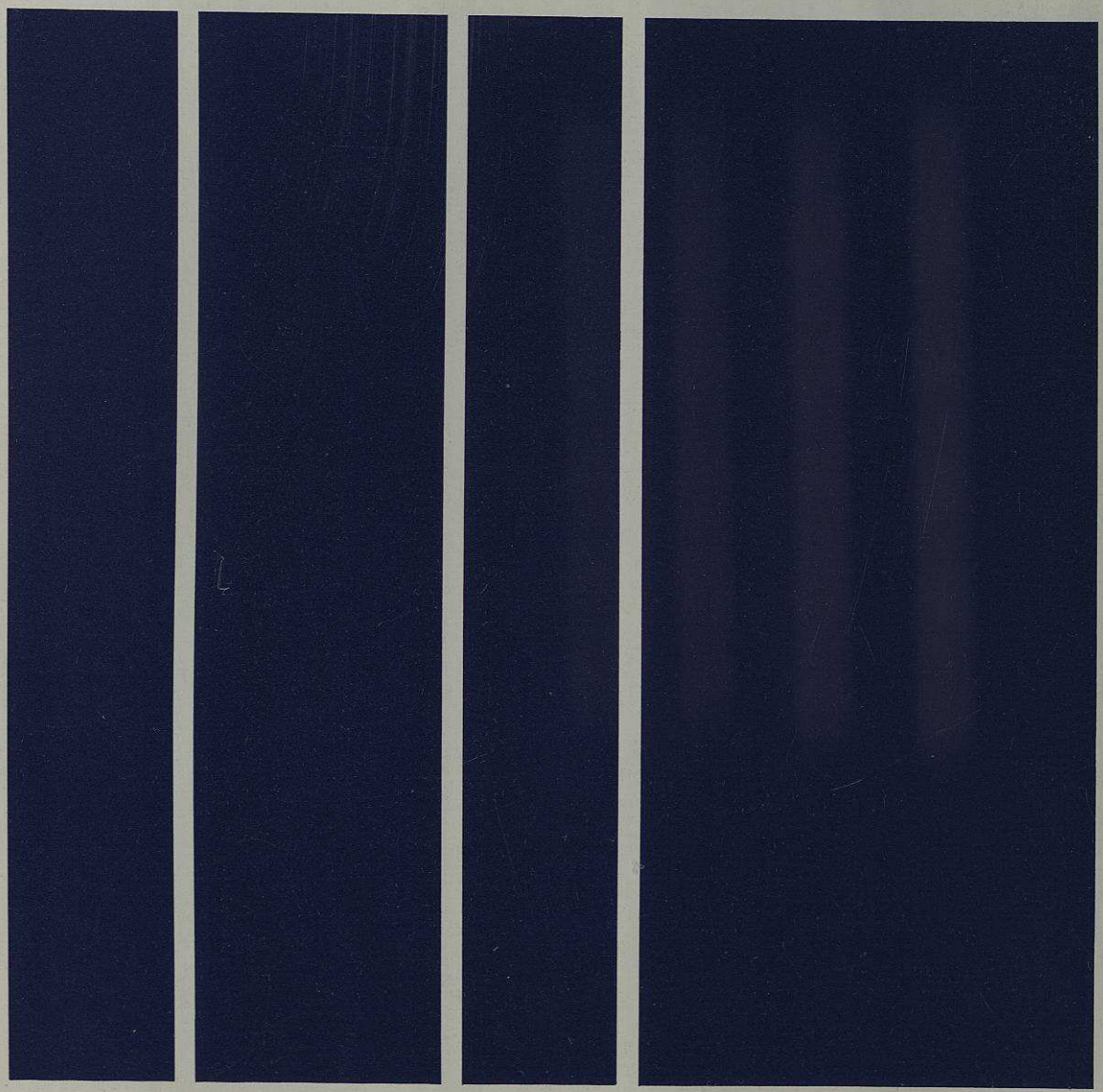
Report Series
No 25

System Development Methods

November 1981



The Butler Cox Foundation

Abstract

Report Series
No 25

System Development Methods

by Elisabeth Somogyi

November 1981

The purpose of this report is to encourage organisations to take a critical look at the way in which they develop systems. The system developer has available today a much greater variety of system development approaches and methods than were available to him in the past. This greater variety has resulted from advances in technology, from past experience of developing and using computer systems, and from theories both about systems and the system development process. This report provides organisations with an overview of the different approaches that they might take to system development, so that they will be better placed to make informed judgements about the relative merits of different approaches.

In this report, we show that the system developer's perception of what a system is determines the approach and the method he will use when developing a system. We also review (and classify) the currently available methods, methodologies and approaches in relation to the two extreme perceptions that system developers can take about systems.

The report considers the well-known technically oriented approaches and also the newer planning and evolutionary approaches. It highlights the relevance of user participation, and it discusses the methods by which this participation can be achieved. In addition, the report reviews the automated tools and support systems that are available to aid the system development process, and it provides guidelines for selecting and introducing new system development methods.

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

The Foundation collects its information through its office in London and also through its associated offices in Europe and the US. It transmits its findings to its members in three main ways:

— Through regular *written reports* that give detailed findings and substantiating evidence.

— Through *management conferences* for management services directors and their senior colleagues, where the emphasis is on the policy implications of the subjects studied.

— Through *working groups* where the members' own specialist managers and technicians meet with the Foundation research teams to review their findings in depth.

The Foundation is controlled by a Management Board whose members include representatives from the Foundation member organisations. The responsibilities of the Management Board include selecting topics for research and approving the Foundation's annual report and accounts, which show how the subscribed research funds have been employed.

Report Series No 25

SYSTEM DEVELOPMENT METHODS

by Elisabeth Somogyi

November 1981

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

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

CONTENTS

CHAPTER 1

INTRODUCTION

During the brief history of commercial data processing, many organisations have discovered that it is all too easy to develop application systems that do not meet their expectations. Most organisations will be familiar with system development projects that have been perpetually 60 per cent complete, or that overran their original budgets for time and cost by several orders of magnitude. And, even when such systems passed from the development stage to the operational stage, they often did not meet the users' real needs, and sometimes they actually made the users' jobs more difficult to perform. In addition, application systems proved to be most difficult to maintain and to adapt to the changing needs of users, and often, maintenance and adaptation could be performed only by selected specialists.

In order to combat the problems mentioned above, most organisations have evolved a set of standard practices that they now use for system development. Organisations usually base these standards on those practices that have proved to be successful in the past, and most 'standards' for system development consist of a set of rules and procedures that are rigorously enforced. The system development standards that most organisations currently use concentrate on the construction of the 'computer system' itself, and they have the following basic characteristics:

— They view the computer system as an entity in itself, and so they take little or no account of the wider environment in which the system will be used.

— They are usually based on a staged and analytical approach to system development.

— They require skilled technicians (analysts and programmers) to actually develop and maintain the application systems.

These traditional (or analytical) approaches for system development have proved adequate for many traditional data processing applications. During the past few years, however, several alternative system development approaches and methods have been proposed. The purpose of this report is to provide organisations with an overview of the different approaches that they might take to system development, so that they will be better placed to make informed judgements about the relative merits of different approaches.

Many organisations may wonder why they should even consider using either an alternative way or a new way of developing systems, particularly if their existing system development standards, after a long and perhaps painful evolution, work tolerably well. We believe that recent developments both in computer-based technology and in the way in which that technology will be used, make it essential for organisations to be aware of the various alternative system development approaches and methods that are available today. We have many reasons for believing this, but our most important reasons are as follows:

— Recent developments in technology have opened up new application areas for

computer-based systems (for example, office applications and decision-support systems), and the types of systems in these areas differ from traditional data processing systems. It is questionable whether any one system development method is the most appropriate method for all the different types of systems that computer-based technology can now be used for.

— Technological advances now make it possible to automate parts of the system development process. Before an organisation can make a sensible decision about what part of the system development process it should automate, it needs to have a good understanding both about system development methods and about the nature of the system development process.

— The continuing shortage of the skilled manpower that the traditional approaches require makes it attractive for an organisation to select approaches and methods that non-specialists can use. Consequently, organisations should be seeking out system development approaches that do not waste valuable and scarce human resources, and that also permit non-specialists to be involved in the system development process.

— The current economic climate makes it essential for organisations to minimise the risk of large-scale and expensive system failures. A knowledge of alternative system development methods can help organisations to select economic and effective methods.

— The established method of developing systems may no longer be the most effective method. Although it is not a trivial task to change the method used for developing systems, organisations should, from time to time, be prepared to review their method. As computer-based systems become more ubiquitous, organisations have the opportunity to achieve substantial gains in productivity by developing systems that are more effective than their existing ones are.

APPROACHES, METHODS, TECHNIQUES AND METHODOLOGIES

We have already used the terms 'approaches' and 'methods' several times. People often use these two terms, together with the terms 'techniques' and 'methodologies', when they speak or write about system development. During the research for this report we encountered a good deal of confusion (and even laxity) in the way in which these four terms are used in relation to system development. It is therefore worthwhile defining in some detail the way in which we use the terms approaches, methods, techniques and methodologies in this report.

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. Structured approaches can be sub-divided 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 it 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 within more than one system development approach.

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 a system development technique will often require that a specific tool be used. For example, those documentation methods that are based on the assumption that system design should be represented in a pictorial and a diagrammatic format, require that both techniques and tools be used in order to draw the diagrams. In this report, we do not discuss system development techniques.

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 proven practices for a given area of activity. As an example, within the structured approach, programming methodologies and system development methodologies have been developed.

STRUCTURE OF THE REPORT

We begin in chapter 2 by showing that the system developer's perception of what a 'system' is determines the approach and the method he will use when developing a system. We show that a range of viewpoints can be taken, and we discuss the consequences for system development of taking one or other of the two extreme viewpoints. We also show that there is a natural progression from one end of the range of viewpoints to the other, and that as this progression takes place, system development approaches and methods will need to evolve.

In chapter 3, we examine the influence that both the way in which work is organised, and the way in which technology is advancing have on the evolution of system development methods. Next, in chapter 4, we classify system development approaches under three broad headings, and we then sub-divide these into more detailed classifications. To our knowledge, no one else has attempted to classify system development approaches in a similar way. We believe that such an orderly classification is badly needed because, without it, organisations cannot compare and select approaches and methods from the numerous choices that are now available.

In chapter 5, we turn our attention to the automated aids that are available to support the system development process. Next, in chapter 6, we provide guidelines that organisations can use to select and to introduce new system development approaches and methods.

Following our concluding comments in chapter 7, we then provide a comprehensive bibliography, whose structure mirrors the structure of the report itself. We have adopted this structure for the bibliography to make it easy for those who wish to do so to obtain further details about the many approaches and methods we mention in the report.

PURPOSE OF THE REPORT AND INTENDED READERSHIP

The purpose of this report is to encourage organisations to take a critical look at the way in which they develop systems. The report is therefore intended to be read by the executive responsible for an organisation's systems' function. It will also be of value both to system development managers and to the managers of those user departments that are likely to become more closely involved with the development of their own systems.

CHAPTER 2

TWO SYSTEM VIEWS AND THEIR CONSEQUENCES

Most data processing systems form part of a larger system. This larger system is the total work system that includes every aspect of a working environment — the physical environment, the way in which work is organised, the operating procedures, the tools and the equipment used, the local practices and customs, the organisational structure, the working relationships both between individuals and between groups of people, and so forth. (System development itself is a work system, and we discuss work organisation as it affects system development in chapter 3.) In the larger environment of a work system, data processing systems therefore have to co-operate with the human elements of the work system.

In the past, however, many data processing systems have been developed with a total disregard of the human elements of the work system in which they are embedded. Most work systems are an integral part of an organisation and they usually support the aims, the goals and the objectives of that organisation. Even so, some of the data processing systems that are being created today disrupt the organisation, and they also do little to help the organisation both to grow and to survive. In this respect, then, those systems are failures. However, without an understanding of the way in which people's viewpoints about systems affect the systems that they develop, it is difficult to understand why this should be.

In particular, both the approach taken to develop a data processing system, and the method used for developing that system are determined by the system developer's perception of what a 'system' is. If his perception of a system does not include certain concepts, factors or elements, then — no matter how vital they may be — the system is likely to be developed without them. Consequently, any discussion of system development approaches requires, as a prerequisite, an understanding of the different viewpoints that can be taken about systems.

Data processing systems are part of the wider classification of technological systems, and we present here two simplified viewpoints about technological systems and the likely consequences of holding them. These viewpoints represent the two extremes of a wide range of viewpoints and, in practice, most actual viewpoints are somewhere between these two extremes. Nevertheless, we discuss only the two extreme viewpoints because we believe that there is a natural progression from one extreme to the other. This progression represents a maturing process in the use of technology, and we discuss this maturing process in chapter 3.

We call the starting point of the range of viewpoints on systems the 'closed system view'. The closed system view is a natural outcome of an environment in which a technology is new, and so the main preoccupation of many people is to create and use that technology. In this environment, a technological system is regarded as a particular assembly of the relevant technology, and the boundaries of the system are well defined. For example, computer manufacturers call the hardware and the operating system a system (and the manufacturer is often referred to as a system supplier), and those who create application systems regard the programs, the files, the computer hardware and the operating system software as the system. In both examples a fairly well-defined

boundary exists between the system and its users. The users, so far as the hardware manufacturers are concerned, are those who create the application systems, and the users, so far as the developers of application systems are concerned, are the managers and the staff of the department for which the application system is being created.

Those who hold a closed system view therefore perceive a technological system as a closed assembly of the technological parts surrounded by a well-defined boundary. This view mostly excludes those other relevant factors that may affect the larger work system (such as the users, the company's organisational structure, the environment within the company, the company's financial and other objectives, etc.).

The other end of the range of system viewpoints is the open system view. Those who hold an open system view regard systems as being larger than the technological parts of the systems. They view systems as highly-complex open-ended entities that may have fixed parts (such as those parts that are fixed in a computer), but they see systems essentially as ever-changing organisms in which different parts interact with and modify one another.

In general, the term 'open systems' is used to describe those systems that interact strongly and continuously with their environment, that have many parts, and that are self-regulatory in nature. The term was first used in general systems theory, based on the work of von Bertalanffy (see bibliography), who described living organisms in terms of complex systems. The nature both of complex technological systems and of human organisational structures is not dissimilar to the nature of living organisms, and systems that contain human elements and machine elements can certainly be classified as open systems.

The three prime characteristics of an open system are that it exists in a state of continuous change, that it has a high level of complexity, and that human elements form a large part of the system. An open system has a high level of complexity for two reasons. First, there are many interactive elements in such a system. Second, the essentially non-structured nature of the human element creates a level of unpredictability that is mostly unacceptable to those who rigidly adhere to a closed system view.

The viewpoint that system developers have about the nature of systems profoundly influences their attitude towards all aspects of the development and the use of technological systems. Before we examine the consequences of holding either a closed system view or an open system view, two points relating to technological systems, and to data processing systems in particular, need to be emphasised:

1. Most system developers have now progressed beyond a strictly closed view of data processing systems. But most of the system development approaches that are currently practised originated from a closed system view, and the closed system view still influences the extent to which system developers are ready to accept the newer methods, and the methodologies for, and the approaches to, system development.

2. Data processing systems are, in fact, parts of open systems regardless of whether they are perceived as such. There is a tendency nowadays to call these systems 'embedded' systems, because their prime characteristic is that they form an inseparable organic part of a larger administrative work system. Those who perceive data processing systems as closed systems will inevitably find that this view conflicts with reality. The larger work system is itself an open system, and if the technological parts of it are to support their ever-changing environment they too should be regarded as open systems.

The consequences of perceiving data processing systems as closed systems are different from the consequences of perceiving data processing systems as open systems. We now explore these different consequences under the five headings of the attitudes to system development, the characteristics of a good system, the life cycle of a system, the attitudes towards maintenance, and the use of technology.

THE ATTITUDES TO SYSTEM DEVELOPMENT

The major consequence of perceiving data processing systems as closed systems is that the technological boundary needs to be defined fully and completely before any development is commenced. The technological part of the work system is then developed both within this boundary and in isolation from any other part of the work system. The development process concentrates on making a preselected technology work within a fixed specification. The development process is therefore an inward-looking activity that precision-engineers the technological parts of the work system.

This activity is carried out by specialists and highly-skilled technicians who consider that outsiders, such as users, have little part to play in the development process. The users are normally asked to specify their requirements precisely (and preferably in technical terms), and the specialists regard any subsequent changes in the requirements as an unwelcome intrusion. The specialists do not appreciate that requirements sometimes need to be changed, and they interpret a user's request for changes as indicating that the user did not specify his requirements carefully enough in the first place.

In contrast, those who hold an open system view usually start by envisaging all of the elements and factors that influence a total work system. The system development process is therefore a high-level process of design that optimises the arrangement of all of the system elements before selecting those parts of the total work system that may benefit from the use of technology. The development process takes into account goals and factors of a financial, technical, human and organisational nature, and these goals and factors are carefully balanced in order to specify those areas of the work system in which technology can be employed.

With an open system view, the technological boundary exists just as it does with a closed system view. The significant difference is that with an open system view the technological boundary does not prevent the system developer considering issues that are outside that boundary. The technological boundary is a natural consequence of a higher-level design, and the system developer expects that changes will occur both during and after the development of the technological parts of the total system.

THE CHARACTERISTICS OF A GOOD SYSTEM

Technically-oriented people who hold a closed system view often equate the degree of excellence of a system with the extent to which technology is used ingeniously, even though the ingenious use of technology may not form part of the original specification. This view usually results in technically-complex systems that are difficult to use, difficult to operate and difficult to maintain. In general terms, those who hold a closed system view can use the best method of development to develop a system that is only as good as its original specification. Thus the system development methods that are used by those who hold a closed system view place great emphasis on the need to define carefully the requirements of the system. Consequently, whether or not a system developed by those who hold a closed system view is a good system depends upon:

— How well the technological system matches its original specification.

— How well the technological system employs the selected technology.

System developers who hold a closed system view make little reference to the environment that surrounds a technological system. As a result, the most obvious measure of how good a system is (which is how well it serves the purpose for which it was created) is difficult to include in any evaluation of the system.

By contrast, system developers who hold an open system view perceive a system as consisting of many parts, and as existing in a continuously changing environment. Such an open system is therefore a good system when:

— It can adapt to change.

— It supports the interaction and the co-operation of its elements.

— It promotes its own survival.

— It can support the goal or the mission for which it has been created.

Individual parts of such a system must therefore be capable of developing, modifying and repairing themselves, and these requirements have interesting consequences for the technological parts of the system. Self-development features and self-repair features are difficult to include in a technological system, although various researchers in universities throughout the world are attempting to develop heuristic technological systems that can learn from their own past experiences. As a result, the technological parts of an open system tend to be developed as closed systems, and both the fixed technological boundary and those technological systems that are built to meet the original specification are likely to become inappropriate as the overall open system evolves. They may, in fact, become so inappropriate that they cause the overall open system to cease to function.

A closed system view of the technological parts of an open system is therefore very limiting. The reason for this is that the specification of the requirements of a technological system rarely includes the need to absorb subsequent changes. The technological parts of an open system are effective only if they satisfy the relevant objectives, and only if they also support the rest of the open system in a changing environment. Consequently, an open system view of technological systems requires that the specifications must include elements of the objectives and the goals set for the overall open system, and these objectives and goals, must, in turn, indicate the need to deal with subsequent changes.

THE LIFE CYCLE OF A SYSTEM

Those who hold a closed system view perceive the life cycle of a system as having the following two major characteristics:

— The life cycle of a system is envisaged as a finite series of three stages. These are usually described respectively as the initiation stage (or feasibility study), the development stage (which consists of analysis, design and implementation), and the operation stage (which usually includes maintenance as well).

— The life of a system terminates when parts of the technology on which it is based

are either changed or used in a different way. A new system then replaces the old system, and the life cycle begins again.

Those who hold a closed system view put most of the effort involved in developing a technological system into the engineering of the system (in other words, into the way in which the technology should be used). The time and the effort put into defining the technological boundary and specifying what the system should do represents a small proportion of the total development effort.

The initiation stage and the development stage are usually separated totally from the operation stage, and those individuals who have the greatest technical knowledge are employed only on the first two of those stages. Maintenance activities receive little attention from these skilled technicians, and the maintenance that is carried out is usually performed by inexperienced personnel. Any changes to the system are usually handled as a maintenance activity, and they are incorporated into the system at the lowest possible level (for example, by making a coding change to a program). Amendments to the system are handled in an ad hoc manner, and it is rare to find that an existing system is reviewed thoroughly in a way that leads to a planned programme of change.

A technological system usually forms a part of an open system, and open systems spend their lives in a state of perpetual change. Technological systems therefore have to be modified at regular intervals to suit the evolving open system. Those who hold a closed system view envisage that this process of regular modification creates successive versions of the technological system.

By contrast, those who hold an open system view envisage that the technological parts of an open system need, from the outset, to accommodate the changes that are bound to occur. The need to accommodate changes can be catered for in three ways:

— The system specification can include the requirement to accommodate specific known changes.

— The development method that is chosen can be one that can be used to create technological systems that are easy to change.

— The life cycle of the technological parts of the open system can be planned on a continuous and iterative basis. The life cycle therefore contains regular reviews that enable a decision to be made to either modify or redevelop the technological system.

The perception of the life cycle of a system by those who hold an open system view is significantly different from the perception of those who hold a closed system view. The four main consequences of holding an open system view are:

— The detailed planning and design stage that precedes the development of a system is far longer than it is for those who hold a closed system view.

— Regular system reviews and subsequent modification to (or even redevelopment of) the system form a natural part of the life cycle of the system.

— The technological parts of the system are designed in a different way from those designed by the holders of a closed system view, because an open system view encourages the use of re-usable and changeable system elements.

— The iterative nature of development is clearly recognised.

THE ATTITUDES TOWARDS MAINTENANCE

The most profound difference there is between those who hold one or the other of the two extreme system views is the attitude they have towards maintenance, and the perception they have of the position of maintenance in the life cycle of a system. We have already said that those who hold a closed system view see system maintenance as the lowest possible level of activity (for example, fixing programs). In contrast, those who hold an open system view encourage regular system reviews and the regular redevelopment of systems. But the most interesting consequence of holding a closed system view is the belief currently widespread in the data processing community that too much effort is being expended on maintenance activities. The emphasis on developing systems with a finite life cycle means that many data processing professionals genuinely believe that there will be a continuous need to develop large-scale new systems. They see maintenance as a necessary evil, and to them the continuing increase in maintenance requirements indicates that the original systems were of low quality.

The main reason that causes data processing professionals to hold this belief is that, during the past twenty years, data processing technology has been progressively introduced into most of the business areas that could be computerised on a large scale. During the time that the technology is being introduced it is inevitable that the effort expended on development will be high compared with the effort expended on maintenance. Eventually, though, those areas will become saturated with the use of the technology, and there will be a shift in emphasis from development activities to maintenance activities. A new way of using the technology (for example distributed processing and networking) may produce a temporary surge of development activity, but the trend will inevitably be towards a greater emphasis on maintenance activities.

The effort expended on maintenance activities (compared with the effort expended on development activities) in the data processing industry is not dissimilar to that expended in other industries once those other industries have come to terms with their appropriate technology. There is, therefore, nothing extraordinary about the level of maintenance activity in the data processing industry. What is wrong, though, is that, in general, the wrong systems are being maintained for the wrong reasons in the wrong way. Maintenance should be regarded as an integral part of the life cycle of a system, and to regard maintenance in this way requires that the system developer holds an open system view.

THE USE OF TECHNOLOGY

In addition to ensuring that technology is used in the developed systems, the system developer can also use technology as a tool in the development process.

The closed system view is most prevalent when a technology is new and expensive. Where this condition applies, the technology to be used in a system is usually selected well before the overall system issues are considered. And because the technology is expensive, it is rarely used as an aid to support the system development process. Thus, because data processing development approaches have their origins in an era when the closed system view prevailed, data processing professionals have been slow to realise that there are advantages in using technology as part of the system development process. This is illustrated by the following:

— Databases and database management systems were available to the users of data processing systems long before the data processing community realised that the same technology could be used to keep a record of the parts of a developing system.

— On-line terminals were installed in user departments well before they were made available to programmers.

— Computer-aided design and computer-aided manufacturing techniques are widely used in most disciplines outside of system development. Design and implementation tools that can be used by system developers are now becoming available, but the data processing community has not yet accepted them.

The open system view encourages the belief that the use of technology is justified only to the extent that it can be applied to make the overall system more stable and more adaptable, and generally to assist it to endure. Thus, instead of technology being the constraint around which a system is built, technology should fit into the overall system. The open system view promotes the idea of technological choice, where the appropriate kind of technology is selected for any parts of the system that need to be computerised (and where computerisation is itself one of several possible options). This means that technology is subservient to the requirements of the overall system. However, technology can be employed in the way encouraged by an open system view only when:

— Technological choices exist.

— The cost of technology is relatively low compared with the cost of other elements of the overall open system (such as people).

— The cost of developing the technological part of the system is not too expensive.

At the present time, the cost of developing the technological part of the system is still too expensive, and the implications of this are twofold:

1. Substantial effort is being made to identify cheaper methods of developing the technological parts of systems.

2. As long as lengthy and expensive development methods continue to be used, attention is focused on the technological development activities. Non-technological issues are rarely considered, and any consideration of technological choices is pre-empted because hardware is selected without any consideration of its impact on the larger open system.

The open system view also makes it easier to include technology in the development process itself. The system development process can be thought of as a function that can be supported by automation. In our view, the true potential of new development methods and approaches cannot be fully appreciated unless the system development process is viewed in an open system framework. In chapter 5, we review the ways in which technology can be used as part of the system development process.

SUMMARY

In this chapter we have shown how the view that a system developer holds about the nature of systems affects the way in which he develops systems. Also, by examining the differences there are between those who hold system views at the extreme ends of the complete range of system views, we have illustrated the different consequences of holding different system views. The system view of a system developer determines which of those system development approaches that we classify in chapter 4 he is prepared to use.

The class of system development approaches that have their origins in an era when the

closed system view prevailed are usually called 'technical (or traditional) approaches'. The class of approaches that are based on an open system view are normally referred to as 'systems approaches', or 'holistic approaches'.

CHAPTER 3

THE FACTORS THAT INFLUENCE THE EVOLUTION OF SYSTEM DEVELOPMENT METHODS

In this chapter we examine the two factors that influence the evolution of system development methods. When computers were first introduced into organisations, the way in which they were used and the way in which systems were developed were influenced strongly by the general approach to work organisation that prevailed at that time. The first factor, therefore, that influences the evolution of system development methodologies is the way in which work is organised.

The second factor concerns the way in which people's perceptions about systems change as they acquire a more mature understanding of the way in which to use computer technology. Developments in technology are bound to modify the earlier perceptions about the nature both of systems and system development.

THE ORGANISATION OF WORK

Formal methods of organising the work both of clerks and manual workers have their origins in the classic approach to work organisation (known as Taylorism) that was developed in the United States at the beginning of this century by Frederick Wilmslow Taylor. Taylor developed his approach during an era when machinery was expensive and labour was both cheap and readily available. Thus, Taylorism is based on the premise that people are disposable and machines are not. In addition, Taylorism assumes that, by their nature, people are lazy, passive and unambitious. With Taylorism, therefore, the roles assigned to employees are menial so as to minimise both the individual's discretion and his opportunity to interfere with the 'system'. The essence of Taylorism is to:

— Separate planning from doing.

— Divide the work into small specialised tasks.

— Append people to machines.

For several decades, work in factories and mines was organised on the Taylor principle, but from the 1930s there was mounting criticism of Taylorism. These criticisms were based both on the inadequacies of Taylorism as a social organisation, and on its ineffectiveness as a management tool. After the seond world war, it became obvious that the wrong work organisation had long-term psychological consequences for workers, and so the shortcomings of the Taylor-type organisation were, at the instigation of a Senate committee, seriously investigated in the United States.

These investigations, and the general dissatisfaction with the approach of appending people to machines led to a search for better methods of work organisation. The best-known new approach is Socio-Technical Design. This approach, which was originally developed by the Tavistock Institute in the early 1950s, aims to balance the human and the technical parts of work systems in order to produce the maximum possible improvement. Several theorists have also contributed to the development of work-organisation methods, and the most notable of these are:

— Abraham Maslow, for his work on the analysis of human needs.

— Frederick Herzberg, for his work on hygiene and motivation factors for humans.

— Douglas McGregor, for his Theory X versus Theory Y.

Nevertheless, Taylor's principles have continued to be used in the organisation of office work, and over the years large clerical departments were organised using classical work organisation methods. Many of the first generation of expensive computers were used to computerise the work of some of those departments. Taylorism operates on the principle that the essentially non-structured nature of people makes it difficult to plan and control complex processes, and so it was almost inevitable that early computer systems were designed according to the same principle. As a result, as many complex but repetitive activities as possible were transferred to the computer, and staff were entrusted only with menial input and output operations. This approach to system development resulted in a general dissatisfaction with early computer systems, particularly since the technically-oriented system developers paid no particular attention to the users' needs.

Early approaches to system development also contained traceable elements of Taylorism. The segregation of tasks, the total divorce of the management task and the planning task from the 'doing', the isolation of the development process from the end user, the separation of activities into large uni-functional departments, and, most importantly, the separation of the development of the computer system from the development of the manual procedures were all in the true spirit of Taylorism.

However, as the new approaches and theories about work organisation were developed during the 1950s, the early attitudes to systems and system development needed to be revised. The three main factors that produced new attitudes towards systems were:

1. Social scientists recognised that it is wrong to treat people as adjuncts to machines, and so the concept of 'the right to work' was extended to include 'the right to decide'. The participative approach to work organisation is a direct consequence of this recognition, and the Socio-Technical Design approach recognises that people should not be appended to machines. Social scientists have provided the underlying ideas that form the environment of industrial democracy, employee participation and employee involvement in which systems now have to be developed.

2. Management scientists developed various organisational models and management techniques that have helped to create the overall environment in which systems are now used. The recent tendencies to decentralise organisations and to use small (and often multi-disciplined) teams of people instead of large uni-functional departments are also significant developments that have helped to form the environment in which systems are now used.

3. The theoretical work on the nature of systems pioneered by von Bertalanffy has emphasised the need for a fundamentally different approach called the 'systems approach'.

DEVELOPMENTS IN TECHNOLOGY

In chapter 2, we indicated that there is a natural progression from a closed system view to an open system view. The rate of this progression depends partly on changes in attitudes towards systems and partly on the speed at which the technology itself is changing.

Information processing technology continues to change at a rapid pace, and already, during the short history of data processing, there have been significant changes in computing practice. For example, many large organisations have made some or all of the following changes:

— They have progressed from a batch processing environment to an interactive processing environment.

— They have changed from using one large centralised computing facility to using several smaller (and dispersed) computers.

— They are now actively managing the data that is stored in their databases, instead of, as previously, merely storing data.

As a result of these changes, new opportunities have appeared for using computers to support the organisation. Thus, as well as being used as tools at the operational-support level, computers are now being used at the decision-support level and for information management. Richard Nolan identified the major milestones of this progress in his paper "Managing the crisis in data processing". (A description of Nolan's work can be found on pages 6 and 7 of Foundation Report No. 11.)

These changes in technology have changed the attitudes both of users and developers of information processing systems. As the technology has penetrated into more and more areas of the organisation, the users have recognised their obligations as 'owners' both of systems and data, and they have demanded better, more flexible and 'friendlier' systems. Consequently, system developers have been forced to take greater account of the requirements of the users of data processing systems, and the importance of the interface between people and equipment has been increasingly recognised.

To illustrate the way in which the development of a technology changes the attitudes of those who use that technology, we now present two views about the effect that technology has on those who develop systems.

The first view was presented by Gordon Scarrott in his Clifford Patterson Lecture to the Royal Society in 1979. He showed that in the early stages of using a technology the design emphasis is on how to make the technology work and on how to use it. During these stages, system design is predominantly exploratory. As the understanding both of users and system developers of the way in which to use the basic technology matures, there is a greater design emphasis on why the technology should be used and what it should be used for. In these later stages, system design becomes more analytical, and the technology is used more selectively. Also, the users of the technology become more involved in the system design process as their understanding of the technology matures.

The second view was presented by B. Hedberg in a paper entitled "The Human Side of Information Processing", which he presented at the Computer Impact Conference in 1978. In the paper, he identified three phases in the development of using technology, and he related each of these phases to the way in which system designers view their task.

In the first phase, the system designer is enchanted by the opportunity to work with a new technology, and he often refers to his task as "working at the frontiers of knowledge". All his effort is taken up with making the technology work, and he will consider only those problems that are associated with the technical design of the system. If the resultant technological system alters the organisational context into which it is introduced, then this will take the designer by surprise, because he will not have

envisaged that such a thing could happen. The design philosophy during the first phase can be summarised as 'exploratory'.

In the second phase, the designer, having by now gained some experience of using the technology, is aware that it can have organisational consequences that the users may not welcome. Consequently, he is now careful to design the system in such a way as to minimise the social implications. Ideally, the remote users of the system will hardly notice that they are working with a computer, and, if organisational changes do occur as a result of the system, they will not be intended by the designer. The designer, in this phase, works like a tailor, trimming his system to fit the situation into which it is being introduced. The design philosophy during the second phase can be summarised as 'defensive'.

In the third phase, the system designer has confidence in his ability to use technology in a sophisticated way. He perceives himself as an agent of change, and he is aware of the complexity and the multi-disciplinary nature of the design task. He knows that when he changes the technology he also changes the organisational environment and alters the working lives of many people. He deliberately designs systems to achieve a result that is good not only in organisational and human terms but also in technological (and also in technical) terms. The design philosophy during this third stage can be summarised as 'strategic'.

From her work on participative design at the Manchester Business School (which we discuss in more detail on page 27), Enid Mumford believes that there is a fourth phase following the three phases that Hedberg identified. In this fourth phase, the specialist expert designer disappears, and he is replaced by users of the technology who design systems in a participative manner. In this phase, the systems that are created take full account both of technical factors and human factors.

From what we have said above it is clear that as people's understanding of the way in which to use technology matures, their perception both of systems and of the nature of system development changes. This in turn means that the approaches system designers take, and the methods they use to develop systems need to be constantly reviewed. Approaches and methods that were adequate at an earlier stage of understanding may no longer be so, and they may in fact be hindering the development of systems that now meet the users' expectations.

Technology also influences the development of systems in another way, by making use of the technology itself in the development process. Much effort is currently being made to support the system development process with automation, and this is one more reason why the approaches and the methods for system development need to be constantly reviewed.

CHAPTER 4

A CLASSIFICATION OF SYSTEM DEVELOPMENT APPROACHES

During the research for this report we came across many possible approaches and methods for developing systems. Many of them were variations of individual general themes, and there would be only limited value in classifying all the approaches comprehensively. We have chosen instead to classify system development approaches under the three broad headings of technical approaches, planning approaches and open system approaches. In many ways this classification reflects the ways in which the overall evolution of system development approaches has been influenced both by the shift from a closed system view to an open system view (which we discussed in chapter 2), and by the factors we identified in chapter 3.

TECHNICAL APPROACHES TO SYSTEM DEVELOPMENT

The first approaches to system development (usually referred to as 'traditional approaches') were originated by technologists who viewed systems primarily as closed systems. These early technical approaches were not based on any systematic research into the nature of system development. Instead, they grew out of the early 'trial and error' experiences of developing computer systems. As a result, these traditional approaches concentrated on the technological parts of the systems, and to a large extent they took little account either of the users of the systems, or of the wider environment in which the technological systems operated, or of the organisation's overall objectives. With these approaches, system development was carried out at a low level, and it was performed in isolation both from the development of the overall work system and from the users of the system.

A truly empirical set of practices evolved, and these were documented as standards. These early approaches to system development are best described as regulatory, advisory and deterministic. They provided system developers with checklists, rather than methods. Although they did involve the users of the systems in agreeing the technical specification, they did not help them to understand either their systems or the development process. In addition, these early approaches maintained a rigid view of the sequence in which the development activities had to be performed, and therefore they did not allow for the developing system to be enhanced by successive iterations of the activities.

Over the years, the traditional approaches have evolved to a stage where today there are diverse and separate approaches and methodologies each of which concentrates on different technical aspects of the system. These approaches and methodologies are usually called analytical approaches because they all have a common analytic characteristic. They all break down the steps of development into a process of repeated analysis and deduction.

Analytical approaches were developed to overcome three particular problems inherent in the traditional approaches. The first problem was the one created by the rigid view of the development process mentioned above, and this was overcome to some extent by improving the mechanism used to control the system development process. In

particular, staged project management methods were introduced, and these methods now permit limited iterations to be performed within the development process.

The second regular and recurring problem with the traditional methods was the problem of setting objectives and specifying requirements for the system. It became clear that development approaches needed to place a greater emphasis on these activities, and that they also needed to allow more time for the activities to be performed. As a result, today's analytical approaches place a greater emphasis on the early stages of system development, and separate methods have been devised for specifying the requirements of systems.

The third problem was highlighted when new concepts and theories (such as structuring, top-down development, entity modelling, data management, etc.) were formulated about the nature of technological systems and their development. As a consequence, it was recognised that a technological system requires a level of design that is as independent as possible from the limitations of the physical environment in which it will be implemented. Thus, logic design developed as a separate set of activities, and the key aspect of many analytical approaches and methodologies is the method by which the logic design is performed.

Most analytical approaches have, however, inherited from the traditional approaches the assumption that systems are developed to be run on specific hardware. For this reason, the point during the development process at which hardware should be selected (and the steps that should be performed to select the most appropriate hardware) either are omitted altogether from these approaches or are included at the very early stages of the development process.

Analytical approaches rely on deductive development, and this, of necessity, generates ever-increasing amounts of paper during the development process. In trying to reduce the amount of paper generated during the development process, most analytical approaches use one of the various pictorial or diagrammatic documentation methods that have been developed.

The most notable analytical approaches are the structured approaches, the database approaches, and the software engineering approaches. In addition, a separate, but partial analytical approach, the requirement analysis approach, has emerged. We now discuss each of these four analytical approaches.

1. The structured approaches.
   The two major types of structured approach are the data-oriented approaches and the functional approaches. Both types of approaches regard structure as the key element of systems, and both were influenced heavily by the theoretical work of Dijkstra, Parnas, Boehm, Jacopini, Mills and others (see bibliography).

   Structured approaches originally addressed the programming area of system development, and they gave rise to the discipline of structured programming.

   The data-oriented approaches derive the system structure from the data structure, and the two best-known structured programming approaches are Michael Jackson's JSP and Jean-Dominique Warnier's Logical Construction of Programs (LCP). In contrast, the functional approaches derive the system structure from the internal functions of the system, and they were developed as the result of work done by several people, the most notable being Myers, Constantine and Yourdon.

   Once the two types of structured approach had addressed the programming area,

both types were then developed to address the problems of other areas of system development. The basic principles of structuring were identified, and attempts were made to integrate these principles into the design and the analysis areas of system development. Various structured disciplines were formalised and packaged with the aim of creating a complete structured methodology for system development.

One of the first attempts to create a structured methodology was IBM's Improved Programming Techniques (IPT). This methodology brought together means and methods of structured programming, structured walk-throughs, team organisation, and a documentation method, HIPO (which was described in Foundation Report No. 11).

Subsequent generations of structured methodologies (both data-oriented and functional) aim to support the total system development cycle. For example, Jackson is currently developing a structured system design methodology, and K. Orr has extended Warnier's original ideas so that they can be used for systems analysis. (The resulting Warnier-Orr methodology is now widely used in the United States.) Warnier himself has also developed structured methods for defining the high-level objectives of systems.

The functional structured approaches that were originally used for structured programming have also been extended so that they can now be used for other areas of system development. For example, Ed Yourdon, working with Larry Constantine, has established structured design as a separate discipline, and Yourdon's further work with Tom de Marco, Chris Gane and Trish Sarson has established the discipline of structured analysis. Several formal structured methodologies have resulted from this work, and the four most notable are:

- The Yourdon and de Marco approach.
- Gane and Sarson's Stradis methodology.
- Metasystem 1000, which was originally developed by Bank of America.
- SADT (structured analysis and design technique), which was developed under the direction of D. T. Ross by SofTech Inc.

Structured approaches are now a well-established technique for developing systems, and they are widely used. The principles of structuring have been encapsulated in logic description languages such as pseudo code, program design language (PDL) and structured English. They have also been used to construct new programming languages, notably Pascal.

2. The database approach
The database approach has its origins in the introduction of management information systems that were based on large databases. It concentrates on the organisation of global, company-wide data, and data analysis techniques are central to the database approach. A well-known methodology that is based on the database approach is the one developed by CACI Inc. under the direction of Ian Palmer. The CACI methodology is used to develop systems in a shared data environment, and it is based on the following six principles:

- A clear distinction between application-oriented tasks and data-oriented tasks.
- Separation of design from analysis to provide flexibility both for business changes and technological changes.

— Orientation towards a development strategy rather than towards ad hoc problem solving.

— Decomposition into small, well-defined controllable tasks.

— Emphasis on simple, standard and diagrammatical documentation, rather than on narrative.

— Interactive use of a data dictionary system for all documentation.

The CACI methodology views system development as a single large undertaking, and this view is based on an unchanging global model of the company. The documentation method is comprehensive, but its full benefits can be achieved only by using a flexible and efficient data dictionary system. Of necessity, such a system needs to be an automated system, and although most of the current data dictionary products provide some of the facilities that the CACI methodology requires, they do not provide nearly enough of them.

3. Software engineering
Software engineering is a highly-disciplined approach to developing software, and it regards the production of software as an engineering problem. It uses rigorous technical methods that are enforced by rigorous management methods, and it is practised mainly by computer manufacturers and suppliers of software, and also by those engaged in the development of military systems. The problems that such software developers have in terms of the size, and the quality and the maintainability of their software, and in terms also of their user base are, of course, different from those that a company's system development department faces. Even so, the general methods of software engineering can often be applied to the development of in-house application systems. Many of the principles and practices of software engineering are, in fact, used in many of the structured approaches.

The methods of software engineering are most useful both for constructing large programs and systems and for managing the large group of people that are often required for constructing such software. In particular, software engineering provides a means of handling the complexity created by sub-dividing large software development projects into a large number of smaller units. Each unit may, in itself, be self-contained and easy to manage, but the complexity arises from the need to control the interface of the individual units with one another.

Quality control of the software product is an integral part of software engineering approaches. The most formalised quality control method is the method of inspection developed by M. E. Fagan of IBM. The inspection method was described in detail by Tom Gilb at the seventh Foundation management conference held in Venice in May 1980, and full details of this presentation can be found in the conference transcript.

IBM's software engineering programme (developed under the direction of Harlan Mills) is an example of the management approach to software engineering. This approach was developed by, and is used by, IBM's Federal Systems Division. There are two key elements to the IBM approach — the management of the software production process itself, and the development of a new kind of work called integration engineering.

The management technique is based on an unprecedented degree of formalisation and fragmentation of the development process, and it is reinforced by a rigid mathematically-based management discipline. The main software production tool is

a factory-like structure (called the 'uniform programming environment') that aims to make the software producer's work more predictable, more controllable and more uniform. This tool is used by the managers who control the production of software. The software producers are part of the tool, not users of it.

IBM use the term 'integration engineering' to define the incremental development and integration of large software products. In essence, integration engineering is a unifying methodology for a top-down, large-scale system development process. It is concerned both with the interfaces between the parts of a complex system and with the quality of the resulting system. Those who practise integration engineering (integration engineers) are concerned also with testing the parts of the system.

IBM claims that its software engineering programme has been successful during the past five or six years, and has contributed to high productivity in the development of large software projects. This claim cannot easily be verified because most of the Federal Systems Division's projects are large defence projects, and so are not in the public domain. The United States government's General Accounting Office has, however, been very complimentary about one particular system that was developed by using the software engineering programme. On the other hand, a survey that IBM carried out of its staff showed that a relatively high percentage of programmers had doubts about the success of this approach.

Another example of an engineering approach to producing and supporting software is the approach that the French software house SOPRA has developed. The SOPRA approach can best be described as industrialised software engineering, and it is based on the re-employment of technical know-how within an overall framework (or philosophy), supported by a selection of methods and tools. SOPRA uses its methodology to develop bespoke application systems and adaptable application packages. It is not available as a commercial product.

4. Requirement analysis (or requirement engineering)
Methods developed for analysing and specifying the requirements of a system fall into two categories:

   — The requirement engineering methods that aim for precision through analysis. These methods are discussed in this section of the report.

   — The methods that allow for initial imprecision, and advocate several iterations to accumulate knowledge about, and understanding of, the requirements. These empirical methods are discussed later in this chapter under the headings of prototyping and system evolution.

Requirement analysis is not a complete system development approach, because the specified requirements form the input to a system development process. Requirement analysis methods were developed as a result of user dissatisfaction with those early computer systems that were developed using traditional development approaches. Methods for determining requirements began to emerge when it was realised that systems cannot be created by simply translating the existing processes to run on a computer system. The most notable research in this area was carried out by Boehm of TRW, and by Professor Teichrow at the University of Michigan, who directs the ISDOS project. (The work of both of these researchers was reviewed in Foundation Report No. 11.) The ISDOS products (PSL and PSA — problem statement language and problem statement analyser) are now used in new requirement specification methods such as Core, which Systems Designers Limited have developed in the United Kingdom.

Core stands for controlled requirement specification, and it was originally defined by Geoff Mullery of Systems Designers Limited in 1978. The Warton division of British Aerospace adopted the method in 1979 and, with their technical co-operation, it has been fully developed and is now available as a commercial product.

The key concept of Core is that it takes account of the different viewpoints that different people have about the same system. Core enables a hierarchy of these different viewpoints to be established at the same time as the initial information about a system is being gathered. The different viewpoints are used to establish a structure both for the information about the system and for the specification of the system requirements. For each level in the structure, the method provides viewpoint structures, action structures and data structures. A simple diagrammatic form of documentation is used to express and to bring together the different viewpoints of users, customers and management in a way that technical people can readily understand.

PLANNING APPROACHES TO SYSTEM DEVELOPMENT

The traditional approaches to system development, and the analytical approaches that evolved from them, all concentrated on the development of the technological system. However, right from the earliest use of computers there were those who took a more open view of the nature of systems, and as a result, a second family of development approaches grew up. Instead of concentrating on the development of the technological system, these approaches concentrated on the selection and the use of computers.

These approaches can be classified as planning approaches, and the early planning approaches were characterised by a preoccupation with hardware, a disregard of the users and a low emphasis on technical development. The last step in many planning approaches is the implementation of the system, implementation in this context being equivalent to development in the technical approaches.

Later, when database management systems became available, the planning approaches focused on management information systems. As a consequence, the concept of data as a corporate resource was developed, and the planning approaches evolved into strategic planning tools for technologically-based information systems.

Early advocates of the planning approach were Jerome Kanter and Henry C. Lucas. They emphasised the need to define the business objectives, to develop strategies, to select the business area for the system and to derive the business system's objectives. However, their methodologies did not particularly help with the development of the technological system once these actions had been taken. Later, Paul Siegel and James Martin, in their various books, addressed lucidly the strategic issues of management information systems.

The current planning approaches concentrate on the strategic planning of information systems. The three best-known planning approaches are:

1. Business Systems Planning
   IBM's Business Systems Planning (BSP) concentrates on the use of data as a corporate resource, and it advocates that data must be managed from an overall organisational viewpoint. Through the use of BSP, an information system is developed by identifying the basic data of various business processes, and this data is used to create a stable information framework on which different applications can be developed. BSP recommends a series of fourteen steps that can be used to develop:

— An information resource architecture.

— An information and data management framework.

— An action plan for system development.

The BSP approach has been developed specifically for use with large central databases, although many elements of the approach could be applied to the process of planning for information systems.

2. Critical success factors
The critical success factors (CSF) approach was developed by John Rockart at MIT's Center for Information Systems Research. The approach concentrates on the information needs both of functional management and top management, and it relies on the existing information systems (formal, informal, manual and automated) to provide the inputs to the approach. These inputs come primarily from four sources:

— The industry that the company is in.

— The company itself.

— The business environment (customer trends, economic factors, political factors, etc.).

— The existing organisational factors in the company.

Information needs are analysed with special emphasis on the following two factors:

— The 'soft' information the company requires, including verbal information, information of all kinds from outside the company, comparative data, etc.

— The information needs of an individual manager, rather than the other needs of either his position or his organisational function.

The critical success factors are determined from the objectives and goals either of the company or of an individual. The related information needs identify the need for management information systems or decision support systems.

3. PRISM (People/Resources/Information System Management)
The PRISM planning system was developed by Ivan George at Deltacom Inc. It emphasises that information system development is not a process that can be isolated from either the purpose, or the function, or the structure of the organisation. With PRISM, the total planning process starts at the top of the organisation, and strategic management areas are defined. For each area, cost-effective management functions (and their associated organisation structure and information systems) are planned, specified and finally implemented. The management systems and the information systems therefore evolve together within a framework of cost-effective resource usage. The concept that underpins the PRISM approach is that an information system cannot be effective without an effective management system. The two systems must therefore be planned and developed together.

The ISAC methodology, which we discuss on pages 30 to 33 as an example of a systems approach to developing systems, can also be considered as a planning approach, because it embraces the planning elements of system development.

Because many planning approaches do not address the detailed problems of technical development, it may be difficult to translate into low-level technical objectives the high-level objectives they identify. Tom Gilb developed the Design by Objectives methodology with the aim of overcoming this difficulty. The purpose of this method is to structure the objectives and then to decompose them into a network of lower-level objectives that can be expressed in terms of system attributes, system functions and system performance. (A description of the Design by Objectives methodology can be found on pages 52 to 56 of the transcript of the seventh Foundation management conference held in Venice in May 1980.)

OPEN SYSTEM APPROACHES TO SYSTEM DEVELOPMENT

The technical approaches and the planning approaches all assume a deterministic, well-regulated linear process of development. This requirement is not unexpected, because all of these approaches have their roots in the early days of computing, when it was regarded as a modern branch of the exact sciences. Thus, the rules of formal logic and mathematics were originally thought to be adequate for designing all computer systems. But a linear view of development combined with a deterministic approach takes no account of the basic nature of people. People act in an unstructured manner, and they develop and learn by trial and error. As a result, requirements in the real world are not always logical, and they tend to change. Also, real organisations are complex and unstructured. In the real world it is not possible to develop the optimum system in a single iteration, and many of the early approaches allowed only for a single iteration.

The data processing community is slowly beginning to realise that system development needs to be an iterative process, with the system designers and the system users holding a continuous dialogue that successively refines their joint understanding of the system. Indeed, this iterative process needs to continue throughout the life cycle of the system. The approaches we have discussed so far in this section of the report do not make it easy to carry out this dialogue. This is because they concentrate on planning problems, organisational problems and technical problems, and they exclude the user almost entirely from the single pass through the development process.

However, work systems that employ technology as one of their elements must be developed by methods that accommodate both the technological parts and the non-technological parts of the overall system within a methodology that can recognise all parts with equal relevance. Such a methodology should also recognise the iterative nature of system development. The realisation by researchers that none of the types of approaches we have discussed so far could be used easily in this way led them to develop new approaches.

These new approaches have been developed with a much more open view of systems than earlier approaches had. Their aim is to remove the boundary that has existed between the user and the development of the technological system, by involving the user completely in an iterative development process. We classify the four categories of new system development approaches that we have identified under the headings of user participation, system evolution, approaches based on new views of the life cycle of systems, and a 'systems approach' to developing systems. We now discuss each of these categories in turn.

User participation

For some time now, the trend has been to involve the user more than previously in the development of his systems. However, involvement implies a secondary role in the

development process, and it also implies that the users do not take the leading role. In the past, the data processing manager filled this role, and because he was a specialist he could not be expected to have more than a limited knowledge of the users' business needs or plans, and of the way those needs and plans related to the wider goals of the organisation.

It follows that it is not enough merely to involve users in the development process. If they are to be really effective in the development process, users, and especially user executives, should lead and participate, and data processing specialists should be involved only as and when they are required. User executives should plan their own systems, and users should both manage their own development projects and take an active part in designing and developing their own systems.

Three recent developments that have made it possible for users to participate in the development process are personal computing, the prototyping method and the participative approach. Personal computing and the prototyping method allow users to participate in the development of the technological system. The participative approach allows the user to participate in the more important areas of planning and designing the environment in which the technological system will operate. We now discuss each of these developments in turn.

1. Personal computing
   Personal computing requires high-level facilities such as simple and easy-to-learn languages, data management facilities, on-line access to databases, and simple screen formatting and report formatting facilities. The user can use these facilities to develop his own systems, with little or no involvement from data processing specialists. Although this development approach is not appropriate for all types of systems, it is suitable for fairly simple applications and stand-alone applications. The major advantage of personal computing is that it provides the user with direct access to computing facilities, and it also provides him with both an insight into, and an understanding of, the intricacies of the method of developing systems. In particular, it enables the user himself to clear the backlog of small, low-priority applications that the data processing department cannot handle because of its lack of resources.

   In general, there are three ways in which a company can provide personal computing facilities for users:

   — By purchasing a small computer that users can use.

   — By allowing users to use a timesharing system.

   — By providing facilities that allow users to use the company's own database.

   When the user uses the company's own database, he is provided with comprehensive query and formatting facilities, and he can be trained to manipulate the data with languages such as APL and Basic.

   Personal computing facilities can also be provided both on mainframe computers and on minicomputers. For example, both IBM and ICL offer personal computing systems on their mainframe computers, and Information Builders Inc. markets the Focus system, which provides a data management facility on IBM machines using IMS files. In the United States, Complete Computer Systems Inc. market a system, called Create, which provides personal computing facilities on a variety of smaller machines.

2. The prototyping method

After the basic hardware and software facilities have been installed, prototyping is a quick and inexpensive method of developing skeletal experimental systems. These facilities may include either a database or a data management system, a high-level (and preferably non-procedural) language, a screen formatter, a report generator, generalised input/output software, and the ability to print out parts of the prototype. The most important prototyping tool is, however, a high-level language that is easy and quick to use for developing and modifying programs.

The prototyping method can be used at several stages of the system development process. Thus it can be used for:

— Specifying requirements.

— Fixing the user interface of a system before system design commences.

— Testing the likely future effect of a system before the system is fully developed.

Prototyping as a method has two key characteristics that are different from conventional development methods:

— A prototype is built quickly, without either a lengthy initial investigation or the documentation of requirements.

— A prototype is exercised by the eventual users of the future system. The aim of the prototype is to make a practical check of the original views about the system, and to modify the prototype if those views prove to have been either incomplete or wrong.

Once a prototype system has been built, it can then be tidied up, documented and used as the final system. Alternatively, the results of the prototype can be transcribed into a conventional specification so that the system can be developed by some other method. In general terms, a prototype should be retained until its original purpose is fulfilled. In practice, this means that a clearly-defined purpose should be attached to prototyping. In deciding whether or not to retain a prototype, it is necessary to understand that a prototype does not include all the aspects of a full system. For example, it does not include:

— Resilience.

— Logical clarity and completeness of design.

— Operational efficiency with large amounts of data or a large number of users.

— Recovery and other technical operational features.

— Optimal technical design.

— Full documentation either of the features that exist in the prototype, or of the requirements it fulfils, or of its operational characteristics.

A prototyping approach is effective only when it is performed with the user's full co-operation. For example, the user needs to agree the system requirements and the user interface, and he needs to assess the likely future effect of a system.

Prototyping provides the means of creating a genuine dialogue between users and system developers. The main advantage of the prototyping method is that it enables the user to exert a vital influence on the system at a stage when a change of direction does not cost a large amount of money or effort.

However, if prototyping is to be effective, both users and system developers do need to have the correct attitude towards it. Prototyping implies less formality than is involved with traditional and analytical approaches, and some organisations and individuals may find it difficult (or even impossible) to operate in such an informal and relaxed mode. In particular, the users need to understand that:

— The flexibility provided by the prototyping method brings with it a responsibility for achieving the desired results, and the user has as much responsibility for the design as the system designer has.

— The prototype is merely a tool to prove a point, and it is not the final system.

3. The participative approach
The participative approach to system analysis and design has been developed by Enid Mumford at the Manchester Business School, and it is based on earlier ideas and practices that were formalised by the Tavistock Institute under the name of Socio-Technical Design (which we briefly discussed on page 13). The basic socio-technical theory says that the successful introduction of new technology is possible only when human and social needs for the work system are recognised in parallel with the technical and the economic factors. Mumford and her colleagues have adapted this basic theory and modified it to suit the development of computer systems.

The rationale of the participative approach is based on four arguments:

— The morality argument, which says that people have a moral right to control their destiny.

— The expediency argument, which says that activities should be (and ultimately are) controlled by those who perform them.

— The location-of-knowledge argument, which says that the people who actually do the jobs are the experts on operational factors (such as the design of tasks).

— The motivation argument, which says that involvement both acts as a motivator and leads to increased productivity and efficiency.

In practice, the participative approach involves the users of the technological system in various levels of analysis and design. The original approach advocated three levels of involvement — consultative, representative and consensus. Mumford found that the approach works best when the users are fully involved in the analysis and are allowed to redesign their own work system as part of the development. She and her colleagues have developed a range of tools for analysis and design to aid the system analysis and design processes. Both the approach and the tools have now been used successfully in several fairly large companies.

System evolution

For many years, researchers have recognised that many systems would benefit if they

evolved over a period of time. This evolution would take place by successive developments of the system to provide an ongoing series of enhancements. Until recently, such an evolutionary approach, which relies on performing large-scale iterative development, was prohibitively expensive.

However, recent developments both in technology and in the application of computers have made the evolutionary approach to system development economically feasible. In particular, the two major recent developments are:

— Advanced technical facilities such as on-line systems, databases, high-level languages, screen formatting aids, report generators, etc.

— The prototyping method.

We have already discussed the prototyping method under the heading of user participation. But prototyping can also be used to develop systems through a series of evolving prototypes. An example of this approach is the Nomad facility that National CSS provides on its time-sharing facility. National CSS calls the development process 'protocycling'. Nomad is a high-level non-procedural language that is used to interrogate, to change and to manipulate a relational database.

Another approach to system evolution is to use high-level technical facilities to develop a fairly sophisticated base model of a given system, and then to modify that system regularly during the first part of its operational life-cycle. Easily-modifiable base-model systems can be used for those application areas in which the requirements never become static, or the final requirements evolve only during the early operational life of the system. This type of evolutionary approach can be used successfully for the development of decision-support systems. (Professor Persson from the Stockholm Business School spoke about the practical application of this type of development approach at the ninth Foundation management conference held in Bournemouth in July 1981. Full details of his presentation may be found in the conference transcript.)

The ideal evolutionary development method would create a base system that could develop a 'system consciousness'. Such a system would sense changes through its own experience when used operationally, and as a result it would modify its own rules and 'learn' a new behavioural pattern.

Existing approaches are still a long way from creating systems that can 'evolve' with change, although Ronald Stamper's Legol project at the London School of Economics is promising in this respect. Legol aims to describe systems in terms of formal and informal rules and their associated data structures, and the Legol language provides a sophisticated tool for developing those systems. The development of the base 'rule system' itself is evolutionary, because it is achieved through several hundreds of successively-refined models. But because the final system operates by analysing the rules that describe the system, it is not inconceivable that Legol may provide a way for the system to recognise automatically changes and modify its own operational rules.

New views of the life cycle of systems

The traditional view of the life cycle of systems is that they progress through a regular sequence of stages (feasibility study, analysis, design, implementation, and operation/maintenance). This traditional view assumes that a single pass through the development stages produces a lasting and usable system. It also assumes that the maintenance stage is separated completely from the development stages. This linear view of the life cycle of systems is now being criticised from many quarters, and new

views and concepts have recently begun to appear. These new views and concepts imply a markedly different non-linear nature of the life cycle.

The basic drawback of the linear life cycle is that it does not recognise that systems exist in an environment of constant change. Even if external changes and organisational changes are disregarded (and both of these can be considerable), the changes that a system itself induces cannot be ignored. Professor M. Lehman at the Imperial College of Science and Technology in London explains the meaning and the effect of this type of change in a simple way. He says:

> "Any computer program is, in part, a model of the environment which it is intended to serve. As soon as it becomes operational, it becomes part of that environment and thus interacts with it. Hence the model that the designer used to develop the program and which forms an intrinsic part of its structure and content is at least partly invalidated. The very installation of the program initiates and accelerates its own decline into obsolescence. Thus, an intrinsic property of software is that it continually evolves and adapts to a changing environment."

A linear view of the life cycle of systems suggests that the requirements of a system can be defined completely and finally by a single pass through the requirement specification activity. However, it is now recognised that there are many situations in which this is not possible.

We have already mentioned that the prototyping method can be used as an aid to defining requirements, but Professor C. Floyd at the Technical University of Berlin believes that there is a much more fundamental problem with defining requirements.

According to Floyd, most specifications deal only with the functional requirements of systems. But, she says, every system has performance characteristics, and some performance requirements (at least some rudimentary ones) ought also to be stated before the system is developed. Although it is possible to state what these performance requirements are, there are no known techniques for interrelating and transfusing them with the functional requirements.

In addition, the requirements should also describe the manner in which the system is to be embedded into the activities of all the groups of people who are affected by it. Floyd calls these the handling requirements, and it is difficult to state precisely what these are. A system (which is normally a well-structured entity) has to perform in an essentially non-structured environment, and this makes it difficult to specify the requirements in a single pass. This lack of structure in the environment can be caused either by individual users using the system in a non-structured way, or by several different users using the system in several different non-structured ways.

Professor Floyd believes, therefore, that the properties that should form the requirement specification are the functional requirements, the performance requirements and the handling requirements. The first two types of requirement are not incompatible with a linear view of the life cycle of a system, but whether or not the handling requirements are being satisfied can be proved only by the use of the system in practice. Thus, the complete specification of the handling requirements may require several iterations through the specification stage and the operation stage.

In other words, Floyd maintains that there is no point in envisaging the life cycle of a system in terms of a major development phase followed by an operational phase that includes 'maintenance' of the system. Instead, she maintains that the life of a system should be envisaged as a series of controlled enhancements that are performed in a

well-planned cycle with a high level of user involvement. The cycle of 'revisions' would occur at planned intervals (for example, every three years), and as few changes as possible would be made in the intervening period. Successive 'system versions' would therefore be developed, and the system developers and the users would both have well-defined tasks to perform in parallel with each other to verify the version of the system created during the current cycle. (These tasks are summarised in figure 1.) Floyd calls this approach to system development the 'process-oriented approach', in contrast to the phase-oriented approach of the traditional and the analytical methods.

The main difference between the process-oriented approach and the phase-oriented approach is a difference in emphasis. With the phase-oriented approach, the predominant concern is with the sequencing of the activities within each phase of development. The sequencing (and the control) of the rest of the life cycle is not addressed. For the remainder of its life, the 'system' is basically the same single entity, which is subjected to minor modifications as and when they become essential. With the process-oriented approach, the design and the implementation phases are mapped onto successive development cycles. During the life cycle of the system there will be several recognisably different versions of the system, and during their operational life these versions will be subjected only to genuinely minor modifications.

A systems approach to developing systems

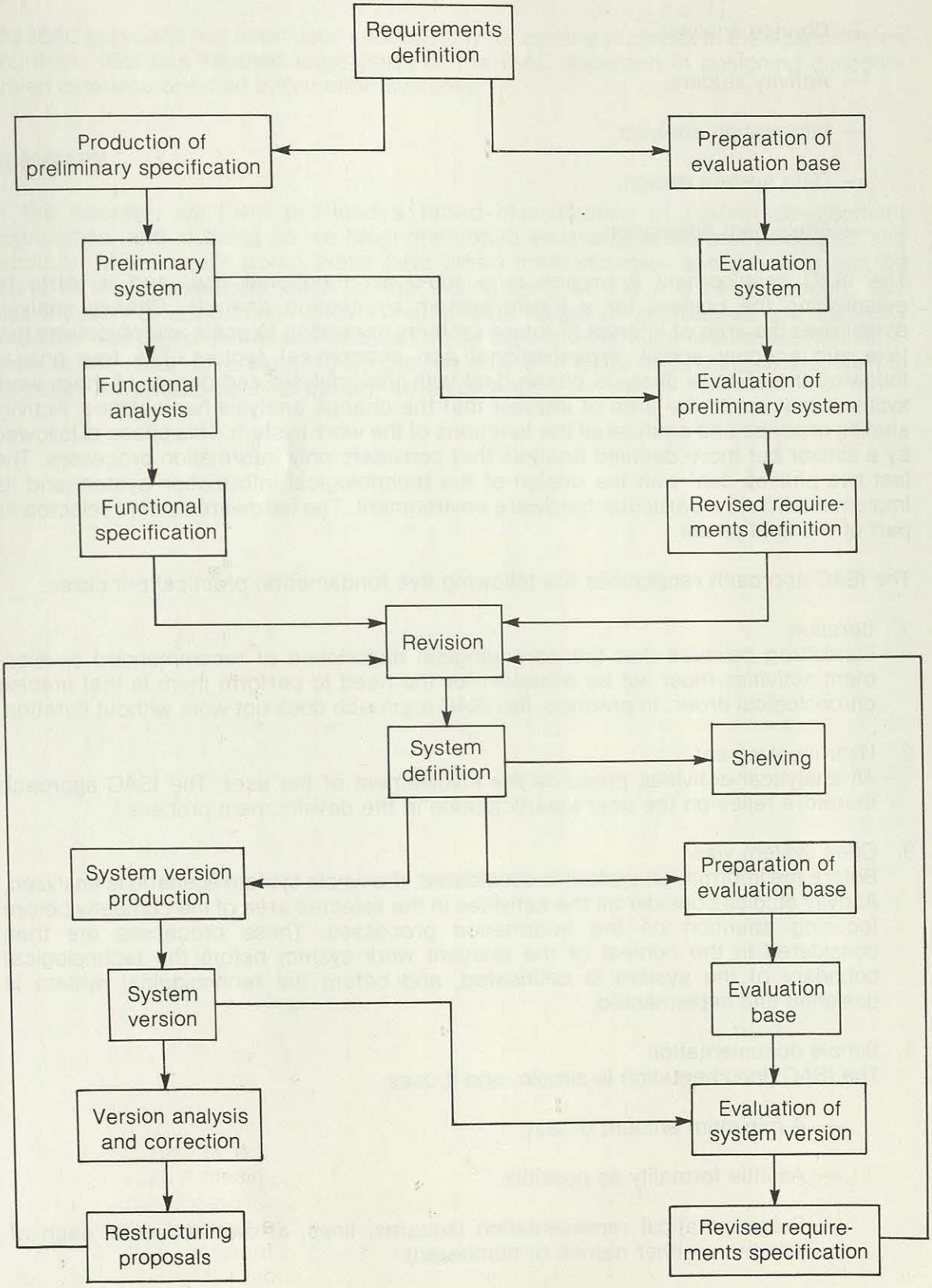
As a result both of von Bertalanffy's work on general systems theory and of the Tavistock Institute's development of their socio-technical systems theory, new approaches to system development have been developed. The main characteristic of these approaches is that they regard systems as human activity work systems. In Foundation Report No. 11 we reviewed the approach developed at Lancaster University under the direction of P. B. Checkland. The Lancaster approach is typical of a systems approach to developing systems.

In the Scandinavian countries, the theoretical understanding of general systems theory has been further advanced by the work of Professor B. Langefors. He identified two different general approaches to systems. The first is the 'infological' approach that follows organisational design concepts for the design of systems, and the second is the 'datalogical' approach, which is the conventional way of finding solutions to organisational problems within the context and the structure of a given organisation. Professor Mats Lundeberg in Stockholm subsequently developed a practical system development method that recognises technical and economic problems, and uses social and organisational criteria as well as technical criteria in system design. This approach is called ISAC (information systems work and analysis of change).

ISAC is one of the few system development methods that start with high-level planning of all aspects of an information system (not just its technological parts) and then take account of the details of technical development as well. An interesting and novel aspect of ISAC is that hardware is not selected until after all the data structures and the computer routines have been designed in the final phase of development.

The principal idea that underlies the ISAC approach is that a new system cannot be envisaged or created without first examining the environment in which it will fit. For information systems the relevant environment is the organisation. The principal technique of the ISAC approach is that of information analysis. This technique is used because its designers believe that information is now the prime source of raw material for organisations. The intrinsic properties of information must therefore be analysed in the context of the organisation that creates and uses it. Without such an analysis, no system (whether it be manual or automated) can be successfully created, because it

Figure 1 The process-oriented approach to system development



would be completely out of context. The ISAC approach recognises five phases of developing an information system and in the terminology of the ISAC approach they are called:

— Change analysis.

— Activity studies.

— Information analysis.

— Data system design.

— Equipment adaptation.

The ISAC development approach is a top-down functional one, and it starts by establishing the context for a future system by change analysis. Change analysis establishes the area of interest of future systems according to goals and objectives that take into account social, organisational and economical factors. The four phases following the change analysis phase deal with the analysis and design of those work systems that are in the area of interest that the change analysis has outlined. Activity studies describe and analyse all the functions of the work system. This phase is followed by a similar but more detailed analysis that considers only information processes. The last two phases deal with the design of the technological information system and its implementation in a particular hardware environment. The hardware itself is selected as part of the final phase.

The ISAC approach recognises the following five fundamental practical principles:

1. Iteration
   Lundeberg believes that the chronological description of recommended development activities must not be mistaken for the need to perform them in that precise chronological order. In practice, the ISAC approach does not work without iteration.

2. User involvement
   All analytical activities presume the involvement of the user. The ISAC approach therefore relies on the user's participation in the development process.

3. Open system view
   Before the information system is considered, the whole system scenario is analysed. Activity studies consider all the activities in the selected area of the company before focusing attention on the information processes. These processes are then considered in the context of the relevant work system before the technological boundary of the system is delineated, and before the technological system is designed and implemented.

4. Simple documentation
   The ISAC documentation is simple, and it uses:

   — A minimum amount of text.

   — As little formality as possible.

   — Diagrammatical representation (squares, lines, arrows and dots, each of which are either named or numbered).

   — Very few symbols.

5. Simplicity
   The same analytical methods and documentation techniques are used throughout the complete development process.

The ISAC approach has been used successfully for several projects. In the Scandinavian countries, IBM has adopted large parts of the ISAC approach in designing dialogue-driven database-oriented information systems.

SUMMARY

In this chapter, we have provided a broad classification of system development approaches, and in doing so we have mentioned several specific methodologies and products. More details about these (and other) methodologies and products can be found in the publications listed in the bibliography at the end of this report. Our classification of approaches shows that a wide range of approaches are currently available, and that different approaches can be used to advantage in different situations. Later in this report (in chapter 6) we provide advice to those organisations that wish to introduce a new approach to system development.

CHAPTER 5

AUTOMATED AIDS FOR SYSTEM DEVELOPMENT

Earlier in this report we indicated that the task of developing systems can be considered as a work process. Any work process can be enhanced and supported both by theoretical knowledge about the basic concepts that underlie the process and by practical aids. In the previous chapter we discussed the system development approaches and methods that have evolved as a result of the growing body of theoretical knowledge there is about the basic nature of systems. These approaches and methods support the system development work process at a conceptual level.

In this chapter we turn our attention to the practical aids that are available to support the system development process. These aids can be provided either for the complete process or for part of it, and they can be manual, mechanised or automated. We concentrate in particular on those aids that provide automated support for the system development process.

Three levels of aid are available. First, there are the individual tools, such as documentation aids, programming languages and compilers. In this chapter, we do not consider aids at this level. Second, there are integrated facilities such as user-oriented languages and high-level languages, both of which can be integrated with databases and data dictionaries. We discuss these integrated facilities in more detail in the next section of this chapter.

The third level of aids comprise what we term 'environmental support'. These comprise the individual tools and the integrated facilities that workers have at their disposal from the environments in which they perform the work. The environment is supportive if the individual tools and facilities are integrated to produce a synergy that both meets the demands of a given job, and enables those who perform the job to do it better. To be supportive, an environment should have the following characteristics:

— It must provide tight integration of the tools and facilities.

— It must be applicable and available to many different users.

— It must create a helpful and constructive work atmosphere.

— It must be flexible, so that it can adapt both to different needs and to different users.

— It must provide sufficient capacity to store and to index information about the diverse uses to which that information will be put.

Environmental support aids will themselves be complex computer systems, and they will normally comprise:

— An on-line facility and high-level languages.

— Some form of data depository (such as a database and its management system).

— Special routines for generating reports, for screen formatting, for assistance and for diagnostic analysis.

— Security features.

— A co-ordinating or supervisory system.

A supportive environment can be constructed either to support the people who perform particular tasks or to support a particular work process. In the first of those environments (a people-oriented supportive environment) the environment is designed to aid a specialist by providing him with the optimum environment in which to perform his work. The specialist will use such an environment in performing tasks with which he is familiar, and the supporting environment will provide automated links between the tools and the facilities he uses for the different tasks. If a complete work process consists of tasks that several different specialists perform, then each specialisation may well have its own distinct supportive environment.

In contrast, an environment that is constructed to support a particular work process (a process-oriented supportive environment) will force those who manage the work process to examine critically each individual task. This examination may show that the sequence of the tasks that were performed before the supportive environment was constructed is no longer relevant. Alternatively, it may show that some tasks are no longer required, or that some tasks that were previously performed by people can be completely automated, leaving no role for people to play.

An effective supportive environment (whether it is people-oriented or process-oriented) does not, however, force people to use a particular method to do their work.

In the field of system development, people-oriented supportive environments have been constructed that provide automated aids to meet the specific needs of programmers, and we discuss these in this chapter under the heading of programming support environments. At the present time, people-oriented supportive environments do not exist for the other specialists (such as designers or analysts) who are involved with other stages of the system development process. However, there are now several process-oriented supportive environments that provide automated aids to support the system development process, and we discuss these later in this chapter under the two headings of systems for building systems, and software development environments commencing on pages 39 and 41 respectively.

INTEGRATED FACILITIES

There are many general facilities that can be used to aid the system development process. These facilities combine several individual system development tools, and, as such, we describe them as integrated facilities. In this section of the report we have singled out the two types of facilities that we believe are particularly important — interactive system development facilities and data dictionaries.

Interactive system development facilities

In the previous chapter we mentioned several possible approaches to system development (such as personal computing and prototyping) that require a combination of co-operating facilities, such as:

— An easy-to-learn (and non-procedural) programming language.

— A database and a data management system, together with the appropriate security and back-up features.

— Screen-formatting and print-formatting facilities.

— A ready-made library both of routine and generalised functions.

Most computer manufacturers offer a software package that contains these combined facilities, and such packages are designed to provide personal computing facilities to users. For example, IBM offers its virtual storage personal computing (VSPC) package, and ICL offers its personal data system (PDS). (Personal computing facilities designed specifically for use by programmers are discussed later in this chapter on pages 38 and 39.) As an interim measure to providing full development facilities to users, several manufacturers provide software products that enable users to interrogate existing databases. (For example, IBM's Generalised Information System and Dataskil's Slave interactive data handler.) In addition, APL and Basic are increasingly being used as user programming languages.

We have already mentioned the Nomad facility of National CSS, which can be used both by users to develop their own systems and by system developers for prototyping. A reasonably new IBM product, which is similar in concept to Nomad, is structured query language/data system (SQL/DS). SQL/DS is a relational database management system that is integrated with a very powerful manipulation language, and its facilities include the dynamic redefinition of data and automatic navigation in the database (namely, searching for what the user wants without the user having to specify the way in which the required information should be retrieved). The language provides facilities for making queries, for defining data, and for loading, updating and controlling the database. SQL/DS has a catalogue of information both about programs and users, such as the parts of the database that different users are authorised to access. The system can be used to develop database applications in an interactive way. It can also be used both for prototyping and (to a limited extent) for personal computing.

Data dictionaries

Most technical methodologies for developing systems recommend that an automated facility be used for storing centrally the information about the different elements of a developing system. This central depository of information often takes the form of a data dictionary system, and ICL's DDS product is an example of such a facility.

DDS is an interactive database for the support and control of application development. It is centred around an IDMS database that holds and interlinks, in the form of models, the following four different types of information:

— Information (in the form of operations and events) about real-world processes, such as business functions.

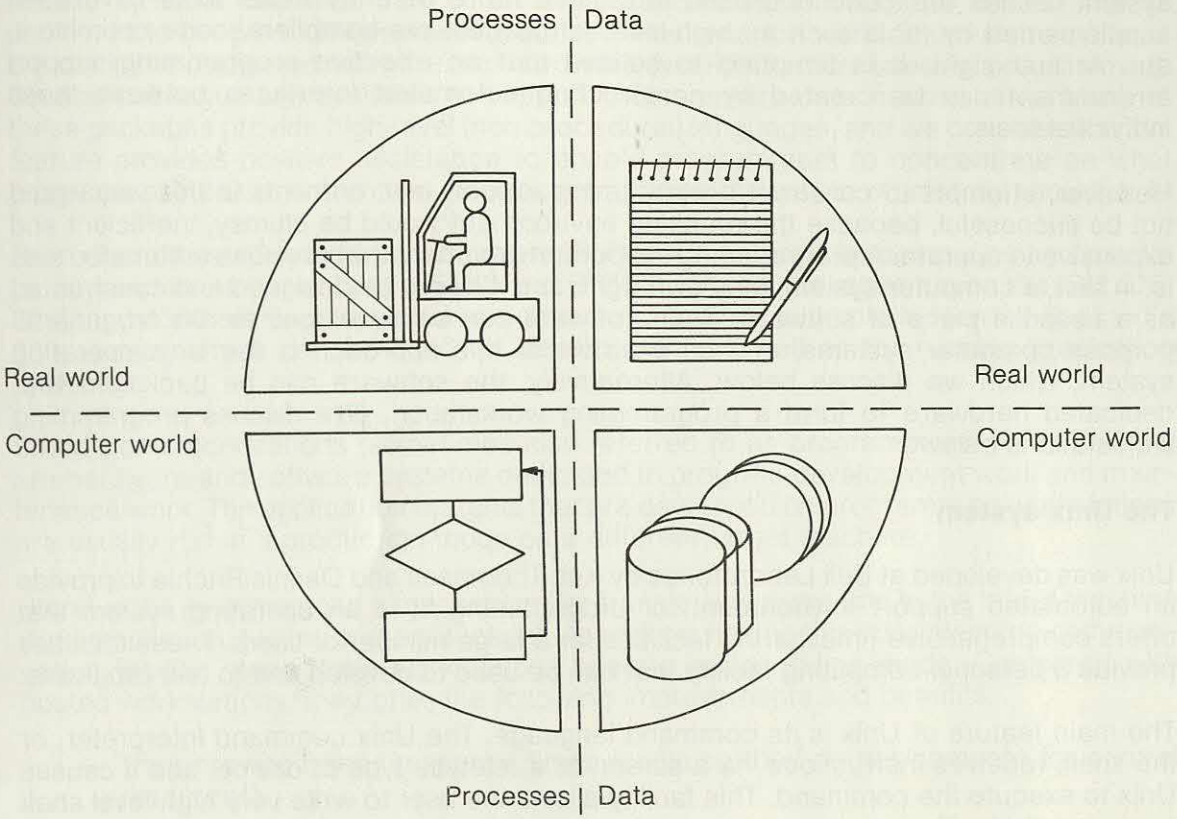
— Information about real-world data (data models).

— Information about computer processes (programs and modules).

— Information about computer data (conventional records, files and IDMS data sets).

ICL represents the DDS database diagrammatically as a circle with four quadrants that connect the real 'world' with the computer 'world', and connect processes with data (see figure 2).

Figure 2 ICL's Data Dictionary System (DDS)



DDS can be used to:

- Maintain records of the parts of a developing system (such as models, designs, data structures, programs, modules, etc.).
- Compile comprehensive documentation about all the computer data.
- Aid in the system design process for data analysis and functional analysis.
- Generate data descriptions for programs.
- Cross-reference all the system elements (both real world and computer).
- Interlink selected parts of the database (for example, linking processes with data, or linking real-world models with computer models).
- Retrieve selected and interlinked items.

ICL is currently enhancing DDS to include facilities similar to those in Cades, which is ICL's internal software engineering system.

PROGRAMMING SUPPORT ENVIRONMENTS

Individual tools for automating parts of the system development process have been available for many years. In the programming area, compilers, editors and operating system utilities are commonly-used tools, and more recently those tools have been supplemented by tools such as high-level languages, pre-compilers, code optimisers, etc. At first sight, it is tempting to believe that an effective programming support environment can be created by constructing automated interfaces between those individual tools.

However, attempts to construct programming support environments in this way would not be successful, because the resulting environment would be clumsy, inefficient and expensive to operate. A programming support environment that provides automated aids is, in fact, a computer system in its own right, and it has to be designed and constructed as a bespoke piece of software. Such software can be developed to run on general-purpose computer systems, and an example of this approach is the Unix operating system, which we discuss below. Alternatively, the software can be packaged with dedicated hardware to form a programming workstation. (We discuss programming workstations below.)

The Unix system

Unix was developed at Bell Laboratories by Ken Thompson and Dennis Ritchie to provide an automated support environment for programming. It is an operating system that offers comprehensive timesharing facilities for a large number of users. These facilities provide a personal computing facility that can be used to develop and to test programs.

The main feature of Unix is its command language. The Unix command interpreter, or the shell, receives instructions via a screen or a teletype type of device, and it causes Unix to execute the command. This facility allows the user to write very high-level shell programs that will execute many operations.

The Unix user can create command superstructures that can:

— Perform the routine steps of a job (standard commands can be catalogued and invoked).

— Develop new application programs from building blocks of routines coded in a conventional way (and which have been catalogued by Unix).

The Unix system provides several co-operating utilities that can be used both to perform simple tasks for the programmer and to provide standard solutions to routine programming problems (such as re-formatting files, sorting, etc.).

The Unix facilities can be used as a rapid way of building high-powered command programs without the need to generate conventional program code. Unix is therefore an ideal tool both for prototyping and for system evolution.

Programming workstations

Programming workstations are dedicated systems that help the programmer do his job. They are designed to replace coding sheets, pencils and templates, and also to replace the tasks of submitting jobs to the computer, searching for test data files on cards or tape, retrieving detailed technical documentation from filing cabinets, etc. They support on-line interactive programming in one or more languages, and they provide facilities for

storing and retrieving data and program modules, and for compiling and testing programs. In other words, they support the programmer's day-to-day job.

Programming workstations can be either hosted systems or stand-alone systems. Hosted systems use that part of an organisation's main computer system that is dedicated to program development work and maintenance work. Stand-alone systems are complete dedicated software and hardware systems that are used for programming. Several packages are available to support programmers on a hosted basis. Many of these packages provide high-level (non-procedural) languages, and we consider that this feature provides positive assistance to enable programmers to concentrate on what programs should do, rather than on concentrating on the way in which they should do it.

Examples of hosted-system packages are IBM's ICCF (interactive computing and control facility) and its earlier versions (ETSS and ETSS II). These packages run on the IBM 370, 3030 and 4300 series of mainframe computers. Another example is Hewlett Packard's 300 system, which can be used for development as well as for running application systems.

Stand-alone workstations (sometimes also referred to as programmers' workbenches) are hardware and software systems dedicated to program development work and maintenance work. The application systems that are developed on programming workstations are usually run in a production mode on a different target machine.

Stand-alone systems need a mechanism for transferring programs to the target machine (for compilation, testing and production). Apart from having this inter-machine communication facility, stand-alone workstations operate in an off-line mode. Compared with hosted workstations, they offer the following improvements and benefits:

— They provide better response times (because they do not compete for central resources).

— They are specifically designed for a given job, and so they are less generalised and better engineered from a human point of view.

— They may be used to develop programs for several different target machines.

— They are more secure and more reliable than a more complex central facility (which has more elements that can go wrong). In addition, the malfunction of the development system could not adversely affect any other hosted systems.

— They tend to be cheaper than an all-embracing central facility.

Examples of stand-alone workstations are the Programmer's Workbench developed at Bell Laboratories (technical name PWB/UNIX), which runs on DEC PDP-11s, and the Pet/Maestro system developed by Softlab GmbH, which runs on the Philips X1150 minicomputer.

SYSTEMS FOR BUILDING SYSTEMS

In Foundation Report No. 11 we reviewed several systems that provide high-level integrated facilities that either support or automate the system development process. In particular, we reviewed the use of PSL/PSA for system analysis, Autogen for automatic code development, and Adam for generating software without programming. In the meantime, several new ideas and products have been developed, and most of these products have the following common facilities:

— Documentation aids.

— An unambiguous language that is used to define the system's design.

— Checking and referencing facilities.

— Mechanisms to generate code automatically from the output of the design process.

— Mechanisms to interlink elements of the design.

We now review briefly four products that fit into this category of systems that either support or automate the system development process.

1. Gamma
   The Gamma system was developed at Software Sciences Limited by David Burns and Dr. M. Falla. It is available as a commercial product, and it is currently at the public trial stage of its development.

   Gamma uses a single database as a library of all the constructs that make up a system. A single high-level language (the Gamma language) is used during the development of a system to enter, store, search, display and edit constructs. The system maintains a hierarchical map of all related constructs, and it provides aids for cross referencing, listing and checking the elements of a developing system. Gamma can include executable code in the library, and it provides code generation facilities for the IBM 370 range of equipment. It also provides links both to Assembler routines and Cobol routines and to a file handling package. Extensive data administration facilities and free-text-editing facilities are included in the system. Future plans include the development of facilities for testing and displaying graphics.

2. Pride
   A high level of automation of the system development process has been achieved by Milt Bryce of M. Bryce & Associates through his system called Pride. Milt Bryce spoke about Pride at the seventh Foundation management conference in May 1980. The transcript of his presentation contains a detailed account of the Pride system.

3. System Builder
   ICL is in the process of developing a new generalised product called System Builder. The concept behind this new development is that all processes within a system are modelled as data structures, and they are envisaged as 'state changes' of the system.

   A high-level (non-procedural) language with no input-output statements is provided, and this language is used to 'build' a system through the use of very high-level generalised functions.

   The system building process consists of defining the structures both for data and processes, with System Builder providing structural checks. System Builder's own compiler provides input-output statements. It also sequences the operations, and optimises the memory requirements and the storage layouts.

4. Elias
   Elias is a system building system provided by IBM, and it is an aid for developing database/data communication applications. It helps to define the database and the screen formats. Elias generates much of the application code of a system by the use

of interactive facilities, by pre-tested 'skeletons' and 'bricks' of code, and by pre-coded database and data communications interfaces. Elias also has the interesting feature of being multilingual, and the help and service functions and dialogues are available in several European languages.

In developing Elias, IBM used the concept of 'Development Management Systems' (DMS). DMS supplies both a framework and 'off-the-shelf' system elements that enable a customised application to be created rapidly.

SOFTWARE DEVELOPMENT ENVIRONMENTS

In this section of the report we discuss two approaches to creating software development environments. The first environment is one in which systems are developed on hardware that is different from the hardware on which the systems will be run. We call this the 'host/target development concept'. The second environment is one in which custom-built systems are constructed from ready-made application segments. We call this the 'customising application systems concept'.

Host/target development

The development of systems that run on a target machine different from the host development machine has attracted much attention amongst commercial and military software suppliers. Foundation Report No. 22 described the methods used to solve this problem for commercial packages. A different set of tools and facilities have been developed for military software, and we mention them here because they are now beginning to emerge as commercial products that have a wider use.

Military systems are mostly on-line real-time systems, and they normally operate on small target machines that are not supported by sophisticated operating system facilities. A special software environment is therefore needed for both their development and their operation. In the United States, the Department of Defense issued directives for the development of such software environments. The result of these directives is the much-publicised ADA language and software support environment.

In the United Kingdom, the Ministry of Defence has successfully standardised the development of real-time military software through the use of Mascot (Modular approaches to software construction, operation and test). Mascot has been used as a standard system since 1975 both by the Ministry of Defence and its sub-contractors. Two of the largest sub-contractors (Software Sciences Limited and Systems Designers Limited) have now made Mascot available as a commercial product.

Mascot is a software development environment that enables on-line real-time systems to be developed on a host machine and then be transferred to a final target machine. The Mascot environment continues to support the system when it is running in the target machine.

A prerequisite for using Mascot is that the system must be designed in a modular fashion. Mascot then generates a network representation of this modular design, and it supports the development of the system by:

— Keeping the developing system elements within the framework of the design.

— Supplying a command language (in the form of primitives) to interlink and to control parts of the system/program.

— Providing a software environment (called the Mascot machine) for the system, both during its development and its operational life. (Mascot actually provides a transfer mechanism to place and to monitor the system in its target environment.)

Outside the field of military systems, Software Sciences Limited has used Mascot successfully to develop distributed microprocessor-based systems (for example, the intelligence for a point-of-sale system).

Systems Designers Limited has developed a Mascot-based product called Context, which is available as a commercial product. Context uses a database both to store the software being developed and to monitor and check it during its operational life. The Context product includes the following features:

— Support for development on a host machine.

— Production of software in the object code of the target machine.

— Transfer of software to the target machine.

— Control and monitoring during the software commissioning stages.

— Maintenance support on the target machine.

— Emulation of the target machine on the host machine.

Customising application systems

In many industries, new products (or 'applications') are constructed from ready-made building blocks. (For example, in the building industry, a builder will not construct his own window frames — he will use standard components.) In the system development industry there are not many approaches that are based on a similar building block concept.

Hewlett Packard has, however, produced 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 application system).

— The facilities (Application Customiser and Application Monitor).

Hewlett Packard has developed the approach for its HP3000 computer, and the approach has been used with Hewlett Packard's Materials Planning and Control System for Manufacturers.

The approach requires a special method to be used 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 tailoring of the system is accomplished by modifying the tables, and 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.

SUMMARY

In this chapter, we have provided a brief insight into the automated support that is available today for the development of systems. Apart from a few isolated products, it is clear that the programming area is the best 'tooled-up' part of the development process. The analysis and the design areas are aided mainly by facilities that allow both documentation and elements of the system to be recorded, and that help in cross-referencing and consistency checking. But there are still very few real analytical and design aids. Computer-aided manufacture of program code is a reality today, but computer-aided design of the system is still in the future.

CHAPTER 6

INTRODUCING NEW WAYS OF DEVELOPING SYSTEMS

In this report we have shown that today there are many different approaches, methods and tools that can be used for developing systems. We have also indicated the types of systems and the types of situations for which particular approaches, or methods, or tools are either suitable or inappropriate. Many organisations have experienced considerable effort in the past few years in standardising on a particular system development approach or method. We believe, however, that if organisations are to develop systems that are truly effective, then they need to use the most appropriate approach for each different system.

In this chapter we provide guidelines that organisations can use to select and to introduce new system development approaches and methods. A critical examination of the available approaches and methods may well reveal that the existing 'standard' approach is not the most appropriate approach. In this circumstance, when a new approach is introduced the existing approach will be replaced. In other words, the organisation's method of creating systems will be changed, and this means that the work system of the system development staff (and possibly of the users as well) will be changed. The introduction of a new system development approach is therefore concerned not only with evaluating and selecting an approach. It is also concerned with managing the change that the new approach will bring to the work systems both of system developers and users.

People are, by and large, conservative. Once they develop a convenient way of doing something, it is not easy to persuade them to change their work habits. The introduction of either a new technique, or a new tool, or of a whole new approach may be resisted simply because it is different, and not because it is wrong or does not make sense. When a new method or a new way of doing a job is introduced into a group of people it unsettles the security of the group, it creates a natural resistance in the group and it reinforces the cultural dependence on the old practice.

Consequently, before we provide specific guidelines for introducing new ways of developing systems, we first explore both the nature of the resistance to changes in system development practices and the way in which that resistance can be overcome.

RESISTANCE TO CHANGES IN SYSTEM DEVELOPMENT PRACTICES

People resist change for four principal reasons:

— People do not take kindly to changes that other people impose on them. (That is, they have a low tolerance to change.)

— People are unwilling to abandon the familiar culture the existing practice provides.

— Some people may misunderstand both the need for the change and the implications of the change.

— People may believe that the change will create a worse situation than the existing one.

A person's willingness to accept a change may well be conditioned by the implications of his accepting that change. For example, many people may, in certain circumstances, resist the introduction of a new method of work, but under different circumstances they may even recommend the introduction of that same method. This interesting dichotomy arises for two reasons:

1. The practitioner of an old method may find that the new method is much better than the old method. He may then wonder why he ever used the old method, and the personal, internal conflict that follows is usually resolved by his rejecting the new method (because this is the cause of the conflict). This natural reaction (known as cognitive dissonance) results from people's inability to view objectively their own practices.

2. People often resist a change in working methods in order to save face. This is most noticeable in those individuals who were instrumental in introducing the old method. If they accept the new method they are openly admitting failure, and (in their own eyes) accepting that their earlier decision or belief was wrong. Thus, the rejection of a new method may be instigated by individuals who do not actually use the method.

We now examine the particular types of resistance that system developers, users and senior managers are likely to have when new system development approaches and methods are being introduced.

Resistance from systems developers

System developers may resist the introduction of new approaches and methods for a variety of reasons, and we list the most common ones below:

1. Many data processing managers, system analysts and programmers have spent several years laboriously acquiring specific technical skills. It is natural that they should want to continue practising what they are good at, and so they will resist the introduction of any new methods that appear to devalue their particular skills.

2. The more conservative members of the data processing community may refuse to believe that the new methods do actually work.

3. An organisation's existing standards manual may advocate that only conventional methods be used. System development staff will therefore be reluctant to even consider any alternative methods.

4. The introduction of a new method into an organisation involves an element of pioneering. Many system developers, like other staff, are not keen to be pioneers, because they know that not all pioneers survive.

5. Unlike the technical approaches to system development, many of the new approaches are not exact, or analytical, or deductive or deterministic. If a system developer is to be able to use technical approaches successfully, he needs to have both analytical skills and deductive skills. The individuals who possess such skills may not comprehend the importance of experimental, or evolutionary or iterative approaches.

6. Data processing managers may resist the introduction of approaches that require

the user to manage the system development process. In addition, many system developers believe that the best way to create organised chaos is to involve users to a greater and greater extent.

7. A new method that implies organisational changes in the system development function (and, in particular, changes in the career structure) may be deliberately misused, or even sabotaged, by some system developers.

Resistance from users

Many new system development methods imply that the user will increasingly be involved in the development process, even to the extent that he will be asked to develop his own system. Users may resist this increased involvement for the following reasons:

1. Some users perceive computer applications as having a mystique that is beyond their ability to understand.

2. Some users are apprehensive of computers in general, and of terminals in particular. This is most noticeable in companies where either users do not know very much about computers or there are not many on-line systems.

3. Some users have misconceptions about the way in which a technological system will react to the use they make of it. They may actually believe that a trivial mistake on their part could ruin either the equipment or the system. Systems that are not 'user-friendly' systems encourage users in this belief.

4. The experience many users have had during the past several years have left them with a feeling of general dissatisfaction about computer systems. Disillusioned users may consider that to be involved with system development will be a waste of their time.

5. Users may perceive personal computing or a greater involvement with system development as a move by the data processing staff to transfer some of their work-load to the users.

6. Users may be fearful of being transformed from passive receivers of systems to active developers of systems.

7. Many users may not be willing to take on the increased responsibility that a more active involvement with system development implies. They may in fact quite like the barrier that a bulky specification (which they cannot understand) creates between them and the system developers. Consequently, they may resist the introduction of a new method that makes their decisions more visible.

Resistance from senior managers

Senior managers will be most involved with a new development approach or method when funds have to be released for purchasing a methodology (and any relevant software facilities), and for training staff, etc. They may resist releasing funds for the following reasons:

1. Senior managers are, in general, not interested in the technical details of a new approach or method. When a method is explained in a simplified form to them, they perceive it as 'plain common sense'. It is difficult to convince them that common sense is not very common in practice.

2. It is difficult to justify the use of an approach or a method by means of a cost/benefit analysis. The future return on investment that will offset the cost of introducing a new approach or method is not easy to quantify. Senior managers may not look favourably on a proposal to commit funds where the benefits, and the timescale in which they will be achieved, are uncertain.

3. An investment in a new approach or method requires that money be spent on ephemeral things, such as training courses, discussion sessions and pilot projects. Many senior managers feel more comfortable when they are authorising investment in items that can be counted as capital assets.

4. Some senior managers may resist the introduction of a new method because they believe that it will upset the status quo, and so will be a disrupting influence on the organisation. They will also resist the introduction of a method if they believe that other executives may use it to gain control of a part of the business for which they themselves are currently accountable.

5. Many of the new system development approaches and methods are not yet in widespread use. Most senior managers are reluctant to commit funds to products such as these.

OVERCOMING THE RESISTANCE TO CHANGES

The list of possible reasons that system developers, users and senior managers may have for resisting the introduction of a new method for system development is a formidable one. J. P. Kotter and L. A. Schlesinger have suggested six methods for dealing with resistance to change, and figure 3 summarises the situations in which each method is commonly used, and the advantages and the drawbacks of each method. These advantages and drawbacks show clearly that not one of the six methods can guarantee that resistance to change will be overcome completely. We believe, however, that most organisations will find that the first three methods (education plus communication, participation plus involvement, and facilitation plus support) are the most effective methods when working practices are being changed.

In most companies, it is easier to overcome resistance to change when the new approach receives visible backing from higher management. To overcome the resistance to changes that the introduction of new system development approaches and methods brings about, the ideas that underlie the approach or the method need to be sold. They need to be sold to senior managers, to middle managers and to technicians and users — and they need to be sold in that sequence. Senior managers need to understand the reasons why the new approach or method is being introduced and the benefits it will produce. Middle managers need a more detailed overview of the approach or the method so that they can clearly assess the implications of using it. Experienced technicians and users need to be convinced of the merits of the new approach or method. Often it is quite difficult to convince them, but once they are convinced they tend to become the most dedicated advocates of the new way.

Resistance to changes brought about by a new method of developing systems will continue after the new method has been introduced, and the resistance can be at least partly overcome by properly motivating those who are using that new method. It is important to recognise that when a new approach or method is used for the first time, it will not be completely effective. Consequently, those who are practising the new method must be skilfully managed, so as to ensure that their enthusiasm for the new method does not wane.

Figure 3 Methods for dealing with resistance to change

Method	Common usage	Advantages	Drawbacks
Education plus communication	Where there is either a lack of information or inaccurate information and analysis.	Once persuaded, people will often help with the implementation of the change.	Can be very time-consuming if lots of people are involved.
Participation plus involvement	Where the initiators do not have all the information they need to design the change, and where others have considerable power to resist.	People who participate will be committed to implementing change, and any relevant information they have will be integrated into the change plan.	Can be very time-consuming if participators design an inappropriate change.
Facilitation plus support	Where people are resisting because of adjustment problems.	No other approach works as well with adjustment problems.	Can be time-consuming, expensive, and still fail.
Negotiation plus agreement	Where someone or some group will clearly lose out in a change, and where that group has considerable power to resist.	Sometimes it is a relatively easy way to avoid major resistance.	Can be too expensive in many cases if it alerts others to negotiate for compliance.
Manipulation plus agreement	Where other tactics will not work, or are too expensive.	It can be a relatively quick and inexpensive solution to resistance problems.	Can lead to future problems if people feel they are being manipulated.
Explicit and implicit coercion	Where speed is essential, and the initiators of the change possess considerable power.	It is speedy, and can overcome any kind of resistance.	Can be risky if it leads to people resenting the initiators.

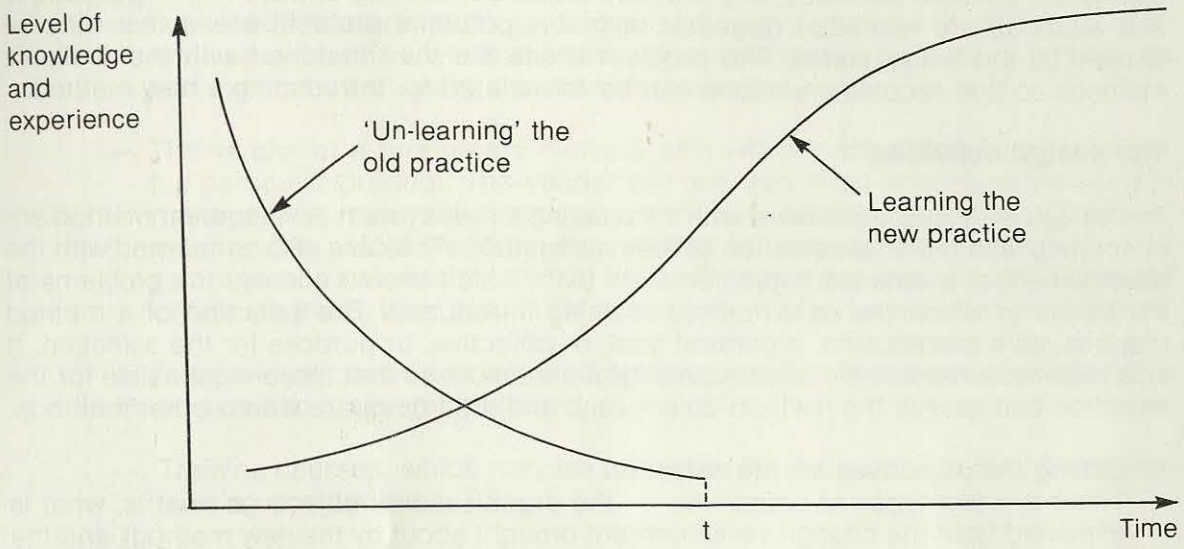
The most important motivating factor for technical people and users is positive management backing. Senior management should ask for regular feedback on the progress with, and the problems involved in, introducing the new method. They should take advice on when and how best to demonstrate their interest, and they should act positively when their support is asked for.

There is, in fact, a simple reason that explains why a new method is not completely effective when it is first used. This reason is the double learning curve (illustrated in figure 4) that always applies. Most organisations select experienced staff to be the first users of a new method. Unfortunately, their experience is often in another method that will prejudice their attitudes to the new method. Even when they believe in the new method, they have first to reduce their dependence on the old method before they can become effective practitioners in the new one. The first users of the new method have, in effect, to 'un-learn' the old method, and until this un-learning process is completed, they will have doubts and second thoughts about the merits of the new method.

GUIDELINES FOR INTRODUCING NEW METHODS OF DEVELOPING SYSTEMS

The introduction of a new method of working into an organisation should not be undertaken lightly. A new system development approach or method will have far-reaching effects on the working lives both of the system developers and the users of the systems that are developed. New approaches or methods should therefore be introduced in a planned way.

Figure 4 The double learning curve that applies to the introduction of a new method



Note: Until time 't' is reached, technicians will have doubts and second thoughts about the new practice.

The plan for introducing a new method of developing systems can be likened to the plan for any development project. Thus the plan for introducing a new method needs to allow for three major activities that are similar to analysis activities, design activities and implementation activities. We now discuss these three major activities as they relate to a project for introducing a new system development method. The process of introducing a new working method is a highly-iterative one. The relevant activities are presented here in chronological sequence but it should not be inferred from this that they need to be executed in that precise sequence.

The analysis activities

In the context of introducing a new system development method into a department or a group of people, the analysis activities are primarily concerned with gathering facts, identifying problems and making recommendations. Three types of facts need to be gathered:

— Facts about the current working practices of the department or group. These facts may be difficult to unearth because they may, in reality, be very different from what the standards manual says they should be.

— Facts about the possible methods that are available. (The ways of gaining a detailed understanding of a method are described on page 51.)

— Facts about the future plan for either system changes or operational changes that may have an effect on systems. These facts may not be easy to gather

because the future plans of the company may not have been made available to the data processing department. Alternatively, if they have been made available, they may not have been transformed into a system strategy.

During the analysis activities, any problem areas concerning current working practices and methods are identified (together with any potential problem areas that may be caused by the future plans). The problem areas are then matched with the available methods so that recommendations can be formulated for introducing a new method.

The design activities

The design activities associated with introducing a new system development method are concerned with the final selection of the new method. They are also concerned with the development of a detailed implementation plan, which should address the problems of the period in which the new method is being introduced. The selection of a method requires, as a prerequisite, a general goal, or objective, or purpose for the selection. It also requires a reasonable understanding of a method, so that those responsible for the selection can assess the method objectively, and then compare it with other methods.

1. Setting the objectives for the selection
   There are two types of objectives — the organisational objectives (that is, what is expected from the changed environment brought about by the new method), and the technical objectives of the method itself. Both sets of objectives are used to create the criteria against which the method will be assessed (as discussed on pages 51 and 52). We set out below a checklist of the possible benefits that may be considered desirable when setting the objectives.

   — Better control of projects or schedules. This could be achieved by a method that makes it easier to define the scope and the duration of the development project, or that provides facilities for inspection and quality control.

   — Better communication between various technical disciplines and between system developers and users (thereby involving the user to a greater extent in the development process).

   — Flexibility of resources, so that there is continuity when staff leave, and so that new staff can be easily integrated into the development project.

   — Higher efficiency in the system development process. This benefit might be achieved by a method that uses predefined building blocks, or that requires less manual (or routine) work, or that makes better use of human intelligence, or that requires less testing and correction work, or that makes use of machine-aided processes.

   — Improved quality of the products produced by the use of the method. Quality objectives might be stated in terms of system reliability, or system flexibility, or easier maintenance.

   — Better documentation. This benefit might be achieved by a method that produces concise graphical documentation that is a by-product of the system development process. Alternatively, the objective might be to produce consistent and simple documentation that will be updated automatically when the product is modified.

   This list is not exhaustive, but it can be used as a basis for determining the detailed evaluation criteria that will be used when a method is assessed.

2. Understanding the method
During the analysis activities, a general understanding of the method will have been gained. During the design activities, a more detailed understanding is necessary, and the following sources of information can be used:

   — Published literature, such as data processing magazines (in particular *EDP Analyzer*, which is a worthwhile source of information both about techniques and experience of using a particular method).

   — The vendor of a proprietary method, who will provide ample material about his particular product. The vendor will also run short briefing seminars and management overview seminars, and will publish some of his users' experiences. Most vendors are willing to provide potential users with references to successful users of the method, but any unsuccessful users are difficult to trace. User groups for a particular method are almost non-existent.

   — Independent consultancies, who are in a position to provide a comparative opinion about several methods.

   — Training courses, which may be run either by the vendor or by independent training companies.

3. Assessing the method
A method should be assessed at two levels. First, the individual features and qualities of the method should be assessed, and second, the method should be assessed by comparing it with other methods. The following checklist can be used for assessing the features and the qualities of a method:

   — The technical quality should be assessed in terms of its consistency, its logic, its interface with other methods and automated aids, its documentation, the parts of the development process that it aids, etc.

   — The ease with which the method can be taught should be assessed.

   — The implications of using the method should be assessed in terms of hardware and software requirements, organisational structure, the management style required and the method's dependence on other methods.

   — The orientation of the method should be assessed to determine whether it is either a technical method that helps the system designer or programmer, or really a communication method that aids communication between the different groups of people involved in the system development process.

   — The time-related factors of using the method should be assessed. These factors include the time people take to become proficient in the method, the time a proficient person takes to use the method, and the time it takes to change the resulting system developed by using the method.

   — The human factors of the method should be assessed. These factors include a consideration of whether people like to use the system, and why they like to use it.

Different methods can be compared with one another in one of two different ways. The first way is to compare each method with a set of desired objectives and benefits. Alternatively, in the absence of well-defined objectives, methods can be

compared with one another. This type of comparative analysis (or feature analysis) aids in acquiring an understanding of the commonality or the diversity that exists in several methods, and so it is a good way of gaining an overview about several methods. It can, of course, also be used to collect knowledge about methods, and to develop the technical objectives of introducing a new method. Comparative analysis is an effective selection mechanism when one of the objectives of introducing a new method is to cause as little disruption as possible.

The implementation activities

The implementation activities are concerned with training the staff who will use the new method, introducing the new facilities, running pilot projects, and modifying standards. The process of implementing a new system development method takes much longer than the process of implementing a data processing project. Experience has shown that the worst possible way of implementing a new method is to train all the staff at one and the same time and then to issue a directive that the new method is to be used from a particular date. People need a considerable transition period to adjust fully to a new method, and it is not unusual for the full benefits of a new method to take two years to achieve.

During the research for this report we spoke to vendors of system development methods, to training companies, and to data processing departments that had successfully introduced a new method. They all agreed that to achieve a successful implementation required the following actions to be taken, and to be taken in the following sequence:

- Appointing a person to be responsible for the implementation.
- Appointing a person (or a team) both to evaluate fully the shortlisted methods and to provide an internal consultancy facility.
- Selecting a pilot project and a project team.
- Training the project team.
- Creating guidelines for using the method.
- Implementing and evaluating the pilot project, and modifying the guidelines if necessary.
- Planning the training of staff, and selecting the projects that will be used to implement the new method.
- Training the technical managers.
- Training the staff.
- Implementing the new method with the selected projects.
- Providing continuous support to the staff who are implementing the method.
- Appraising the success of the implementation, and modifying and extending the guidelines if necessary.
- Incorporating the method into the standard training programme.
- Providing advanced training for those staff who are now experienced in using the method.

We now provide specific guidelines for selecting pilot projects, and for supporting a gradual implementation of the method:

1. Guidelines on selecting pilot projects and the pilot project team
   The selected pilot projects should be technically simple to allow the project team to concentrate on the new method. Consequently, pilot projects should be:

   — Short in duration (typically three to six months in duration).

   — Not time critical.

   — Not innovative.

   The pilot project team should consist of carefully selected people who, apart from having their proven technical ability, should be:

   — Predisposed to the introduction of the new method.

   — Aware that the new method is being tested.

   — Capable of evaluating the method as well as the end result of using the method.

   — Able to act as in-house consultants and trainers.

2. Guidelines on planning for a gradual implementation
   It will take time to develop in-house expertise on using the new method. Until such expertise has been developed, it is essential that experienced external support should be available. Thereafter, some larger companies develop their own in-house team of experts who may, for several years, provide guidance to project teams on the use of the method.

   Organisations should expect that there will be problems with using the new method during the first few projects. They should also expect that those problems will not have been encountered during the pilot project. The problems will be caused by the newly-trained (and probably less experienced) staff who are using the method, because the project tends to be more difficult than the pilot project was. It is sensible therefore to arrange for members of the pilot project team to be included as members of the project teams for the first 'live' projects.

   A potentially dangerous situation often develops during the first live project. As already mentioned, this project tends to be more difficult than the pilot project was, and it has a project team that is partly inexperienced in the new method. Those who already have experience of using the method will be keen to demonstrate their expertise to the other team members, and so they may be inclined to 'over-use' the method. This tendency is likely to confuse other team members. When new problems occur that have not been encountered during the pilot project, the team will not be equipped to deal with them easily. As a result, a panic situation is likely to occur part-way through the project. It is most important to handle this panic situation correctly, and the following actions need to be taken:

   — Management support should be both visible and positive.

   — The project should be rescheduled.

   — No one should be reprimanded or removed from the project team (even when a team member asks to be removed).

Provided that the early panic situations are managed correctly, the correct balance between experienced and inexperienced staff should be established from about the third project that uses the new method. From this time onwards, the new method should begin to realise its anticipated benefits.

CONCLUSION

In this chapter we have shown that the task of introducing a new system development method is not a simple one. Those responsible for introducing a new method need to recognise that there will be resistance to the new method from system developers, from users, and from senior managers. The introduction of a new method needs, therefore, to be planned in a way that is not dissimilar to the plan for any other development project. In particular, the plan should focus on the early stages of implementing the method, but it should also recognise that it may take several years to realise the full benefits of the new method.

CHAPTER 7

SUMMARY AND CONCLUSIONS

This report has shown that the system developer has available today a much greater variety of system development approaches and methods than were available to him in the past. This greater variety has resulted from advances in technology, from past experience of developing and using computer systems, and from theories both about systems and the system development process. The methods and approaches we have identified in this report can be classified broadly into those that concentrate on the development of the technological system itself, and those that concentrate on the vital planning process that should precede the introduction of a technological system.

Advocates of a particular system development approach or method can always show that, for specific circumstances, it produces impressive results. However, today's range of system development approaches and methods poses for the system developer the four major problems that we identify and discuss below.

The first problem is concerned with the fact that there is a choice of system development approaches and methods. In the past, the system developer had only one empirically-based development method available to him. Today, although he may be skilled both in choosing the technology, and in understanding the implications of his choice, he is less skilled in choosing a development approach. In addition, he will find that there is little overall guidance available to help him make his choice.

The second problem is concerned with the great differences there are in the breadth of scope the different approaches offer. For example, some approaches (such as ISAC) deal with the complete planning and development of a system, whereas other approaches (such as the structured approaches) deal only with the technical development of a system. By contrast, many of the newer approaches (such as the participative approach or prototyping) concentrate only on a particular aspect of system development, and they provide little guidance (or no guidance at all) for other areas of the development process. As a result, many of the newer approaches can be used only in conjunction with other approaches, or methodologies, or methods. It is not an easy task to match and to interface different approaches, and there is little practical experience and guidance available to the system developer to assist him in this task.

The third problem is caused by the fact that there is little solid experience of using some of the newer approaches. The ideas and the reasoning behind the newer approaches are powerful and convincing, but they are not yet supported by an equally impressive record of experience. Advocates of a particular approach inevitably emphasise the merits of the approach, but each approach will have its limitations. Unfortunately, those limitations will not emerge until a substantial amount of experience of using the approach has been gained. In addition, there are, as yet, few critical and objective assessments of alternative approaches that the system developer can use to help him select the most appropriate approach for his particular circumstances.

The fourth problem concerns the process of introducing a new system development approach. The introduction of a new approach needs careful planning and skilled management of the changeover tasks. If these vital tasks are not given sufficient

attention, undesirable organisational consequences may result, and the final result may be that the new approach will be rejected. Most system developers have little experience of changing their own working methods, and when they are planning and managing the introduction of a new system development approach they can learn much from system users. In the past, system users' working methods have been changed (if only inadvertently) by various computer systems.

We have shown, however, that choice in the area of system development methods is not restricted to a choice of approach. Today, the system developer also has available a wide variety of automated development aids, and most of the problems we identified above apply also to the task of selecting the appropriate aids. In addition to facing these problems, the system developer may not yet be clear as to how he can best fit the selected aid into the development process in the most effective way. His quandary in this matter is illustrated by the way in which high-level languages, databases, data dictionaries, and other similar aids are currently being used equally for prototyping, for personal computing and as development aids for certain other areas of system development. When and where the system developer should use these aids is a matter for his individual choice and interpretation.

In addition, the problems we have identified so far are exacerbated because system developers in the future will increasingly be developing system solutions that are outside the narrow confines of traditional data processing. Many of the approaches and the methods we have discussed in this report apply equally to the development of wider information processing systems, such as word processing systems, office systems and videotex systems. These types of systems often offer genuine alternatives to traditional data processing systems. However, experience has shown that these non-data processing systems can often be more effective if they co-operate with (or are backed-up by) a conventional data processing system, with both types of systems sharing (and contributing to) the same data resource. At the present time, there are no known methodologies that either permit a clearcut choice to be made between these alternative system solutions, or that make it easy to interface the data processing parts and the non-data processing parts of the total system.

Some of the problems we have identified (and particularly those that result from the newness of some approaches) will disappear with time. But it is highly unlikely that an overall system development methodology will ever be offered that will eliminate the need for the system developer to choose the right approach and the right aids for a given system development situation. Increasingly therefore the system developer will be faced with the need to choose between several alternative development approaches and aids. If he is mentally committed to follow the single approach that he is most familiar with, he will find it impossible to make a rational choice, because his blinkered attitude will not allow him to give objective consideration to any other alternative. This observation applies also to the selection of a system solution, or to the selection of hardware, or to the selection of a development aid.

Ultimately, the chosen system solution, development approach, development aids and hardware represent 'tools' at different levels either in the system development process or in the system itself. In this report, we have made it clear that tools are appropriate only if they fit effectively into the wider environment in which they will be used. If instead tools are selected without proper consideration of the environment in which they will be used, the most important reason for justifying their use is neglected. If they prove to be useful, this will be only by chance. System developers already have considerable experience of choosing system solutions and hardware, and they need now to acquire corresponding skills in choosing development approaches and aids.

We began this report by emphasising that the view that a system developer holds about systems determines the choices that he will make concerning development approaches and aids. The closed system view was held by most system developers at the time when the traditional (or analytical) approaches were developed, but we believe that, today, it is desirable for system developers to hold a more open view of the nature of systems. An open system view will enable the system developer to detach himself from the details of individual approaches and aids, and this detachment will enable him to make the most effective choice of approaches and aids. An open system view will therefore allow the system developer to make his choices firstly on the basis of their relevance to the surrounding environment in which they are to serve, and secondly on their technical merit.

Our research for this report has identified many new and different system development approaches. Each of these approaches has its strengths and weaknesses, and organisations will have to decide for themselves how best to use the different approaches. We believe that not only is an open system view desirable, it is also the only possible view of systems that can be held if the new and different approaches are to be used effectively in employing computer-based technology.

BIBLIOGRAPHY

GENERAL READING

Abbatt, J., Campbell, C., Jones, A. H., Land, F. F., *New approaches to systems analysis and design,* BCS business information systems specialist group, UK working paper, 1980.

Beer, S., *Brain of the firm,* Allen Lane, Herder, 1972.

Beer, S., *Platform for change,* Wiley, 1975.

Donaldson, H., "General scenario for future dp (1980-1990): realising technological trends within the business", *Infotec state of the art review, 1980.*

Hedberg, B., "The human side of information processing", *Proceedings of conference on computer impact,* North Holland, 1980.

Hedberg, B., Mumford, E., "The design of computer systems. Man's vision of man as an integral part of the system design process", *Human choice and computers,* North Holland, 1975.

London, K., *The people side of systems,* McGraw-Hill (UK), 1976.

Nolan, R. L., "Managing the crises in data processing", *Harvard Business Review,* March/April 1979.

Nygaard, K., "The impact of social movements", *The Computer Journal,* Vol. 23, No. 1, February 1980.

Scarrott, G. G., "From computing slave to knowledgeable servant: the evolution of computers", *Proceedings of the Royal Society, London,* A369, 1-30 (1979).

"Educating executives on new technology", *EDP Analyzer,* Vol. 18, No. 11, November 1980.

"How to use advanced technology", *EDP Analyzer,* Vol. 17, No. 9, September 1979.

"Introducing advanced technology", *EDP Analyzer,* Vol. 18, No. 3, March 1980.

"The coming impact of the new technology", *EDP Analyzer,* Vol. 19, No. 1, January 1981.

CHAPTER 3

General system theory

von Bertalanffy, L., "General system theory", *General systems, year book of the society for the advancement of general system theory,* Vol. 1, 1956.

von Bertalanffy, L., "The theory of open systems in physics and biology", *Science,* Vol. III, 1950.

Katz, D., Kahn, R. L., "Common characteristics of open systems", *Systems thinking,* Penguin Modern Management Readings, Penguin Books Ltd, 1972.

Weinberg, G. M., *An introduction to general systems thinking,* John Wiley & Sons, N.Y., 1975.

Young, O. R., "A survey of general systems theory", *General systems,* Vol. 9, 1964.

Socio-technical approach

Emery, F. E., Trist, E. L., "Socio-technical systems", *Systems thinking*, Penguin Modern Management Readings, Penguin Books Ltd, 1972.

Herbst, P. B., *Socio-technical design strategies in multidisciplinary research*, Tavistock Publication, 1974.

Kelly, J. E., "A reappraisal of socio-technical system theory", *Human Relations*, Vol. 31, No. 12, Tavistock Inst., 1978.

Work organisation

Herzberg, F., Mausner, B., Snyderman, B. B., *The motivation to work*, John Wiley & Sons, N.Y., 1967.

Maslow, A., *Motivation and personality*, McGraw-Hill, 1954.

Mayo, E., *The human problems of an industrial civilization*, Macmillan, N.Y., 1933.

McGregor, D. M., "The human side of enterprise", *Adventures in thought and action*, (proceedings of the Fifth Anniversary Convocation of the School of Industrial Management, MIT, Cambridge, Mass., April 1975), Technology Press, Cambridge, Mass.

Scott-Myers, M., "Who are your motivated workers", *Harvard Business Review*, January-February, 1964.

Taylor, F. W., *Scientific management*, McGraw-Hill, N.Y., 1947.

"The new industrial relations", *Business Week*, May 11, 1981.

CHAPTER 4

Structured approaches and requirement engineering

Gane, C., Sarson, T., "Structured Methodology: What we have learned?", *Computerworld Extra*, 1980.

Gane, C., Sarson, T., *Structured systems analysis: tools & techniques*, Improved System Technologies Inc., N.Y., 1979.

Jackson, M. A., *Constructive methods of program design*, M. Jackson Systems, London, 1979.

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

Jackson, M. A., *Some principles underlying a system development method*, M. Jackson Systems, London, 1981.

de Marco, T., *Concise notes on software engineering*, Yourdon Press Monographs, 1979.

de Marco, T., *Structured systems analysis,* Yourdon Press, N.Y., 1978.

Myers, G. J., *Composite/structured design*, Van Nostrand Reinhold, N.Y., 1978.

Orr, K. T., *Structured system development*, Yourdon Press, N.Y., 1977.

Page-Jones, M., *The practical guide to structured systems design*, Yourdon Press, N.Y., 1980.

Peters, L. J., Tripp, L. L., "Comparing software design methodologies", *Datamation*, November 1977.

Rudkin, R. I., Shere, K. D., "Structured decomposition diagram: A new technique for systems analysis", *Datamation*, October, 1979.

Warnier, J. D., *Logic construction of programs*, H. E. Stenfert Kroese Bv., Leiden, 1974.

Weinberg, G. M., *The psychology of computer programming*, Van Nostrand Reinhold, N.Y., 1971.

Yourdon, E., *An analysis of the M. Jackson design methodology*, Yourdon Inc., N.Y., 1977.

Yourdon, E. (ed.), *Classics in software engineering*, Yourdon Press, N.Y., 1979.

Yourdon, E., Constantine, L. L., *Structured design*, Yourdon Inc., N.Y., 1975.

Yourdon, E., *Structured systems development*, Yourdon Inc., N.Y., April, 1980.

Yourdon, E., *Structured walkthroughs*, Yourdon Press, N.Y., 1978.

Core seminar material, Systems Designers Ltd., U.K., 1981.

JSP & JSD — An introduction, M. Jackson Systems, London, 1980.

Meta system 1000 reference manual, Structured Methods Inc., N.Y., 1979.

"Program design techniques", *EDP Analyzer*, Vol. 19, No. 3, March, 1979.

Requirement specification techniques study, Baseline reports, Systems Designers Ltd., U.K., 1979.

SofTech and the structured analysis and design technique, SofTech Inc., Mass., 1978.

Stradis reference manual, McAuto/IST, Missouri, 1981.

"Structured software development", *Infotech state of the art report, 1979.*

Structuring the analysis and design process, the key to more effective applications development, SofTech Inc., Mass., 1976.

"The production of better software", *EDP Analyzer*, Vol. 17, No. 2, February, 1979.

Database approach

Flavin, M., *Information modelling*, Yourdon Inc., N.Y., May, 1980.

Flint, D., *Trends in database management systems*, Butler Cox Foundation, Report Series No. 12, June, 1979.

Martin, J., *Computer database organization*, Prentice Hall, 1977.

Palmer, I., *System development in a shared data environment*, CACI Inc., International, London, 1980.

Software engineering

Avizienis, A., "Can reliable computing through fault tolerance be attained in the 1980s?", *Infotech state of the art review, 1980.*

Charignon, P., *A new approach to pay packages: The Pacha System*, Sopra, Paris, 1980.

Charignon, P., *Thoughts on the development of generalised packages*, Sopra, Paris, 1980.

Fagan, M. E., "Design and code inspection to reduce errors in program development", *Infotech state of the art review, 1980.*

Kopetz, H., "Design for maintainability", *Infotech state of the art review, 1980.*

Manchester, P., "Software engineering management at IBM", *Computerworld U.K.*, 4 February, 1981.

Rustin, R. (ed.), *Debugging techniques in large systems*, Prentice-Hall, N.J., 1971.

Smiley, D. H., "Process standardisation in information systems engineering", *The Australian Computer Journal*, Vol. II, No. 2, May, 1979.

Thomas, N., "A hard plan inside IBM"; "Why FSD maths add up to management"; "The FSD tools to make the software ford"; "Education to integration", *Computing*, 26 March, 1981.

"Key issues of software engineering", *Infotech state of the art report, 1979.*

"Progress in software engineering (Parts 1 & 2)", *EDP Analyzer*, Vol. 16, Nos. 2 & 3, Feb/March, 1978.

"Software development", *IBM systems journal*, November, 1980.

Planning approaches

Gilb, T., *Design by objectives course manual*, 1980.

Lucas, H. C., *Computer based information systems in organisations*, McGraw-Hill, 1973.

Martin, J., *Principles of data-base management*, Prentice-Hall, 1976.

Noguenda, G. R. D., "On the methodologies for analysis of information systems", *MA thesis report in systems in management*, University of Lancaster, Dept. of Systems, 1979.

Siegel, P., *Strategic planning of management information systems*, Petrocelli, N.Y., 1975.

Zani, Dr. W., "Blueprint for M.I.S.", *Harvard Business Review*, November-December, 1970.

Planning for information system development, IBM customer executive education.

PRISM, Deltacom Inc., Southampton, PA., 1979.

"What information do managers need?", *EDP Analyzer*, Vol. 17, No. 6, June 1979.

Personal computing

Martin, J., *Application development without programmers*, Savant Institute, 1981.

Tagg, R. M., "Query languages", *Infotech state of the art conference, June, 1981.*

Voysey, H., "Who pulls the string?", *Computer Management*, September, 1980.

Wasserman, A. I., "User software engineering and the design of interactive systems", *proceedings 5th International Conference on software engineering*, San Diego, Ca., March, 1981.

"Programming by end users", *EDP Analyzer*, Vol. 19, No. 5, May, 1981.

"Supporting end user programming", *EDP Analyzer*, Vol. 19, No. 6, June 1981.

"Spending money outside", *Computer Management*, September, 1980.

Prototyping and system evolution

Alexander, C., *Notes on the synthesis of form*, Harvard University Press, 1964.

Bally, I., Brittan, J., Wagner, K., "A prototype approach to information system design and development", *Information and Management*, Vol. 1, 1977.

Brittan, J., "Design for a changing environment", *The Computer Journal*, Vol. 23, No. 1, February, 1980.

Gutierrez, J. O., "Justification and theoretical evaluation of experimental techniques in information systems", *MSc. thesis report in analysis, design and management of information systems*, London School of Economics, 1981.

Hanau, P. R., Lenorovitz, D. R., "A prototyping and simulation approach to interactive computer system design", *Proceedings of ACM*, 1980.

James, E. B., "The user interface", *The Computer Journal*, Vol. 23, No. 1, February, 1980.

Jenkins, A. M., Naumann, J. D., "A prototype model as a MIS design technique", *Discussion Paper No. 131*, School of Business, Indiana University, September, 1980.

McCracken, D. D., *A guide to Nomad for applications development*, National CSS Inc., 1980.

McCracken, D. D., "Software in the '80's. Perils and promises", *Computerworld Extra*, 1980.

McCracken, D. D., "The Nomad approach", *Datamation*, May, 1980.

Stamper, R., "Evolutionary development of large systems", *Legol working paper*, London School of Economics, 1981.

"Developing systems by prototyping", *EDP Analyzer*, Vol. 19, No. 9, September, 1981.

Nomad — A creative approach to information, National CSS Inc., 1979.

Protocycle — A dynamic applications development methodology, National CSS Inc., 1980.

Participative approach

Land, F. F., "Adapting to changing user requirements", *Long Life Systems, Vol. 2, Infotech state of the art review, 1980.*

Kling, R., "Social analyses of computing: Theoretical perspectives in recent empirical research", *Computing Surveys*, Vol. 12, No. 1, March, 1980.

Mumford, E., Henshall, D., *A participative approach to computer system design*, Associated Business Press, London, 1979.

Mumford, E., "Participative system design: A description and evaluation of a consensus approach", *paper presented at the Trade Union Conference in Vienna, 1979.*

Mumford, E., "Participative system design: Practice and theory", *IFIP, TC8 Working Paper*, Manchester Business School, 1980.

Mumford, E., *The participative design of new technology: Four design tools to assist the design process*, Manchester Business School, 1980.

Mumford, E., Land, F., Howgood, J., "A participative approach to the design of computer systems", *Impact of science on society*, Vol. 28, No. 3, 1978.

Life cycle views

Floyd, C., "A process-oriented approach to software development", *Systems architecture, proceedings of the Sixth ACM European Regional Conference ICS*, 1981.

Lehman, M. M., "An environment of program development and maintenance — programs, programming and programming support", *Systems architecture, proceedings of the Sixth ACM European Regional Conference, ICS*, 1981.

Lehman, M. M., "Introduction to the software engineering state of the art and its future", *Infotech state of the art review, 1980.*

Lehman, M. M., "Programs, life cycles and laws of software evolution", *proceedings of the IEEE*, Vol. 68, No. 9, September, 1980.

"Life cycle management", *Infotech state of the art report, 1980.*

Systems approach

Campbell, B. T., "An examination of three systems analysis and design methodologies", *Project report, MSc. in analysis, design and management of information systems*, London School of Economics, 1981.

Checkland, P. B., "Towards a systems based methodology for real world problem solving", *Journal of systems eng.*, Vol. 3, No. 7, April, 1972.

Lundeberg, M. et al., "A systematic approach to information system development", *Information systems*, Vol. 4, 1979.

Methods and techniques in DB/DC projects, IBM Nordic Education Centre, Seminar: EA32, Handout: 1977-11-14.

"The analysis of user needs", *EDP Analyzer*, Vol. 17, No. 1, Jan. 1979.

CHAPTER 5

Belady, L. A., "Manual, machine-aided and mechanical processes in software development and maintenance", *Infotech state of the art review, 1980.*

Bourne, T. J., "The data dictionary system in analysis and design", *ICL Technical Journal*, November, 1979.

Bryce, M., *Pride — ASDM overview, automated systems design methodology*, M. Bryce & Assoc., Inc., Cincinnati, Ohio, 1981.

Falla, M. E., "Using Gamma to support structured analysis", *Gamma project working paper G2/WP39*, Software Sciences Ltd., November, 1980.

Gutz, S., Wasserman, A. I., Spier, M. J., "Personal development systems for the professional programmer", *Computer*, April, 1981.

Inzelstein, S., "Management of information systems", *IBM large systems conference*, Montreux, 1980.

Kernighan, B. W., Mashey, J. R., "The Unix programming environment", *Computer*, April, 1981.

Maddock, R. O., *Software development using Mascot*, Software Sciences Ltd, April, 1980.

Manchester, P., "Relational databases", *Computerworld UK*, March, 1981.

Manchester, P., "Unix", *Computerworld UK*, 8 April, 1981.

Osterweil, L., "Software environment research: directions for the next five years", *Computer*, April, 1981.

Pierce, R., "Computer aided software design: the Gamma system", *Computer Bulletin*, March, 1980.

Stecher, P., Allenstein, V., "The application controller concept: A first experience", *The Computer Journal*, Vol. 24, No. 2, 1981.

Teichroew, D., "Computer aided software development", *Infotech state of the art tutorial, March 1977.*

Teitelman, W., Masinter, L., "The Interlisp programming environment", *Computer*, April, 1981.

Voysey, H., "Why not use the database design tools?", *Computerworld UK*, 3 December, 1980.

Wasserman, A. I., "Automated development environment", *Computer*, April, 1981.

Winston, L. E., "A novel approach to computer application system design and implementation", *Hewlett-Packard Journal*, April, 1981.

Zahn, C. T., *C notes, a guide to the C programming language*, Yourdon Press Monograph, N.Y. 1979.

"A new view of data dictionaries", *EDP Analyzer*, Vol. 19, No. 7, July, 1981.

"Application system design aids", *EDP Analyzer*, Vol. 19, No. 10, October, 1981.

Asset — An introduction to the PSL/PSA software system, Systems Designers Ltd., 1981.

Context user guide, Systems Designers Ltd., December, 1980.

IBM DP systems handbook, GE19-5234-2, September, 1980.

ICL — Data dictionary system, technical overview, International Computers Ltd., 1980.

"Programming work stations", *EDP Analyzer*, Vol. 17, No. 10, October, 1979.

Slave — Product overview, Dataskil Ltd., July, 1980.

"Stand-alone programming work-stations", *EDP Analyzer*, Vol. 17, No. 11, November, 1980.

"The production of better software", *EDP Analyzer*, Vol. 17, No. 2, February, 1979.

Unet communication software for Unix, 3 Com Corporation, California, August, 1980.

Xenix, Logica, U.K., 1981.

CHAPTER 6

Feeney, W., Sladek, F., "The systems analyst as a change agent", *Datamation*, November, 1977.

Kotter, J. P., Schlesinger, L. A., "Choosing strategies for change", *Harvard Business Review*, March/April, 1979.

Yourdon, E., *Structured systems development*, Yourdon Press, N.Y., April, 1980.

"Program design techniques", *EDP Analyzer*, Vol. 17, No. 3, March, 1979.

"Structured systems development", *Infotech state of the art report, 1979.*

The Butler Cox Foundation

Report Series No 25 System Development Methods

The author of this report would welcome the opportunity of discussing the findings of the report with small groups of members. Would you therefore please list below (and overleaf if necessary) any points on which you would like the author to expand at such a discussion. Also, would you please record your interest in follow-up work on the subject of this report, highlighting those aspects of the topic that are likely to be of most interest to your organisation. We will then contact you to arrange a suitable venue and date for the discussion.

Completed by .....

Job title .....

Organisation .....

Address .....

.....

Please return this form to your local address of the Butler Cox Foundation shown on the back of this report.

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

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
La Fondation Butler Cox
Tour Akzo, 164 Rue Ambroise Croizat,
93204 St Denis-Cedex 1, France
☎ (1) 820.61.64, Telex 610789 ASFRA

United States of America
Butler Cox & Partners Limited
216 Cooper Center, Pennsauken, New Jersey 08109, USA
☎ (609) 665 3210

Switzerland and Germany
Butler Cox & Partners Limited
Morley House, 26-30 Holborn Viaduct, London EC1A 2BP
☎ (London) 583 9381, (Zurich) 302 0848

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