous projects involving both large mainframe computers and smaller distributed systems.

Photoset and printed in England by Flexiprint Ltd., Worthing, Sussex.

THE BUTLER COX FOUNDATION

Butler Cox & Partners
Butler Cox is an independent management consultancy and research organisation, specialising in the application of information technology within commerce, government and industry. The company offers a wide range of services both to suppliers and users of this technology. The Butler Cox Foundation is a service operated by Butler Cox on behalf of subscribing members.

Objectives of The Foundation
The Butler Cox Foundation sets out to study on behalf of subscribing members the opportunities and possible threats arising from developments in the field of information systems.

The Foundation not only provides access to an extensive and coherent programme of continuous research, it also provides an opportunity for widespread exchange of experience and views between its members.

Membership of The Foundation
The majority of organisations participating in the Butler Cox Foundation are large organisations seeking to exploit to the full the most recent developments in information systems technology. An important minority of the membership is formed by suppliers of the technology. The membership is international with participants from Belgium, Denmark, France, Italy, the Netherlands, Sweden, Switzerland, the United Kingdom and elsewhere.

The Foundation research programme
The research programme is planned jointly by Butler Cox and by the member organisations. Half of the research topics are selected by Butler Cox and half by preferences expressed by the membership. Each year a short list of topics is circulated for consideration by the members. Member organisations rank the topics according to their own requirements and as a result of this process, members' preferences are determined.

Before each research project starts there is a further opportunity for members to influence the direction of the research. A detailed description of the project defining its scope and the issues to be addressed is sent to all members for comment.

The report series
The Foundation publishes six reports each year. The reports are intended to be read primarily by senior and middle managers who are concerned with the planning of information systems. They are, however, written in a style that makes them suitable to be read both by line managers and functional managers. The reports concentrate on defining key management issues and on offering advice and guidance on how and when to address those issues.

Additional report copies
Normally members receive three copies of each report as it is published. Additional copies of this or any previous report (except those that have been superseded) may be purchased from Butler Cox.

Previous reports
No. 1 Developments in Data Networks
No. 2 Display Word Processors*
No. 3 Terminal Compatibility*
No. 4 Trends in Office Automation Technologies
No. 5 The Convergence of Technologies
No. 6 Viewdata*
No. 7 Public Data Services
No. 8 Project Management
No. 9 The Selection of a Computerised PABX
No. 10 Public On-line Information Retrieval Services*
No. 11 Improving Systems' Productivity
No. 12 Trends in Database Management Systems
No. 13 The Trends in Data Processing Costs
No. 14 The Changing Equipment Market
No. 15 Management Services and the Microprocessor
No. 16 The Role of the Mainframe Computer in the 1980s
No. 17 Electronic Mail
No. 18 Distributed Processing: Management Issues
No. 19 Office Systems Strategy
No. 20 The Interface Between People and Equipment
No. 21 Corporate Communications Networks
No. 22 Applications Packages
No. 23 Communicating Terminals
No. 24 Investment in Systems
No. 25 System Development Methods
No. 26 Trends in Voice Communication Systems
No. 27 Developments in Videotex
No. 28 User Experience with Data Networks
No. 29 Implementing Office Systems
No. 30 End-User Computing
No. 31 A Director's Guide to Information Technology
No. 32 Data Management
No. 33 Managing Operational Computer Services
No. 34 Strategic Systems Planning
No. 35 Multifunction Equipment
**These reports have been superseded.*

Future reports
No. 37 Expert Systems
No. 38 Selecting Local Network Facilities
No. 39 Trends in Information Technology
No. 40 New Ways of Presenting Information

THE BUTLER COX FOUNDATION

REPORT SERIES NO. 36

COST-EFFECTIVE SYSTEMS DEVELOPMENT AND MAINTENANCE

CONTENTS

THE BUTLER COX FOUNDATION

REPORT SERIES NO. 36

COST-EFFECTIVE SYSTEMS DEVELOPMENT AND MAINTENANCE

REPORT SYNOPSIS

Both the nature and the scope of most organisations' computer applications will expand dramatically over the next three to five years. The pressure on management services staff is already here: in many organisations, users are demanding that flexible new systems be developed urgently.

Responding to these demands means selecting methodologies, taking advantage of appropriate available aids and, in various ways, improving development productivity. It does not mean simply increasing the number of people engaged on the development: for most organisations this is not possible and, even if it were possible, it would not be advisable.

Development and maintenance are the two parts of the complete system life-cycle. They are strongly interrelated, since false economy in development can lead to substantial extra costs in maintenance.

Modern systems development practices aim to achieve reduced maintenance costs through better structured and documented systems and automated procedures. These procedures may increase development costs, but can result in lower maintenance costs and hence improved overall life-cycle cost-effectiveness. This is important since maintenance typically accounts for over 50 per cent of system cost.

In working towards cost-effective systems there is a bewildering choice of approaches. System developers are offered project management disciplines, structured analysis and design methods, structured programming, application generators, application packages, development toolkits, novel operating systems and end-user computing — too many alternatives for most organisations to investigate, let alone install and use.

Large computer users can benefit from one or more of these alternatives. But the uncritical adoption of a single solution can be no panacea for development/maintenance problems. Choosing the right approach is the subject of chapter 4 of the report.

The traditional staged development approach is thorough, but cumbersome and slow. It is regulated by formal standards, and supported by a limited number of automated tools. Three non-traditional approaches — collaborative, iterative and end-user development — have emerged in attempts to overcome the traditional shortcomings.

In collaborative development the system requirements and design are developed by a team of users in close association with professional systems staff. This takes time, but it enhances user commitment and applies user knowledge directly.

In iterative development a prototype of the system is built, using advanced tools, and is then refined to form the operational system. End-user development (reviewed in Foundation Report No. 30) embraces a variety of approaches in which users develop their own systems.

Managers should accept that the non-traditional approaches are here to stay and are likely to contribute more extensively in the future. Projects can be matched to the appropriate approaches on the basis of a number of key characteristics: commonality of requirements, generality, impact on the business, complexity of requirements, performance requirements and clarity of requirements.

Whatever approach is adopted, the active participation of users in the development process should be encouraged. Users must be able to identify their own requirements, and prototyping — the building of a working model to test assumptions — is increasingly used for this.

Prototypes can be used for various types of system, provided that appropriate software tools are available. They are well suited to small business applications such as stock control and decision support systems; and less suited to large, complex systems.

Five examples of methodologies that are affecting systems development are described: data analysis methodology, systems development methodology, structured analysis and design, information systems work and analysis of change, and PRIDE/ASDM. These are outlined in chapter 5 of the report, which also discusses the effects of increasingly automating systems work and the use of systems building tools.

These tools include system generators, language-independent program generators, DBMS-based tools, integrated toolkits and discrete tools such as query languages.

The use of computer-based tools, together with improved access to computers, is leading to the integration of the separate tasks of the analyst, designer and programmer. Program code may be generated by a wider variety of people, and once again it will be feasible to combine the roles of analyst and programmer. This change in the role of systems staff will be a main feature of data processing over the next five years.

Another change in role concerns the relationship between systems staff and systems users. Systems staff bridge the gap between the user and the machine. But the gap-bridging becomes easier as the machine facilities expand. Sometimes the gap can be closed by providing users with appropriate computer-based tools, whereupon the role of the systems staff becomes that of identifying how the machine facilities can best be exploited by the users.

How can cost-effectiveness in systems development and maintenance be measured and improved? Measures of effectiveness (in terms of timeliness, quality and quantity) and efficiency (in terms of staff performance, equipment, methods and costs) form a useful structure for managing performance. Managers should establish formal programmes to monitor the appropriate performance data; systems management by intuition is no longer adequate.

Certainly there are difficulties in systems performance measurement — the "product" may be intangible, different phases are interdependent, the creative process does not lend itself to measurement, and comparisons may mislead if the products are dissimilar. But these difficulties can be overcome if well-defined measures or metrics are established. Appropriate measures for systems, equipment, personnel and projects are discussed in chapter 2.

We find that improvements in cost-effectiveness can best be achieved through a broadly based improvement programme. This should consider such factors as project management, environmental (personnel) factors, data management, new methods and tools, application packages, new approaches to development and the role of the user.

Work attitudes of systems staff have a strong influence on productivity, as we noted in an earlier Foundation Report. And work attitudes are themselves strongly influenced by organisational factors. Among the many organisational factors reviewed in chapter 3 are the importance of assigning high management priority to staff motivation; the value of inspection procedures; and the benefits of organising maintenance as a separate function.

In the final chapter of the report we give guidelines for management aimed at promoting cost-effective system development and maintenance. Strategically, the choice of system to be developed in the first place is a critical factor.

At departmental level, our recommendations include:

—Implement training programmes to improve staff skills and motivations.

—Introduce system metrics to provide objective measurement of performance.

—Introduce quality assurance procedures.

—Adopt automated project management aids for large and complex projects.

—Increase the degree of automation within the data processing department.

Among the guidelines given for the project level are:

—Adopt collaborative, iterative and end-user development approaches.

—Break long-delivery projects into smaller elements.

—Raise productivity by using off-the-shelf software.

—Adopt formal methodologies such as data analysis.

—Cut lead times by adopting system-building tools.

Using the approaches outlined in the report in the context of a co-ordinated programme, we conclude, there is no reason why most data processing departments cannot double their development and maintenance productivity in three to four years' time, and raise it by a further 400 per cent in six to eight years.

PREFACE

Before a new application system is developed, several crucial decisions need to be taken. These include the method and the tools that will be used to develop the system. Also, in recent years, it has become increasingly relevant to consider the type of staff (systems developer and user) involved in the development process. These early decisions have far-reaching effects on system costs — yet they are often ignored.

Intended readership

The report is intended for the executive responsible for an organisation's systems function. It will also be of value to systems development managers, senior systems staff, and managers of user departments which are closely involved with systems development and maintenance.

Our approach to the research

The research contained three main elements:

- Interviews with suppliers, progressive user organisations and industry experts.
- Working group meetings with Dutch Foundation members.
- A study of the literature on cost-effective systems development and maintenance.

Purpose of the report

We approached the research with three main objectives:

- To identify the key factors that influence cost-effectiveness.
- To evaluate the success of organisations in exploiting these factors.
- To give guidance to Foundation members on how to improve the cost-effectiveness of their systems development and maintenance processes.

Structure of the report

Chapter 1 reviews the broad issues of cost-effective development and maintenance and identifies the requirements for improving cost-effectiveness.

Chapter 2 discusses the issues of efficiency and effectiveness in relation to system processes, and reviews the factors that affect cost-effectiveness.

Chapter 3 discusses the potential contribution of improvements to the organisational environment.

Chapter 4 evaluates alternative approaches to systems development and maintenance and identifies criteria that can be used in the evaluation.

Chapter 5 evaluates alternative methods which can be used to support the different approaches, and discusses the use of enhanced computer access and software tools.

Chapter 6 identifies potential future developments and indicates the main conclusions drawn from our research.

CHAPTER 1

INTRODUCTION

A recurring theme of the Foundation's work has been systems development productivity. Foundation Report No. 11 — Improving Systems Productivity — examined the overall issues of productivity in the system life-cycle. Foundation Report No. 25 — System Development Methods — reviewed the methods that organisations can use for developing systems. For many organisations, the question of how to develop and maintain systems in the most efficient and cost-effective manner remains a crucial issue. Data processing departments are experiencing growing pressures from users to provide flexible systems quickly. These pressures emerge as users become aware of new opportunities, often through direct contact with computer suppliers.

Decisions about the method or methodology that will be used to develop a new system, and about the tools and techniques that will be used to aid the development process, have far-reaching effects on the development and subsequent maintenance costs. Yet the factors that affect the development process are not widely understood.

In this chapter, we describe the significant factors that can contribute to cost-effective systems development and maintenance. We begin by explaining what we mean by cost- effective systems development and maintenance, and other relevant terms. We go on to describe the nature of the problem because, for most information systems managers, both development and maintenance present problems.

THE MEANING OF COST-EFFECTIVE SYSTEMS DEVELOPMENT AND MAINTENANCE

Systems development and maintenance are terms that mean different things to different people. So we begin with some definitions.

Systems development and maintenance

We define systems development as that part of the system life-cycle which precedes the changeover to a new system. (The system life-cycle spans all stages from project initiation to system replacement. It encompasses both systems development and systems maintenance.)

Systems maintenance is that part of the system life-cycle which follows changeover and which changes the operational system. We define systems maintenance as the process of modifying an existing operational system while leaving its primary function intact.

Of these two definitions, it is the second — system maintenance — that is the more controversial. By way of amplification, the following activities all fall within the scope of our definition:

- Redesign and redevelopment of parts of an existing system.
- Design and development of an interfacing system which requires some redesign of an existing system.
- Modification of the systems documentation, file structure or programming code.

It is useful to categorise systems maintenance, so that valid comparisons can be made between different kinds of maintenance work. There are two main categories:

- System enhancements, which change and improve the system functions.
- System repair, which leaves the system functions unchanged.

System repair can itself be further subdivided into:

- Corrective maintenance, resulting from processing, performance or implementation failures.
- Adaptive maintenance, resulting from changes in configuration, input or data files.
- Perfective enhancements, to improve performance or maintainability.

Approaches, methods, techniques and methodologies

Before discussing what we mean by cost-effective systems development and maintenance, it is useful to explain some further terms: approaches, methods, techniques and methodologies. Here our definitions are consistent with those given in Foundation Report No. 25 — System Development Methods.

System development approaches

An approach provides a general direction for doing something. In system development, an approach provides a general framework within which development is carried out, and this framework is based on fundamental beliefs. These beliefs may be axiomatic in that they do not necessarily have to be proven. A hierarchy of system development approaches can be constructed, based on the orientation of a particular set of approaches. (For example, all structured approaches aim to derive the system by examining its structure. These approaches can be subdivided according to the criterion used to derive the structure.)

System development methods

A method is an orderly arrangement of ideas that aids a particular activity (such as system design or system analysis). A method usually contains an inherent logical assumption, and is based on a theoretical concept. Thus, a system development method is used to practise a system development approach. (Indeed, a system development approach cannot be practised without a system development method.) Some system development methods can be used in several system development approaches.

System development techniques

A technique provides a predominantly mechanical way of doing something. System development techniques therefore provide the detailed guidelines for using a system development method, and such a technique will often require that a specific tool be used. For example, documentation methods that are based on the assumption that system design should be represented in pictorial and diagrammatic format require that both techniques and tools be used in order to draw the diagrams.

System development methodologies

A system development methodology is a collection of interconnecting methods and techniques, normally within the framework of an approach. A methodology represents a packaging of practical ideas and practices for a given area of activity. As an example, within the structured approach, programming methodologies and system development methodologies have been developed.

Defining cost-effective systems development and maintenance

Effectiveness is a measure of usefulness, or value.

The term cost-effectiveness is a measure of value for money. When something is cost-effective, it represents good value for money. Value for money can be improved either by increasing effectiveness for a given cost, or by reducing cost for a given level of effectiveness.

It is important to realise that effectiveness relates to the systems process, which is the means by which the system is produced, and not to the product itself, which is operated to achieve a business objective. It is also important to note that effectiveness is not the same as efficiency. Efficiency measures the consumption of resources during a process. (Improving staff productivity is a means of improving efficiency, but not necessarily a means of improving effectiveness.)

Having explained what we mean by systems development and maintenance, and by the term cost-effectiveness, we are now in a position to state the meaning of cost-effective systems development and maintenance. Put simply, it means achieving good value for money during the process of systems development and systems maintenance undertaken throughout the system life-cycle, from inception to replacement.

Cost-effectiveness can be improved either by increasing the development cost — if this results, say, in greater reliability of the operational system — or by decreasing the development cost, if what the user really needs is a simple system that satisfies only (say) 80 per cent of his requirements. It follows from this view that the biggest contribution to systems cost-effectiveness comes from the correct decision on which system to build. This issue falls outside the scope of this report. In this report we identify the most appropriate development approaches to adopt on the assumption that the choice of system has already been made.

Finally, it is worth noting that the early stages in a project are the most critical because:

- —Most of the problems that arise in the system maintenance stage can be traced back to earlier stages, especially to requirements definition.
- —The cost of correcting errors is much higher during the later stages of a project (see Figure 1).

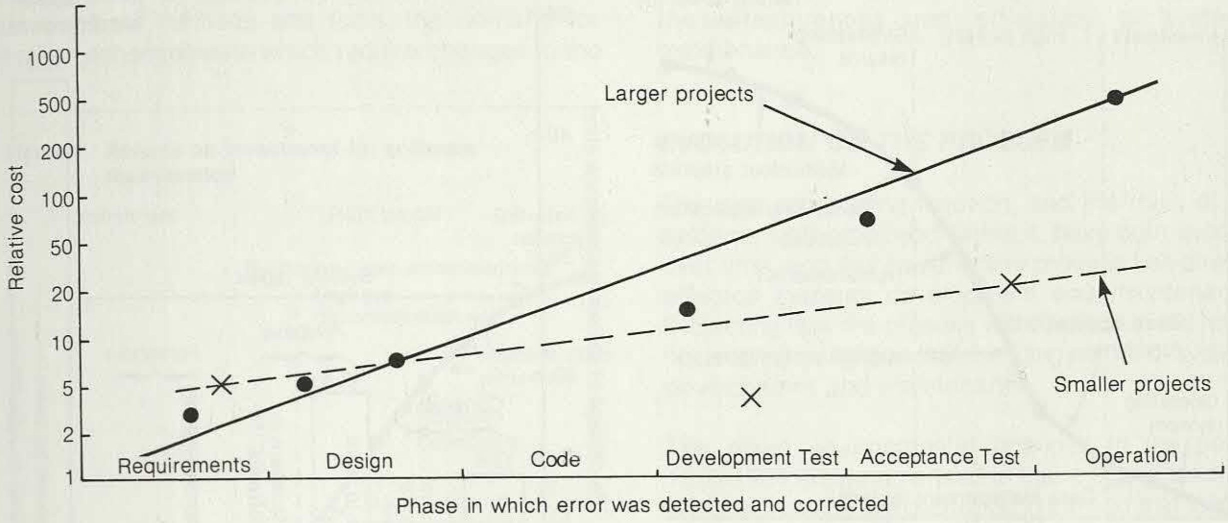
SYMPTOMS OF THE PROBLEM

In this section we consider the pressures that are being exerted on data processing departments to produce cost-effective systems. These pressures are forcing senior managers to look critically at the effectiveness and efficiency of their approaches to both systems development and maintenance.

Systems development

Both the nature and scope of most organisations' computer applications will expand dramatically over the next three to five years. Today, computers are being applied to a variety of management information systems. At the same time the widespread use

Figure 1 Increase in the cost of system changes throughout the life-cycle



(Source: Boehm (1980))

of computers in office and production environments is opening up a whole new range of applications. Improved user education and awareness, and increased sales activity by suppliers, is reinforcing this trend. In many organisations, frustrated users are making continual demands on overstretched management services staff for the development of new systems. There are three ways to solve this problem.

One way is by improving the efficiency (productivity) of systems development undertaken by professional systems staff. But high levels of productivity are difficult to achieve on large-scale projects because of communication and integration problems. Productivity aids are available, but they are relatively expensive and may not be justifiable on small-scale projects. In most organisations, systems staff will continue to play the major part in the development (and maintenance) process. Often there is a long elapsed time from user request to successful operation (many months is typical and several years not unusual). Elapsed time may be reduced by carrying out some sub-stages in parallel but this must be planned and controlled very carefully or inconsistencies and further delays may result.

A second way of solving the problem is by exploiting re-usable software (standard code or application packages). Suppliers are offering standard packages for a growing range of applications. The two main advantages of the package approach are that implementation can be quick, and at a known cost. The main disadvantage is that the package is unlikely to meet precisely the users' needs. Users must decide whether the benefits outweigh the shortcomings.

A third way of solving the problem is to encourage users to develop some of their own systems. This solution falls beyond the scope of this report. (It is worth noting, however, that suppliers are improving their products' ease of use, so that end users can construct some of their own solutions with the minimal involvement of system staff.)

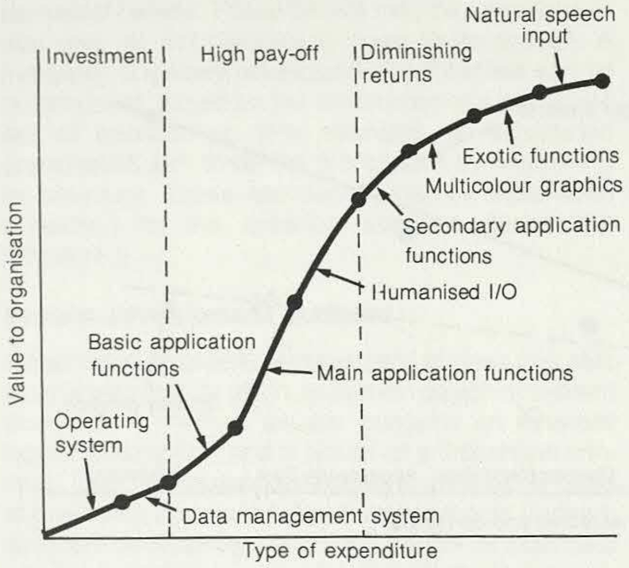
One of the major factors that inhibits cost-effective development for users is the poor selection of applications. Historically, the data processing department has acted as a reviewing body for computer projects, although end users are increasingly developing many of their own systems. This trend could lead to more relevant systems being developed. But there is also a danger of effort being wasted on ineffective systems because of the inexperience of end users in selecting projects for computerisation. Figure 2, overleaf, provides a rough guide to evaluating a system's effectiveness and its development costs.

The evaluation is based on the return on investment which is provided to the user by particular system functions. For instance, the operating system is an investment feature: it has to be financed, but it provides few direct benefits for the user. Application functions can be regarded as having a high pay-off: for relatively small incremental investments, they result in relatively large returns. The return on investment begins to diminish beyond a certain point, when the incremental cost of system development exceeds the incremental value to the user.

Systems maintenance

It is important to distinguish between the cost-effectiveness of systems development, and cost-effectiveness over the system life-cycle. Reduced develop-

Figure 2 Cost of producing the facility



(Source: adapted from Boehm (1981))

ment costs do not necessarily lead to improved life-cycle cost-effectiveness. Modern systems development practices emphasise a reduction in maintenance costs through better structured and documented systems, automated procedures and so forth. These procedures may actually increase development costs. But they could result in lower maintenance costs, and hence improved life-cycle cost-effectiveness (particularly for long life-cycle systems).

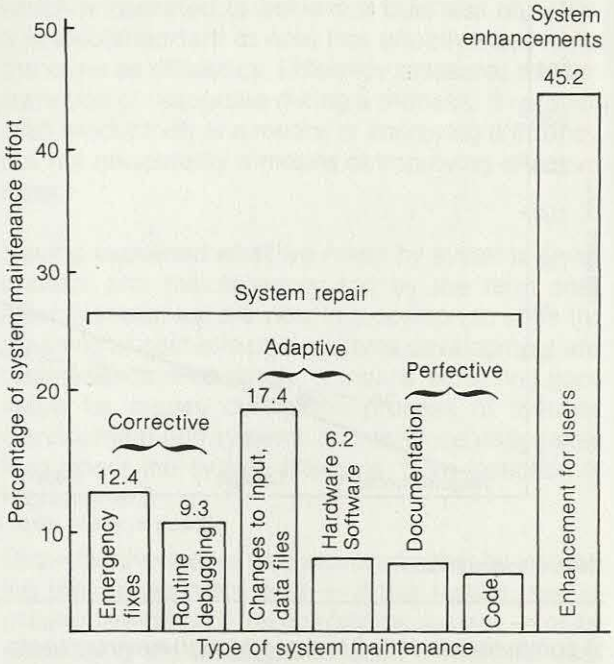
Foundation Report No. 11 concluded that maintenance typically is responsible for over 50 per cent of the cost of a system. Although it may be difficult to estimate or predict accurately, systems maintenance is not a cost that can be ignored. It absorbs an increasing proportion of scarce systems staff. This is unacceptable in today's environment with, typically, a growing backlog of potential applications.

Distribution of maintenance effort and cost

Estimates of the magnitude of systems maintenance costs vary considerably from system to system and from installation to installation. A recent survey in the USA (reference 1) indicates that, for data processing installations in business, the average ratio of development to maintenance costs is 47:53.

Figure 3 shows that, within system repair, corrective maintenance (emergency program fixes and routine debugging) accounts for about 20 per cent of the maintenance effort; adaptive maintenance accounts for about 25 per cent of the effort and perfective enhancement accounts for about 10 per cent. About 45 per cent of the maintenance effort is devoted to software updates (enhancements for users).

Figure 3 Distribution of system maintenance effort



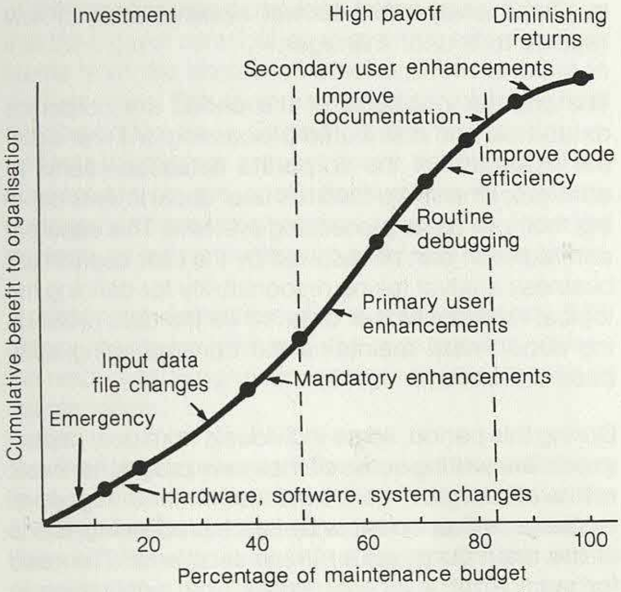
(Source: Lientz and Swanson (1978))

The conclusion to be drawn from these percentages is that achieving error-free system development does not eliminate a significant requirement for system maintenance. Corrective maintenance consumes a relatively small proportion of the overall system maintenance effort. Almost half the system maintenance effort is devoted to enhancements for users which result in changes to the specifications. It is this factor which dictates that maintenance levels remain constant over time — as user demands increase and change.

It is possible to consider system maintenance in terms of relative cost-benefits for the tasks undertaken. In figure 4, the investment segment consists of those maintenance activities which must be performed if the system is not to deteriorate in value. The activities include emergency program corrections, hardware changes, operating system and data changes and mandatory enhancements as a result of legislation. The high pay-off segment of the curve consists of high-priority enhancements for users. These include improvements in efficiency, reliability and documentation together with a set of secondary improvements which provide a lower but still positive ratio of benefits to costs. The diminishing-returns segment of the curve consists of a backlog of 'desirable' features such as limited-demand reports, and rewriting poorly structured, but stable, code.

The decision on how much effort to put into maintaining particular systems rests with each organisation. Our research indicates that once an organisation has determined an appropriate level of maintenance this will remain constant over time. As corrective maintenance is reduced by use of, say, modern development methods and tools, the demand for system enhancements which require changes to the specification increases. It is clear that flexible data structures and report generation capabilities can facilitate system enhancements, thereby improving the effectiveness and efficiency of systems maintenance.

Figure 4 Returns on investment for software maintenance



(Source: adapted from Boehm (1981))

EVOLUTION OF THE PROBLEM

The data processing function, and the roles of the systems staff employed within it, have both evolved over time, and this evolutionary process has directly affected systems development and maintenance. Projecting how the process will continue in the future helps to throw light on the changing nature of systems development and maintenance.

The major environmental changes in the period 1950-1985 are summarised in figure 5, and the more important trends are described in the text that follows.

1950-1960

The first commercial computer applications were typically accounting and payroll. They were justified on the basis of a reduction in repetitive clerical work. The systems were implemented by analyst/programmers, with part-time user participation. The analyst/programmers were concerned largely with the technical aspects of computing, often failing fully to understand the users' information processing needs.

Figure 5 The changing data processing environment

Era	*Technology/control factors*	*Functions of user departments*	*Functions of DP departments*
1950-60	1. Computers are new tools 2. Lack of understanding of computers	1. Limited participation on design of DP systems 2. Maintain data base	1. Select and maintain hardware 2. Control the design effort 3. Full-time participation on project teams 4. Employ programmer/analysts
1960-70	1. Proliferation of technology 2. Project teams emerge	1. Increased participation on project teams	1. Select and maintain hardware 2. Control the design effort 3. Full-time participation on project teams 4. Employ analysts 5. Employ programmers 6. Maintain data base
1970-75	1. MIS 2. Teleprocessing 3. Better understanding of computer usage	1. Joint control of project team 2. Continued participation on project teams	1. Select and maintain hardware 2. Full-time participation on project teams 3. Employ analysts 4. Employ programmers 5. Maintain data base
1975-85	1. Distributed processing 2. Corporate data base 3. High-level languages	1. Joint control of project team 2. Full-time participation on project teams 3. Own analysis 4. Limited programming	1. Select and maintain hardware 2. Full-time and part-time participation on design 3. Employ analysts — more emphasis on user to undertake own analysis 4. Employ programmers 5. Maintain data base

The data processing department selected and maintained the computer hardware.

1960-1970

In this era, the concept emerged of project teams dedicated to the design of data processing systems. The project manager was part of the data processing organisation. User personnel were assigned to the project on a part-time or full-time basis.

With the proliferation of new software (operating systems, languages, utilities) and hardware (storage devices such as discs and tapes, input devices such as key-to-disk systems, and output forms such as microfilm), merely keeping up-to-date with the technology became a full-time job. At the same time, it became clear that a greater understanding of business functions was needed in order to design effective systems. A separation of functions offered a solution to the problem, and programmer/analysts were largely replaced by specialist programmers and specialist analysts.

1970-1975

During the early 1970s most organisations implemented some form of management information system (MIS).

In some advanced organisations, project teams developing MISs changed significantly during this period. Some users became actively involved in the management of systems projects. This happened for three reasons. First, user acceptance of designs for computer systems was often a problem, and this problem was reduced if the user-manager was involved in the design. Second, for users, the mystery of what computers could do gradually disappeared. And third, user managers wanted to control systems that affected their jobs.

Technical advances in this area revolved around teleprocessing. Systems were designed to provide online retrieval, validation and updating of data files. In leading organisations, users themselves gained direct access to data for the first time.

1975-1985

The current period has itself brought some significant changes. Many users now have experience in specifying and approving major computer systems. In some cases, users have participated in designing data processing systems. In a few organisations, the business analyst (the former systems analyst) now reports to a user manager.

The popular concepts of the period are corporate databases and distributed processing. (At first sight, the existence of the corporate database seems to contradict the trend towards user departments defining their own data processing systems. This apparent contradiction can be resolved by the user department business analyst taking responsibility for defining the logical relationships of data, while the data processing department maintains the corresponding database software.)

During this period, some individuals from user departments are writing some of their own programs. Basic retrieval (extract, sort, and list) with a high-level language, either online or by batch processing, is one of the main purposes of these programs. The need for such a retrieval tool varies from application to application. Users find retrieval an excellent tool for solving both unique and repetitive problems. Some enthusiastic users also appreciate the convenience of the process, because they no longer have to endure long delays while the data processing department processes their requests. Increasingly, some users are writing non-critical and relatively simple applications themselves.

In short, during this period some advanced users have experienced for the first time the advantages of developing their own systems and programs. The expectations of users resulting from this evolution are providing a major impetus to the development and maintenance of cost-effective systems.

CHAPTER 2

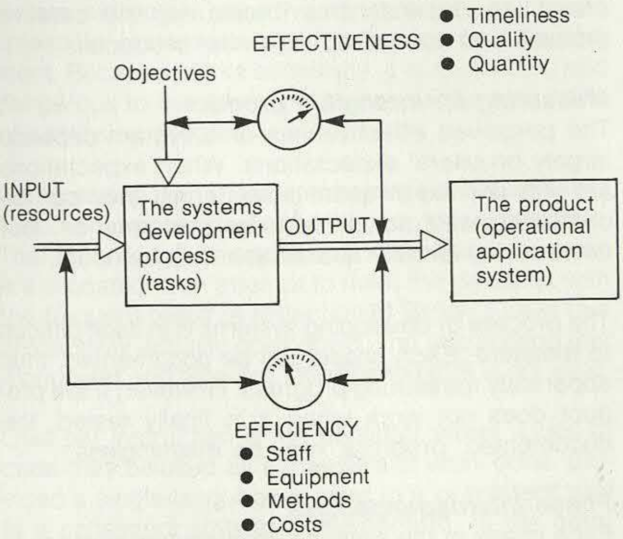
UNDERSTANDING AND MEASURING THE COST-EFFECTIVENESS OF SYSTEMS DEVELOPMENT AND MAINTENANCE

The effectiveness of a system is measured in terms of the benefits that it provides to the user. It follows that the biggest contribution to systems effectiveness stems from the choice of which system to build in the first place. This topic was considered in Foundation Report No. 34 — Strategic Systems Planning — and falls beyond our present scope. In this report we concentrate on the cost-effectiveness of systems development and maintenance processes once the decision on which system to build has been taken.

A related but distinct issue is that of efficiency. Efficiency measures how well the organisation utilises the resources assigned to systems development and maintenance.

The relationship between effectiveness and efficiency is represented diagrammatically in figure 6. In this chapter we discuss the different measures of effectiveness and efficiency. We also identify the measurements that are appropriate in a formal performance measurement programme.

Figure 6 The measurement of effectiveness and efficiency



ASSESSING EFFECTIVENESS AND EFFICIENCY

In this section we discuss the factors which affect the measurement of effectiveness and efficiency. We also discuss the difficulties of performance measurement.

Factors in assessing effectiveness

Systems effectiveness can be assessed in terms of three factors: timeliness, quality and quantity.

Timeliness

Timeliness reflects the extent to which the objective can be accomplished in time to be effective. Typical measures of timeliness include:

- —Responsiveness to users. This measure can be applied to activities during systems development and maintenance.
- —Schedule compliance. This measures the progress of a project. Poor schedule compliance often results in a loss of user confidence in the data processing department, from which it may be difficult to recover. Experience indicates that adding additional staff to projects is not an effective way to solve this problem. (Brook's Law states that adding additional staff to a late project during its final stages in an attempt to make up time almost certainly makes the project later still.)

Quality

Quality reflects the extent to which the objectives of a system are achieved. Typical measures of quality include:

- —Acceptability. Each system must contain a minimum set of essential features if it is to be effective. Measurements of user acceptance are generally subjective, but it is possible to establish a consistent procedure for evaluating acceptability.
- —Process complaints. Following the completion of a phase, the number and nature of complaints can be used to measure process effectiveness and to highlight problems such as poor evaluation of user requirements and inadequate documentation.

Quantity

Systems effectiveness can also be measured quantitatively. Typical quantitative measures include:

- —System throughput. This is measured by the number and size of systems developed over a specific period of time.
- —Extent of backlog. This is measured by the number and size of outstanding systems applications. (Many potential users may not request systems because of the backlog.)

Factors in assessing efficiency

Efficiency can be assessed in terms of four factors: staff, equipment, methods and costs.

Staff
It is not easy to measure quantitatively the performance of individual programmers and analysts. Nonetheless, where it can be done the results may be revealing. The point is well illustrated by a study (reference 2) conducted in 1966 which analysed the performance of 12 experienced programmers. The performance variations ranged widely between individuals as is illustrated by the following ratios between the worst and best person:

debug time	computer time	coding time	code size	running time
26:1	11:1	25:1	5:1	13:1

Despite the fact that such variations are probably not so extreme among today's programmers (as a result of the widespread use of high-level languages and the observance of stricter standards) they do still exist. Measuring quantitatively the performance of systems analysts is by no means easy (individuals are usually assessed subjectively by experienced project managers). Were it to be done, there would probably be a similar variation between individuals as in the case of programmers — a variation of, perhaps, 4:1.

This difference in productivity between individuals does offer a major opportunity for improving efficiency by replacing those individuals who are not really suited to the systems environment. Our research indicates that most organisations are not taking advantage of this opportunity.

Equipment
Throughput and staff performance can be measured in situations where different levels of equipment are used. The level of equipment can be measured in terms of capacity, facilities and arrangement. Our research indicates that equipment enhancements can improve the productivity of systems development and maintenance.

Methods
Throughput and staff performance can also be measured in situations where different methods are being used. These measures can help to identify appropriate methods of systems development for different types of system. Ideally, any method should include:

—A framework for evaluation.

—A uniform presentation of results.

—Consistent terminology through each stage of the project.

—Clear objectives.

—The achievement of objectives through systematic application of the method.

—Suitability to the development environments.

—Appropriate skills and expertise.

Costs
Individual systems and the total application portfolio can be monitored on a cost basis. Equipment and staff resources can be directly translated into financial terms. Comparative costs of different methods are more difficult to define and monitor but, as we indicated in Foundation Report No. 25, separate evaluation programmes can be set up and costs can be monitored over time.

Performance measurement

Effectiveness and efficiency measures form a useful structure for managing performance. Measuring performance is fundamental to management control in any organisation. Yet most organisations lack a formal programme for performance measurement, so systems management remains an intuitive process. To correct this situation, managers should establish a formal programme for indentifying and collecting the appropriate performance data.

Difficulties of performance measurement

There are four main difficulties of performance measurement: measuring an intangible product, phase interdependencies, measuring the creative process and comparing dissimilar products.

Measuring an intangible product
The perceived effectiveness of a system depends largely on users' expectations. When expectations are well-defined in quantitative terms, they can be compared with actual systems performance. But users rarely identify quantifiable requirements.

The process of developing systems is in itself difficult to measure. Each phase can be documented, thus apparently measuring progress. However, if the product does not work when it is finally tested, the documented 'progress' may be meaningless.

Phase interdependencies
Each phase in the systems development process is based on preceding phases. The process is also dependent on the specific requirements of individual systems. The necessary flexibility in the process makes only the most general measure of effectiveness worthwhile.

Measuring the creative process
The creative aspect of system design is undefined

and discontinuous and does not lend itself to measurement. Since all phases of a project contain tasks requiring creative effort, performance measurement must be viewed at best as a step function (with results becoming measurable at certain stages of the project). In contrast, the input of resources is continuous.

Comparing dissimilar products

Three characteristics of an application system make it unique to a particular organisation: technology, structure and size. Comparing the design effort required by a small, highly structured, low-technology project with that required by a large, unstructured, high-technology one is unwise.

The need for practical metrics

The problems raised in the previous section can be overcome if well-defined measures or metrics are established. These measures require conscious evaluation to provide a basis for comparison and should incorporate inspections, audits, and continuous record keeping.

Much of the discussion in this report is concerned with improving the productivity of systems development and maintenance. Some people argue that improving productivity leads inevitably to an improvement in cost-effectiveness. Our view is that productivity can lead to gains in cost-effectiveness — but this is not inevitable.

Productivity metrics — such as cost per instruction — are extremely sensitive to variations in project timescale, system size and the development environment. Because of this sensitivity, it is misleading and dangerous to use these measures as the sole basis of assessing progress.

Compressing the timescale of a project in an attempt to force up productivity is often counter-productive. That is the case when additional manpower resources are allocated in an attempt to meet the deadline, with the frequent result (a reflection of Brook's Law) that productivity actually goes down. The assumption that manpower and time are interchangeable is not valid.

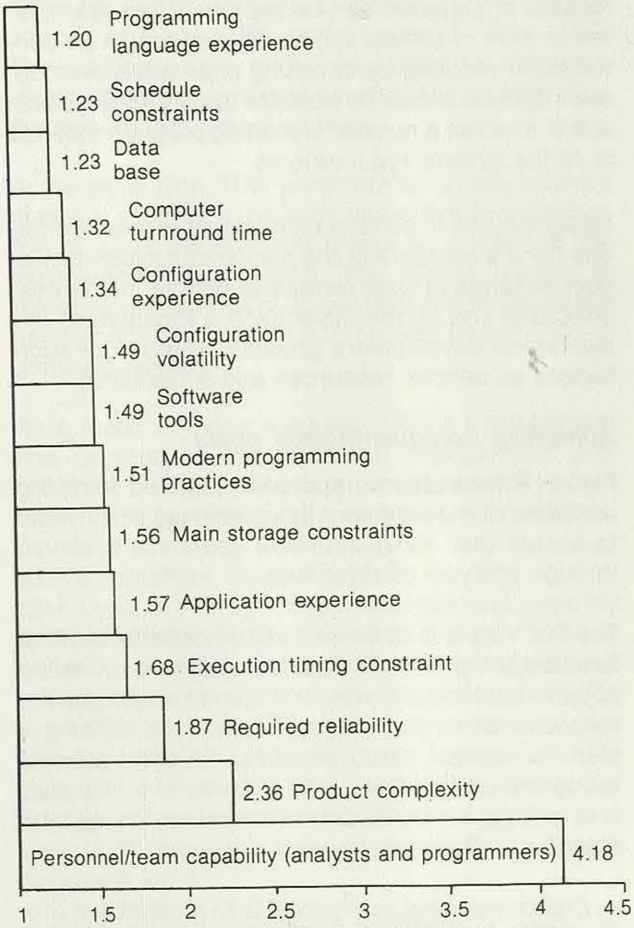
Cost per instruction is not constant either. Lines of code may be used as a measure of work done, provided a single language is used in a consistent way in a consistent software environment. In the great majority of cases, though, where changes are contemplated, a better system metric must be found.

Factors for improving productivity

An overall view of the factors contributing to software productivity is given in figure 7 (based on work carried out at TRW Systems in the USA). It shows the ratio of the least to the most productive rating. The productivity ranges provide a means of identifying the high pay-off areas that should be emphasised in a productivity improvement programme.

From these ratios we conclude that an effective productivity improvement programme involves more than just introducing improved techniques for systems development. The highest ratio in figure 7 is for project staff. Thus cost-effectiveness must be concerned primarily with maximising the contribution of systems analysts and programmers. Appropriate measures must not just increase productivity in the short term (for instance, through increased controls) but must have a longer-term effect as well (for instance, through engaging higher-quality staff).

Figure 7 Software productivity ranges



(Source: Boehm (1981))

Each of the factors given in figure 7 can be measured and weighted in accordance with its effect on system development and maintenance effort. But doing this in a practical and effective way is not always easy.

INSTALLING AND USING METRICS

Many organisations have difficulties in establishing programmes for systems measurement. In this section we describe three different approaches to systems measurement and offer some practical guidelines for installing and using system metrics.

The use of successful metrics provides two primary advantages for systems management. One is a sound start-point for management planning and project control. The other is that many of the system cost parameters are project related, so that by concentrating on the most appropriate parameters the cost-effectiveness of a system can be improved. For example, the amount of software to be developed can be controlled or reduced by choosing alternative development options. It may be possible to purchase a package or to adapt a number of existing program routines to fit the system requirements.

Using metrics in combination should provide on the one hand a measure of the successful systems product (in terms of such factors as people, resources, structure) and on the other hand a measure of the successful development process (in terms of such factors as people, resources and scheduling).

Achieving non-quantifiable goals

Figure 8 illustrates an approach (derived from the concepts of management by objectives) which aims to ensure that non-quantifiable goals are achieved through analysis of objectives.

The first step is to define the overall objectives of the function or the system. The next step is to refine the objectives at lower levels, and then to determine the means of achieving them. This involves defining a plan to assign responsibilities, identify actions, recognise assumptions, and so forth. The final step is to monitor the development process relative to the objectives. The main benefits are:

—Explicit personal commitments to product and process objectives.

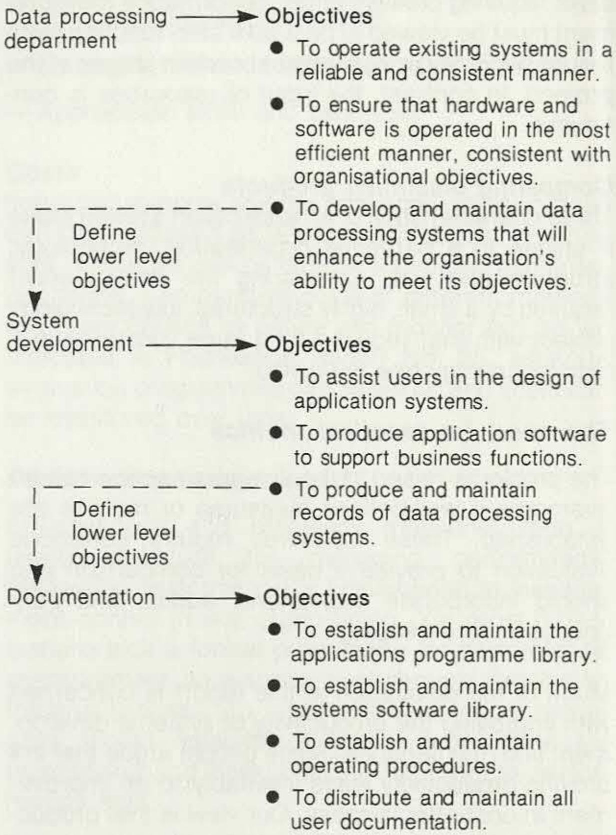
—A well-defined sequence of progress.

—A framework for checking achievement.

—Checkpoints for reconciling qualitative measures with quantitative measures.

A design-by-objectives methodology which translates high-level business objectives into low-level technical objectives has been developed by Gilb (see reference 5). The purpose of the methodology is to structure the objectives of a system and then to decompose them into a network of lower-level objectives that can be expressed in terms of system attributes, functions and performance.

Figure 8 Examples of functional objectives



Data processing department → Objectives

- To operate existing systems in a reliable and consistent manner.
- To ensure that hardware and software is operated in the most efficient manner, consistent with organisational objectives.
- To develop and maintain data processing systems that will enhance the organisation's ability to meet its objectives.

Define lower level objectives

System development → Objectives

- To assist users in the design of application systems.
- To produce application software to support business functions.
- To produce and maintain records of data processing systems.

Define lower level objectives

Documentation → Objectives

- To establish and maintain the applications programme library.
- To establish and maintain the systems software library.
- To establish and maintain operating procedures.
- To distribute and maintain all user documentation.

The methodology is associated with an evolutionary approach to design, planning and implementation. It is based on the early delivery of high priority subsystems or improved attributes. (The approach is distinct from that of prototyping as each delivery is concerned with a well-defined controllable part of the final system.)

A system is progressively defined in terms of its functional and attribute requirements. The functional requirement is basic to the system solution. The attribute requirement indicates the important characteristics or qualities of the function. Attributes can be defined in terms of resources such as time, money, manpower, system capacity, and qualities such as reliability, ease of use and timeliness.

Gilb stresses the need to identify the critical attributes — those which if not properly controlled would threaten the viability of the project or final system. He also emphasises the need to quantify the objectives in a way that is easy to understand.

The fundamental element of the methodology is the 'attribute specification'. All the tools recommended by Gilb are based on this. The general principles of attribute specification are that:

—All critical attributes must be identified and controlled throughout the project.

—All attributes must be measurable.

—Attribute specifications must be clear and unambiguous.

—The attributes must be specified early in the project plan.

—The requirements of any one attribute must be evaluated against the others (because the planned levels of all attributes are interdependent).

—Priorities must be established to resolve conflicts between attribute objectives.

The attribute specification can then be used to provide tools for analysis and design. The three major tools are:

—Function/attribute/techniques analysis: this maps the relation between system goals (function and attributes) and the means suggested for meeting these goals (the techniques).

—Quota Control Analysis: this provides a rough estimate of progress towards achieving attribute goals. It is a highly generalised cost-estimating technique which can be used at any point in the development process.

—Multi-element component comparison and analysis (MECCA), which allows comparisons to be made between alternative systems. A weighted score is derived for each alternative system, and the best alternative is the system with the highest score.

Each defined attribute has to be measured in some way. Concepts which cannot be immediately quantified must be partitioned into sub-concepts until it becomes obvious how to measure them. Unless otherwise specified the measure should be something that can be carried out in acceptance tests, or at the point where the changes start to have an impact on the end user.

The aim of the methodology is to provide managers with a decision accounting method in areas of evaluation which are usually treated in a less formal manner.

Case studies of performance evaluation

The efforts of two large organisations to implement formal measurement programmes illustrate how success can be achieved.

TRW Systems' programme for productivity improvement

TRW Systems' programme for productivity improvement has been described by Stuckle (reference 6). The programme was implemented after TRW had performed a detailed analysis of 63 large software projects. A cost estimation programme has been developed to help identify the major cost-sensitive parameters (see figure 7 on page 9) and to evaluate which of these parameters can be influenced by a productivity improvement programme.

TRW undertook a 'productivity audit' of its projects to determine the weighted averages of parameters characteristic of systems under development and maintenance. The audit measured not only the present situation, but also several future alternative scenarios at different levels of investment for productivity improvement. The resultant analysis indicated that a co-ordinated programme concentrating on just a few key parameters would improve productivity by a factor of 3.4 by 1985, and a factor of 7.8 by 1990. The analysis also provided guidelines for determining which parameters to emphasise as part of the productivity improvement strategy.

At the same time, TRW undertook an activity analysis aimed at assessing the likely reductions in project effort at each development phase as a result of the improvement programme. The results pointed to development savings of 39 per cent and maintenance savings of 46 per cent (excluding any savings due to software re-use).

As a result of these analyses, TRW's management has decided to implement an improvement programme. The goals are to improve productivity by a factor of 2 by 1985, and a factor of 4 by 1990.

IBM's Function Point Analysis

IBM's use of Function Point Analysis was reported to the Share/Guide Conference in October 1979. In order to measure productivity, IBM defined and measured systems and costs. For each system, the number of inputs, enquiries, outputs and master files delivered was counted, weighted, summed and adjusted for complexity. The objective was to develop a relative measure of the function value (to the user) which is independent of the particular technology or approach used.

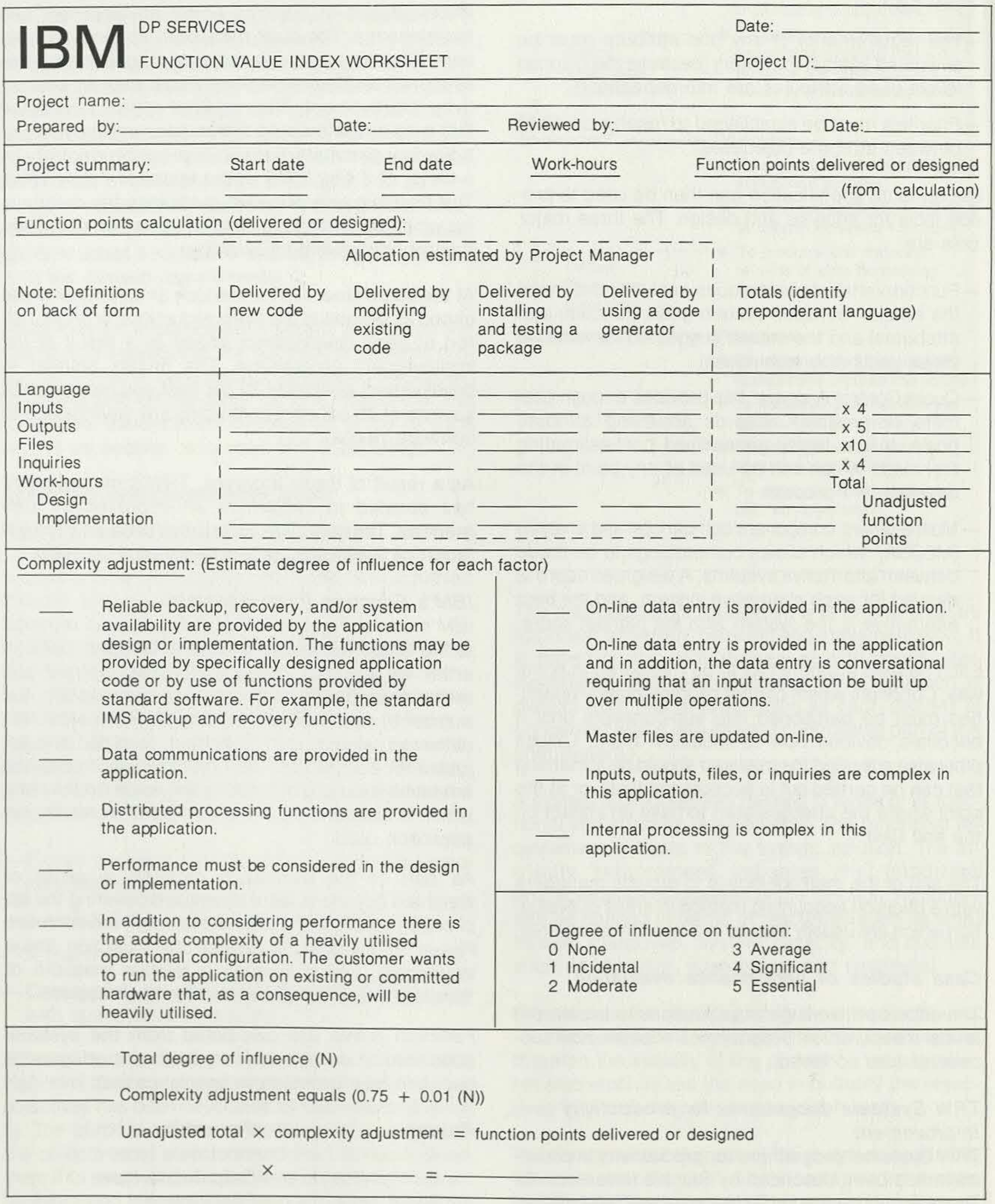
As part of the estimating process, a series of weighted questions were developed covering the application function and the development environment. Figure 9, overleaf, illustrates the Function Value Worksheet, which enables a relative measure of 'function value' to be derived for each system.

Function points are calculated from the systems specification as follows (an adjustment of up to 25 per cent may be made in special cases):

Points	*For each:*
4	Input data type.
5	Output data type.
4	Enquiry type.
10	Master file.

This approach defines a 'function' which is based on external system attributes. It has been used to determine the relative productivity of different languages and technologies for projects of different sizes.

Figure 9 IBM's function value worksheet



IBM DP SERVICES
FUNCTION VALUE INDEX WORKSHEET

Date: ______
Project ID: ______

Project name: ______
Prepared by: ______ Date: ______ Reviewed by: ______ Date: ______

Project summary: Start date ______ End date ______ Work-hours ______ Function points delivered or designed ______ (from calculation)

Function points calculation (delivered or designed):

		Allocation estimated by Project Manager				
Note: Definitions on back of form	Delivered by new code	Delivered by modifying existing code	Delivered by installing and testing a package	Delivered by using a code generator	Totals (identify preponderant language)	
Language						
Inputs						x 4
Outputs						x 5
Files						x10
Inquiries						x 4
Work-hours						Total
Design						Unadjusted function points
Implementation						

Complexity adjustment: (Estimate degree of influence for each factor)

___ Reliable backup, recovery, and/or system availability are provided by the application design or implementation. The functions may be provided by specifically designed application code or by use of functions provided by standard software. For example, the standard IMS backup and recovery functions.

___ Data communications are provided in the application.

___ Distributed processing functions are provided in the application.

___ Performance must be considered in the design or implementation.

___ In addition to considering performance there is the added complexity of a heavily utilised operational configuration. The customer wants to run the application on existing or committed hardware that, as a consequence, will be heavily utilised.

___ On-line data entry is provided in the application.

___ On-line data entry is provided in the application and in addition, the data entry is conversational requiring that an input transaction be built up over multiple operations.

___ Master files are updated on-line.

___ Inputs, outputs, files, or inquiries are complex in this application.

___ Internal processing is complex in this application.

Degree of influence on function:

0 None	3 Average
1 Incidental	4 Significant
2 Moderate	5 Essential

______ Total degree of influence (N)

______ Complexity adjustment equals (0.75 + 0.01 (N))

Unadjusted total × complexity adjustment = function points delivered or designed

______ × ______ = ______

FACTORS THAT INFLUENCE COST-EFFECTIVENESS

Our research strongly suggests that improvements in cost-effectiveness can best be achieved by a broadly based improvement programme. Many factors need to be evaluated and key factors identified before an effective programme for improvement can be implemented and sustained. We now describe each of the factors to be considered.

Project management

In Foundation Report No. 8 — Project Management — we argued that system development is best conducted in an organised and stable environment where the ground rules are established clearly in advance, and then maintained. 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.

Good project management will not directly improve the quality of a system, but it will reduce the risks of time and cost overruns. Poor management can increase system costs more than any other factor.

Environmental factors

Environmental factors are concerned not with systems work itself but with the analysts' and programmers' perception of it. In order to improve productivity, a positive attitude must be promoted amongst the staff.

Increased control, fragmentation and deskilling of system development can demotivate staff. Such methods often increase productivity in the short term, but have a larger negative impact in the long term because they produce staff who are interested in neither professional growth nor in the organisation's objectives.

Work conditions can be improved by the use of such facilities as software tools. The use of these tools does improve staff morale, as they remove some of the tedious tasks and increase the analyst's or programmer's control over their own output.

Working conditions can have a negative impact — if conditions become too bad, people will leave. This, in turn, can reduce productivity because a high proportion of staff may be unfamiliar with the organisation and its objectives.

The most significant environmental factor in terms of productivity is the analysts' or programmers' perception of the work objective. As far as possible, systems staff must be encouraged to recognise that their ultimate objective is to assist users to manage their activities effectively. The use of modern methods and facilities which we discuss in chapter 5 can encourage this attitude.

Data management

In Foundation Report No. 32 — Data Management — we examined the topic of data as a company resource, and identified several benefits that arise from that concept. They included:

- Reduced time needed for applications development.
- Involvement of users in data analysis (enabling them to contribute to data model design and application development).
- Greater flexibility in database design.

Additional resources are needed at the outset, to carry out data analysis and database administration while existing staff are being trained in the new methods. Additional software tools, such as a data dictionary, a database management system and data design aids may also be required. In theory, the number of systems development and maintenance staff should eventually be reduced as productivity rises, development becomes quicker and easier, the maintenance load reduces and end users satisfy many of their own requirements. Whether this happens in practice remains open to question. The ability to satisfy user demands more easily has the effect of generating further demands from users.

Foundation Report No. 12 — Trends in Database Management Systems — confirmed that a database approach can improve productivity by reducing the cost and effort of development and maintenance and by allowing a faster response to user requirements. To achieve the full benefits of data management, however, an organisation needs to invest heavily in manpower, training, tools and techniques.

Methods and tools

Too often the whole topic of systems productivity is associated with the implementation of new methods. Modern systems development methods can improve systems productivity, but they should be regarded as only one element in a complete programme.

Most modern development methods help to improve productivity in parts of the development process, but not all of them improve overall life-cycle productivity. Because of this, methods based on the traditional staged approach to systems development have limited application.

Figures 10 and 11, overleaf, illustrate the results of a 1979 Guide (IBM user group) survey of the use and evaluation of improved practices in about 800 installations. Figure 10 illustrates the effects of new methods on various system and life-cycle characteristics. It indicates that the greatest improvement has been in the quality of code, and in early error detection. The effects on programmer productivity and maintenance costs are strongly positive. About 50 per cent of installations reported 'some' improvement and about 30 per cent reported 'great' improvement.

Figure 11 gives estimates of further improvements which might be achieved if the methods are introduced extensively. About 40 per cent of the 800 installations could achieve an additional 10-25 per cent productivity gain; about 12 per cent could achieve an additional 25-50 per cent.

Note that the effect on staff morale indicated in Figure 10 is overwhelmingly positive. Nineteen per cent improved greatly, 50 per cent improved somewhat, 28 per cent showed no effect and three per cent experienced a negative impact. We conclude that measurable positive results can be achieved if new development methods are introduced carefully.

Figure 10 Use of modern methods: effects on system and life-cycle

Factor considered	Number of organisations				
	Improved greatly	Improved some	No effect	Negative impact	Total respondents
Project estimating and control	63	294	206	8	571
User communication	89	227	252	3	571
Organisational stability	47	193	303	10	553
Accuracy of design	166	297	107	3	573
Quality of code	206	287	94	2	589
Early error detection	213	276	87	4	580
Programmer productivity	165	350	80	6	601
Maintenance time or cost	178	272	108	11	569
Programmer or analyst morale	108	292	160	20	580

(Source: Guide survey)

Figure 11 Use of modern methods: potential for further productivity improvement

Relative to current level	Number of organisations
decrease	8
same	132
≤ 10% increase	153
10-25% increase	264
25-50% increase	82
50-100% increase	18
100% + increase	1
total respondents	658

(Source: Guide survey)

Closely associated with, but separate from, the use of new methods is the use of software tools. Using software tools on their own can help to improve productivity. Using them in combination with other techniques can be even more worthwhile.

Application packages

In Foundation Report No. 22 — Applications Packages — we concluded that much of the increasing demand for application systems can be satisfied by the use of packages. There are six powerful reasons why an organisation should make greater use of application packages:

- They release scarce professional staff to work on unique or special applications.
- They enable the maintenance activity to be subcontracted to the package supplier.
- They enable more systems staff to work on priority systems.
- They are usually well supported and well maintained (the best packages have a large user population).
- They are available more quickly then bespoke systems.
- They are initially more reliable and cheaper than bespoke systems.

But there are four main disadvantages to the use of packages:

- Problems of matching the package to user needs.
- Problems of integrating packages with other application systems. (These problems may include difficulties in establishing a data management policy.)
- Conflict with other elements of an organisation's systems strategy, such as hardware and systems software.
- Problems of maintenance with some suppliers.

We believe that the advantages often outweigh the disadvantages, and that applications packages, or indeed any re-usable code, will continue to be an important method of improving productivity for the foreseeable future.

The development approach

The advent of database management systems, application generators and report generators has led to new approaches to system development that differ from the traditional staged approach. (We describe these alternative approaches in chapter 4.)

The new approaches require greater commitment and involvement from the user in the development process. A great deal of effort and technical skill may be required to create the database or the communications network, but the provision of application programs can be so easy and cheap that no maintenance will be undertaken. It may be simpler to discard the inadequate programs and develop new ones.

The new approaches require a different contribution from the systems analyst. In many cases the analyst will act as technical adviser to the users. For instance, the users may be responsible for the identification of data elements, with the analyst responsible for setting up and loading information into the files.

The role of systems staff is to bridge the gap between the user and the machine. As the machine facilities expand, the job of 'bridging the gap' becomes technically less demanding. Sometimes the gap can be closed by providing users with appropriate computer-based tools. At that point the relationship between the user and systems staff is changed: the role of the systems staff is to identify how the machine facilities can best be exploited by the users.

There has been considerable debate on how to involve users in systems development. Nevertheless, user involvement certainly promotes system acceptability and thereby effectiveness.

CHAPTER 3

IMPROVING ORGANISATIONAL FACTORS

In Foundation Report No. 11 we noted that the work attitudes of system development staff are significant determinants of productivity. Organisational factors have a strong influence on work attitudes.

In this chapter we discuss the organisational factors that influence work attitudes and summarise the lessons learnt from our research.

MANAGEMENT ATTITUDES

Many of the factors that should be considered by management in promoting a positive systems environment were discussed in Foundation Report No. 31 — A Director's Guide to Information Technology. The following is a summary of the more pertinent factors.

Ensure commitment to the information systems function at all levels

Senior managers should expect and demand appropriate actions from line managers and systems staff to realise profitable use of information technology. Senior managers must set the stage, in an organisational sense, for information technology to prosper.

Develop on the basis of systems plans

In most organisations data processing and other systems trail behind the business decisions that they have to implement. Though primacy must be given to business requirements, support systems must be considered at an early stage.

Plan strategically

This theme has recently been examined in detail in Foundation Report No. 34 — Strategic Systems Planning. The report notes that many organisations fail to understand the nature of strategic planning. Managers often think that it is about future decisions but the focus should be, in Drucker's terms, on making 'current decisions in the light of future needs'.

Recognise the need for infrastructure

Senior managers must differentiate between individual projects or uses of information technology and the infrastructure required to make them work. Both are important.

Manage technical staff as a company resource

Technical staff must be treated first and foremost as company employees and resources — otherwise they tend to burrow into the technology. All technical staff need business training and some need career development outside the systems area.

Emphasise the link between systems and business goals

Systems plans must be integrated with business plans. The proposed changes engendered by new systems must have the necessary management commitment to ensure their implementation.

Create and sustain objectivity

Managers must ensure that timescales and cost estimates are not falsely optimistic. Productivity can be seriously degraded by unrealistic scheduling or unnecessary slippage.

Clarify the role of central expertise

In large organisations, the role of central expertise in relation to local operating divisions must be clarified. This factor is central to the issue of development approaches addressed in chapter 4.

Concentrate on the key issues

Key issues in 1983 are not concerned with hardware policy. Rather they are concerned with aligning information technology with business aims, the delivery of working systems and the problem of technology absorption.

Be positive

Senior managers must adopt a positive attitude towards information technology. The real task of managers is to 'grasp the nettle' in order to ensure success.

Identify responsibilities of end-user and systems staff

In large organisations every system must have a line departmental manager as its sponsor. This is particularly critical where a central unit sponsors a system for use in a number of dispersed units such as local offices and depots. In too many such cases the sponsor acts more as a barrier than a communication channel. The existence of a sponsor should not prevent systems staff from having access to both end users and their local managers.

MOTIVATION

Most productivity studies have found that motivation is the most important of all influences on productivity. Given the strength of this assertion we need to find ways to motivate systems staff. But motivation is a complex issue. There are many theories of motivation (including Maslow's hierarchy of needs, McGregor's theory X and theory Y, and Herzberg's motivation and hygiene theory). The overriding practical consideration, however, is that the personal objectives of systems staff must be aligned with organisational objectives.

Motivating systems staff

Studies (references 7 and 8) have shown that data processing staff are more highly motivated by 'growth' needs than by social needs. Although Herzberg's distinctions between motivating factors and hygiene factors generally hold for systems staff, there are also significant differences between systems staff and other office workers. These studies indicate that data processing staff are more strongly motivated by opportunities for technical supervision, by peer relations and by personal life than by recognition, responsibility, salary and status. The differences are more pronounced among analyst/programmers than they are among project leaders and managers. The results indicate that data processing managers should not expect their subordinates to be motivated in the same way as themselves.

Reconciling individual and organisational objectives

It follows from the significantly higher levels of productivity achieved by highly motivated people that data processing management must emphasise staff motivation as one of their highest priorities. It is the responsibility of data processing management to promote an efficient and co-ordinated development process so that staff capability, motivation and team work can be maximised. Such a commitment must be maintained over a period of time. Once-off exercises are often counter-productive. Managers must demonstrate their continued commitment by investing in better systems tools, recognising and rewarding good performance and enforcing good practices. The prime essential of any motivation programme is that management commitments must be both consistent and sustained over a long period of time.

Case history — a failure in motivation

This case history concerns a major heavy engineering company that is divided into nine divisions, each with a management board. There are about 500 system development staff in total, about 130 of whom are located in a strong central systems department. This department is responsible for defining standards, evaluating tools and techniques and ensuring adherence to common policies.

As part of a staff improvement programme the company evaluated and accepted the BIS MODUS system. MODUS is a systematic methodology for information system planning and development which incorporates procedures, techniques and documentation. A high-level steering committee gave its support to the choice, and a central techniques group visited each of the systems departments to run courses introducing and establishing MODUS.

However, MODUS has not been used effectively — not because of its lack of quality as a product — but because the implementation programme did not gain the full acceptance of middle managers and project managers.

The company is now seriously reconsidering its position. The message is clear: without management commitment, improvements in the systems process cannot succeed. Managers must be genuine in their desire to change systems procedures and methods.

PROJECT MANAGEMENT

Good project management cannot directly improve the quality of a system but it can reduce the probability of failure and the likelihood of time and cost overruns. There is considerable evidence that systems staff often regard project management methods and systems development techniques as acting against their own interests. Foundation Report No. 25 specifically addressed this problem and identified guidelines for introducing new development methods.

It is useful to discuss project management under three headings: management quality, planning the project and controlling the work.

Management quality

No study has yet put forward a convincing definition of management quality, nor succeeded in establishing precisely its impact on staff productivity. The penalties of poor management, however, are clear. They can lead to a doubling of system development costs as a result of assigning the wrong people to jobs, demotivating staff by failing to reward good performance, allocating staff to projects in advance of defining their responsibilities, failing to resolve high-risk elements in time, and failing to provide adequate support.

Planning the project

Planning the project entails breaking it into segments, then estimating the resources needed for each segment. Several proprietary aids are available to assist with the task of determining work stages, estimating and monitoring progress against plan. Two examples are the Putnam SLIM model and PROMPT II.

The Putnam SLIM model is based on Putnam's analysis (reference 16) of software life-cycle costs in terms of the levels of people assigned to a project over its duration. Putnam concludes that for a job of a given complexity, there is an associated minimum timescale.

PROMPT II utilises a life-cycle model to determine work stages and provides, at the end of each stage, formal reports to management. The stages are project initiation, conception (this addresses business justification and resources needed), functional definition, project definition (an outline technical design, identifying the work and resources required), development, systems test, user acceptance and maintenance.

For PROMPT II to be effective, a plan is required to address training needs, overheads, suitable pilot projects, consultancy support and costs. A manager should also be designated for each project stage.

Controlling the work

Again, several proprietary products are available to assist the task of controlling the work. BIS's MODUS, Arthur Andersen's Method-1 and Philips' PRODOSTA-R are three examples. All three stress the need to manage the projects using a steering committee. The committee controls the budget and decides whether to continue the project at each stage. Another characteristic is a strong separation between the initial stages that are concerned with functional aspects of the system, and subsequent stages concerned with technical aspects.

A relatively simple way of controlling work and raising systems quality is through formalising the procedures for checking quality and compliance with standards. This type of approach is referred to as inspection.

The aim of inspection is to detect errors in system components and documents. Several inspections should be conducted during the systems life-cycle. Early stages of the development project as well as the design and programming stages should be included. Inspections are characterised by the use of checklists and summary reports. An inspection team typically includes a group leader responsible for process planning, moderating, reporting and follow-up activities. Other members of the team are the person responsible for designing (or implementing) that part of the system, and the person responsible for testing the item being inspected.

The five basic steps involved are: planning, preparation, inspection meeting, rework and follow-up. The steps do not vary for inspections conducted at different development stages but the responsibilities of individuals on the inspection team will change as the life-cycle progresses.

Our research indicates that inspections are an effective method of increasing product quality (reliability, usability and maintainability). Experience with the technique indicates that it is effective in projects of all sizes. The best results are achieved when the inspection leader is experienced in using the inspection technique.

One IBM study reported a 23 per cent improvement in programmer productivity with inspections compared to walkthroughs. The study also reported 38 per cent fewer errors in the operational software compared with the use of walkthroughs as a method of detecting errors.

WORK ORGANISATION

In trying to improve the cost-effectiveness of development and maintenance, it is worth considering the

Figure 12 Distribution of project effort by activity

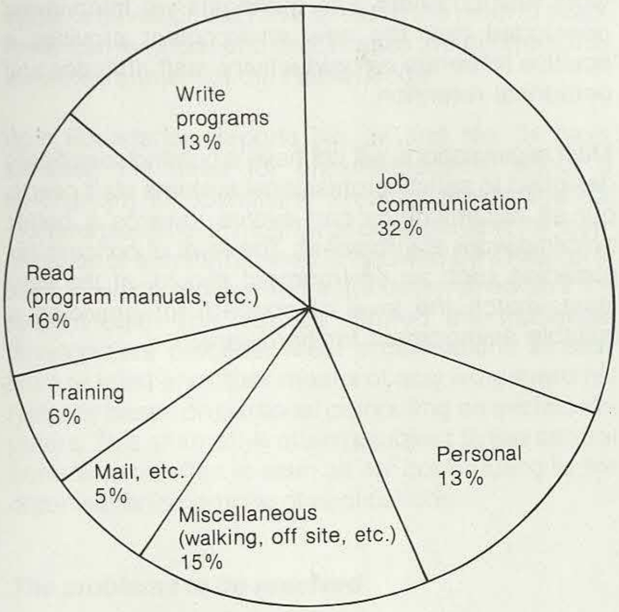
(Source: Boehm (1980))

nature of tasks undertaken by systems staff. Once this has been done, the appropriate work environment can be created.

Figure 12 shows the distribution of effort by activity for two small application systems projects. The analysis indicates that activities such as reading, reviewing, meetings and enhancing (changing or adding to) the original specifications consumed roughly 40 per cent of the development effort for both projects. These ancillary project activities need to be estimated before development work commences.

Figure 13 shows the results of a Bell Laboratories time and motion study for 70 programmers. It indicates that roughly 30 per cent of the programmers' worktime is devoted to overhead activities such as training and personal business.

Figure 13 Analysis of programmer activities



Choosing objectives

Figure 14 illustrates the results of a programming experiment indicating that programming performance is highly sensitive to objectives. In this experiment, five teams were given the same assignment but each team was required to maximise a different objective. When the programs were completed and evaluated, the results indicated that each team (with one exception) charged with responsibility for maximising an objective, did in fact do better in that respect than the other teams. None of the teams, however, performed consistently well on all of the objectives.

The main conclusions we can draw from this are that:

—Programmers are highly motivated towards achievement. If achievement is defined in terms of project objectives, programmers will generally attempt to achieve those objectives.

—In practice, different objectives conflict with each other.

The conflict between different objectives must, therefore, be resolved if life-cycle planning is to be successful.

ORGANISING FOR MAINTENANCE

Most organisations still undertake maintenance within system development groups. Our research indicates that, where maintenance is organised as a separate function, the result is usually an overall reduction in the need for resources. In terms of the organisational environment, the main conclusions from our research are:

—Maintenance should be organised as a separate function. The responsibilities of development and maintenance staff must be clearly defined.

—Co-ordination between development and maintenance staff is essential.

Figure 14 Analysis of programmer performance against objectives

	Performance ranking				
Team objective to optimise:	*Effort to complete assignment*	*Number of statements*	*Memory required*	*Program clarity*	*Output clarity*
Effort to complete assignment	①	4	4	5	3
Number of statements	2-3	①	2	3	5
Memory required	5	2	①	4	4
Program clarity	4	3	3	②	2
Output clarity	2-3	5	5	1	①

(Source: Weinberg (1974))

—Requests for system changes must be handled through formal procedures.

—Staff attitudes to maintenance can be improved (or made worse) by management actions.

WORK ENVIRONMENT

Foundation Report No. 20 — The Interface Between People and Equipment — discussed aspects of the work environment that affect people's attitudes towards their jobs. These environmental factors apply to the performance of systems staff as much as they do to end users. The physical work environment has a significant influence on systems productivity. Work conditions tend to act as 'a Herzberg hygiene factor'. Above a certain level they are not a powerful motivator; but below that level they are a powerful demotivator.

Perhaps the most ambitious attempt to provide a work environment specially suited to the needs of systems staff is the architectural design and development of the IBM Santa Teresa Laboratory (reference 9). The buildings, offices, furnishing, electrical and telephone connections of the Santa Teresa Laboratory were all designed to meet a set of requirements derived from studies of software development activities. These requirements included:

—Communications (both intra-project communication such as office proximity and conference rooms, and external communications such as voice and data telecommunications).

—Seclusion (personal offices with acoustic isolation, adequate ventilation, windows and individual control of the environment).

—Furniture (such as work surfaces which accommodate the use of computer listings and interactive terminals).

—Computer connections (terminal connections to every office and easy access to communication links and hard-copy devices).

—Security (controlled access to the site, data processing facilities and project facilities).

—Technology (flexibility to encompass future hardware and software advances such as powerful personal workstations).

Several organisations we visited in our research have recently established less comprehensive but similar work environments. The managers we interviewed concluded that the new environment provides a positive influence on productivity, staff attitudes and personnel retention.

Most organisations will not have a building specifically designed to satisfy professional systems staff needs, but all organisations can evolve towards a better systems work environment. The level of concern for providing such an environment should, at the very least, match the level of concern for providing a suitable environment for hardware.

CHAPTER 4

CHOOSING THE RIGHT APPROACH

In this chapter we look at the various ways in which systems might be developed and maintained. We consider the relevance of each alternative and offer guidelines for selecting the appropriate approach.

THE PROBLEM OF CHOOSING BETWEEN ALTERNATIVE APPROACHES

Each organisation has its own culture and ethos. Within an established cultural framework some approaches (and methods) work better than others. For instance, members of a Dutch Foundation working party on systems development have stressed the need to adapt American methods and practices to the different cultural environment of the Netherlands.

Both Foundation Reports No. 24 and No. 34 have stressed the need for organisations to adopt a mechanism for identifying systems needs and for systems planning. In most large organisations the identification of a systems need is followed by a fixed process in which requirements are determined and the system built. This can be termed the traditional development process. Most organisations already have at least one other means of acquiring systems, typically based on personal computing on microcomputers. This alternative often is subject to few central controls, and often is seen as not contributing to the organisation's portfolio of applications.

The problems to be resolved

Traditional data processing departments have become remote from the user community they profess to serve. Two factors in particular have inhibited contact between systems staff and users: over-specialisation and inadequate specifications.

Over-specialisation and bureaucracy
The complexities of data processing have led both to specialisation, and a need to combine and integrate the work of different specialists. The large size of many system functions has led to detailed controls, procedures and extensive documentation requirements. Both complexity and size contribute to bureaucracy, and with bureaucracy comes reduced responsiveness. Data processing organisations usually have become bureaucratic not because of poor management but because of the complexity of hardware, software and application systems.

Inadequate specifications
With traditional development methods, the specification of business requirements is a very difficult task. On the one hand it is difficult for users to define clearly and precisely what they want. On the other hand, their requirements tend to change over time. On large complex projects, communication problems between team members compound the difficulties.

Confusing alternatives
Faced with these problems, users and system developers are offered many confusing alternatives, such as:

—Project management disciplines and methods.

—Structured analysis and design methods.

—Structured programming.

—Application generators/higher-level languages.

—Application packages.

—Development toolkits.

—Novel operating systems (such as UNIX and PICK).

—End-user computing.

In practice there are too many alternatives here for most organisations to investigate, let alone install and use.

Management commitment

In our view most large computer users can benefit from one or more of these alternatives. But we strongly caution against the uncritical and comprehensive adoption of a single solution as a panacea for development and maintenance problems. A good deal of time is wasted through failure to recognise the scope and limitations of a proposed tool or method. Nevertheless, it is vital for systems management to commit themselves once they have chosen an approach. This commitment should be expressed in:

—Public endorsement of the approach.

—Support for the new methods that accompany the approach.

—Clear and reasonable expectations of the benefits to be achieved.

The remainder of this chapter provides guidelines for choosing the right approach. (In the chapter that follows we give guidelines for choosing the most suitable methods and automated tools.)

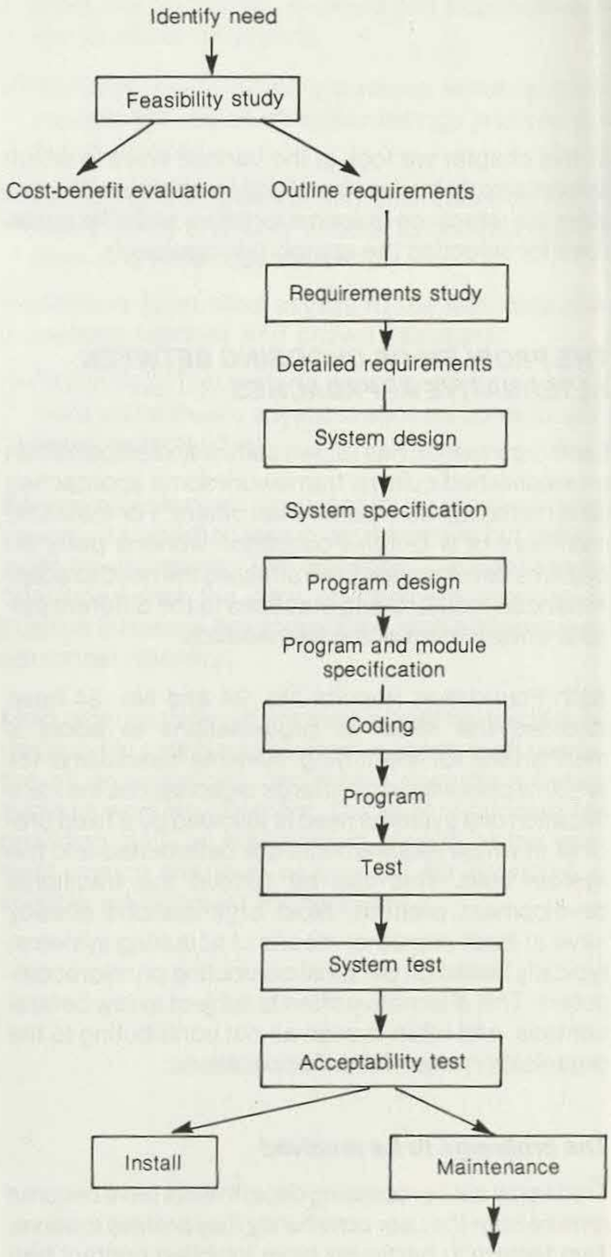
THE AVAILABLE ALTERNATIVES

Several different approaches have been devised to try to ensure the successful development of computer systems. More recent approaches have been designed to overcome the shortcomings in the traditional staged development approach.

The traditional staged approach

The traditional staged development approach is presented schematically in figure 15. In most organisations this approach is regulated by formal standards, and is supported by a limited number of automated tools such as editors, compilers, program libraries, TP monitors, interactive test aids and report generators. The process is thorough but it is also cumbersome and slow. Communication problems can arise between the user and development staff, as well as within the development teams themselves.

Figure 15 Traditional staged system development



Collaborative development

In collaborative development, the system requirements and design are developed by a team of users and their representatives, in close association with professional systems staff. This approach was pioneered by Land and Mumford. Collaborative development is time-consuming, but it enhances user commitment and allows user knowledge to be employed directly.

Case history — user involvement in a collaborative development

This case history examines the experience of a Dutch group of companies. Each company in the group has a medium-term business plan which provides the basis for an associated systems development plan. The implementation of this plan is the responsibility of a supervisory committee. This committee monitors the progress of active projects and takes executive decisions about their conduct.

The initial stages of a development project are undertaken by operating company staff (large systems are developed for mainframe computers). Each company has its own section of business information analysts who are independent of the central systems staff. The feasibility study is undertaken by a project group consisting of one or two business information analysts and one or two end users. The next stage, functional design, is undertaken by an enhanced project group which is responsible for system structure and data analysis. Working in conjunction with the project group on a part-time basis is the user group. This group is responsible for supplying information to the project group. They review project work such as input/output formats, volumes and algorithms for which they provided information in the first place.

Central systems staff become involved only towards the end of the functional design stage to provide advice on database design, run-times and other technical matters. Once the functional design is complete, the systems staff have responsibility for developing the automated parts of the system. Responsibility for developing the clerical parts of the system remains with the project and user groups.

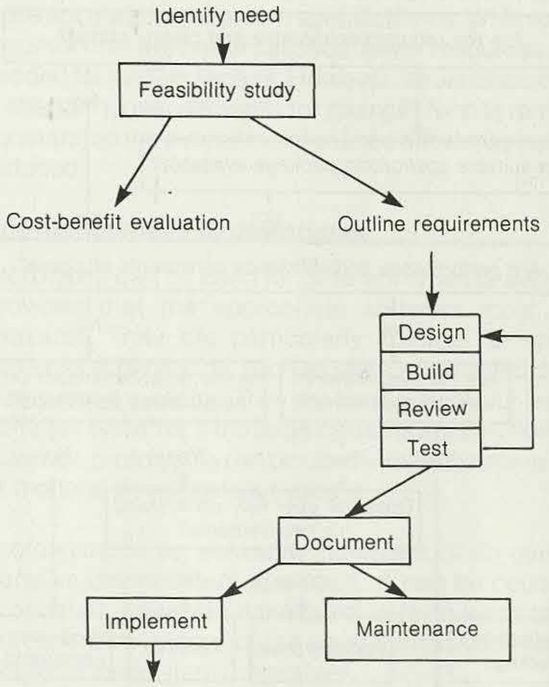
There are now few problems of user acceptance. The users have complete responsibility for functional design, while central systems staff develop the automated parts of the system in accordance with the functional design specification.

Some problems were experienced on large projects where the elapsed time between functional design and implementation was prolonged. In these cases, users lost interest because of the lack of a visible product in a reasonable timescale. Consequently, all systems are now partitioned so that acceptance testing on the early sub-systems begins less than nine months after completion of functional design.

Iterative development

In iterative development, a prototype of the system is built using advanced tools and is then refined until it satisfies the user. Figure 16 shows the iterative process diagrammatically.

Figure 16 Iterative system development



End-user development

This type of development includes a variety of approaches, all of which are initiated by the end user. It includes users who have a report-writing facility to define reports for themselves, and users who develop their own applications using a modelling package such as FCS-EFS and languages such as APL and Basic. This type of approach, which is outside the scope of this report, was fully reviewed in Foundation Report No. 30.

SELECTION OF DEVELOPMENT APPROACHES

Cost-effectiveness (and productivity) is often inhibited when all systems are developed in one standard way. Since this approach is typically appropriate for the most sensitive, large and complex systems, it is often inappropriate for systems that are small, local or aimed at decision-support applications.

We believe that organisations should recognise the four kinds of development process described in the preceding section: traditional, iterative, collaborative and end-user. Many organisations already use several kinds of development approach. Managers and system developers in these organisations should recognise that the non-traditional approaches are not some kind of peripheral anomaly. They are already contributing to the organisation's portfolio of systems and will contribute more extensively in future.

Standard procedures should identify the nature of a new system (or an enhancement request) at an early stage. The request should then be tackled using the appropriate development approach. The complete system need not be developed using only one approach. Often it will be sensible to divide a system into:

- —Operational processing and database maintenance, which will be developed using the traditional staged approach.
- —Standard management reports, which will be developed iteratively.
- —Ad hoc reports, which will be specified by line managers from time to time using a query language or some other end-user facility.

Selection criteria

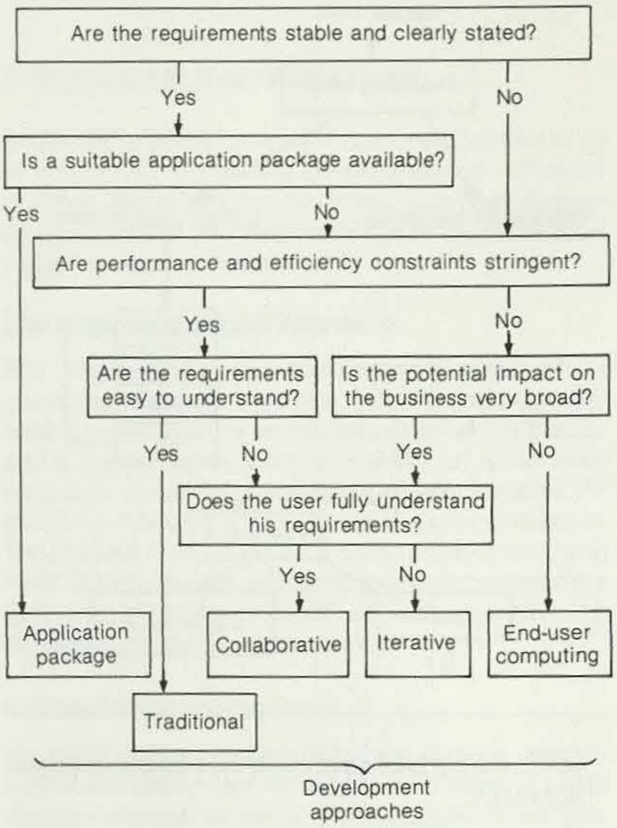
We can identify some general rules for allocating projects to different development approaches. Figure 17, overleaf, summarises how development needs can be assessed on the basis of a small number of key characteristics: commonality of requirements, generality, impact on the business, complexity of requirements, performance requirements and clarity of requirements.

Commonality of requirements

A system requirement may be common to a number of businesses, or it may be unique. Systems with unique requirements usually have to be specifically developed. Systems with common requirements may be able to make use of a package shared with other offices or businesses. An apparently commonplace requirement may be rendered unique in practice by features such as a bonus scheme in a payroll system. Equally, an apparently unique requirement may closely resemble a need commonly met elsewhere.

In recent years some extremely flexible application packages have appeared. As noted in chapter 2, the

Figure 17 Selecting a suitable development approach


Are the requirements stable and clearly stated?
Yes
No
Is a suitable application package available?
Yes
No
Are performance and efficiency constraints stringent?
Yes
No
Are the requirements easy to understand?
Is the potential impact on the business very broad?
Yes
No
Yes
No
Does the user fully understand his requirements?
Yes
No
Application package
Collaborative
Iterative
End-user computing
Traditional
Development approaches

package option is more relevant than ever before.

Generality
Where the particular requirements are unique they may, nevertheless, sometimes be seen as an example of a more general pattern. Thus many financial models may be seen in terms of spreadsheets. In these cases, the need may be met by providing a suitable end-user facility such as an electronic spreadsheet.

Impact on business
The system may have implications for many offices or business functions; alternatively, its effects may be localised. Systems with widespread effects are usually sponsored by a central authority and developed by central systems staff, while systems with only local effects may be developed locally.

For systems having a broad effect, the most appropriate approach is usually either collaborative or traditional (or, occasionally, iterative). However, with the availability of advanced system building tools such as NOMAD, LINC and MAPPER, there are now a number of large operational systems (mostly among engineering companies) that have been wholly developed by sophisticated users with little involvement from systems staff. There have also been a number of conspicuous failures to produce systems in this way.

In the longer term we are convinced that users will be able to develop most of their systems reliably. This will result from an improvement both in the tools and in understanding how they should be used. We advise organisations to experiment with end-user development so that data processing staff can monitor the experiments and learn the lessons quickly.

In general, it is desirable that systems with a narrow impact should be developed by their users. Our research for Foundation Report No. 30 made it clear that organisations must now formulate policies and plans to guide (and perhaps control) the development of end-user computing. One of the key elements of these plans is the formation of an end-user support service. Such a service can be used either to control and restrict the activities of end users, or to assist end users in enhancing their computing. The same Foundation Report strongly recommended that this service should be staffed by high-calibre people, having the potential to progress to management positions.

Systems with narrow impact which will be replicated in many dispersed units should be developed under the supervision of a central sponsor. Often this arrangement helps to discourage dispersed units from developing their own systems, except when agreed by the sponsor and the central data processing department.

Complexity of requirements
At one extreme an application may be highly structured, or at the other extreme rather simple. Sometimes, apparently complex requirements may be expressed as a set of simple processes. Complexity in the applications domain suggests iterative or collaborative development. Complexity in computing requirements suggests central and conventional development. Simplicity suggests user development or central development using advanced tools.

Performance requirements
Performance requirements are obviously critical areas for success in system design. The required levels of system response and resilience (especially with large files or large transaction volumes) can often be achieved only through considerable expertise in design. Strict performance targets generally require a disciplined approach and the use of conventional languages (or even assembler) in critical parts of the system. The less rigorous the operational constraints, the greater freedom will exist over both the choice of development methods and the choice of tools.

Clarity of requirements
If the requirements for a system are clear and stable, traditional development is probably the right choice though the most productive methods and tools should

be used. If the requirements lack clarity then development should begin with a prototype:

— If development is to be traditional then the prototype should be discarded when it has served its purpose.

— If development is to be iterative then the prototype will evolve into the finished system.

USING PROTOTYPING TO PROMOTE USER INVOLVEMENT

Regardless of which approach they adopt, systems departments should encourage the active participation of users in the development process. Users must of course be able to identify their own requirements. Increasingly, prototyping is becoming a viable method of achieving this objective.

In the context of this report, prototyping means building a working model of a system — a working model which can be created quickly and relatively inexpensively, and which enables a set of assumptions to be tested. Prototyping is an iterative approach based on trial and error. It can help to clarify user requirements, verify the feasibility of system design and develop the final system.

Requirements for prototyping

One important requirement for effective prototyping is a change in the traditional attitude of both systems staff and users. Both need to be closely involved in an iterative process to specify and develop the required systems — rigid specifications no longer exist. Prototyping demands different procedures, different skills and different tools.

Different procedures are needed:

— To identify the basic requirements.

— To develop a working model.

— To utilise and refine the model.

— To upgrade the prototype to an operational system (if iterative development is used).

Different skills are needed because, despite the reduced importance of formal interviewing, communication remains the key to success.

Different tools are required because prototyping must be supported by the appropriate automated tools (such as a data management system, an application development language, an application generator or re-usable code).

Resources required

Compared with traditional methods, the prototyping approach uses extra staff resources (10 to 20 per cent extra) during the early stages of a project. But it requires fewer staff during later stages because of more accurate requirement specifications. When user requirements are better satisfied, fewer resources are needed for system repairs. However, as we discussed in chapter 1, user demands for changes tend to remain constant, so the overall maintenance effort may not be reduced.

Appropriateness of prototypes

Prototypes can be used for different types of system provided that the appropriate software tools are available. They are particularly suitable for small business applications, such as stock control and decision support systems. They are least suitable for large, complex systems. If the large systems are partitioned, however, prototyping can be used within discrete areas of the total development process.

Prototyping is an essential ingredient of an overall iterative development approach. It can be counterproductive, however, if the initial version leads to an unresponsive system, or to a short-term solution which neglects longer-term objectives.

A further danger of prototyping is that users may be so content with the prototype that it becomes the operational system. Prototypes are not intended for that purpose. A prototype may not, for instance, contain all the fail-safe and recovery features of the eventual system. Prototypes may fulfil only the main requirements, and not the subsidiary ones which often 'make or break' a system when it is used in a live environment. It is vital that prototyping activities follow a well-defined plan.

CASE STUDY: POSITIVE STEPS TO INVOLVE USERS

The key element in all the new development approaches is the increased involvement of the user. One approach that has been successful in Scandinavia, the Netherlands and Belgium is the use of Staffan Persson's Bottleneck Analysis (reference 10).

In this approach, small groups of employees are progressively consulted during the definition of business procedures. The overall systems structure is built up by an amalgamation of these definitions. Four stages are involved:

— Identifying relations between the information system, the work organisation and the company objectives.

— Identifying and describing the operational procedures and organisation.

— Undertaking a design/definition programme with the users.

—Utilising an automated tool to define data and its structures. A relational database is the best form of tool to model the structures, although there are performance limitations for most implementations.

Even where a system is not clearly defined, Dr Persson undertakes development projects on a fixed price basis — something which is possible, he argues, because the main parts of all systems can be defined early in the project. Because the development is undertaken in close collaboration with the user, a potential failure can be detected and corrective action taken at an early stage.

Akzo Systems BV in the Netherlands has successfully developed a number of large and small systems utilising the Bottleneck Analysis approach. Akzo's development method COSAM (Co-operative Organisation and Systems Development Method) is supplemented by the documentation and system support tool SWISS. SWISS is used for the definition of each of the data items; the definition of the relations, keys and dependent data; program generation from data structures (dialogue generation for display programs and report generation for list programs); and finally for menu generation. The result is an information system that can be implemented gradually.

COSAM consists of nine stages, the last three of which are supported by SWISS (see figure 18).

The result is an approach that involves more resources than traditionally in the early stages, but fewer in later stages.

Figure 18 Stages of the COSAM development method

Stage 1 Preparation — naming functions
— task description
— discussing social and organisational consequences

Stage 2 Describing the present system

Stage 3 Analysis and improvement

Stage 4 Functional design — data dictionary
— bubble charts
— Backmann diagrams

Stage 5 Developing the operational plan

Stage 6 Implementation

Stage 7 Prototyping
Stage 8 Reviewing } Using SWISS
Stage 9 Rebuilding/improving

CHAPTER 5

ADOPTING SYSTEM DEVELOPMENT METHODOLOGIES AND AUTOMATED TOOLS

Having described in chapter 4 the different approaches to system development and discussed the advantages of each one, we turn now to examine the available methods and automated tools for improving the productivity of system development.

METHODOLOGIES TO SUPPORT THE APPROACHES

In this section we briefly consider the different approaches before describing the methodologies that are available to support them.

Traditional staged development and collaborative development approaches

Although the need for standards and a disciplined approach in both traditional staged and collaborative development is now widely recognised, our research indicates that these are not always practised. In many organisations there is still considerable scope for improvement in the application of standards. Structured analysis and design disciplines have been introduced with considerable benefit in many organisations using the traditional staged development approach. Particular advantage can be gained in the construction of large and complex systems in well-managed organisations. The value of these disciplines has been less significant in less well-managed environments. Structured methods have been developed to deal with the problems and systems of the l970s. But in many circumstances the same benefits can be achieved by increasing the level of staff skills.

In traditional staged development, a prototype can sometimes be a useful aid to defining system requirements. Only the more complex or vague parts of the system need to be prototyped. The final system can then be built in the conventional way, and the prototype discarded.

Iterative development

As we indicated in chapter 4, iterative development may be used for a wide variety of applications, for both new developments and enhancements. Since experience is still limited, users should be cautious about this approach. (It is clear that conventional documentation standards are inappropriate for iterative development, but it is not yet clear what the new standards should be. Such methodologies as PRODOSTA-R and PRIDE/ASDM (which we describe later in this chapter) encompass the iterative approach, but they still have a traditional approach to documentation.)

Iterative development requires advanced system building tools or languages. It has been successfully practised with APL, ALL, NOMAD, RAMIS and other system building tools. But this form of development is inconsistent with the existence of separate analysts, designers, and programmers. The systems staff involved must be able to fulfil these various roles in rapid succession.

End-user development

Today, end-user computing is regarded increasingly as an alternative, and legitimate, system development approach (as we described in chapter 4). In Foundation Report No. 30, on end-user computing, we identified guidelines to enable an organisation to get the maximum benefit from this activity, whilst at the same time minimising the associated risks.

AVAILABLE METHODOLOGIES

During our research we identified five examples of methodologies that are significantly affecting systems development. The five are as follows:

- —Data analysis methodology.
- —Systems development methodology.
- —Structured analysis and design.
- —Information systems work and analysis of change.
- —PRIDE/ASDM (which embraces project management, systems development processes and automated aids).

The main characteristics of the five methodologies are summarised in figure 19, overleaf, and each is then described in turn.

Data analysis methodology

Data analysis is a methodology developed by CACI. It incorporates data and process analysis. The development cycle consists of six phases, of which analysis and design are specified in detail. The methodology is most suitable for the development of systems in a shared data environment. These sys-

Figure 19 Characteristics of five development methodologies

	Data analysis methodology	System development methodology	Structured analysis and design	Information systems work and analysis of change	PRIDE/ASDM
Applies to all stages of development	–	✓	–	×	✓
Includes project management tools	×	✓	×	×	✓
Includes automated tools and techniques	–	–	–	×	✓
Assists in documentation	✓	✓	✓	–	✓
Applicable to large and small systems	–	–	–	✓	✓
Intended for shared data systems	✓	✓	–	×	–
Aimed at systems staff	✓	✓	✓	–	✓
Aimed at users	–	–	–	✓	–
Top-down approach	×	–	✓	✓	✓
Structured approach	–	✓	✓	–	✓
Includes entity modelling	✓	✓	×	–	–
Includes data-flow	✓	✓	✓	×	–
Uses a data dictionary	✓	✓	–	×	✓

✓ = definitely included × = definitely not included
– = partially or peripherally included

tems are likely to be complex and to use database software.

The methodology is intended for systems staff. It is comprehensive, with the emphasis on tools and techniques rather than on documentation requirements. It emphasises the use of diagrams and the need to record information within a data dictionary.

The data analysis is based on a conceptual data model represented in terms of data entities, attributes and relationships. The process analysis is similar to the Yourdon structured approach. All activities are discrete and are actioned in series. There is little allowance for iterative work, and a top-down approach is not used.

Systems Development Methodology (SDM)
Systems Development Methodology was developed by BIS and by Learmonth and Burchett. It is a highly structured, data-driven method of systems analysis and design which covers all phases of systems development. It incorporates well-defined procedures for both logical and physical design and relies heavily on the data analysis approach.

The major techniques used in the methodology are data flow diagrams, logical data structuring, third normal form of data analysis, data dictionary, walkthroughs and reviews.

The methodology is aimed at all types of system. It uses the data-driven approach and is concerned with understanding the data and its transformations. It is independent of hardware and applies to all kinds of file and database structures. At present, no major software aids are incorporated except for the data dictionary, and standard utilities for testing and checking. The methodology provides program specifications that could be translated into operational programs in any language.

System Development Methodology is a comprehensive and complex methodology which incorporates a large number of stages, tasks and techniques. It is aimed at systems staff, who need to be experienced and skilled in order to apply the various techniques. The user is involved mainly through discussion with the analyst together with walkthroughs and reviews.

Structured Analysis and Design
Structured Analysis and Design (SASD) covers the analysis and design phases of the development process. The Yourdon company, which provides training and consultancy for SASD, refers to it as a set of process-oriented techniques. The techniques include data flow diagrams, structure diagrams and structured English. These can be combined with products, such as data dictionaries from other suppliers.

The analysis and design phases are broken down into well-defined sub-phases with checkpoints at which users can check on the previous phase. The design phase produces a set of program specifications with supporting design documentation and operational procedures in the data dictionary. The methodology is intended for use on medium to large projects where the problems of complexity and communications with the user are significant. These systems often require the solution to be partitioned via functional decomposition.

Some large SASD users have written graphics software that allows the interactive development of data flow diagrams. Some of these are now becoming available as commercial packages.

The approach requires trained and skilled systems analysts. Data flow diagrams play an important role in allowing the analyst to demonstrate models of the system for verification by the user.

Information Systems Work and Analysis of Change
Information Systems Work and Analysis of Change

(ISAC) is a problem-oriented approach to systems development, created by Professor Lundeberg. The methodology is user-driven and systematic. It charts work processes and leads to a diagrammatic representation of both an activity and an information model. No automated tools are commercially available to support the methodology.

To understand wider problems and implications than those specified by the scope of the system, analysis is undertaken by breaking the problem down into smaller units. The initial steps in the process can be performed only by business managers and analysts who have a wide understanding of the business functions of the organisation.

ISAC gives no detailed guidelines on how to perform each phase, but examples are available. Maintenance is not specifically covered in ISAC as a separate phase.

The use of ISAC does not require detailed technical knowledge until relatively late in the development process, after data structures and computer routines have been designed. The user is closely involved in analysis and design. The methodology has been used in large and small organisations. But it is potentially most useful in complex situations where a top-down analysis of problems reduces them into manageable sub-problems.

PRIDE/ASDM

A high level of automation of the systems development process is incorporated in PRIDE/ASDM as developed by Bryce & Associates. This is a systems development methodology which incorporates project management methods. The system is similar in concept to METHOD-1 and PRODOSTA-R, but its great advantage is that it incorporates a data dictionary feature and automated design facilities.

The methodology divides systems development into nine phases, ranging from the initial system study through to the audit of the installed system. An automated dictionary/directory stores the definitions of systems, organisational entities, data, procedures, and programs. It generates data diagnostics and phase documentation. The design method is based on the concept of grouping outputs by the time cycles in which they must be produced.

In the automated design process, the outputs are defined first as the analyst performs the initial system study. Each of the outputs is defined in terms of a cycle, offset within a cycle and response time requirements (for example, a daily report, produced at 1 pm with a response time of ten minutes). The system analyses the dataflows, relating them to their sources. As the design advances the system can be divided into sub-systems and more detailed design undertaken for each sub-system.

Again data definitions are input to the system, together with the procedures necessary for producing the desired outputs. The system identifies errors of inconsistency or omission. This whole process is iterative, with the design progressing on a trial and error basis.

This comprehensive methodology offers considerable advantages for the development and maintenance processes and has been used successfully on both large and small projects. So far the major productivity improvements relate to the design of batch systems, although the suppliers are enhancing the automated facility to provide assistance for on-line system design.

INCREASING THE DEGREE OF AUTOMATION

Significant productivity improvements can be achieved by increasing the degree of automation available to support systems staff. As the technology advances, so the quality of system design aids will continue to improve, becoming easier to use, requiring less effort, and assisting in the logical design.

Advantages of increasing the degree of automation

Many advantages arising from increasing the degree of automation can now be enjoyed by systems staff. A wide variety of products are available to automate or support the process of system design and construction. Almost all these products work in one (or more) of three ways:

- —They improve the user or developer's access to computer power.
- —They provide pieces of code, or whole programs, that would otherwise have to be written specially.
- —They allow data and functions to be defined in a more convenient way than in conventional languages, often in a way that the user can understand.

Access to computer power may be provided through programmers' workstations and software workbenches, through front-end development systems and through the use of interactive compilers, editors, debuggers and interpreters.

Re-usable code may be provided through standard modules for file access, calculation or business functions; through data centre utilities; through a library of standard program skeletons; and through operating systems, DBMSs and TP Monitors ('middleware').

Better ways of defining functions were first provided by report generators and query processors. Later, file definition capabilities and dedicated very high level languages were introduced. The most recent developments are comprehensive systems development packages and facilities which short-circuit the normal development process by imitating a familiar manual data structure such as a spreadsheet or a box of file cards. Products currently on the market provide the following benefits:

- —They improve the image and self-esteem of systems staff through increased professionalism.
- —They support prototyping and thereby ensure that the system matches the user's requirements more closely.
- —They reduce the proportion of routine and tedious activities, thereby enabling systems staff to be more creative.
- —They reduce the lead time from project initiation to system delivery.
- —They improve the overall productivity of the data processing function.
- —They identify careless errors (inconsistency, omission, duplication).
- —They enable changes to be made with a minimum of disruption.
- —They require a disciplined and structured approach (and therefore help to enforce standards).

Improving access to computers

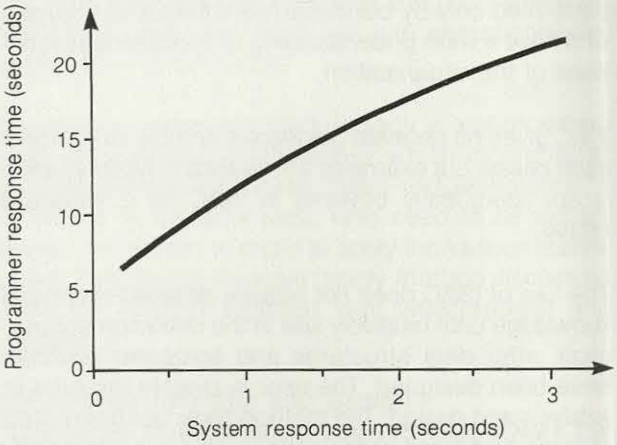
In many organisations, programmers and analysts are still poorly supported by their computers. Easy and responsive computer access can improve the productivity of maintenance and enhancement work, as well as new development work. The areas that are ripe for improvement include:

- —Response times.
- —Turnround times.
- —Access to documents.
- —Ratio of terminals to systems staff.
- —Source code libraries.

Response times may be reduced by tuning or upgrading an existing mainframe. But this improvement is usually obtained more economically by using a minicomputer-based front-end system.

In a recent project within IBM, one development team was given priority access for TSO (timesharing) work. The team was allocated a dedicated office adjacent to the computer, using fast local line connections. A terminal was allocated to each programmer. The study noted that the time taken by programmers to reply to the terminal (programmer response time) fell dramatically with system response time — see figure 20. Productivity was increased by 58 per cent indicating a saving in total effort of 37 per cent. Of equal significance, the equipment cost per module did not increase — the increased rate of use of machine resources was balanced by the saving in time.

Figure 20 Analysis of programmer response time against system response time



(Source: IBM (1982))

In addition to the productivity gains, the project also benefited from an almost complete elimination of overtime and the early installation of the finished system.

Our research indicates that a number of organisations have noted this effect in reverse. Heavy mainframe usage often results in a degraded service, causing programmers to lose concentration and eventually to become demotivated.

In principle a combination of micro-based workstations, locally shared resources and mainframe access would provide the best facilities. But the software to support such an environment does not yet exist.

A substantial reduction in response time changes the nature of the interaction with the computer. Menu selection, for instance, is efficient when response times are under one second, but is less efficient when response times are five seconds. If textual search can be made sufficiently fast, it will be preferred to the manual scanning of printouts. (Though the use of printouts for debugging has declined in some organisations, it remains a central element in most development work. It is therefore desirable that a fast printing service should be provided for system developers.)

In many organisations there is considerable scope for increasing the ratio of terminals to programmers. One terminal per working programmer is a reasonable target. Often the terminal can be cost-justified by increased productivity.

Online access to specifications and other documents speeds up retrieval and updating activities. Computer support also makes it easy to manage several parallel versions of a document. Good library facilities ease the programmer's job and encourage the sharing of code and skeletons. The use of electronic messaging systems for memoranda and systems documentation can also improve communication between team members.

The Philips Maestro system

Maestro has been designed to run as a stand-alone system linked to the host computer. Currently each Maestro system can handle up to 20 workstations.

Maestro supports program development starting with specifications and continuing through module design, coding and testing. Documentation can be created and maintained throughout the process. A text editor is provided for making additions, deletions, changes and global replacements in a body of code. It is also useful for handling textual information, such as requirement statements and system (or program) specifications. For example, analysts can develop the specification for a new system in text form. The top-down design features, plus the text editor, allow them progressively to correct and enhance the specification.

Maestro provides a variety of functions to support the programming process. It encourages structured programming by providing five control 'constructs' for designing a program. The system can draw structure diagrams of the logic that has been expressed in terms of these control constructs. It allows for the marking of sections of code, text and data, and each of these sections can be indexed. Movement between sections is undertaken by one key stroke. A shorthand feature enables abbreviations to be used for common data-names etc. Significantly, an audit trail is maintained of all changes to each program; and the system monitors the usage of facilities.

Maestro is relatively expensive; each workstation costs about $6,000 for the configuration described in the case history below. But the productivity gains experienced by existing users confirm that the system can improve the cost-effectiveness of the development and maintenance process.

Case history — justifying increases in the degree of automation

A major consideration in increasing access to computers and introducing automated tools is that they may require substantial capital expenditure, which may be difficult to justify in advance.

During our research we contacted a major manufacturing organisation to discuss its systems development environment. The company, which employs more than 70 development staff, has recently made a financial case, justified over five years, for purchasing a dedicated Philips Maestro system to provide one terminal per programmer/analyst. The company and staff have agreed that a 10 per cent increase in development productivity will be achieved, and development timescales are being adjusted accordingly. This productivity improvement will be used as a basis for evaluation in the next round of annual staff assessments (and salary reviews).

The system costs about $100,000 and consists of 16 terminals, 2 x 64M bytes discs and a Philips P7000 processor. An additional payment is required for annual software rental. Each individual is given three days training on the system and five days are allowed for familiarisation over a period of three months. The decision to acquire the system was taken after the company had considered using interactive facilities on the mainframe. This alternative was rejected because of the poor response time (and low priority) of development work on the mainframe.

In addition to the increase in productivity, the most significant advantages of the Maestro system observed during a trial period of nine months were:

—Instantaneous response for editing.

—Terminals with function keys.

—Consistency of approach through the development process.

—Enforcement of standards and procedures.

—Automatic document generation.

—The ability to cross-reference between levels of program definitions within the system, thus providing flexibility of work patterns.

INCREASING THE USE OF COMPUTER-BASED TOOLS

There is no doubt that modern system building tools (SBTs) lead to much higher productivity than conventional languages such as COBOL and PL/1. Some of these tools are referred to as fourth generation languages, and some are also referred to as non-procedural or declarative languages. Conventional languages such as COBOL require the programmer to specify a procedure for achieving a solution as well as strict processing logic. With non-procedural languages, the developer describes 'what' needs to be done and leaves the individual 'how' steps to the software.

The improvements derive from a number of features, including:

- —A rational syntax, which enables more efficient translation into executable code.
- —Integration with a database management system (DBMS), which eases the accessing and updating of files.
- —Non-procedural instructions, which ease the use of languages by reducing the level of technical expertise and training needed.
- —Integrated debugging aids, which ease the testing procedures.

Advanced SBTs are needed for iterative development but they can also be used to improve stages in the traditional development approach.

The selection of appropriate tools depends upon a number of factors associated with an organisation's application portfolio. These include:

- —The nature of the workload: how much is batch processing and how much is on-line? What is the balance between on-line enquiry and file update transactions?
- —The need for operational efficiency: what are the response time requirements? How heavy are the tools on main memory capacity?
- —The need to interface with other systems written with conventional languages.
- —The computer used (because tools are available on only a limited range of computers).
- —Whether systems staff or the end user will be the prime developer.

As a result of our research, we can distinguish between five kinds of SBT. These categories are not exclusive, as illustrated in figure 21.

Figure 21 Categories of system building tools

Category	Example of tool	Supplier
System generators	Application blueprint Customised application system	ICL Hewlett Packard
Language independent program generators	DELTA	Sodecon AG
DBMS-based tools	ADF ADS FOCUS MANTIS NATURAL RAMIS LINC	IBM Cullinet Information Builders Inc. Cincom Adabas AG Mathematica Inc. Burroughs
Integrated toolkits	LINC ALL MAPPER NOMAD	Burroughs Microdata Sperry Univac National CSS Inc.
Discrete tools	Query languages such as SQL Various text aids	IBM Various

System generators

System generators are appropriate to particular kinds of application — often they cannot readily interface to existing files and may be restricted to a proprietary DBMS.

It is essential to evaluate the environment into which the generator is to be fitted if the potential productivity improvements are to be realised. If a generator does not closely fit an application or provide appropriate interfaces, then substantial elements of own coding may be required. This can result in a hybrid application which is not much quicker to develop and may be difficult to maintain.

There are two types of system generators, both of which require programming expertise:

- —Program generators: these create source code as stand-alone programs. These statements can be modified directly, without using the generator. Further, an application may be built up from a combination of results from the program generator and direct coding in the source language.
- —Application generators: these require a run-time portion of the generator to run the application programs; they generally provide faster run-times through the use of pre-compiled code. It is usually very complicated to incorporate programs written in conventional languages with programs generated by application generators.

Both ICL and Hewlett Packard provide examples of products which have been developed to increase the flexibility of system generators and to enhance integration capabilities. The main component of ICL's system, known as an Application Blueprint, is intended to provide users with a 'core' structure designed for the application area concerned. The two main components of the Application Blueprint are a business model for the application area, and a database design with sample reports and enquiries supported by ICL products, both documented in the ICL Data Dictionary System (DDS).

Blueprint offers a standard startpoint from which to modify, expand and build software. The next stage in this process is likely to be the introduction of a product (Application Master) which will generate on-line database applications.

Hewlett Packard also offers a product that utilises the building block concept. This product, which is based on Hewlett Packard's 'customisable' application system concept, consists of a set of application packages that can be made into a tailor-made system by means of a system-aided assembly facility.

The Hewlett Packard approach has three elements: the concept (tailoring), the set of packages (the 'customisable' applications system), and the facilities (Application Customiser and Application Monitor). Hewlett Packard has developed the approach for its HP3000 computer, and it has been used with the company's Materials Planning and Control System for Manufacturers.

The approach involves a special method for constructing the packages themselves, as well as a sophisticated application facility for assembling and using the packages to meet specific requirements. The packages are developed in a parameterised form, and the application facility maintains a set of tables for each application. These tables make up the 'application data dictionary', and they define the application and its operational environment. The system is tailored by modifying the tables; no source code is ever modified. With this approach, the customisable application is a truly dictionary-based system. The dictionary is the depository of all the parts that make up the system.

In practice, the customisable application system concept needs two kinds of automated facilities — one to tailor the application and another to run it. The Hewlett Packard approach provides two 'tools' — the Application Customiser (AC) for tailoring the system and the Application Monitor (AM) for operating and controlling the tailored application system.

Language-independent program generators

Language-independent program generators are appropriate where there are a number of target environments and where execution efficiency is particularly important. They have a place in the traditional development process, particularly when structured techniques are used. Their value in maintenance and enhancement work is doubtful.

DELTA, a program generator (see reference 11) has been used on some very large systems, including a 320 person-months banking package for Sperry. One DELTA statement expands into three or four Cobol statements. Production rates can be 3.5 statements per hour (generating 10-14 Cobol statements per hour).

DBMS-based tools

DBMS-based tools are especially appropriate when the DBMS is already in use. They may be used to provide enhancements to DBMS-based systems, for new developments and in iterative developments. Some of them, such as RAMIS II and FOCUS, may be used also for end-user development. These non-procedural languages provide users with facilities to create applications and set up database files.

RAMIS requires relatively little training (typically three days). Development timescales can be reduced to one third of the COBOL or PL/1 equivalent. The response time of RAMIS programs is poor within a timesharing environment, and these programs require a relatively large amount of main memory.

RAMIS can be useful for implementing rapid changes and for ad hoc reporting. One study (reference 12) claims that RAMIS is valuable in system conversions. In an example cited by the study, changes were made to 40 RAMIS programs in just four hours. However, experience shows that RAMIS is not suitable for real-time updating, multiple terminal updating or very high volume systems.

FOCUS is marketed to end users as well as being presented to data processing staff as an SBT. For this purpose the 'Dialogue Manager' facility allows systems staff to build a 'black box' between the system and the user. FOCUS is written in a mixture of Fortran and Assembler and it generates source code. There is a common syntax to all output modules. The data dictionary allows FOCUS to describe virtual data structures and relationships.

Both RAMIS and FOCUS allow complex business applications with links to COBOL or PL/1 routines, for instance, to be created but this demands considerable programming expertise.

The SBTs vary in their usefulness depending on two factors:

- —Their intended purpose: are they to be used for high performance, operational applications, or for ad hoc applications with few performance or efficiency requirements?
- —The person intended to apply the product: is the person to be a systems professional, or the end user?

Figure 22, overleaf, illustrates the way in which some available products are related to the supplier's intend-

Figure 22 Alternative categories of system building tools

	Intended purpose	
Intended development person	To deliver high-performance and efficient operational applications	To develop ad hoc applications with low performance and efficiency requirements
System staff	ADF ADS ALL MANTIS	NATURAL
End user	LINC MAPPER	FOCUS NOMAD RAMIS

ed market. We next outline five of these products — MANTIS, NATURAL, FOCUS, MAPPER and LINC.

MANTIS
MANTIS, which is marketed by Cincom Systems for IBM compatible machines, is used chiefly by systems staff to develop high performance and efficient operational applications. It combines a good structural language with screen development tools. It can be linked to TOTAL and other IBM-based databases. Currently, MANTIS contains no report generator and is available for on-line systems only.

NATURAL
NATURAL is a full programming language, comparable to COBOL, that was originally designed as a query language and report writer. It was developed by Software AG. The distinctive features of the language facilitate access to ADABAS database records, control of terminals and the process of program writing. NATURAL programs are smaller than COBOL equivalents, and their respective design philosophies are different. Under some circumstances NATURAL programs may behave in ill-defined ways or may terminate, and it is often difficult to make provisions against these problems. NATURAL is used by many end users as a query language or report writer; it is also used by systems staff to develop ad hoc applications.

FOCUS
FOCUS, a DBMS-based tool aimed at end users, is available from Information Builders Inc. for use on IBM compatible machines. It provides sophisticated file accessing and interlinking facilities, but is subject to performance constraints because it contains an additional layer of software. It is not appropriate for high-volume, high-performance systems.

MAPPER
MAPPER is a Sperry SBT running on large series 1100 computers. It is an on-line system which can be used to create files, reports and processing procedures that lie between high performance, efficient applications and ad hoc applications with no efficiency constraints. Its simple functions can be quickly learned by end users, typically in two days, but learning all functions requires considerably more time. MAPPER employs a collection of shared files or tables which constitute a simple form of relational database, which is not integrated with Sperry's DBMS 1100 database system.

LINC
LINC is a Burroughs product which uses the DMS database in a way that is incompatible with normal DMS use. Currently, LINC DMS files are not available to non-LINC COBOL programs. Also the reporting function is batch oriented. LINC is intended for efficient, operational applications where the end user carries out the bulk of the development work.

Integrated toolkits

Integrated toolkits, which provide a range of facilities, have their greatest value for new developments. Some toolkits such as ALL can access alien files and may thus be used to enhance existing systems. Some toolkits such as NOMAD impose substantial run-time overheads; others such as LINC do not.This latter group may even produce more efficient systems than with conventional languages, because the routines are written in Assembler and can use pre-compiled code.

Entry level costs for most of these systems is above $100,000. For that investment, however, significant productivity gains are claimed. One user of LINC has experienced a 70 per cent reduction in programming and testing activity and a 50 per cent reduction in overall project effort. Similar claims are made for the improvements in maintenance of LINC generated programs.

Another organisation initially estimated that a conventional development project, to develop a production control system using COBOL, would take six person-months of systems work and two person-years of programming. In the event the system was developed using ALL, and in eight weeks a systems analyst and the production manager had 50 per cent of the application working.

ALL, NOMAD and LINC may be used for prototyping and the iterative development of systems. Specifications are not 'frozen' until the users have seen a tangible product and gained 'hands on' experience.

Most organisations we visited during our research claimed that the productivity improvements gained by the use of integrated toolkits would allow them to develop systems more quickly and hence to handle more user requests. Any consequent reduction in staff numbers usually was expected to take place through natural wastage.

Discrete tools

Discrete tools such as query languages and text aids may be used in various contexts and must be evaluated individually.

One product from IBM is the structured query language SQL/DS. This is a powerful manipulation language which is linked to a relational database management system. Data from an IMS or DL/1 database must be extracted and imported into SQL/DS before the user can make enquiries.

Performance considerations

All the advanced software tools reviewed above raise two significant issues in relation to machine performance:

—How efficient is the code produced by software tools?

—How fast is the response?

One IBM study (reference 13) has indicated that the machine cycles used in running many applications for their lifetime are less than the machine cycles used for assembling or compiling them. This is true of both advanced software tools and conventional languages. A further IBM study (reference 14) indicated that 50 per cent of the application programs accounted for only two per cent of the machine execution time. For over 90 per cent of the application programs, development and maintenance costs exceed lifetime execution costs by a factor of 10. This would indicate that for most application programs machine efficiency is not a significant factor.

Advanced software tools which use pre-coded modules or modify skeletal programs will be machine efficient. As we indicated above, tools of this type can often create object code which is more efficient than code produced via COBOL or PL/1.

Pre-compiled modules can reduce response times substantially. The use of re-entrant code in a multi-user environment means that there is substantially less paging than with conventional programs, for which every code module is different. In his latest book (reference 15) James Martin cites an IBM installation where PL/1 applications were replaced with the same applications written in DMS (IBM's Development Management System — a tool for CICS applications). Eight applications ran concurrently and shared modules of DMS. The average response time dropped from 3.4 seconds to 0.6 seconds and the range of response times dropped from 7.1 seconds to 0.8 seconds.

THE CHANGING ROLE OF ANALYSTS AND PROGRAMMERS

Computer-aided tools are available now for both systems analysts and programmers. For analysts they include application generators, high-level languages such as MANTIS and integrated toolkits such as ALL. For programmers, they include program generators such as DELTA, generalised software environments such as UNIX and PICK, and programming workstations.

Generalised software environments can be used as a rapid way of building high-powered command systems without the need to generate conventional program code. Programming workstations are dedicated systems that help the programmer to do the job. The relevance of these systems was discussed in Foundation Report No. 25.

The use of computer-based tools, together with improved access to computers, is leading to the integration of the separate tasks of the analyst, designer and programmer. As the task of programming becomes technically less demanding and computer resources become more widely available, program code may be generated by a wider variety of people. Once again it will become feasible to combine the role of the analyst and programmer. This change in the nature of the role of systems staff will be one of the main features of the data processing environment over the next five years.

SUMMARY

The case for increased automated support should be made at an organisational rather than at a project level. The experiences described in this chapter indicate that productivity improvements of more than 25 per cent (and sometimes more than 50 per cent) can be achieved by increasing the degree of automation. If these improvements are translated into numbers of systems delivered or staff reductions, then a positive financial case can be made.

CHAPTER 6

TRENDS IN SYSTEMS DEVELOPMENT AND MAINTENANCE AND GUIDELINES FOR MANAGEMENT ACTION

There have been significant changes over the past ten years in most data processing organisations — particularly in the larger ones — as we pointed out in chapter 1. Changes over the next ten years are likely to be even more significant. In this chapter we first look at trends in the systems environment, then summarise the implications of these trends for the data processing department, and finally set out guidelines for improving the cost-effectiveness of systems development and maintenance.

TRENDS IN THE SYSTEMS ENVIRONMENT

In this section we consider those factors which are most likely to influence the systems environment over the next ten years.

Improving hardware price-performance

Hardware price-performance will continue to improve, helping to dispel the concern of many data processing managers about the relative inefficiency of automated tools and techniques. The cost of main store and processing power required by these tools will often be justified by productivity gains.

As a result, the degree of automation within data processing departments will significantly increase. In effect, machine costs will be substituted for people costs.

Increased software capability

Software will continue to be a growth sector in the information technology industry. The result will be an increase in the capability and variety of tools as well as their ease of use. On the one hand, this will enable organisations to select tools most closely matching their requirements. On the other hand, it will exacerbate the task of evaluating and selecting the right tool.

As a consequence, data processing departments will become sources of expertise in techniques, determining the software appropriate for each application and providing consultancy and training services to users.

Diminishing role of programming

With improvements in hardware price-performance, reductions in the costs of software products and the increasing availability of high-level system building tools, the programming role will diminish in importance. Analyst/ designers will be able to generate applications without needing to program in the accepted sense of today. Rather, the job of programming will shift towards program and application generation, undertaken by analyst/designers using non-procedural facilities.

Increasing significance of the data resource

The more general recognition of data as a key company resource will encourage the spread of databases. Systems analysts' expertise will grow in the fields of data analysis and data structures (though not in technical matters, which will continue to be the responsibility of the database administrator).

Increasing role of the user

Business competitiveness will grow rather than diminish. As a result, the traditional system life-cycle of five to seven years will reduce significantly. This will affect the underlying economic justification of new systems development, and underline the importance of rapid system delivery. In turn this will encourage users to develop more applications themselves. Users will become more involved in systems processes and systems management. The emphasis will shift away from traditional development and towards collaborative development.

Users' expectations of business systems will continue to rise. This will result from the spread of microcomputers with easy-to-use packages (such as VisiCalc and Wordstar), from the spread of simple-to-program colour graphics, and from the growing familiarity with computers of school leavers. Data processing departments will have to improve their delivery performance merely to retain current levels of user satisfaction.

IMPLICATIONS FOR DATA PROCESSING DEPARTMENTS

The trends we have described in the preceding paragraphs imply that data processing departments should:

— Reduce the lead time between project initiation and system delivery in order to meet user demand.

— Become more responsive to users' system requirements, both at the beginning of a project and during its evolution.

—Reduce the backlog of applications.

The need for these improvements is now urgent. Data processing departments that fail to achieve them will put their futures at risk.

GUIDELINES FOR IMPROVING THE COST-EFFECTIVENESS OF SYSTEMS DEVELOPMENT AND MAINTENANCE

In this section we set out guidelines for improving the cost-effectiveness of the systems development and maintenance process. We classify these guidelines in three levels: strategic, departmental and project-level.

Strategic-level guidelines

The discussion in this report has been deliberately wide-ranging. Improvements in productivity and in the cost-effectiveness of systems cannot easily be achieved. A critical factor is that appropriate systems are selected for development in the first place, so as to maximise the contribution to the business. This is a matter of vital importance which is discussed in Foundation Report No. 34.

Departmental-level guidelines

Departmental-level guidelines can be applied to the management of systems development and maintenance. These guidelines include:

—Implement training programmes with the aim of improving the skills and motivations of systems staff.

—Improve staff productivity by weeding out substandard performers.

—Introduce system metrics to provide an objective basis of performance measurement (of staff, methods, tools and techniques).

—Introduce quality assurance procedures (including formal audits of development and operational systems, reviews of documentation and design procedures, software libraries and change procedures).

—Adopt automated project management aids for large and complex projects.

—Improve the work environment in order to raise the level of staff motivation.

—Organise staff specifically to undertake maintenance work, emphasising the problem-solving aspects of the job, and try to attract high-quality staff.

—Increase the degree of automation within the data processing department (renting for a trial period those products whose benefits cannot easily be quantified beforehand).

Project-level guidelines

Project-level guidelines include:

—Adopt and emphasise collaborative and iterative development approaches, recognising and accepting end-user computing as a valid approach.

—Use the criteria set out in chapter 4 on page 23 to help identify the approach most appropriate for new project proposals.

—Break long-delivery projects into smaller elements, each able to be delivered in a shorter time and each retaining user involvement.

—Raise productivity levels by using off-the-shelf software.

—Adopt formal methodologies, such as data analysis (which is useful whether or not a database approach is involved).

—Support the development process by adopting specific development tools in the form of additional terminals, main memory and disc space; or a dedicated development machine or programmer workbench; or documentation program library and testing aids.

—Help to cut lead times (and to increase productivity, user participation and the match between requirements and outputs) by adopting system-building tools.

CONCLUSION

Most data processing departments can increase their development and maintenance productivity by 100 per cent in three to four years, and by an additional 400 per cent in six to eight years, if a co-ordinated programme is undertaken.

Approaching the problem of systems development and maintenance in a piecemeal and ad hoc fashion may produce short-lived gains that diminish or even disappear when another productivity aid is implemented.

BIBLIOGRAPHY

1. McNurlin, B., "Easing the software maintenance burden", *EDP Analyser*, August 1981.
2. Grant, E. and Sackman, H., "An exploratory investigation of programmer performance, *SDC*, September 1966.
3. Boehm, B. W., Software engineering economics, Prentice Hall, 1981.
4. Gilb, T., Software metrics, Winthrop, 1977.
5. Gilb, T., Butler Cox Foundation Management Conference, Edinburgh, April 1983.
6. Stuckle, D., Butler Cox Foundation Management Conference, Edinburgh, April 1983.
7. Cougar, J. D. and Zawacki, R. A., "What motivates DP professionals?", *Datamation*, September 1978.
8. Fitz-Eng, J., "Who is the DP professional?", *Datamation*, September 1978.
9. McCue, G. M., IBM Santa Teresa Laboratory — Architectural Design for Program Development, *IBM Systems Journal*, Vol. 17, No. 1, 1978.
10. Persson, S., Butler Cox Foundation Management Conference, Bournemouth, July 1981.
11. Thurner, R., Butler Cox Foundation Management Conference, Bristol, October 1978.
12. McNurlin, B., "Replacing old applications", *EDP Analyser*, March 1983.
13. Kendall, R. C., Management perspectives on programs, programming productivity, Proceedings Guide 45, 1977.
14. Kendall, R. C., Management perspectives on programs, programming and productivity, IBM, 1978.
15. Martin, J., Application development without programmers, Prentice Hall, 1982.
16. Putnam, L. H., The real metrics of software development, IEEE paper, 1980.

GLOSSARY OF TERMS

An application generator — Generates program code for specific applications by using user-supplied parameters for direct processing of pre-coded routines.

An application package — A set of programs for use in data processing systems. In practice the distinction between application packages and some other types of proprietary software is blurred. For example, some application packages include a high-level report generation language, and some application packages use a database management system for file management tools.

Collaborative development — An approach in which the system requirements and design are developed by a team of users in close collaboration with professional systems staff.

Data analysis — The activity of identifying entities together with the data that describe them and describing these in a data model.

Data analysis methodology — There are two main activities within the methodology (functional analysis and entity analysis) which are normally completed in parallel as complementary operations.

End-user development — This type of development includes a variety of approaches, all of which are initiated by the end user. It includes users who have a report-writing facility to define reports for themselves and users who develop their own applications using a modelling package or programming language.

Entity analysis — The activity that identifies resources and information needed by the organisation. (For example, staff, equipment, orders, personal details, etc.)

Functional analysis — The activity which defines the functions of the business (for example handling orders, maintaining plant, paying staff, etc).

Inspection — A manual analysis technique in which systems or programs (requirements, design or code) are examined in a very formal and disciplined manner to discover errors.

Iterative development — An approach in which a prototype of the system is built using advanced tools. The prototype is progressively refined until it satisfies the user.

A metric — A measure which can be applied to a system or software environment. There are two important stages in defining metrics: to agree on the measuring concept (for example absenteeism) and to agree an economical and accurate tool for measuring the property (for example staff attendance records).

Normalisation in data analysis — The process of achieving the highest possible levels of data independence. (This approach derives from the work of E. F. Codd of IBM's research laboratory in San Jose.) In third normal form (TNF), any one entity will have only one value for an associated attribute type and each associated attribute type will describe only the entity type in question.

A program generator — Creates source code giving stand-alone programs.

Prototype — A working model of a system, which can be created quickly and relatively inexpensively and which enables a set of assumptions to be tested.

A report generator — A processing program which can generate object programs for report generation dependent upon a set of pre-defined parameters.

A system generator — Produces executable software for a particular computer environment directly from a set of functional specifications.

Traditional staged development — An approach in which systems are developed by undertaking a series of discrete stages. Each stage ends with a formal cut-off point, at which a specific set of project documentation is produced.

Walkthrough — A manual analysis technique (mainly for programs) in which a module author describes the module's structure and logic to an audience of colleagues.

Butler Cox & Partners Limited
Morley House, 26-30 Holborn Viaduct, London EC1A 2BP
☎ 01-583 9381, Telex 8813717 BUTCOX

Belgium & The Netherlands
SA Butler Cox NV
Avenue Louise – 479 – Louizalaan,
Bte – 47 – Bus.
Bruxelles 1050 Brussel
☎ (02) 647 15 53, Telex 61963 BUTCOX

France
Butler Cox SARL
Tour Akzo, 164 Rue Ambroise Croizat,
93204 St Denis-Cedex 1, France
☎ (1)820.61.64, Telex 630336 AKZOPLA

United States of America
Butler Cox & Partners Limited
P.O. Box 590, Morristown, New Jersey 07960, USA
☎ (201) 285 1500

Switzerland and Germany
Butler Cox & Partners Limited
Morley House, 26-30 Holborn Viaduct, London EC1A 2BP
☎ (London) 583 9381

Italy
Sisdoconsult
20123 Milano – Via Caradosso 7 – Italy
☎ 86.53.55/87.62.27, Telex 311250 PPF MI

The Nordic Region
Statskonsult
PO Box 4040, S-171 04 Solna, Sweden
☎ 08-730 03 00, Telex 127 54 SINTAB