A SYSTEMIC APPROACH TO ESTABLISH FUNDAMENTAL PRINCIPLES OF COMPUTER APPLICATIONS PACKAGES DESIGN

BY

IHAB ABDEL RAHMAN TAWFIC

B.Sc. Alex., M.Sc. Aston

Thesis submitted to the University of Aston in Birmingham for the Degree of Doctor of Philosophy

MANAGEMENT CENTRE

July 1975

This thesis is dedicated to the memory of my father, Abdel Rahman Tawfic, who had faith in me.

SYNOPSIS

This work attempts to create a systemic design framework for man-machine interfaces which is self consistent, compatible with other concepts, and applicable to real situations. This is tackled by examining the current architecture of computer applications packages. The treatment in the main is philosophical and theoretical and analyses the origins, assumptions and current practice of the design of applications packages. It proposes that the present form of packages is fundamentally contradictory to the notion of packaging itself. This is because as an indivisible ready-to-implement solution, current package architecture displays the following major disadvantages. First, it creates problems as a result of user-package interactions, in which the designer tries to mould all potential individual users, no matter how diverse they are, into one model. This is worsened by the minute provision, if any, of important properties such as flexibility, independence and impartiality. Second, it displays rigid structure that reduces the variety and/or multi-use of the component parts of such a package. Third, it dictates specific hardware and software configurations which probably results in reducing the number of degrees of freedom of its user. Fourth, it increases the dependence of its user upon its supplier through inadequate documentation and understanding of the package. Fifth, it tends to cause a degeneration of the expertise of design of the data processing practitioners.

In view of this understanding an alternative methodological design framework which is both consistent with systems approach and the role of a package in its likely context is proposed. The proposition is based upon an extension of the identified concept of the hierarchy of holons* which facilitates the examination of the complex relationships of a package with its two principal environments. First, the user characteristics and his decision making practice and procedures; implying an examination of the user's M.I.S. network. Second, the software environment and its influence upon a package regarding support, control and operation of the package.

The framework is built gradually as discussion advances around the central theme of a compatible M.I.S., software and model design. This leads to the formation of the alternative package architecture that is based upon the design of a number of independent, self-contained small parts. Such is believed to constitute the nucleus around which not only packages can be more effectively designed, but is also applicable to many man-machine systems design.

* Definition: A holon is a self-contained, stable open sub-system.
More details in Chapter 2.

ACKNOWLEDGEMENTS

The author wishes to express his gratitude to Professor S.L. Cook for his interest and advice during the study.

He would like to express his deep and sincere appreciation to his supervisor Mr. J.C. Watt whose inspiration and guidance were invaluable to this work and for his diligent supervision and patience throughout the entire course of the author's postgraduate study.

Mrs. H. Watt's genuine support to the author and his family during the same period has been much appriciated.

In addition, he would like to thank all the companies visited and their staff for their co-operation and support.

Last, but not the least, his debts are specially due to his two daughters Reem and Mona who have been a great source of comfort., and to his wife Suzy, the author expresses his sincere gratitude for her continuous encouragement and support.

TABLE OF CONTENTS

Cont.

Cont.

Cont.

Cont.

TABLE OF FIGURES AND ILLUSTRATIONS

Figure

Cont.

Table

INTRODUCTION

This thesis is about the design of man-machine systems. It is particularly concerned with the role and use of data processing equipment in management information systems, and specifically with the problem of designing the man-machine interface in this context.

The philosophical and methodological emphasis of the research came about as the result of earlier efforts to conduct an empirical study of the design and implementation of commercial computer applications packages (1). Preliminary work in this direction (2) showed that there was a lack of fundamental conceptual frames of reference, and hence it appeared difficult, if not impossible, to conduct a meaningful empirical study. Concepts and theories play an important role in laying down the basis for a programme of analysis and empirical examination. It appeared that there was a serious mismatch between the relatively important practical roles packages play in business, and the very low interest directed towards the development of rigorous design theories and

---

(1) An application package can be defined as a set of inter-related programmes or modules - generally sequences of instructions. It is generally designed and planned so that when operational it may form a part of a system. It should be capable of handling similar - but not identical sets of input to produce required sets of output needed to perform one or more functions within different organisations in order to be sold commercially.

(2) Cf. Appendices 1 and 2.

concepts for this expanding area of systems application. This was also confirmed by dissatisfaction expressed by many users (during preliminary explorations for the empirical study) with some of the characteristics of current package architecture regarding intelligibility, modifiability, adaptability, etc. Discussions with analysts and designers of users computer systems exposed, in varying degrees, confused understanding and/or misuse of systems concepts and terms. Some analysts justified the acceptance of inadequate packages on the grounds of not being able to design their own, and this seemed to be a symptom of serious underlying problems. Many analysts appeared to have a deep involvement with computing technicalities at the expense of identification of users needs, and this seemed likely to cause problems of miscommunication.

It was therefore decided to redirect the main effort towards investigating the philosophical and theoretical bases of man-machine interface design. In other words, the main theme instead of being primarily concerned with package design has become the more general one of interface design, but with particular reference to packages when practical illustration or demonstration is called for.

It is rather important to mention that this study treats a package or a computer programme as a system specifically when it is loaded into the central processing unit and other relevant peripherals. Accordingly, the hardware is considered as an essential environment for the package as a system. Without hardware the package will lose its purpose and hence its systems

nature, as any other open system will do when separated from its environment.

The thesis therefore attempts to construct a systems based framework for the design of man-machine interfaces. It concentrates on:-

i) The modelling of systems and their conceptualisation. This involves the study of the relation between model building and problem formulation. This implies an examination of whether a particular design method when used within a certain philosophical context is capable of producing models that reflect the fundamental characteristics of the modelled entities. This is a question of methodology and definitions of systems characteristics, e.g. structure, behaviour, environmental influence etc. This is subsequently referred to as the conceptual environment.

ii) The role, influence and impact of machines, e.g. computers, on their parent systems, e.g. organisations and vice versa. This leads to the question, what is mechanisable and to what extent? This is referred to below as the technical environment.

iii) The development of a design method that is capable of yielding designs having properties to match the characteristics of their parent systems. This is referred to subsequently as the organisational environment.

It is argued that any attempt to build a conceptual framework for use by practitioners to aid them in the design of man-machine systems, should try to satisfy four requirements. First, the language and concepts with which the framework is built must be internally consistent. Secondly, as much as possible of the language and concepts must be drawn from those already recognised and in use in the systems field, in order to minimise difficulties for potential users of the framework. Thirdly, the use made of these terms and concepts should as far as possible be compatible with present accepted uses and interpretations, or confusion will be caused. Fourthly, there should be some demonstration that the framework can be helpful in describing and approaching a real example of man-machine interface design.

The way in which the thesis attempts to deal with each of the four requirements is as follows:

1) Internal Self-Consistency of terminology and concepts.

Consistent usage of terminology and concepts is particularly critical in the systems field because it is a young field without a well-established and coherent language system and because it draws its ideas from such a wide range of disciplines, each within its own different languages and idea systems. One way of insuring internal self-consistency in building a new framework is to rely largely or entirely on creating and defining one's own

language and concepts from a few very fundamental ideas. (To some extent this is a path followed by Ackoff (3). That approach may however discourage users and following researchers, who may be reluctant to take on the onerous task of learning the new language, and is not used here as will be clear from the statement of requirement 2. Concepts will be found to have been widely drawn from the systems literature. It is hoped that the structure of the thesis and the way the terms are used within it helps to show that the usage is internally consistent; or failing that, that the clarity with which they are displayed enables the reader to detect any inconsistencies which will need treatment in further research.

2. Use of well established terminology and concepts.

It will be very clear from the thesis that it draws on language and ideas already used by others. References are liberally given to allow sources to be checked. The aim has been in each case to use the term in the same basic way as generally accepted usage (or sometimes, the originator), and this may in most cases be readily checked. However it must be stated that the use and meaning of a term or concept in systems literature is not normally restricted to either a unique use or meaning. A case in mind is the philosophical and scientific arguments about the meaning of terms such as information, operation, activity, etc. This is also the case concerning what is a system? Is it right to refer to any entity, once it is made up of more than one part, as a system? Can it be treated as

---

(3) Ackoff, R.L., "Towards a System of Systems Concepts", Management Science, Series A - Theory, Vol.17, No.11, July, 1971, pp.661-671.

such without causing many difficulties? Such ambiguity of systems terms and concepts has been identified as still a major limitation in this field and attempts are currently being made (4) to overcome this problem.

3. Broader Implications and Interpretations of Terminology and Concepts.

Although this thesis attempts to use terms and concepts to carry the same basic meaning as in general usage, it does not follow that the broader implications and interpretations of the term in general usage (or by the originator) are accepted here. Where this difference is seen as important, it is discussed in the text. However, as will be indicated below, this is by no means covered exhaustively in the thesis and there is much scope for further exploration and development. It is hoped the references given will provide an adequate base for such further work.

4. Applicability of the Framework to practical design problems.

To show the relevance of the present philosophical and theoretical treatment of interface design, a real situation is considered throughout. This is the design problem of applications packages. This also may show that a reasonable understanding of systems and other related concepts can be useful in tackling ill-structured problems by developing new ideas.

---

(4) For example see Young, O.R., "A Survey of General Systems Theory", General Systems, Vol.IX, 1964, p.61.
Also see Ackoff, "Towards a System of Systems Concepts", op.cit.

It is imperative however to point out that the proposed framework has not been experimentally checked. This again is discussed further below.

A Comment on the above requirements

The four requirements dealt with above were put forward as being feasible for a research study of this nature. Ideally they might well have been extended further. In particular, for example, a more exhaustive search of the laternative interpretations of each of the terms and concepts in the literature might have been made and documented, any incompatibilities discovered being pursued and in some way resolved. Similarly, applicability of the framework in practice would mainly be tested in relation to a wide range of practical examples of interface design and perhaps shown to have weaknesses as well as strengths.

This thesis does not claim to have gone that far. The building of a conceptual framework of this kind, it is argued, is a very large task that cannot be completed in one PhD thesis. The framework put forward here is seen as a testable hypothesis, partially tested in the thesis but needing much further testing by researchers who may follow on. Thus the thesis sets out to show also that the effort involved in such further testing is worthwhile.

In other words, the framework put forward is not supposed to be without defects or shortcomings. It is recognised that it

does not resolve all problems. Such a proposition would in fact be anti-thetical and inconsistent with the main idea of the thesis which is that no definition or solution or design is perfect or complete. Consequently eventual application of the framework can result in discovering new problems due to its inevitable incompleteness.

Not all difficulties in application, of course, will be due to defects in the conceptual framework; there are inevitable difficulties in the systems approach itself, due to the inability to formulate perfect or complete definitions of real phenomena. This, in turn, leads to inaccurate definitions, boundaries, elements and relationships.

THE STRUCTURE OF THE FRAMEWORK

In the light of the foregoing, this thesis attempts to construct the design framework from the viewpoint of the three environments, vis the conceptual environment, the organisational environment and the technical environment. The characteristics of each are summarised in the following.

1. The Conceptual Environment of the Framework

A mental image of a real phenomenon or situation is a form of a model as developed by the person involved, e.g. the designer. The conception of the model may involve the activation of a set of preceding activities like the identification of the phenomenon, its description, the selection of the appropriate method that can handle it, and so on. However, the effective use of such activities is considered to be constrained by the designer's own set of conceptual frames of reference and their natures. Thus Chapter One is devoted

to discussing the complex topic of change and its likely effects on models, and how such models should be designed in order to cope with a changing environment. Accepting that change is inevitable, it is argued that the internal structure of the model should be distinguished by flexibility to enable it to meet the requirements of change. Other design criteria, namely content and bias, and their effect on the flexibility of the model are examined. This has implied a brief study of interdependence between the process of defining, methods and concepts. Within these outlines the current form of a package as a ready-to-implement solution for an identified problematic situation is examined.

Chapter Two has two purposes. One is to identify the basic assumptions behind the concept of packaging. The second purpose is involved with the problem of understanding a given package or model under normal operational conditions. As for the first purpose, the thesis hypothesizes that the basic philosophy behind the concept of packages is that of analogy. This hypothesis is built on two assumptions which may form the foundation of packaging: a) the feasbility of designing a specific model that can correspond to procedures and fulfil needs of all members that may be included in an arbitrarily chosen set of N potential users whose procedures are assumed to have some similar properties; b) certain needs and procedures of any potential user within the set are very similar or even identical to those of any other user within the same arbitrary set of users. Consequently, only those relevant aspects of the concept of analogy are considered. Concerning the question of understanding a given package or model the concept of

black-box is reviewed. This involves a discussion about the concept's characteristics and limitations, and whether the concept on its own can yield a unique determination of the structural pattern of a given package. This treatment derives much of its importance from the idea that many analysts and designers rely heavily on this concept, thinking that it can resolve much of their difficulties. This has brought to the discussion the notion of structural organization to identify some means through which the given package or model can be analyzed. Again, this has suggested the study of the concept of hierarchy as a possible conceptual means. The Chapter ends by introducing the concept of holons and their organization; Koestler defines a holon as a self-contained, stable open-subsystem (5). These holons provide: 1) a satisfactory conceptual definition or description for a part or module; 2) a flexible overall structural pattern of a package.

2. The Organisational Environment of the Framework

This is the environment within which the purpose and usefulness or otherwise of the model as an interface between the user and the computer can be determined. Unless its users are satisfied with its performance the model may lose its purpose. Thus a reliable conception of the nature of the eventual interplay between the model and its user is of absolute design importance.

---

(5) Koestler, A., "The Tree and the Candle", In, W. Gray, and, N.D. Rizzo, (Eds.), "Unity Through Diversity", (Gordon and Breach, Sci. Pub., N.Y., 1973), Part I, pp. 287-314.

The scope of this environment covers two main areas. One is the study of the likely interaction between a user and a model that may be allowed through the different versions of the concept of management information systems, M.I.S.'s. Chapter Three examines typical definitions of M.I.S.'s as proposed by designers. In this Chapter an M.I.S. is treated as the user-oriented environment of a package. The decision whether to buy a ready-to-cimplement model or package or attempt to make one's own model is also investigated. The decision is analyzed to identify and examine the likely consequences of opting for and/or relying on ready-to-implement models in a comprehensive manner as presented in the form of packages. The effect of this model form on the development and versatility of users' data processing and design expertise and skills is also considered.

The second area covered by the organizational environment is that concerning the identification of users' characteristics. It is argued in this thesis that without adequate identification of users' characteristics it may become absolutely difficult to understand or appreciate users' needs. This lack of understanding of users' requirements may lead to the design of the wrong models. At best this may lead to the design of models that may not fit in the structures of their users. This is dealt with in Chapter Seven by attempting to test the compatability of the design requirements - drawn so far - with the needs of decision making.

3. The Technical Environment of the Framework

This is the environment which constrains or limits the scope of the model. It exercises this role by dictating the content of the model in the form of specific features. Content is therefore defined to refer to: specifications, i.e. the model's language, size, structure and organization; scope, i.e. acceptable input and the possible output; generality, i.e. whether it corresponds to a unique real situation with reference to a specific user; etc. Thus the relationship between the intended purpose of the model and its content on one hand, and that between content and the design requirements, e.g. flexibility, on the other, should all be carefully examined. Chapter Four discusses the relationship between software and packages and studies the impact of the latter on the former. Also the manufacturers' desire to extend their influence to control users' freedom of choice is examined. Methods to prevent this from happening are suggested.

Chapter Six attempts to define a package or model in terms of its communication function. This is considered to be compatible with the requirements of the other two environments as well as those of the concept of holons. Also this Chapter analyses the current state of modularity and proposes the use of the technique of structured programming.

Chapter Five considers a typical package, namely CLASS, and discusses it along the lines put forward in the above three environments.

The thesis concludes by rejecting the present package architecture on the grounds of three main reasons. The first reason is concerned with the design properties. In this respect current package architecture makes it very difficult, if not impossible, for users' analysts to modify or adapt such packages to fit in their environment. This is attributed to the ad hoc, pseudo-modular, over-complicated and inadequately documented and tested designs. As pseudo-modular, the constituent parts of the package become more dependent upon each other and thus increasingly inseparable. In view of these qualities and in addition to the content implanted by the manufacturer the package becomes extremely biased to and dependent on its originator's backing. The second reason is involved with rigidity. Current architecture leads to duplication of efforts, due to the inseparability of the parts of the package, and this is anti-thetical to the fundamental idea behind packaging. The third reason for rejecting present package architecture is the effect of package rigidity on its user by reducing his number of degrees of freedom, which can lead to communication as well as psychological problems. Also for these very reasons users' data processing design skills and expertise degenerate.

The thesis concludes by proposing the design of small and independent modules as an alternative design approach. It is concluded that through this new architecture packages may be freed from most of their deficiencies.

However, the usefulness of this study will not be fully realised unless further research is carried out to construct more precise propositions which are testable. This is likely to have two facets. One is to assess the practical effectiveness of the proposed package architecture. The second is more general than this and may involve testing of the philosophy of the framework in relation to a wide range of actual interface problems. Also it may be possible to extend the scope of the former facet to examine the reliability of the package concept in general which is not questioned in this thesis.

CHAPTER 1

PROBLEM SOLVING

The purpose of this chapter is to study the essential design requirements to cope effectively with change. These requirements have two main facets. One is that covering the designer's own conception of change in general and more specifically in relation to his, or his clients, needs. The other is that involving the designer's ability to discriminate and select methods to design models to satisfy the identified needs. Consequently, by treating a package as a ready-to-implement model or set of definitions, its design requirements are investigated. In this context, it is argued that the package is bound to reflect the conceptual frames of reference of its designer. Also the impact of design methods on package properties such as flexibility is considered. Thus the emphasis of the discussion is centred around the interdependence of definitions, methods and solutions. First, the process of defining is briefly outlined and some of its characteristics are pin-pointed. This is followed by an overview of methodology. This is because methodology can be regarded as the source of frameworks within which concepts and techniques may be used. In this respect two methods are reviewed, viz. the elementistic and the systems. Finally the roles of analysis and synthesis and the nature of problems are all examined from the viewpoint of systems thinking.

THE PROCESS OF DEFINING

A definition is essentially a statement describing an abstract entity or a material object, or any combination of both. Such a statement may take numerous forms, for example mathematical expressions and verbal descriptions. Whatever form a definition may take, it is regarded to be imperfect and ultimately imprecise (1). Consequently definitions are constantly reconsidered and reformulated. As a new definition is formulated often it is accompanied with more dimensions and/or new insights (2). Such new insights open new horizons and make it possible to redeem the definition in question to unveil new aspects. It has been said that defining is circular (3) because definitions do not emerge from vacuum, rather they are founded in terms of some known concepts. Accordingly definitions show either implicitly or explicitly varying amounts of subjectivity. This remains to be true as long as they contain such an element of selection, i.e., personal judgment; they may always be liable to review.

---

(1) Westaway, F.W., "Scientific Method", (Hillman-Curl Pub, N.Y., 1937), p.19.

(2) Popper, K.R., "The Logic of Scientific Discovery", (Hutchinson & Co Ltd, London, 1972), p.54.

(3) Ackoff, R.L., "Scientific Method- Optimizing Applied Research Decisions", (John Wiley & Sons, N.Y., 1968), p.170.

CIRCULARITY OF DEFINING AND RELATIVITY OF DEFINITIONS

Revision of definitions may be initiated in the light of new propositions, observations, or needs. Whatever reason that there may be behind it, constant reconsideration helps in formulating a more precise description of the phenomena in question (4). This argument as it applies to definitions in general is particularly applicable to computer application programmes. In this respect, the programme is considered to comprise a set of definitions which is constructed and designed primarily to support or perform a certain function. The set of definitons depends to a large extent upon the knowledge of the definer (5). The interdependence between the definer and the definition is illustrated by Russell when he says:

> "Common words, even proper names, are usually descriptions. That is to say, the thought in the mind of a person using a proper name correctly can generally only be expressed explicitly if we replace the proper name by a description. Moreover, the description required to express the thought will vary for different people, or for the same person at different times.* The only thing constant ... is the object to which the name applies" (6).

Therefore the problem is further compounded by trying to define concepts which are not objects and which are not constants. This understanding of change and the variability and relativistic characteristics of definitions implies the selection of a certain design approach which is capable of producing designs which are easy to change and modify.

---

* Emphasis is not in the original.

(4) Whitehead, A.N., "Process and Reality", (Indiana Univ. Press, 1971).

(5) Russell, B., "The Problems of Philosophy", (Oxford Univ. Press, London, 1973), pp. 25-32.

(6) Ibid.

Thus adequate defining should not be seen as an easy, ad hoc task to perform. Indeed it embodies the whole process of inquiry. One of its basic requirements is clear understanding of the body of concepts that may form the conceptual framework of designers in order to use them appropriately.

Methodology

Good method is a prerequisite for good design. Methodology is the science which studies the various problem solving and design methods that are generated by the various philosophical approaches (7). Thus a clear understanding of an approach and the appreciation of its limitations and usefulness are primary conditions for the appropriate application of its method. This, in turn, can lead to deeper consideration of design properties such as consistency, discrimination and effectiveness.

The current design practices of computer systems analysts/ designers reflect: a serious misconception of some of the basic design methods; a confused understanding of the conventional, i.e.

---

(7) Ackoff, "Scientific Method", op.cit., p.6.

elementistic, and systematic approaches (8). This criticism is directed towards the manner in which methods are confused rather than the methods themselves. Thus, considering the present problem, i.e. package design, it may be appropriate to emphasise from the beginning that the usefulness of the elementalistic approach in examining parts can be made more effective when it is used within the context of the systemic approach that caters for wholeness which may lead to the identification and hence the investigation of the proper parts. Consequently, the problem of package design as conceived here is not only that of writing a number of instructions but also the studying of how such instructions are related to its designer, and its environment, e.g. the organization needs of its potential user, other programmes, to hardware, etc.

As was implied earlier definitions and designs evolve. Some lose their purpose or relative importance and so attract little, if any, attention and hence fade away. Others are maintained and elaborated to cope with change. Such an evolutionary process may involve conceptualization of new relationships. These relationships can indicate new dimensions

---

(8) It may suffice at this stage of the discussion to mention that most users of packages complain from the lack of adequate documentation of their packages irrespective of the sources of such packages. This indicates besides other things a comparable lack of appreciation of the consequences of the inevitable interaction that takes place between a package and its invironment.

and/or similarities between two or more phenomena previously considered unrelated. Consequently, in the light of such novelty, the original statement or design undergoes certain changes in order to be adapted to suit the new requirements. As a result, and parallel to this evolution, methods, techniques and tools have been continuously worked upon to render better understanding and thus descriptions/definitions/designs of the problems concerned. Therefore, the creation, development and formulation of the science of methods can also be traced back in exactly the same manner human societies and evolution do. Thus in the following two modes, i.e. the elementalistic and systemic, are briefly reviewed because of their identified influence upon today's programme design characteristics.

1. The Elementalistic Conception

The application of this approach involves the breaking down of the investigated entity (whether material or conceptual) into its basic elements. Thus the emphasis of the study is bound to be placed upon parts rather than wholes. The main proposition is that examination and understanding of parts permit an understanding of the whole. This may be exemplified by the following Descartes' rule:

> "If we are to understand a problem perfectly, we must free it from any superfluous conceptions, reduce it to the simplest terms, and by a process of enumeration, split it up into the smallest possible parts" * (9).

---

* Emphasis is not in the original.

(9) Anscombe, E., Geach, P.T., (trans.), "Descartes Philosophical Writings", (Thomas Nelson & Son Ltd, London, 1971), p.179.

Possible interpretations of this rule may include:

1. the characteristics and properties of the whole can be perfectly inferred from those of the individual parts;

2. interactions between the parts, i.e. relationships, are less important than the parts themselves. Some interpreters go farther than that by suggesting that this and other similar rules assume the non-existence of part interactions (10);

3. the whole is the sum of its parts. Thus an entity can be resolved into, and hence reconstituted from, the parts put together (11).

However, in some areas of biology and social organization the complete reliance on this approach as a source of understanding has been regarded as inadequate (12). It is argued that this inadequacy is due to the observed complexities, e.g. strong part interactions, of biological and social phenomena. Even in physics the aim to identify and define the simplest unit of matter

---

(10) Bertalanffy, von L., "General Systems Theory: Foundations, Development, Applications", (Allen Lane, The Penguin Press, London, 1971), pp. 16-17.

(11) Bertalanffy, von L., "Problems of Life", (Watts & Co., London, 1952); Churchman, C.W., "The Design of Inquiring Systems", (Basic Books Inc., London, 1971).

(12) Ibid, p.5.; Ashby, W.R., "General Systems Theory as a New Discipline", General Systems, 1958, Vol.3, pp. 1-6.

is far from being reach (13). It has also been suggested by Weaver that being part-oriented, the elementalistic, or classical approach as he calls it, can lead to fragmentation or unorganized complexity (14).

---

(13) In physics as well as other hard sciences, laws and definitions are liable to change, then it can be considered as illogical to design a man-machine system such as a package without being changeable.

(14) Weaver, W., "Science and Complexity", American Scientists, 1948, 36, pp. 536-544. Bertalanffy, "General System Theory", op.cit., p.33.

2. The Systems Conception

The systems' view of the world is that of a complex whole. Conceptually this is based upon two highly related notions. The first is that of the systems' concept. The other is that of analogy and isomorphism. The term system is defined to be a complex of elements in mutual interaction (15). Isomorphism is that of one-to-one correspondence between two or more different phenomena (16). Therein, isomorphism is considered to be a sub-class or special case of the wider encompassing area of analogy (17).

It has been suggested that the two concepts, i.e. systems and analogy, together form the pillars of the contemporary general systems theory and systems research in general (18). Also the concept of systems is often associated with other concepts such as hierarchy, differentiation, purposiveness, etc. Each of these concepts is used to provide means by which an entity can be conceived of and examined. All of these concepts must be treated with caution, since they are not adequately rigorous and certainly are not relateable to the physical world in the same way as corresponding concepts in, say, physics. Thus the concept of systems is used only as a conceptual framework (the systems approach) that can handle complexity by considering entities as wholes (19). The approach is not fully satisfactory but what are the other alternatives? Thus, the impact of systems thinking upon

---

(15) Bertalanffy, "Problems of Life", op.cit.

(16) Rapoport, A., "The Uses of Mathematical Isomorphism In General Systems Theory", In G.J. Klir, (Ed.), "Trends in General Systems Theory", (J. Wiley, N.Y., 1972), pp. 42-77.

(17) Bunge, M., "Analogy, Simulation, Representation", General Systems, 1970, Vol.15, pp. 27-34.

(18) Bertalanffy, von L., "The History and Status of General Systems Theory", Academy of Management Journal, 1972, pp.407-426; Ashby, W.R., "General Systems Theory as a New Discipline", General Systems, 1958, Vol.3, pp. 1-6.

(19) Weaver, "Science and Complexity", op.cit.

the development of model building should be found in every discipline. For example, analogy between organisms and human groups or sub-groups, organizations, can be drawn. Various other analogues of organizations are nowadays available. Thus, there is Beer's Cybernetic model (20); Churchman's communication model (21); March and Simon theory of organizations (22); Forrester's industrial dynamics (23). In that sense, organizations are regarded to constitute open systems (24). The main characteristic of which is dynamics, i.e. continuous interaction with the environment. This enables the organization to maintain, at least its survival, i.e. maintains its steady state. Such indicates teleological or purposeful behaviour.

Thus considered as interacting systems, man-machine complexes are understandably permitted to exhibit regulative behaviour and consequently are expected to show variable degrees of adaptability in face of disturbance.

---

(20) Beer, S., "Decision and Control", (J. Wiley, London, 1966), Part II; "Cybernetics and Management", (English Univ. Press, London, 1959.

(21) Churchman, C.W.; Ackoff, R.L.; Arnoff, E.L., "Analysis of the Organization", In Litterer, J.A. (Ed.), "Organizations: Systems, Control and Adaptation", Vol. II, (J. Wiley, N.Y., 1969), pp.274-286.

(22) March, J.G.; Simon, H.A., "Organizations", (J. Wiley, N.Y., 1958).

(23) Forrester, J.W., "Industrial Dynamics", (M.I.T. Press, Cambridge, Mass., 1961).

(24) Katz, D.; Kahn, R.L., "Common Characteristics of Open Systems", In Emery, F.E., (Ed.), "Systems Thinking", (Penguin Books Ltd., 1970), pp.86-104.

The development of the systems approach has induced a change in the attitude towards structure. The mutual dependence between structure, behaviour and genesis (i.e. origin, mode of formation or generation) has been clearly identified (25). Consequently, the idea maintained here is that design processes which act as genesis determine structure; structure puts the limits and scopes of behaviour which, in turn, may introduce changes in structure. According to this understanding flexible structuring of the parts that constitute the system gives it better chances of survival in a changing environment.

Furthermore, the dynamic approach towards structure has shown some significant design implications; of these, it is suggested that hierarchical ordering of structure can present an effective way of coping with complexity (26). For the purposes of stability, a hierarchical structure is considered to be governed by two modes of interactions. These are the horizontal interactions that connect within each one level its various parts so as to maintain the coherence of the whole; vertical interactions between different levels are necessary for control purposes.

---

(25) Gerard, R.W. "Units and Concepts of Biology", Behavioural Sci., 1958, 3, pp. 197-206.

(26) Simon, H.A. "The architecture of Complexity", In Litterer, J.A., op.cit., pp. 98-114.

The resultant dynamic interaction precludes the delineation of a clear sharp line, to wit a boundary, which discerns either horizontally a sub-system from another, or vertically one level from another. Instead, fuzzy, i.e. greyish area, determination of boundaries may be regarded as sufficient (27).

SIMPLIFICATION AND APPROXIMATION

Simplification and approximation constitute a major property of systems thinking and design because under no circumstance, exact measurements, even of physical forms, can be assured. This is mainly due to the infinite number of variables associated with any particular event (28). It can be maintained that considering structures as hierarchies implicitly assumes an element of simplification. Moreover, it has been proposed that the notions of hierarchy and simplification are complements (29). As time lapses the complexity of a system with certain levels of hierarchy may increase according to the intensity of the interaction

---

(27) Stratton, A., "Total Systems Analysis", First Internat. Research Conf. On O.R., 1973, Univ. of Sussex.

(28) Ashby, W.R. "Systems and Their Informational Measures", In G.J. Klir, (Ed.), op.cit., pp.78-97.

(29) Pattee, H.H., "The Evolution of Self-Simplifying Systems", In E. Laszlo, (Ed.), "Relevance of General Systems Theory", (George Braziller, N.Y., 1972).

between the system and its environment. If the system succeeds in maintaining this interaction, its complexity, which results from growth or change, may reach a point where the system either moves up to a new hierarchical level capable of handling such complexity, or it may collapse under its own weight.

Simplification is, therefore, quite a useful concept in systems research and study; on one hand without simplification or approximation, no model, including computer programmes, may ever be conceptualized. On the other hand, the act of modelling of phenomena stems mainly from attempting to perceive analogies, recognition that something is like something else. The ultimate purpose of these models and/or analogies is to attain clear understanding of the considered phenomena. Nonetheless, to devise an acceptable analogue is not in itself attainable for every phenomenon. Indeed, perfect analogues may not exist in the real world. It has been suggested that the closer a particular model is related to human behaviour, e.g. management information model, the farther it moves away from exact analogies (30).

However, over-simplification can lead to unforeseen consequences. To illustrate, one of the fundamental premises, upon which current design practices of computer application packages are based, is the assumed strong similarity, even isomorphism, between

---

(30) Rapoport, A., "The Search for Simplicity", In E. Laszlo, (Ed.), op.cit., pp.15-41.

different companies. This is a misunderstanding of systems thinking which can lead to misuse of concepts, e.g. analogy, and has been guarded against by maintaining that models, regardless of their origins, are neither exhaustive, unique, nor certain (31). The validity of such characteristics can be examined through the concept of information. Characteristically information is said to be an incomplete concept because it depends completely upon the interpretative system of its recipient which alone can judge whether the information has meaning (32). More important, this interpretative system changes continuously with time (33). According to this understanding not only companies differ from each other, more significantly an individual company undergoes continuous change.

SOME DESIGN IMPLICATIONS

Whatever case it is, models may, and in fact are, grouped into classes according to noted similarities with respect to their properties (34). From the viewpoint of computer application packages, the criterion which discerns a general or multi-purpose model from a specific or special-purpose one is here considered to have a three-fold nature, i.e. content; bias; flexibility.

---

(31) Beer, "Decision and Control", op.cit., p. 104.
Bertalanffy, "General Systems Theory", op.cit., p. 100.

(32) Vickers, G., Sir, "A Classification of Systems", General Systems, 1970, Vol.15, pp. 3-6.

(33) Ibid. This is to be covered more fully in Chapters 3, 6 and 7.

(34) Ackoff, R.L.; Sasieni, M.W., "Fundamentals of Operations Research", (J. Wiley, N.Y., 1968).

Content of a model/programme: It can be described in terms of specificity, precision, correspondence/connection/representation with specific/concrete/unique/real situation/company. That is the stronger the correspondence between a model/programme and a real situation, the more content the model is supposed to have. Also the term content is chosen for its generality, i.e. content may indicate or refer to specificity, or precision, etc., but the reverse does not necessarily hold.

Bias: This term is used to indicate preference or inclination towards something which is not necessarily justifiable from the viewpoint of rationality. This is not to say that rationality itself is without limits, this is discussed throughout limits, this is discussed throughout the thesis.

Flexibility: This may refer to the model's modifiability and adaptability

It has been suggested that as content increases generality may decrease (35). Both content and generality are considered to be two important criteria that determine the scope of a package. This can be explained as follows: A programme can be considered as being general or specific depending on whether it is designed to meet the requirements of several users, or it is designed for a specific user respectively. Also a programme can be distinguished as being general or specific by considering the number of functions which it can perform.

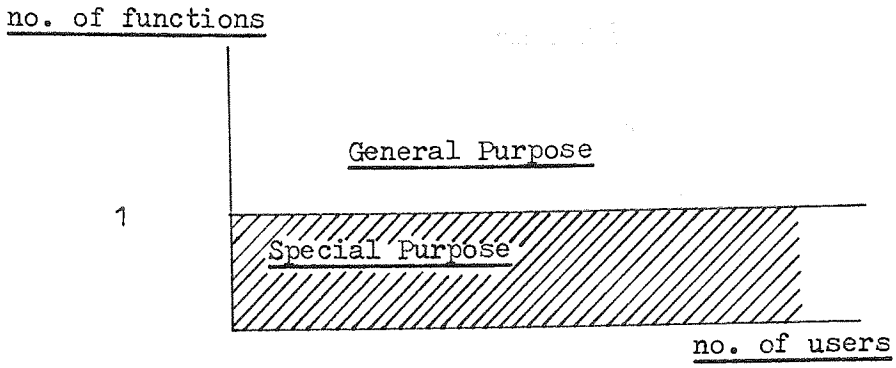
1. According to the Number of Functions (36)

When the package is either externally or internally supplied, then it is general purpose if it covers more than one functional area and irrespective of the number of users such as management information, production control, Pert/cost, etc. The package is special purpose if it covers only one functional area such as scheduling, inventory control, etc. This is shown in Figure (1).

---

(35) Boulding, K.E., "General Systems Theory - The Skeleton of Science", In W. Buckley, (Ed.), "Modern Systems Research for the Behavioural Scientist", (Aldine Pub. Co., Chicago, 1968), pp. 3-10.

(36) The scope of the function is immaterial. Therefore the definition is applicable to modules irrespective of their size or level. In this case the only critical factor is the number of, say, operations performed by the module.

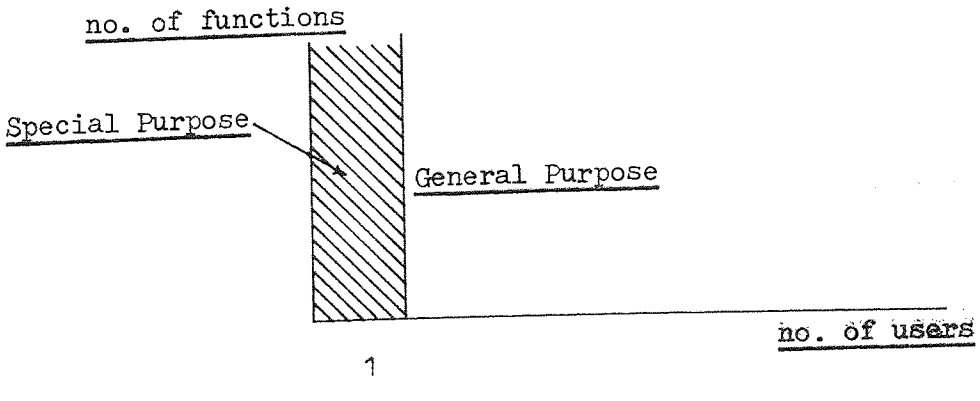


Emphasis upon no. of functions

Figure (1)

2. According to the Number of Users

When the package is externally supplied, then it is general purpose since it implies an applicability to more than one company and irrespective of the number of its functions. Consequently an internally supplied package is considered as special purpose since its applicability is restricted to only one user. This is shown in Figure (2).

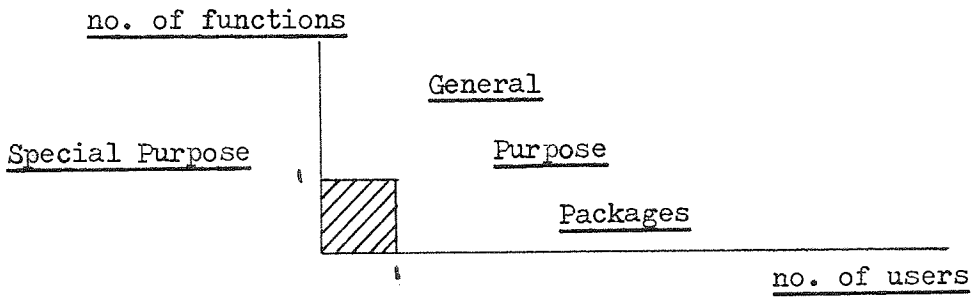


Emphasis upon no. of users

(Figure (2)

3. The Two Dimensions are Considered

In this general case when Figures (1) and (2) are super-imposed upon each other all packages become general purpose except that which is internally supplied to perform a specific function, as shown in Figure (3).



Superposition

Figure (3)

This also explains the relationships between content, generality and specificity. When a package is highly specialized it tends to be highly specified since it is designed to perform a specific function. However, its range of application depends mainly on the amount of content it may contain. Content imposes constraints which may or may not be compatible with the user's own structure. Thus a general property which distinguishes a general purpose package is contentlessness (37). This property ascertains

---

(37) In that respect, the package resembles mathematical models; black-box concept; open-system scheme; even definitions of concepts such as information; system; etc. each is contentless thus it has an extensive applicability.

the right of the individual user to derive his own meaningful version of the package within its general framework. Therein, the problem becomes that of synthesizing or relating the package to his own procedures.

The second element in the triplet is bias which is related to content. Package design, like models, depends upon a number of factors. First, the familiarity of the package-designer with the real phenomena he needs to project. Second to this is the designer's conceptual background. Thirdly, the designer's objectives, i.e. why he is designing the package and what should it achieve. If the scope of the designer is highly specialized, this may lead to a significant reduction in the applicability of the package in question and the effort needed to off-set this quality may prove to be costly. Thus the bias of the package designer is inevitably going to be reflected in the package itself. That is, the more content a package may have, the more biased it may become.

Flexibility is the last aspect of the proposed triplet which differentiates between general and special-purpose packages. Once again, flexibility per se depends upon the preceeding two properties; the more content a package may have, the more rigid-structure it exhibits as content delimits, or perhaps reduces, the number of degrees of freedom built into the package. And as bias increases, so does rigidity proportionally. Thus flexible structure should be regarded as an indispensable charactertistic of general-purpose packages.

The conviction is, however, that a truly general-purpose package like that of a definition and that of a system of systems, is a theoretical gauging or datum point.

ANALYSIS, SYNTHESIS AND PROBLEMS

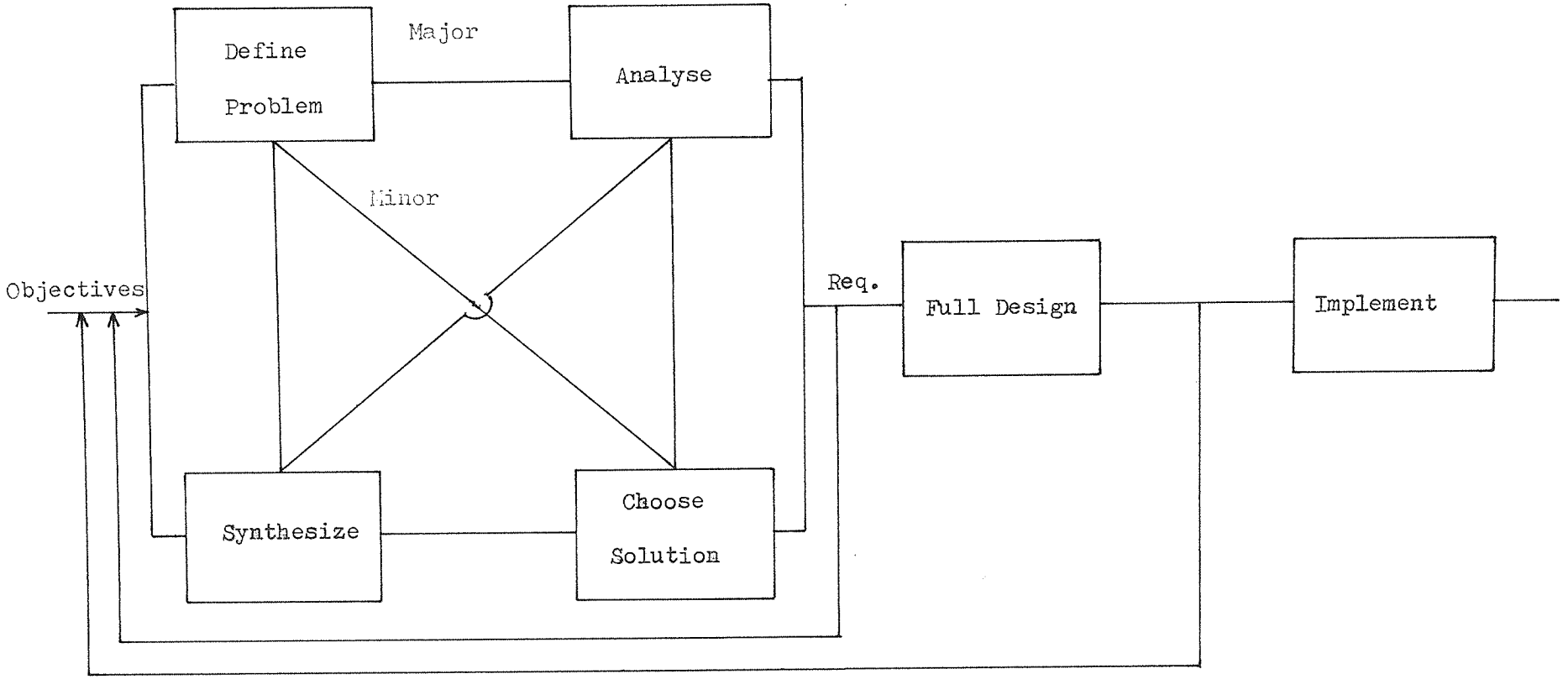
The process of systems design is similar to that of inquiry. That is, design should supposedly activate the processes of analysis and synthesis to define and solve identified problematic situations. The following brief discussion will basically involve an interpretation of the interdependence between (a) analysis and synthesis; (b) definition and solution.

ANALYSIS-SYNTHESIS RELATIONSHIP

Both analysis and synthesis are prerequisites of clear understanding (38). Analysis is needed to provide in-depth knowledge and detailed accounts of the individual components and elements; whereas synthesis integrates and unifies that knowledge to pin-point deficiencies and misunderstandings. Thus analysis and synthesis complement each other. It is identified here that a characteristic of an inquiring process is that of dynamic-interdependence between its components. This understanding is founded upon the realization that analysis stimulates synthesis and synthesis generates further analysis' requirements and so on ad infinitum, as shown in Figure (4).

---

(38) Kant, I., "The Critique of Pure Reason", (Great Books of the Western World, Vol.42, 1952), p.16.



Dynamic-Iterative Approach

Figure (4)

Furthermore, there is no clear sharp line which marks an end, say, of analysis, and the start of the other, i.e. synthesis. Instead it may be conceived of as being consisting of a fuzzy area. As a consequence, any separation between the two is considered to be arbitrary as regards both, spatial and temporal dimensions.

However, the artificial separation of the processes of analysis and synthesis is regarded as originating from the same sources as the classical elementalistic approach. Therein, the emphases are upon individual elements, which resulted in the development of the summative principle, viz. the whole is equal to the sum of its parts (39) which is apparently in disagreement with the Gestalt principle. The latter states that, the whole is greater than the sum of its parts. At any rate, it has been suggested that the summative principle is incapable of offering adequate descriptions of complex entities such as man-machine systems (40). This is because knowledge of parts may not necessarily provide sufficient knowledge about the whole. Even more, the pattern of behaviour of an individual component depends upon, and is affected by, the number of other individuals to which it is related. Thus, the outcome of synthesis may be different from the results obtained from analysis. This basic understanding of systems philosophy is not always fully appreciated by

---

(39) Bertalanffy, "Problems of Life", op.cit.

(40) _____, "General Systems Theory", op.cit., p.16.

existing computer systems designers and/or analysts. This is demonstrated by the huge resources they allocate for the development and testing of modules compared to these they allocate for integrating or linking modules to each other (41).

EXISTING ANALYSTS MISCONCEPTIONS

The main problem seems to be a lack of an explicit philosophical approach and the unthinking use of a mainly mechanistic approach derived from traditional education. Maslow considers concentration on the mechanistic/atomistic approach to be a very serious handicap. He says:

> "... of the two modes of thinking, the atomistic and the holistic, it is the holistic which is the sign of the healthy, self-actualizing ... Insistence on the atomistic mode is in itself a form of mild psychoneurosis" (42).

If the atomistic thinking is regarded as inappropriate approach, then confusing the two modes can be seen as a sign of still more serious deterioration: many analysts talk in systems terms but still behave mechanistically. The reason for emphasizing this point is the relative influence systems analysts/designers can exercise on their companies. Through their tasks they can make or break such companies. They are assumed to interact with all levels of management in order to identify and

---

(41) This is based upon attitudes expressed by working systems analysts/designers who occupy leading positions in a series of interviews. An illustration is given in Table 1, Chapter 3.

(42) Maslow, A.H., "Motivation and Personality", (Harper & Row, N.Y., 1970), Preface.

fulfil its various needs. In addition, they are supplied with computers which often constitute large investments, specially for smaller firms. Consequently, an analyst/designer's job can be understandably regarded at least not less demanding, if not more critical, than for example an engineering job. Yet most engineers are expected to undertake high technical training, e.g. university/ polytechnic degrees. Conversely, a large number of existing systems practitioners have only a formal training equivalent to the O-level. They are often conceptually constrained to the mode of thinking they used to utilize during their earlier schooling period. It can be maintained that this mode is anything but a systems thinking. As a result, systems analysts learn by experience or trial and error. Besides the costs that may be incurred, it has often been questioned whether experience on its own right does or does not suffice purposes of interpretation and understanding (43); particularly when uncertainty is encountered as situations are never exactly repeated (44). The disadvantage of relying heavily upon experience becomes obvious when analysts are challenged by novel situtations. In the absence of clear appreciation of the available methods and adequate frames of reference analysts often opt for ad hoc definitions and designs. The result is weak designs that may be held responsible for exceeding the estimated

---

(43) Churchman, "The Design of Inquiring Systems", op.cit.

(44) Beer, "Decision and Control", op.cit., pp.102-103.

costs (45). Thus weak designs coupled with high costs explain most of the mistrust, on the part of the users, of computers in general (46). It is believed here that the reason behind these drawbacks can be attributed to the misconception by systems analysts of the nature and dimensions of the problems they are supposed to tackle. In practice and in many computer-oriented systems analysis text-books the idea is that given a problematic situation, it can be defined with a bit of pressing. However, problematic situations are not identified entities, rather they are complexes of relationships. It is thus implied that problems portray systems of differing magnitudes and dimensions. The result of the process of isolating a certain problem depends upon parameters such as adequacy of frames of reference, judgment and value sets. When these parameters are either ignored or misconceived, problems may equally be misformulated and hence a serious conflict may develop ebtween the analyst and the user, say the manager. This difficult situation should be of great concern to the analyst because it is he who is going to design a computer solution for the problem as he sees it. If the design proves to be defective it is the analyst who is going to get the blame.

---

(45) Wolverton, R.W., "The Cost of Developing Large-Scale Software", I.E.E.E. Trans. On Computers, 1974, Vol. C-23, No.6, pp. 615-636.

(46) A significant number of users have asserted that analysts/designers frequently offer them what they do not want, on the assumption that they, the users, do not know much about computer systems techniques.

PROBLEM DEFINITION VERSUS PROBLEM SOLUTION

It follows that the term problem definition implicitly indicates a system definition and so it can be maintained that no problem definition is absolute. Because of the inevitable incompleteness in defining, a problem may never completely be solved; it can only be solved partially at a particular point in time with respect to some specific viewpoint.

Dewey has maintained that a problem well defined is half solved (47). It is proposed here that problem definition and its solution are in fact a whole. They are two interacting and inter-dependent components; a development or refinement in any one of them will certainly be reflected on the other resulting in still a further refinement in the first, and so on indefinitely (48). The underlying characteristic, however, is the perpetual group translation of the whole which is supposed to be generally directed towards the unattainable target which is the perfect definition and hence the perfect solution. Then and only then the recursive process ceases. That is, in the hypothetical case of a perfectly or completely defined problem, it is, then and only then, the problem is perfectly or completely solved. At the other extreme the least defined problem is the least solveable.

---

(47) Dewey, J., "Logic: The Theory of Inquiry", (Henry Holt & Co., N.Y., 1938), p.108.

(48) This reflection can be direct or indirect, conscious or unconscious, etc. Also the relationshio between problem definition and solution is very similar to that of analysis and synthesis.

CONCLUSIONS

The need to cope with change is reflected in the iterative nature of defining. This iteration also characterizes the systems approach because of the inevitable incompleteness of the systems and other related concepts. However, the systems approach is regarded as more suitable for man-machine systems design because it puts much emphasis on wholes rather than parts and also due to the complexity of such systems. Thus clearer understanding on the part of systems designers of systems philosophy will result in better appreciation of the effectiveness and limitations of the systems approach.

Appreciation of the effectiveness and limitations of each of the most widely known methods will be reflected in the adaptability and modifiability of a package. Also these qualities will affect the interactions between the structure of the package and its behaviour. The factors which affects this interaction as well as the scope of the package are: content; flexibility; bias. The relationships between these properties are:

The more content a package has,

the more rigid-structure its exhibits.

The more content a package has,

the more biased it may become.

Rigidity increases proportionally with bias.

Systems analysts better understanding of systems philosophy will be reflected in better systems design. Also their understanding of the relativistic nature of problems as well as the interdependence between the processes of analysis and synthesis will all be reflected in the analysts output.

CHAPTER 2

THEORETICAL DESIGN CONSTRUCTS FOR APPLICATIONS PACKAGES

This chapter has two main purposes. The first is to identify the basic assumptions behind the concept of packaging. This is expected to reveal the current designers' conceptions regarding packaging. Then the discussion sets out to examine the validity of the identified assumptions from the viewpoints of their practical and logical implications and limitations. The second purpose of the chapter is to investigate into the essential requirements that may be needed in carrying out satisfactory adaptation/maintenance processes. Adaptation is studied from the point of view of users' designers.

It is hypothesized here that the basic philosophy behind the design of a general purpose model or package is essentially analogical. This hypothesis is based on the following two assumptions: a) the feasibility of designing a specific model that can correspond to procedures and fulfil needs that may be included in an arbitrarily chosen, (by the designer), set of N potential users whose needs are assumed to have some similar properties; b) needs and procedures of any potential user within the set are very similar to, or even identical to those of any other user within the same arbitrary set of users.

Hence for the purposes of analysis and understanding only those relevant aspects of the concept of analogy are considered. The treatment involves a very limited use of set theory notation and preliminaries. The purpose of using set theorems is to facilitate the process of defining some secondary design criteria. These secondary properties, such as order - i.e. sequencing of instructions in case of packages and organization in general, are studied in relation to the previously suggested set, i.e. flexibility, independence and impartiality, in order to find out what are the requirements of the latter.

As for the second purpose, the discussion reviews briefly the concept of black-box. This is considered because of two main reasons. Firstly, its wide popularity among existing systems analysts/designers. Hence the review centres on the concept's fundamental characteristics, requirements, useability and limitations. Secondly, the discussion examines the solving power of the concept, i.e. whether the concept on its own right is sufficient, e.g., for a unique determination of the structural pattern and contents of a certain model/package. The treatment derives much of its importance from the fact that many analysts/designers rely heavily on this concept, hoping to resolve whatever problems they may have.

Also the problem of adapting a model/package to a specific user's context is studied. In other words, under

conditions of insufficient data pertaining to structure, how can an adaptor - i.e. user's analyst/designer - tackle this problem with any real confidence? This brings to the discussion the subject of structural organization in order to identify some means by which a given package/model can be analyzed and studied. To this effect the concept of hierarchy is considered as one of such conceptual means.

The concept of holons and their organization is examined regarding: 1) its ability to offer a satisfactory conceptual definition/description for a part/module whose properties are compatible with those of flexibility, independence and impartiality; 2) the flexibility of the overall structural pattern of the whole model/package and whether this concept does avoid some frequent drawbacks, such as rigidity, that may be associated with stereotyped hierarchical organization of entities.

BLACK-BOX AS A DESIGN CONCEPT

A black-box is that box whose internal structure is unknown and whose inputs and outputs are identifiable. For this reason the problem of black-box is frequently referred to as an input-output problem.

The term black-box is used to refer to two situations. The first is the use of the term in relation to the original package designer. This designer is supposed to know the algorithm and content which he is going to incorporate into his model. What is probably unknown to him is actual users' practices and thus can be treated as black-boxes. The second possible use of the term is that made by any user's analyst who tries to understand the model. In this case, the analyst is supposed to know the procedures and needs of his organization. What is probably unknown for him is the actual internal structure of the model itself. Thus he may consider the model as a black-box. However, as the problem investigated here is that of design, and since both cases involve design processes, then the term black-box as used here may refer to either case.

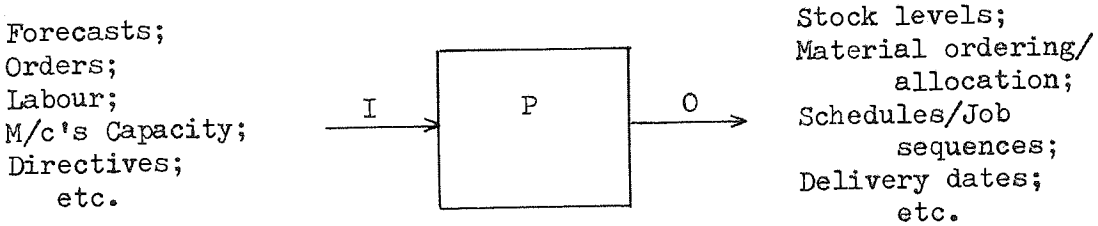
The different forms a black-box problem can assume are generally distinguished from each other according to the amount of information available at a given point in time pertaining to: (a) the set of the identifiable external quantities, e.g. input and output, environmental factors, etc.; (b) the properties of the contents of the box, e.g. its elements, components, couplings, etc., which decide the box's transfer function. Thus, as a consequence, black-box problems are associated with various amounts of uncertainty since full comprehensive knowledge about the box is unattainable (1).

---

(1) Bunge, M., "A General Black Box Theory", Phil. of Science, 1963, 3, pp. 346-358.

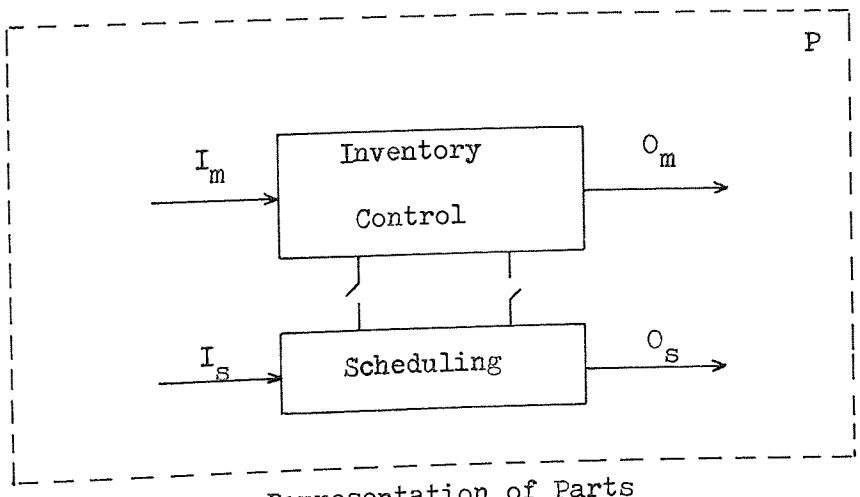
Uncertainty is, therefore, coupled to what is inside, i.e. the structure, rather than what is coming in and out of the box.

Thus any package or system can be represented by the use of blocks or boxes. The number and level of the boxes used to illustrate the package in question depend mainly upon the set of objectives, abilities and background of the designer. Hence at a higher level of design a production control package can for example be depicted as a single box as shown in Figure (5). At a lower level of design, i.e. more detailed, the same package can be viewed as consisting of a set of related boxes, each representing a certain aspect of the package, e.g. inventory control, scheduling, etc., as in Figure (6).



A Package as a black-box

Figure (5)



Representation of Parts

Figure (6)

As established earlier, the behaviour and structure of a package are interdependent. However, obtaining information mainly about that package's behaviour does not necessarily lead to the determination of the structural pattern that is responsible for such behaviour (2). Thus a characteristic property of black-box concept is that the only way the behaviour of a box can be approximately determined is by examining its past to identify a pattern which can be projected in the future (3). Such a process involves a significant amount of uncertainty since the behaviour itself continues to be unexplainable (4).

Consequently a black-box representation, on its own, of a package prevents, or at best, makes it extremely difficult to develop a clear systems understanding which emphasizes the identification of the various relationships that constitute the package (5). As a result, this necessitates that the primary objective of the designer/analyst is to transform the black-box into - a grey - if possible white-box. Thus the usefulness of the black-box concept to the designer must be confined to its simplifying power which transforms complicated events into figurative and relatively simple models.

---

(2) Ashby, W.R., "An Introduction to Cybernetics", (Methuen & Co Ltd, London, 1971), p.89.

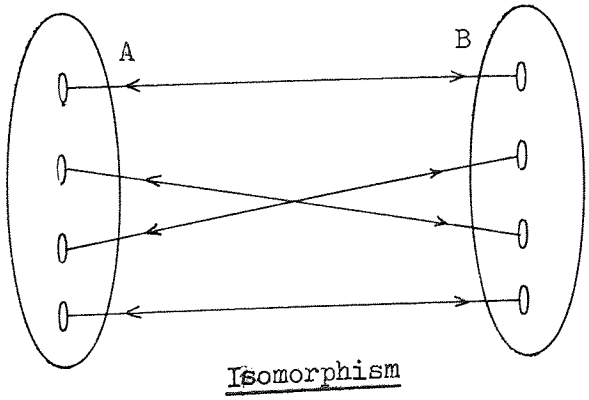
(3) Ibid, p.93.

(4) Vickers, "Classification of Systems", op.cit.

(5) This point is stressed here because of the prevailing attitude of the interviewed data processing managers and analysts which accepts packages as black-boxes due to the inaccessability of basic design and structure data.

AN OVERVIEW OF ANALOGY

Given a package in the form of a black-box, its internal structure can be delineated by applying certain analogical procedures. There must be two boxes before analogy can be started. Thus two black-boxes, say 'A' - manual activity - and 'B' - a computerized model of the same activity, are said to be analogous when either: (a) 'A' and 'B' have some common objective properties, i.e. are equal in some respects; or (b) there is a correspondence between the parts or properties of 'A' and those of 'B'.(6) If the result of the tests is positive it is then said that 'A' is the analogue of 'B' and vice versa. However, there is a spectrum of analogies ranging from the most perfect, viz. isomorphism at one end, to the weak, namely plain - analogy at the other end. Isomorphism is defined as: two sets are said to be isomorphic to each other if a one-to-one correspondence can be established between the elements of one and those of the other and if all the relations defined on the elements of one hold also among the corresponding elements of the other (7), as shown in Figure (7-a).



Isomorphism

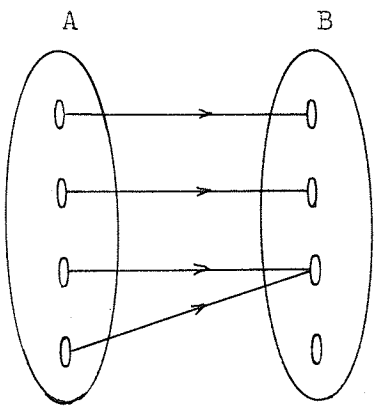
Figure (7-a)

---

(6) Bunge, "Analogy, Simulation, Representation", op.cit.; Ackoff, "Scientific Method", p.109, op.cit.; Beer, "Decision and Control", op.cit., p.111.

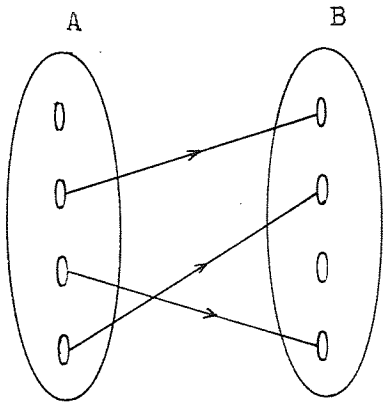
(7) Rapoport, A., "The Uses of Mathematical Isomorhpism", In G.J. Klir,

Also, one-to-one can be imperfect, thus yielding a many-to-one type of relationship as in Figure (7-b), whereas, plain-analogy, i.e. a some-to-some relationship, Figure (7-c), is achieved when some elements of 'A' are paired-off to some elements of 'B'.



Homomorphism

Figure (7-b)



Plain

Figure (7-c)

These constructs can be used to interpret some modes of current computer systems design. It has been mentioned earlier that a significant number of systems designers learn through trial and error. That is they start learning about a certain applications, perhaps for the first time, when they are assumed to design a new computer version. In many cases the purpose of initiating the process of design is to get rid of some of the deficiencies that exist, and not for the sake of computerizing such an application. Since they are inadequately equipped to conduct such an inquiry either one of the two possibilities is likely to occur. The first may take the form of an isomorphic translation of the situation into

computer programmes thus reproducing the same difficulties but in different forms. This outcome can be strengthened by two different forces. On the one hand, the user's employees potential influence upon the outcome due to their role as the designer's main source of the practices and technicalities of the problem. Lack of knowledge weakens the designer's image by reducing his abilities to convince his interviewees and to discriminate between reliable and distorted information. On the other hand, fear of change on the part of the employees can push them to withhold some key information, specially when the designers image is weak, in an attempt, either intentionally or unintentionally, to frustrate that change (8). In this context, isomorphic design is regarded as an indication of weak ad hoc approach which is not capable of defining the problem.

The second possibility is that of homomorphic representation of the problematic situation. By the time the designer has finished that design, he would have developed a conceptual model that is mainly based upon his own experience. Also by that time he would have changed jobs. In his new exercise the designer may try to mould the new application to fit in his model. This bias is often reinforced by the designers tendency to project a strong image as he has already learned by experience. Consequently the resulting design could be a many-to-one relationship between the new - but similar problem and

---

(8) Zandar, A., "Resistance to Change - Its Analysis and Prevention", In P.P. Schoderbek, "Management Systems", (The Wiley Series In Mgmt. & Admin., 1967), pp.200-203.

the designer's own model. Such an approach could have serious consequences as it may implant difficulties that did not exist before.

Because of the characteristic weakness of the plain-analogy which is demonstrated in its definition, i.e. some-to-some, plain analogy imposes less constraining forces upon the designer. Thus it offers, when it is regarded as one of the conceptual bases of model building, the designer more degrees of freedom as it may liberate him from his own model which results in reducing his bias and consequently enabling him to produce more flexible and less problematic design.

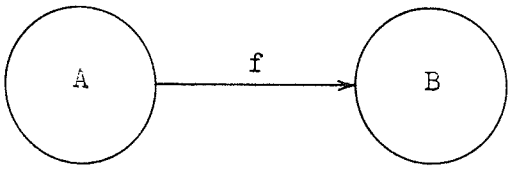
The implications that may be drawn from the above discussion can be summarised as follows. First, in the case of two black-boxes, say an activity and an application package, at least one of them has to be worked upon to reduce it in order to increase its similarities with respect to the second so that a reasonably strong analogy may hold (9). Second to this, and as a consequence of it, only that box which is more comprehensive is subjected to the process of simplification in order to establish isomorphism between the two. Also this may indicate that, having a wider scope, the activity is almost always going to be selected to undergo such a change, which, if not conducted with utmost care, can lead to serious consequences. Third, the whole process is characterised by considerable amount of bias because of the subjectivity of the involved attitudes.

---

(9) This interpretation is based upon Ashby's definition of homomorphism, see Ashby, "An Introduction to Cybernetics", op.cit., p.103.

ACTIVITY-PACKAGE MAPPING

The above mentioned process that is used to establish an analogy between the two boxes A and B is frequently referred to as mapping of A into B - Figure (8). In the following, some of the



Activity-package mapping

Figure (8)

constructs of the set theory are used in an attempt to examine whether the current form of packages is suitable for human organization.

Let an organizational activity 'A', such as scheduling, or inventory control, or payroll, etc., be a non-empty set of a family of non-empty sets '$\mathcal{A}$' (10), such that:

---

(10) In this context, the family $\mathcal{A}$ may be taken to include, for example, all scheduling activities that may exist in a certain environment.

$$\mathcal{A} = \{A_i\}_{i \in I}$$

where 'I' is the index set, representing the universe of discourse, e.g. the environment, of one and only one such activity, so that for each element i E I there corresponds a set A_i. Now, let '$\mathcal{B}$' be another family of sets, such as the family of all scheduling application packages that are available, so that:

$$\mathcal{B} = \{B_j\}_{j \in J}$$

Accordingly, if the activity 'A' is to be mapped into the package, 'B' i.e. f : A⟶B, then the mapping assigns to each set $T \in \mathcal{A}$ a unique set $f(T) \in \mathcal{B}$. In other words, the function f : A⟶B induces a function $f : \mathcal{A} \longrightarrow \mathcal{B}$. However, the two functions are essentially different. The first being a mapping function, where the second is a set function, i.e. its domain consists of sets (11).

However, for the activity and the package to be similar, further conditions must be preserved. The two sets must be, (a) ordered; (b) there must exist a one-to-one correspondence between the elements which preserve the order relation (12).

---

(11) The same argument can be applied to the more detailed case of mapping: i - A particular scheduling activity within a certain company which can be treated as a family of sets whose member-sets are, e.g., the procedures for calculating, allocating capacities, etc. Into, ii - a scheduling package, such as CLASS, which can also be treated as a family of sets whose member-sets are, e.g., the various algorithms that calculate, allocate capacities and produce schedules.

(12) Lipschutz, S., "Set Theory and Related Topics", (Shaum's Series, McGraw-Hill Co, N.Y., 1964).

Order can be partial or total. A partially ordered set consists of a set 'A' and a specific type of a relation 'R' in 'A', i.e. (A, R). The relation R is a comparability relation, i.e. either of any two elements in the set 'A' precedes or dominates the second, i.e. symbolically $a < b$ and $a \neq b$ or $a < b$ respectively. Whereas, a total order in a set 'A' is a partial order in 'A' with the additional property that: $a < b$, $a = b$ or $a > b$.

Therefore an ordered set 'A' is similar to an ordered set 'B', i.e. $A \simeq B$, if there exists a function $f : A \rightarrow B$ which is one-to-one and onto (13), and which has the property that, for any elements $a, a' \in A$: $a < a'$ if $f(a) < f(a')$. The function 'f' is called a similarity mapping.

SOME PRACTICAL IMPLICATIONS

In the light of the above constructs, knowledge of input and output of both an activity and an application package is not enough to establish with confidence the strength of the analogy between them. Thus, it is obvious that the amount of knowledge available about the contents of the two, i.e. the activity and the package, will decide on the level of the analogy. This implies that the designer must be familiar with the structure of both, viz. the activity and the package. However, in practice the

---

(13) A function $f : A \rightarrow B$ is called a one-to-one and onto if for every element $a \in A$, where A is the domain of the function, has one and only one unique image $b \in B$, where B is its range, with no two elements sharing an image.

attainment of such knowledge is made difficult because of two main reasons. The first is attributed to the lack of design information about the package which ought to have been provided by the package designers (14). The second reason is that due to the involvement of the human element in the activity.

An examination of some real activities and a number of applications packages has underlinedthe basic difference between the two. Any computer application package 'B', indeed any computer programme, can be defined as a well-ordered set. An ordered set is called well-ordered if it has a first element (15). Specifically, if 'B' is an ordered set with the property that every subset of 'B', e.g. a module T such that $T \bigcap B = T$, contains a first element, then 'B' is called a well-ordered set. Obviously, this is particularly applicable to applications packages, since every package and every one of its constituent modules, irrespective of size, has a pre-determined first element, e.g. an instruction or statement.

Conversely, not every real activity, in practical terms, is ordered. In fact, the most one may hope for is partial order, perhaps apart from military organizations. The reason behind this

---

(14) In many cases this type of information is non-existent, as in the case of I.B.M. - CLASS package.

(15) Lipschutz, op.cit.

assertion is that as one ascends the hierarchical tree, an activity becomes less determinable and more unspecifiable. In other words, the activity's associated patterns of behaviour become less repetitive implying less programmability/definability. Even at its lowest level, it is not unusual for an activity to have more than one first element. These elements can also be unidentifiable. Because the structure/organization of the activity itself, under conditions of pressure, may be squeezed, altered and/or some of its parts may also be skipped (16). Such behaviour can be explained in terms of the activity's partial-order. Due to its incomplete order, i.e. the relationships between its various parts is not exhaustively defined, an activity retains higher degrees of freedom than these when it is completely ordered. Thus, partial-order, which may seem at the first look as a defect, leads to flexibility, whereas well-order, which also may seem as a healthy sign, in fact leads to rigidity. The difference between the degrees of freedom in the two cases, i.e. the degree of constrain, accounts for most of the difficulties encountered by applications packages, and computer systems applications in general. Therefore, the concept of order can be interpreted so as to distinguish good from bad or ad hoc package design. Good, i.e. flexible design is that characterized by partial-order, whereas bad, i.e. rigid design is that associated with order. Partial-order implies two valuable

---

(16) This can be explainable in terms of skills. The activity pattern of a skilled person differs from that of another person. Also sometimes the skilled person's pattern does not conform to a prescribed set of criteria. Rather it varies according to the circumstances. This flexible and dynamic ability differentiates between, e.g., two skilled persons as well as skilled and semi/un-skilled persons.

inter-related design criteria. The first is that of designing independent parts. This can be ascertained by weakening as much as possible the relationships among the parts of the package. The second is that of designing small parts. This is because as the size of a part increases, the relationships within this part strengthen due to order increase, i.e. dependence increases, until reaching that limit where the whole package becomes in effect a single, well-ordered part. Thus it becomes rigid.

These derived design criteria can be effectively used to examine the viability of packages judging by its current state. The assumption that members of the family of sets '$\mathcal{A}$' of a certain activity* are all similar, i.e. $A_i \simeq A_j$, is practically and organizationally questionable, if not for any other reason than the influence of the subjectivity of the human component. That is, the human component of, say an activity 'A' in company 'i' - 'A'_i', may not necessarily comply with that of company 'j' - 'A'_j'. Yet, the same standard achieved by either(A_i/A_j)may also be reached, or even surpassed by the other(A_j/A_i). Consequently, objectives, methods and procedures may all differ from one company to the other as 'i' varies over the universe of discourse of '$\mathcal{A}$', i.e. 'I'. The question becomes even more persistent as a certain activity, 'A'_i', itself varies with time.

* As defined on p.41.

In the absence of a clear design philosophy, package designers, and computer systems designers in general, have been satisfied with either of the previously discussed methods, viz. isomorphic, i.e. copying, or homomorphic, i.e. moulding, approaches. Both are defective. This has resulted in the current package form which is a statement of self-contradictory design. The truth of this assertion is demonstrated by the fact that almost any available package is said to have comprised a number of individual small parts. However the size of such part is on the average found to be in excess of several thousand core locations (17). That is, in many cases one part of a package can, on its own, constitute a separate and complete package. Such a design complicates the subsequent implementation procedures to fit the package into its user's organisational context. This is largely due to its inherent rigidity that results from the sheer dependence of its parts. This rigidity forces wither one of two outcomes, i.e. the user changes his activity to fit the package, or the package has to be changed. In both cases knowledge of the internal structure of the package is essential. However, this information is not always sufficiently available (18), therefore rendering the package as a proper black-box. Assuming that the user has committed himself to a package, he often finds that the costs required either to adjust the package, if allowed, to his own needs, or to adjust his activity to suit the package, exceed the estimated savings he has initially envisaged.

---

(17) Examples are I.C.L.'s NIMMS package which consists of a number of large modules, with each covering a wide area such as inventory control. Another example is I.B.M.'s CLASS which covers the shceduling activity.

(18) I.B.M. does not supply adequate information of the internal structure of its CLASS package.

Therefore the main justification for packaging, viz. saving by not having to duplicate programming effort, is challenged. The blame is shared by both the designers for not having clear design methodology, and the users for accepting such weak and often incomprehensible designs. However, this main justification for pursuing a package approach can be reinforced by clear understanding of the criteria of order; flexibility, independence; size.

STRUCTURAL CONSTRUCTS

In the preceding discussion the close relatedness of the already reviewed concepts of open-system; black-box; analogy and the derived design criteria, i.e. order and independence, is investigated to show how flexible structure is needed to increase the viability of packages, specially when they are supposed to interact with varying problematic situations.

DATA AND THEIR INDICATIONS

It follows from above that the ability to discern between the various types of data that may be available to a designer/user about a certain package is indispensable for him for a number of

reasons. These may include:

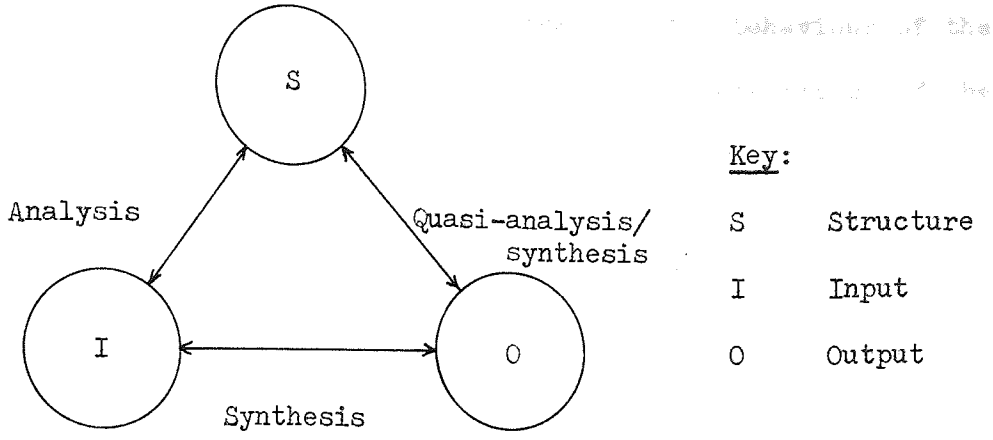
1. The type of data decides how far the user can explain/understand the behaviour of the package.

2. The available data can be taken as an indication of how the designers think of the packaging approach, because:

   i. each type of data may reflect the design methodology and frames of reference, for example whether the designers recognise the inevitable need for change;

   ii. Data can also reflect the designers' attitude towards users, i.e. whether the designers regard users as being identicals.

In general, data about packages may belong to either one of two broad classes, viz. external or internal. External data encompasses these of input and output, where internal are these concerning the structure of a package. The relationships between these types can be depicted as a threesome of boxes, as shown in Figure (9), with the following characteristics. In this game, at



Data-types

Figure (9)

best, only any two boxes may be seen through at any one time. If the two boxes, 'I' and 'S' are given, then box 'O' can be predicted. The process involved is that of analysis. Alternatively, when boxes 'I' and 'O' are known, the process of inferring 'S' is designated as synthesis whereas the third possibility, i.e. involving boxes 'O' and 'S' can be referred to as quasi-analysis or quasi-synthesis (19). However, this articulation can be misleading and a reservation must be stated. Neither any one box is exhaustively known, nor any one box is completely unknown.

Consequently the type of data that is available about a certain package can influence the activation of specific inquiring processes, whose main objective is to predict and control the

---

(19) These two terms are proposed to fill the gap that is identified by Valkenburg, M.E. Von, "Introduction to Modern Network Synthesis", (J. Wiley & Sons, N.Y., 1965), Chap.1, Sect.1.

behaviour of that package. In this respect, the behaviour of the package can indicate some of the underlying characteristics of the structure of the package. This is due to the mutual dependence of both, i.e. structure and behaviour. However, identifying that almost all data that are available to package users are these of the external type; also identifying that not all users are interested in knowing too much about the internal type of data, the question becomes that of designing a structure that facilitates the task of the user when he wants to conduct any of the three inquiring processes.

ORGANIZED COMPLEXITY AND HIERARCHICAL STRUCTURE

Structure implies, and is implied by, order and organization. Thus, it is a definition of the set of relationships that may exist between the different parts of a package. The order in which the parts are organized and the type of linkage/coupling between them determine the properties of the structure. For example, if a pattern of connections or couplings between some parts changes over a given period of time, then the prevailing structure can be regarded as dynamic. Conversely, if that pattern remains virtually unaltered, it is often designated as fixed or static.

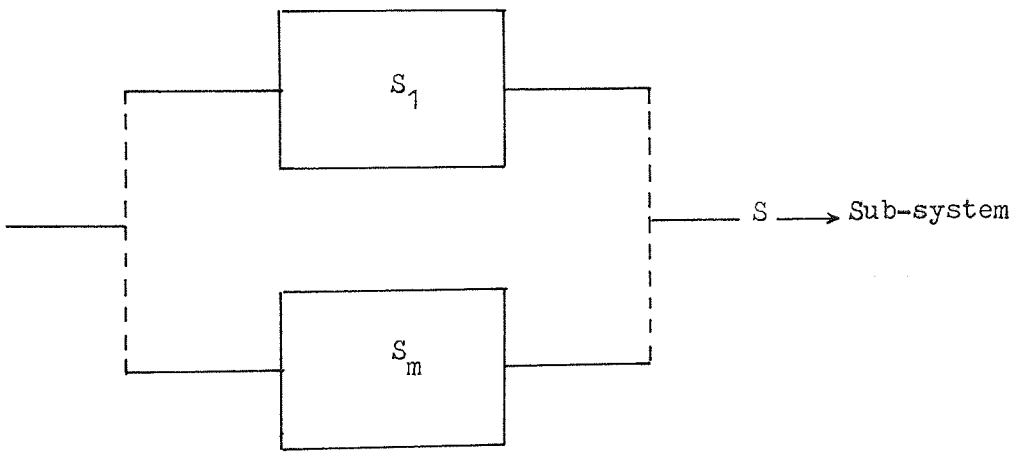
A useful concept for both designing and understanding structures is that of hierarchy. A hierarchic structure is that comprising a number of super and sub-ordinate levels. The principal

criterion of hierarchical ordering is the stability of the various sub-systems that make-up the package. This does not preclude the interactions between these sub-systems but accounts for the overall coherence of the package. However, a hierarchic structure being fundamentally based upon some pre-determined criteria of ordering can lead to rigidity. This can result mainly from the designers' failure to distinguish explicitly the implications of dynamic behaviour within a hierarchy. There are two types of hierarchical dynamics (20). One is the high frequency dynamics within individual components that results from strong couplings between the constituents of a component. The other is the low frequency dynamics between the sub-systems. Thus, the structure becomes rigid when the designer does not discriminate precisely between sub-systems and components. Unfortunately the current trend among systems analysts/designers is the misnomer, i.e. the imprecise use of terms such as systems, sub-systems, etc., on the ground that every system is a sub-system of some larger system and vice versa. To illustrate, I.B.M. produces two packages PICS and CLASS. On one hand, PICS is a production control package which consists of eight sub-systems of which scheduling is one. On the other hand, CLASS is a scheduling package, but it cannot be incorporated into the former.

---

(20) Simon, H.A., "Architecture of Complexity", op.cit., pp.98-114.

Thus, good appreciation of the concept of hierarchy will significantly facilitate the two inquiring processes of analysis and synthesis. In many practical cases the aim of conducting such processes is to identify and replace defective parts whether they are subsystems and/or components. However, the operation required to identify/define a sub-system is the opposite of that required for the component (21). The former is that of decomposing the package into isolated sub-systems, thus the given package/system is considered to be the disjunction of its isolated sub-systems, i.e. if 'Z' is the package, e.g. PICS, and 'S' is the maximal set of isolated sub-systems, e.g. the set containing scheduling, inventory control, etc., then $Z = OR(S)$, as shown in Figure (10). Conversely, the package/system



Disjunction of Sub-systems

Figure (10)

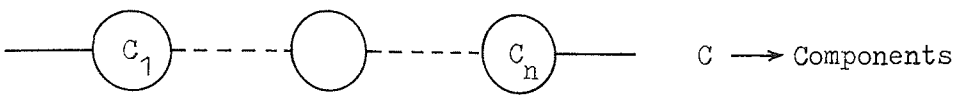
'Z', can be resolved into its independent components (22), thus the

---

(21) Wymore, W., "A Wattled Theory of Systems", In G.J. Klir, (Ed.), op.cit., pp.270-300.

(22) Ibid.

given package/system is considered to be the conjunction of its independent components, i.e. if 'S' is the maximal set of the package's independent components, such as the routine to determine safety stock and re-order points, then Z = AND(S), as shown in Figure (11). Thus the two operations, the determination of sub-systems and components, are duals, as indicated by the OR and AND



Conjunction of Components

Figure (11)

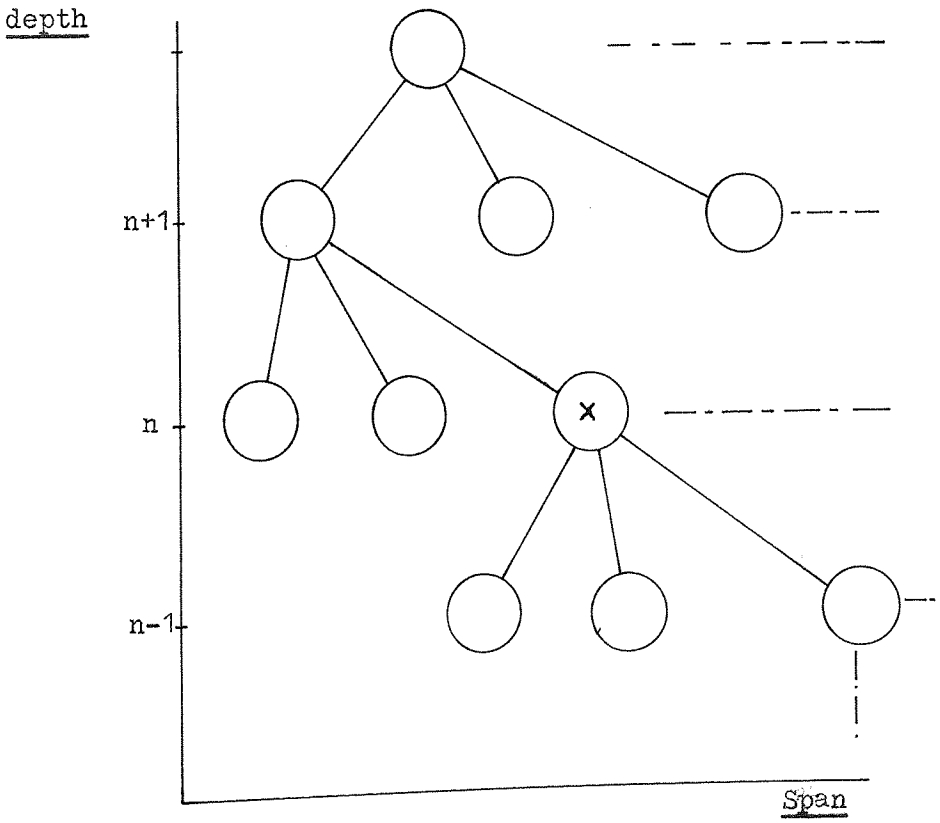
operations. Consequently sub-systems and components are not synonyms. This can be interpreted to suggest few design criteria. First, a proper hierarchic design of packages, i.e. OR based, will ascertain the following: (1) the continuing operation of the package even when more than one of its sub-systems fail to work; (2) detection, replacement or modification become significantly less complicated and less costly; (3) implementation can be carried-out in independent stages by building and adding new sub-systems successfully; (4) work can be delegated to different teams each designing a sub-system. Secondly, the AND-based design is that component oriented. To be effective this needs a considerable awareness of the role of each one component, and a comprehensive documentation is a necessity for designers to be able to follow the logic of the design. Unfortunately, this is the current practice of design which is further aggravated by the obvious lack of documentation. This results in unorganized ad hoc designs as demonstrated by many user-

made programmes and packages such as CLASS, PROMP, etc. where the modules are very big.

However, the question now is what is exactly meant by an independent component, and how does it relate to its system/sub-system. The more recent concept of holons is believed to give adequate answers to these and other design questions.

THE HIERARCHY OF HOLONS: AN EXPOSITION

Koestler, in an attempt to resolve the problem of misnomer, has introduced the concept of a holon (23).

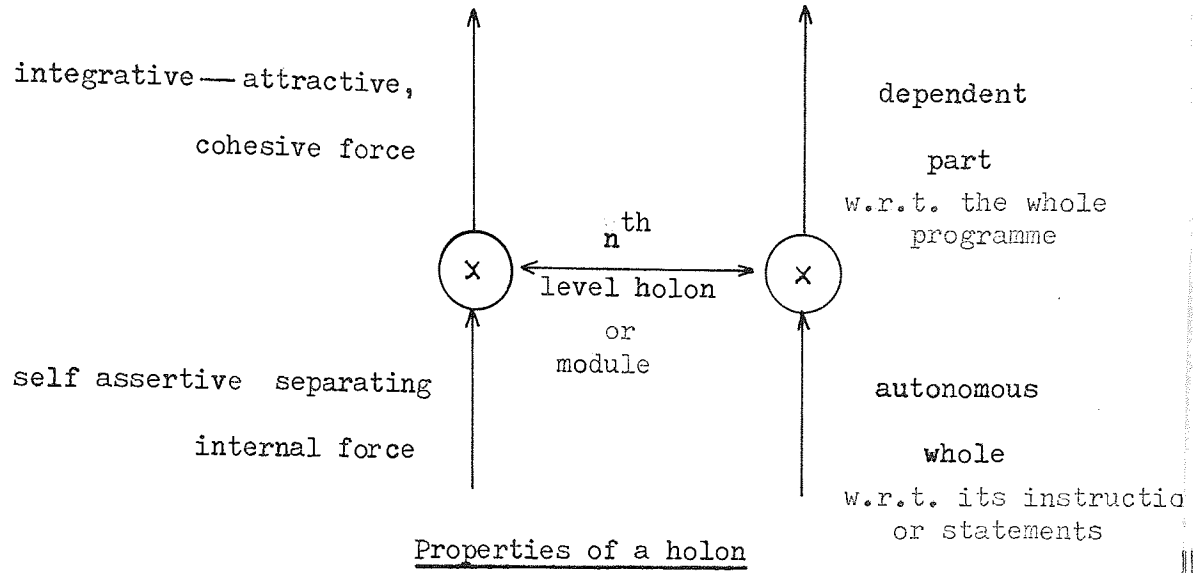


Holon Hierarchy

Figure (12)

(23) Koestler, A., "The Tree and the Candle", In W. Gray and N.D. Rizzo, (Eds.), "Unity Through Diversity", (Gordon and Breach Sci. Pub., N.T., 1973), Part I, pp.287-314.

He derived the term holon from the Greek hole - meaning whole, with the suffix 'on' suggesting a part or particle (24). A holon is a sub-whole which, relative to its subordinate components, behaves as



Properties of a holon

Figure (13)

self-contained whole; with respect to its superordinated controls as a dependent part, as shown in Figure (13). Thus a holon indicates a stable sub-whole which displays rule-governed behaviour and/or structural Gestalt constancy. Also, a holon is meant to supply the missing link between atomism and holism.

One of the basic propositions of the concept is that a hierarchically organized whole cannot be reduced to its elementary parts, but it can be dissected into its constituent branches on which the holons are the nodes of the tree and the lines connecting them

---

(24) Ibid.

represent the channels of communication. Hence

> "...holons are self-regulating 'open systems' governed by a set of fixed rules accounting for the holon's coherence, stability, and characteristic pattern of structure and function" (25).

This set of rules is referred to as the canon of the holon. Consequently, the canon determines the fixed, invariant aspect of the open-system in its steady state (26). However, the canon is in essence equivalent to the constraints that may be imposed upon the holon. These constraints should not exhaust the holon's total degrees of freedom, because if they do the holon will lose its flexibility and becomes unable to cope with any unprogrammed disturbance. This implies the need for even the lowest holons to be allowed some sort of freedom. In contrast, present forms of packages exhibit an overwhelming concentration upon the aspects covered by their canons. Thus packages come with fixed file structures, pre-determined algorithms, record layouts, and even input/output formats. That is, in effect dictating certain structures, and consequently behaviours, which may or may not be compatible with these of their users. Also such elaborate canons reduce the ability of their packages to perform adequately under overloading

---

(25) Ibid.

(26) This description of the canon is based upon an interpretation of Klir's fourth definition of a system, i.e. the definition of the permanent-behaviour, G.J. Klir, "An Approach to General Systems Theory", (Van Nostrand Reinhold, N.Y., 1969), p.43.

conditions (27). However, if designers consider packages as hierarchies of holons, with each holon being allowed to play its roles, i.e. integrative and autonomous, these packages may become much more flexible. In the following the organizational implications of the holon's double-character are considered.

1. Output Hierarchy : Triggers

Output hierarchies are these which operate on the trigger-release principle. Accordingly, a relatively simple implicit or coded signal releases complex, pre-set mechanisms. Examples are sub routine call statements; core-dump, file-setup commands, etc. Each initiates the execution of a set of instructions. The signal from higher echelons does not carry all the details of the actions of the holon. It only triggers the holon into action. This impulse will cause the holon to reactivate its relevant sub-units in the appropriate sequence according to the received message, and guided by feedback from its environment (28). This is essentially what is known as top-down communication with the associated property of magnification. However, implicit in the output hierarchy is the emphasis upon autonomy of parts.

---

(27) The common reason held among these who do not use packages or have used to and abandoned them is the inflexibility of packages.

(28) Koestler, op.cit.

2. Input Hierarchy : Filters, Scanners and Classifiers

Input hierarchies work on the opposite principle of that of output hierarchies. In other words, instead of using trigger-release mechanisms, input hierarchies employ series of filters, scanners, and classifiers. Their effect is that of reductionism. Thus input hierarchies are similar to the bottom-up communication mode.

Obviously hierarchies are neither exhaustively output nor input but combinations of both. Thus the resulting mixes account for the intelligibility and simplicity of the structures as well as their viability as outlined below.

Arborization and Reticulation

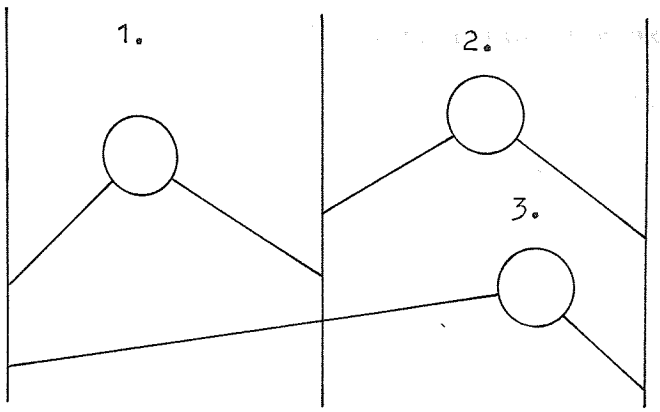
A production control activity comprises sub-activities such as scheduling, sequencing, inventory control, forecasting, costing, etc. Each can be conceived of as a tree. A tree constitutes a vertical structure. The nodes where the branches of such trees meet form horizontal networks at several levels. For example, the output of forecasting can be fed to the inventory control, thus forming one node, and the output of scheduling can be fed to inventory control as well as forecasting, resulting in two extra nodes and so on, as shown in Figure (14). Without the trees there could be neither entwining nor networks (29) and thus no

---

(29) Ibid.

Vertical Structure

Horizontal Networks



Scheduling Inventory Control Forecasting

Key:

1. Material Requisition
2. Material Purchase
3. Planning

Arborization and Reticulation

Figure (14)

inter-dependences. Also, without the networks, trees would be isolated and hence no integration of functions. Thus arborization and reticulation are complements. The design indication is clearly that of the nodes. Being a tree implies upwards and sidewards growth that results in the development of new nodes.

Mechanization and Structure

An activity such as production control and its sub-activities become, on successively lower levels of the hierarchy, increasingly mechanized and predictable. Thus a sub-activity or a holon on the n^{th} level has more degrees of freedom permitted by its canon, than a holon on the (n-1) level. However, in a changing environment time

comes when mechanized holons face situations that call for transforming mechanized activities into mental activities, for example, when stocks reach an unacceptable level or when schedules slip behind. If the holon is fully mechanized, control will have to be transferred to some higher, i.e. less mechanized holon, conversely if the holon is partially unmechanized it may be able to handle the situation. Consequently, mechanization can inflict rigidity by transforming the activity into a machine. In this light the designer should be alert and prevent such a mechanization from reaching to the apex of the hierarchy. By doing so he keeps rigidity under control and within its specified limits and therefore allowing for future development.

CONCLUSIONS

It can be concluded that:

1. The designer conceptual model ought to be flexible to minimise the effect of his bias and to avoid either, a - copying existing problems into the potential computer version;
   b - inflicting new problems as a result of moulding the situation into his own pre-conceived model.

   However, either one of them is a sign of lack of firm methodology.

2. Order is seen to induce rigidity. This applies to both, i.e. the human activity and the package. Therefore ordering must be only partial otherwise the human component will not have enough freedom to cope with contingencies. Also flexible structuring of a package increases its effectiveness as it facilitates subsequent modifications.

3. As size increases, independence decreases, and rigidity increases. Therefore small modules will ensure both independence and flexibility.

4. The problem of structuring a package within the context of flexibility, independence and impartiality is much reduced by using the holonic concept of a package, as a self-contained, rule-governed stable open-system, which implies:

   a - it must be open-ended at the top to allow for growth,

   b - its canon must be flexible to survive unplanned circumstances,

   c - it must be hierarchical to facilitate identification of defective components,

   d - its components or holons are self-contained,

   e - it cannot exist in isolation thus provision must be made for the inevitable entwining with other holons, i.e. packages, programmes, etc.

   Also, the concept of holon re-affirms the concluded effect of order upon flexibility by showing that automation, which is a form of order, reduces flexibility.

Throughout of the remainder, various aspects and implications of the arborization and reticulation will be discussed.

CHAPTER 3

APPLICATIONS PACKAGES AND MANAGEMENT INFORMATION SYSTEMS

This chapter is devoted to the discussion of two items. The first involves the examination of typical M.I.S. definitions and approaches and the second examines the relationships of packages to M.I.S.

Considering an M.I.S. as one of the environments with which an application package is to interact, thus the problem of M.I.S. definition and its relationship to data and performance is examined to determine where the application package lies. After establishing the relationship between M.I.S. and the package, some design effects of the former on the latter are drawn through analysing the basic features of the two prevailing M.I.S. design approaches, i.e. top-down and bottom-up.

The discussion continues by examining the decision whether to buy applications packages or make one's own programmes. Two factors are considered to have influence upon the decision: 1) the decision maker's own bias; 2) the manufacturer's weight. The investigation extends further to attempt to reveal the causes and effects of using packages, or ready-to-implement solutions upon the flexibility, versatility of data processing departments and experiences of data processing staff.

RELATION OF M.I.S. TO APPLICATIONS PACKAGES

The concept of management information systems, or M.I.S., is discussed for two main reasons: 1) a management information system is the framework within which applications packages are slotted; 2) the concept of management information systems is a source of generating and evaluating design methods and approaches since it is the focus of attention of a multitude of disciplines such as Cybernetics, Psychology, Decision-making, etc. Particularly it constitutes an interesting and challenging application for systems philosophy. Thus whatever design methods and approaches that may emerge from M.I.S.'s will affect these of packages.

A company can be considered as being analogous to a communication network. Also it is today's concept to regard a computer, within the context of the network, as being similar to the human mind with respect to the human body (1); the analogy is based upon the similar function of data processing performed by each. Both, it is suggested, maintain their function by supporting neuron/communication networks through which data are received and transmitted.

---

(1) Argyris, C., "Management Information Systems: The Challenge to Rationality and Emotionality", Management Science, 1971, Vol.17, No.6, pp. B-275/292.

In the context of the organization, the resulting communication network is usually referred to as management information system. The designer's definition of the primary role of the computer is believed to affect the flexibility, impartiality and independence of the M.I.S. Thus, if he thinks that the computer's role is that of a problem-solver/decision-maker (3) then, his tendency will be to emphasize the need and usefulness of increasing automation. This is often implied by dissociating the manager/user from the M.I.S. Through increasing automation and less involvement of the user the structure of the M.I.S. becomes more like the conventional machine with its inability to change or adapt itself to cope with new environmental perturbations. Alternatively, if the designer considers the computer as a useful aid or tool for problem-solving/decision-making then, the role of the manager/user as the prime decision-maker may not be undermined. This can lead to a more flexible M.I.S. through its inclusion of the user/manager and his adaptability.

In either case the scope of M.I.S. is influenced by the underlying attitude. According to the former, the scope of M.I.S. should be delimited by what can be economically defined/programmed. In the latter the question becomes irrelevant since the scope does not depend on the computer and the emphases should rightly be directed to what can be defined/programmed to support management.

However, practical as well as published evidence show that scopes of existing M.I.S.'s do not go beyond computerizing parts of some of the activities of the involved companies. These computerized parts, in the main, do not exceed the operational level (2).

As may be expected there exists a myriad of M.I.S. definitions. Representing different viewpoints but strangely enough almost all imply automation. Thus, a typical definition is that

> "An M.I.S. is an automated system which presents to the manager information, both internal and external to the business, that aids him in making a specific set of routine decisions" * (3).

An M.I.S. so defined is, obviously, limited in scope and handicapped because: 1) the definition confines the use of M.I.S. to routine decisions only which are out-weighed by non-routine decisions; 2) it further limits M.I.S. to automated components which are generally outnumbered by unautomated ones. Such an M.I.S. cannot be considered more than a small number of computer

---

* Emphasis is not in the original.

(2) Gorry, A.; Morton Scot, M., "Framework For Management Information Systems", Sloan Management Review, 1971, Vol.13, No.1, pp.55-70.

(3) Stern, H., "Information Systems In Management Science", Management Science, 1970, Vol.17, No.2, pp. B-119/123.

programmes or packages that may perform functions such as payroll; 3) the exclusion of the manager/user degenerates the M.I.S. into a rigid set of programmes which may or may not sustain their relevancy and/or usefulness, thus disregarding the basic idea of M.I.S. as being management, i.e. user, information system.

Also, the question of the difference between data and information is not purely academic because the answer determines the attitudes of the designer towards the role the M.I.S. is supposed to undertake. In this respect data are regarded to be facts which can be used as a basis for reasoning and thinking. Information is the meaning extracted from data by means of interpretation, and thus it is meaningful only to the person who derives that meaning at that time. Accordingly, the designer's concepts of data and information can both be identified from and characterize his design. Once again there are two cases of interest. The first is the case where M.I.S. is defined to exclude the user, i.e. the user does not constitute a component of M.I.S. In this context, and according to the present understanding, M.I.S. cannot produce information unless the term information is used indiscriminately, as often happens, as a synonym of data. Otherwise, a definition such as

> "Information is the intelligence or system output which results from the conversion of data into a 'product' which enables management to take action appropriate within a particular frame of reference" (4)

is utterly defective. This is because data and information are not

---

(4) Kriebel, C.H., "Design of Management Information Systems", In J.F. Pierce, (Ed.), "Operations Research and the Design of Management Information Systems", (Special Ass. Pub., 1967), pp.375-390.

distinguished from each other on the basis of their respective forms, i.e. whether being arranged into tables, graphs, etc. Thus data arrangements do not suffice the conditions to automatically convert them into information until they have been acted upon by management/user. Also, the assertion that an M.I.S. without its user is capable of producing information is impregnated with bias, specifically, in favour of M.I.S. at the expense of its user. Because if the user is presumably provided with information then his role implicitly becomes equally degenerated into an automatic form which can be triggered on and off without thinking as dictated to him by the M.I.S. Obviously such an attitude is bound to produce futile and barren designs, and promote unhealthy conflict between the designer and the user. Consequently, the misunderstanding of data and information can, as it does in practice, mislead many designers to rate their designs, i.e. parts of M.I.S., as higher than their users. This is based upon the belief that the designer, despite his defective understanding of systems, is the one who defines/solves the user's problems.

The second case is that in which the user is seen within the boundary of M.I.S. Accordingly, it can be maintained that the M.I.S. converts data into decisions. The process of conversion implies the generation of information at the point where the decision maker lies and not outside it. In this context, the user satisfaction can be increased as the attitude of the designer becomes

less biased. Such an attitude is strengthened and implied by the appreciation of the user's role and impact upon M.I.S. Thus, failing to grasp the relativistic nature of information is seen as one of the main reasons for the reported failure of M.I.S.'s.(5). Churchman emphasises this by saying

> "They (i.e. computer-based information systems) fail simply because their measure of performance is in terms of the transactions, rather than the benefit. The true benefit of an information system must be measured in terms of the meaning of information to the user (6).

Consequently, the communication network encompasses both automated/unautomated components and the user, e.g. the manager.

Because of its complexity, novelty and high rewards, management information systems design as a proper area of application attracts the attention of many systematists. Again two prevailing approaches can be identified. The first, i.e. top-down, is a version of the systems approach. It has not gained yet practical popularity perhaps because it requires clear understanding of the basic concepts of systems as well as the situation at hand, before the approach can be successfully applied. The second, i.e. bottom-up, is component-oriented and therefore is complementary to the first. Misconception of bottom-up approach constitutes the common practice, ironically enough, among computer systems designers/analysts. Before reviewing the two approaches it should be clear that either approach can activate and use the other.

---

(5) Ackoff, R.L., "Management Misinformation Systems", Management Science, 1967, Vol.14, No.4, B-147/156.

(6) Churchman, C.W., "The Systems Approach", (Dell Pub. Co., N.Y., 1968), p.112.

TOP-DOWN AND BOTTOM-UP APPROACHES

In the top-down approach the emphases are placed upon wholeness and the process of defining the user's set of objectives. Gradually working from that point downwards, the objectives are broken-down successively into subordinate objectives at the various levels. Until reaching that level below which it becomes elementary and a waste to proceed. However, simultaneous to this process, another procedure takes place; the latter involves the attempts to map holons as needed.

A major premise of this approach is that there is no guarantee that can be given to ensure the realisation of all needed components or holons. Consequently, it is quite probable after completing the whole process to find a number of missing components or holons.

Top-down per se reflects the inter-dependence and dynamics of the processes of defining or solving and analysing and synthesysing. In other words it seems more likely to fulfil the requirements of the inquiring process previously outlined. Also a successful application of this approach depends wholly on the ability to define objectives and the relative understanding and rigour of the investigator's conceptual framework.

Alternatively, the bottom-up approach is concerned with the identification of what is available in terms of functions, resources, etc. Guided by these constraints, objectives can be formulated and then, the procedure is supposed to start. Combining

or linking a number of equivalent and/or inferior sub-functions or holons together, it is assumed, may suffice the requirements of developing a superior holon at a higher level. By repeating this process on these superior holons results in the development of still higher holons. This repeats until the required apex is reached.

According to this interpretation a major premise of the approach is that there is no guarantee that objectives are to be satisfactorily fulfilled. In this context it is not unusual to consider objectives as being excessive or unachievable and, thus, they may be forced to change. Consequently, objectives may have to comply with what can be achieved rather than what ought to be achieved. The discrepancy between the two sets may account for user dissatisfaction.

The approach seems to revolve around the concept of optimasation. Optimisation may be activated in two ways. First the overall optimisation that aims to strike an acceptable balance between what is desired and what is achievable under certain conditions. The second is that concerned with sub-optimisation, say, between the various components of the application. However, optimisation should not be used as a scapegoat for inadequate definitions or designs. Such inadequate designs may result as consequences of misunderstanding of optimisation and/or careless application of the approach. These, in turn, may lead to interface-incompatabilities and conflicting holons. Less careful application of this approach may result in a form of disorganised complex of holons rather than a harmonised whole.

The two approaches, top-down and bottom-up, seem to converge, at least, at one point. Namely, that of both being iterative insofar as objectives are concerned. Both, admittedly, start with somewhat vague objectives and both aim to strike an acceptable balance between the inherited uncertainties and resources, on the one hand and satisfactory design on the other. As soon as the processes of iteration start the two approaches immediately start to diverge rapidly. In case of top-down, iteration is less conspicuous and less frequent due to the close contact between each level reached and its objectives and the influence of the overall goals. No action is taken without a need signalling its relevance and thus no superfluous holons are produced, only those needed, within the total framework, are investigated. On the other hand, having to start from, nearly, the lowest level, i.e. the most distant point from the objectives, the designer's chances of going astray increase, so asking for reliable and effective feedback mechanisms to keep himself, and others, within limits. The introduction of feedback loops, in addition to the extra costs it may incur, complicates the design (7).

---

(7) Feedback loops which are designed to control the members of the design/implementation team are also seen in conventional programming languages such as FORTRAN in the form of GOTO statements. The use of such statements complicates the logic of the programme and can reduce its life span by making it very difficult to follow even for the person who wrote it.

Thus, each approach sets out a method by which M.I.S. holons are supposed to be arranged and coupled to each other. Any one holon, in this respect, can either be tailor-made to that particular application, or, alternatively, it can be an extraneously produced application package. Whatever combination the M.I.S. may comprise will influence its characteristics and prospects through the properties of the individual holons concerning precision, robustness, time-space performance, proving, tuning, etc. However, management, as the prospective user of the M.I.S., may only be interested in service, reliability, up-to-date application, and cost.

Inevitably the application of either approach to a problematic situation may result in the identification of a number of new computer applications. To fulfil such requirements, management has to decide either to buy or to make them. By the former, i.e. to buy, it is meant the use of ready-to-implement applications packages that are externally supplied. By the latter, i.e. to make, it is meant any sort of in-house definitions/designs of the required application by using, basically, the internal resources of the company concerned. Because of the serious consequencies of whatever option the management may go for, the question is discussed at some length in the following.

FACTORS AFFECTING THE DECISION WHETHER TO BUY OR MAKE

The relative importance of the decision to buy an application package or to do one's own design work is derived from the likely impact of its outcome on the future of data processing departments regarding their effectiveness and adaptabilities (8). An outcome, i.e. either to buy or to make, is preferred in reference to a number of critical factors. These may include (a) the concept of defining/solving; (b) the decision-maker's own experience, attitude, and bias; (c) the role of the manufacturers and suppliers of software and hardware; (d) the implications, advantages and disadvantages of general-purpose packages; (e) compatibility between the desired package and those already in operation; (f) the abilities and resources available to the company concerned.

A particular conceptualization of definitions and solutions is bound to be reflected in the process to arrive at a certain decision. Static appreciation of definitions leads to static consideration of solutions resulting in a sort of undermining the significance of modifiability and flexibility. On the other hand, dynamic realization of the properties of definitions and solutions emphasizes the inevitable need for change as a consequence of the perpetual interaction between

---

(8) Data processing department is used as a reference to any computer department including management services divisions.

the package, i.e. an imported definition/solution, and its environment, thus acknowledging the benefits of flexible designs. Consequently, the three criteria of design, viz. flexibility, bias, and independence, should be considered by the decision-maker when deciding either to make or to buy a package.

1. Decision Maker's Bias

Like definitions, decisions cannot be separated from, or discussed without direct reference to the person who formulates them. Once again, the person involved in this type of decisions, i.e. whether to make or to buy, is constrained by his conceptual framework and influenced by his own experience. This combination results in his individuality. Nevertheless, individuality indicates uniqueness which in turn can be credited to bias. Either consciously or unconsciously, the person concerned undergoes a continuous struggle between his objective and subjective views. The difference between the two is that the former depends upon the comprehensiveness of the decision-maker's knowledge of the precise outcome of all conceivable alternatives, whereas the latter this knowledge is incomplete and more or less incorrect (9). Consequently, questions such

---

(9) Simon, H.A., "A Behavioural Model of Rational Choice", Quarterly J. of Econ., 1955, Vol.69, pp.99-118; "Administrative Behaviour", (Macmillan, N.Y., 1947).

as rational versus irrational and personalistic versus impersonalistic decision making become directly pertinent to the decision of whether to make or to buy.

There are two schools of rational decision-making (10). These are the classical-rational and neo-rationalist schools. The principles of the former include: 1) the decision maker should consider all alternatives and their consequences, and makes a choice according to a scale of preferences; 2) his objective is to maximize gain expressed in some quantitative parameter related to these preferences; 3) he believes that any decision process can be eventually modelled. However, if all alternatives and consequences are known plus the ability to model the decision process then whatever decision is involved can be automated. Thus, the one who believes in this theory is bound to be biased to automation. However, the latter, i.e. the neo-rationalist school, is based upon the principle of bounded rationality which postulates that managers do not often proceed to the point of maximization in decision making, but individual managers pursue the decesion process only as far as the first satisfying solution that presents itself, the criterion of satisfaction being entirely personalistic (11).

---

(10) Gremion, C., "Toward a New Theory of Decision Making", Internat. Studies Mgmt. Org., 1972, 2(2).

(11) Simon, H.A., "The New Science of Management Decision", (Harper & Row, N.Y., 1960).

Thus, rational decisions can be either personalistic or impersonalistic (12). In the former decisions are typified by both incompleteness and unspecificity of the process itself, whereas in the latter decisions are completely specified and laid-out beforehand.

Accordingly, imporsonalistic decisions being independent of the decision maker himself, can, therefore, be automated. The personalistic decisions being incomplete depend essentially on, the decision maker's reasoning, judgment, intuition and reflection. It is imperative to emphasize that the foregoing dichotomies are all essentially relativistic and personalistic decisions are usually considered by some higher decision maker.

As far as the decision to buy or make is concerned, it is regarded to be personalistic. In practice there cannot be a clear well specified set of rules according to which a data processing manager can decide on either course with reasonable certainty of success. In fact the whole issue rests upon the manager's own attitudes and judgment (13). His assessment of the abilities, skills and resources subordinate to him is almost totally personal. Attributes such as the manager's social

---

(12) Eilon, S., "What Is a Decision", Management Science, 1969, Vol.16, No.4.

(13) Most managers maintained that the matter is decided upon within the scope of their individual circumstances insofar as co-operation with producers, their own departmental experience, available resources, time limitations, etc. are all concerned.

values, way of communication, familiarity with the practices and developments in data processing, his understanding of the role of his department, his attitudes towards his subordinates, are all liable to influence the response he gets from his staff and users. This in turn affects the manager's own assessment. Another factor that usually has a bearing on the assessment is the manager's attitude towards a particular manufacturer. Individual managers may have personal bias in favouring particular manufacturer.

The intention behind the foregoing is nothing more than to establish that the decision either to buy or make is based upon personal preferences rather than a rigorous set of criteria. Thus, nothing can be offered to the manager to safeguard him against taking the wrong decision better than confirming the importance of having clear conceptual framework from which consequences can be imagined and weighed therefore fulfilling the requirements for rationality as attempted below.

2. Manufacturer's Influence

One of the objectives of package producers is to maintain, if not increase, their respective market shares (14). One approach is by offering software applications compatible

---

(14) Package manufacturers can be divided into: a) hardware manufacturers which, by and large, cover wide areas of applications; b) software houses, which are mostly involved with operation functions; c) service bureaux; d) users groups.

with popular hardware. Obviously, hardware manufacturers are, in this respect, the most influential. They enjoy stronger relationships with users or potential users than other computer organisations. Also, they control more resources and guide most of the trends as far as computing is concerned.

A computer is one of the most flexible available machines. The number of possible applications to which a computer can be used is, indeed, very large. Realizing the potentiality of this property, hardware manufacturers to be more competent started to offer software support (15). The idea being that a user may not only rely and depend on the manufacturer for hardware supply, he is also persuaded to depend on the manufacturer for his software needs.

The impact of hardware on the user's company is far less dangerous than that of software, in particular software applications. This can be appreciated when the role software plays in a company is understood. First software applications, i.e. M.I.S. in its limited definition, constitute the media through which data processing department interacts with other departments. Equally true through these applications various departments can interact with each other. Thus, software does not only affect hardware; more serious than that its consequences tend to extend over the

---

(15) The term software is used here not only to indicate operating systems, but also to refer to applications. This is discussed in the next chapter.

whole organization. Consequently, establishing control over software is regarded to be more exacting and potentially more rewarding than controlling hardware. Therefore, an unaware user may find whatever automated activities he may have under the control of the software supplier which is particularly alarming if the user has a form of data-base and/or integrated/ total management information system.

Package designers rely heavily on the assumed sameness of all companies that may make use of their packages. This is for the purpose of extending the usability of packages. However, this was shown earlier to be misleading in two respects. The first is due to the variation in human behaviour from one company to the other and from one time to the next. The second is the implications of the content, e.g. algorithms; bias, e.g. emphases depend upon the attitude of the designer; rigidity, e.g. the extensibility of the package, which are all built into the package (16).

In this context perhaps it is expected that a general-purpose package may intentionally incorporate a relatively high degree of flexibility to reduce both the unavoidable differences between potential users and the built-in bias. Unfortunately, in practice, such a package seems to suffer from both. Therefore,

---

(16) Cf. Chapter 1, p.20.

a keen user who sets himself to adopt such a package may have to employ very competent staff which can be costly (17). The user by doing so is most likely to get a hostile reaction from the supplier by withdrawing his support (18). Thus, leaving the user with what amounts to a black-box problem. In many cases the costs incurred in solving that problem equal, if not exceed, those required to design the user's own programme.

Besides direct control of hardware and software manufacturers aim to influence users' staff through training schemes. Existing practitioners, e.g. systems designers, programmers, get their basic knowledge about design, languages, etc. from such schemes, which are designed to introduce bias to the trainees in addition to instructing them. Thus, the manager must be critical when he gets recommendations from his staff and examines them within the context of the staff's backgrounds.

Therefore the data processing manager must be alert not to relinquish his control over his department and indirectly other involved departments to a manufacturer or supplier by not considering the consequences of opting for ready-to-implement solutions.

---

(17) Duncan, Discussion, "Software Engineering", Infotech State of the Art Report No.11, Maidenhead, Berks., pp.171-172.

(18) Holden, G.K., "Factfinder 13: Production Control Packages", (N.C.C. Pub., 1973), p.53.

THE AFTERMATH OF DECIDING ON A PACKAGE

Far reaching consequences of relying upon packages are recognisable. Of these are the effects upon experience, versatility in terms of human and hardware terms, and independence. However, the assertion made by many data processing managers that they have to use packages because of their lack of competent systems designers is believed to be the start of a vicious circle. This is because the more a company opts for packages, the less experience its practitioners may gain (19), and the more a company believes in extrinsic solutions, the less need it may recognise to go for more expensive staff. This may be attributed to the imaginary reduction in design, analysis and programming functions associated with the acquisition of packages. As a result, when presented with a fresh problematic situation, the company may find it difficult to resist the tendency to opt for the seemingly-easy way, i.e. another package, which again may further contribute in limiting its staff experience. This crucial relationship is best explainable in terms of the Law of Evolutionary Potential. The law states

> "the more adapted a population (presently, systems designers) becomes to a particular environment (i.e. packaging) the less adaptable it is when faced with other environments" (20).

That is, through what may account for a constant interaction with

---

(19) Since most packages are very difficult to understand even if detailed information is available.

(20) Weinberg, G., "Natural Selection as Applied to Computers and Programs", General Systems, 1970, Vol.15, pp.145-150.

packages, systems designers may develop certain skills required for the handling and operation of packages, but on the expense of other more important abilities such as design itself. Two possibilities may emerge but lead to the same outcome. The first is that of motivated designers who seek design experience. In this case the designers may find a package-environment monotonous and futile that may drive them to look for more challenging jobs elsewhere thus leaving the company with less able designers. The second is the loss of confidence of the remaining designers to undertake design responsibilities. Handicapped with their conceptual understanding of systems and the particular area of application, in addition to lack of experience in design, designers may be too reluctant to accept or propose to assume proper design responsibilities. Thus, through the loss or stagnation of design faculties, versatility of the staff declines, leaving the environment to exercise a sort of selection. The term selection is used in the sense that the environment may push for more packages leading to higher still specialization of the population, i.e. designers, in packages. Logically, as this situation persists, the whole company becomes more and more dependent upon the suppliers and their ready-to-implement solutions whose effects do not vanish at this point, but extends all over the concerned areas of application that may cause serious conflicts between these areas and data processing.

The same argument can be practically applied to hardware. By delimiting available hardware to the running of a particular package, or application to this effect, only these parts and components of hardware used get maintenance and, thus, are kept in shape, whereas other unused parts accumulate errors as time progresses. Consequently, when a new and basically different application is tested, hardware may fail to survive the test (21). Therefore it logically follows from the above that high specialization must be avoided since it brings with it rigidity in the form of limited freedom to act. However, if systems concepts are adequately understood by systems practitioners, including managers, this conclusion should have been clear in their minds since high specialization is refuted by the systems first principles. Thus, variety of applications helps to keep hardware from becoming highly adapted to a specific application, and through the use of its own resources a company can maintain and develop the versatility of its staff that may lead to greater independence.

The decision then whether to go for a package or not should be weighed independently of whether a company does or does not use other packages at that time. Also, the issue is not so much that of know-how and resources enjoyed by the suppliers as it is an issue of self-development and perseverance by the company concerned. However, in practice, the decision is based upon an awareness of lack of experience and investment priorities.

---

(21) Ibid.

Table (1) showing the attitudes of five data processing managers

Table (1)

Users' Estimated Software Effort Distribution

	A	B	C	D	E	Avg.
Design + Analysis %	15	30	20	40	25	26
Programme Coding + Auditing %	60	40	50	40	35	45
Testing + Integration %	25	30	30	20	40	29

who use packages to 'how they allocate their resources regarding the three main software development phases' illustrates this. The first four entries give programme coding and auditing the biggest slice so supporting the previously questioned programming cost savings (22). In view of the current researches in this area, it is recommended not less than 45 to 50% of the total effort should be directed and allocated to testing and integration (23). Seen within the frame work of selection, users by over-estimating the difficulties associated with coding, tend to overlook the real costly problem of testing and correction. This coupled with the fact that there will always be bugs in any programme (24), provide

---

(22) Cf. Chapter 2.

(23) Boehm, B.W., "Software and its Impact: A Quantitative Assessment", Datamation, 1973, May; also Wolverton, op.cit.

(24) Cf. Chapter 6.

practical evidence that they, perhaps, use inaccurate and unrealistic criteria. For even in cases of charge-free packages, e.g. CLASS, the user may consciously or unconsciously over-estimate what he gets. In other words, although the package itself may be given free, the hardware on which it runs, its maintenance, are all paid for. More important is the price he may pay in the form of degenerating his experience abilities, and his stronger dependence on extraneous sources.

CONCLUSIONS

An applications package is a part of an M.I.S. which is defined to include the whole organisational activities irrespective of both routine and/or automation. The output of such a package should be treated objectively, i.e. as data, and not subjectively, i.e. information, in order to ensure that it is the user who controls the package and not vice versa.

Proper or effective application of either approach, i.e. top-down or bottom-up, may require the use of the other. The ability to identify such interdependence between the two approaches is much enhanced by clearer understanding of systems philosophy and its concepts. Failing to identify such an interdependence can result in over-complicated and ad hoc designs that management nowadays complain from.

The decision to buy or make is personalistic since the manager has not got a specified set of criteria that he can apply, knowing this manufacturers try to tempt managers by offering applications packages, sometimes charge-free, to retain their customers. This is because through software the manufacturers ensure the continuing dependence of customers. This dependence marks the start of a vicious circle that can seriously be damaging since the more a company opts for packages, the less experience its designers may get and the less attractive it may become to competent designers to stay, this, in turn, pushes for more ready-to-implement solutions.

CHAPTER 4

APPLICATIONS PACKAGES AND SOFTWARE

This chapter discusses software as being the second environment, besides M.I.S., of an application package. Software is the environment that supports and activates the package when need arises and imposes restrictions upon future developments in the package.

The idea of user's control over software is investigated particularly because software, in its limited definition consists of compilers, operating systems, etc., is provided by the manufacturer. Having established earlier the consequences of a user not controlling his software, two models are examined to provide two things: 1) to establish the dependence between software and applications programmes; 2) to show how the user can increase his control.

Software structure is discussed and the relationships between its components are examined through two models. Also the concept of holons is compared to these models to test its applicability.

RELATION OF SOFTWARE TO APPLICATIONS PACKAGES

Software, i.e. the whole corpus of programmes, is studied here because it influences application programmes, including packages, in a number of ways such as: 1) it determines what programming languages can be used, size of programmes, mode of processing, etc.; 2) it provides support in the form of sort programmes, core dumps, error flags, loading and execution of programmes, compiling, etc.; 3) it constitutes a centre of attention and consequently it generates design methods and techniques. In this respect, an application package in addition to its dependence upon the M.I.S. it also depends upon software as far as effective use of hardware and other programmes are concerned.

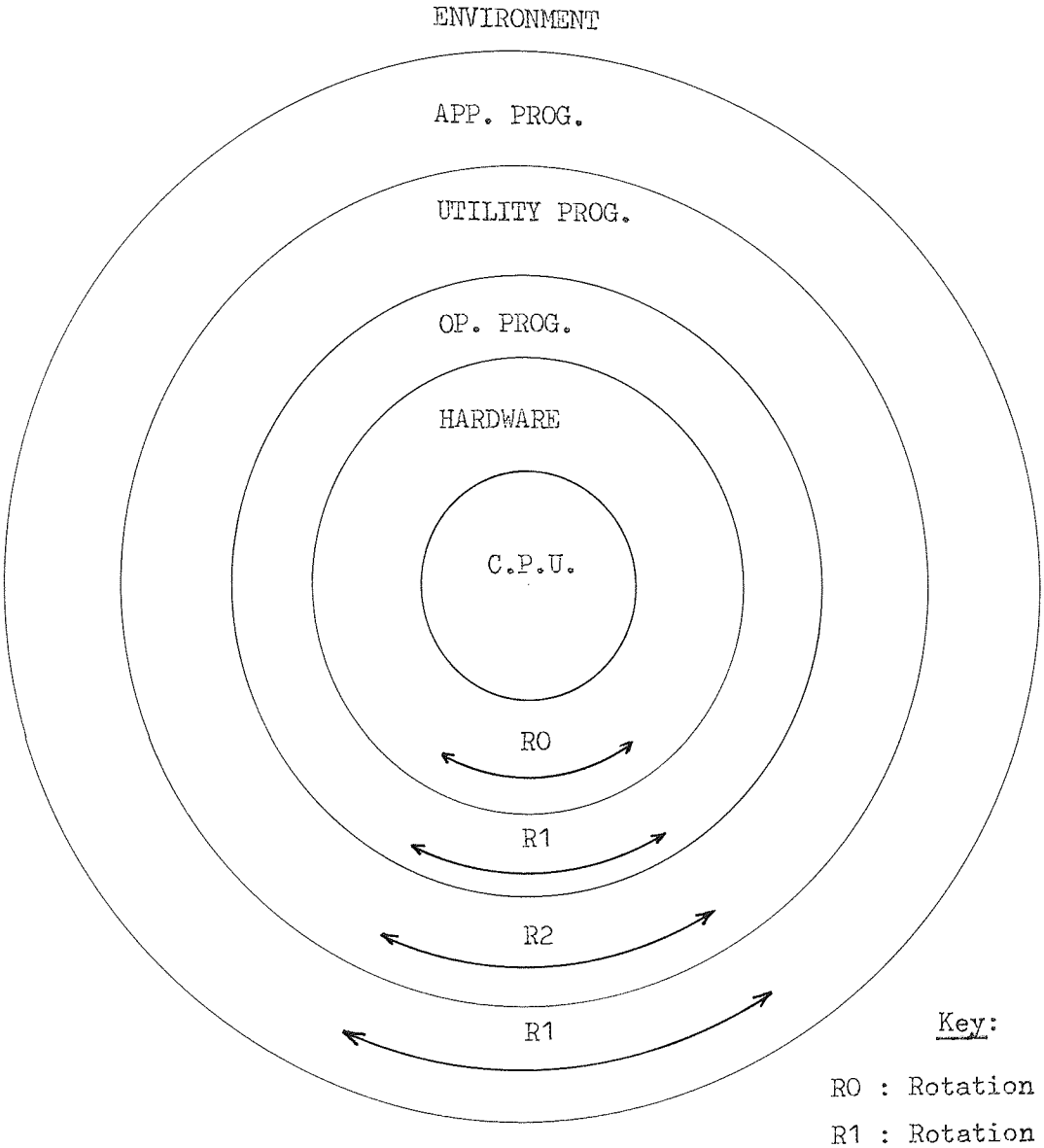
Software can be dissected into its three main constituents as shown in Figure (15). The parts are, (a) operating programmes, (b) utility programmes, (c) applications programmes. In this figure the principal constituents of both hardware and software are represented by a set of concentric discs. Each disc is free to rotate around the centre, i.e. the central processing unit, irrespective of other discs. Thus, when a user triggers one of his signals, i.e. a request/demand, a number of co-ordinated rotations of a number of discs occur in order to release and to activate the required mechanism (1). That is, the

---

(1) The term user is used in its general sense to include anyone who belongs to the M.I.S.

outermost disc, the user-software interface, which may in some environments be done through special programmes as in time-sharing operation or human activity in other less automated environments, receives the message. This message subsequently transforms into a set of signals. These signals can be transmitted either serially or in parallel depending upon the characteristics of software's structure. The resulting signals, however, are initiated to activate the rotation of the individual discs so that all parts of the required mechanism are properly aligned. Consequently, the manner in which these discs are coupled to each other determines the level of flexibility built-in the available software. Also the mode of coupling, i.e. whether strong or weak, may indicate whether or not the user controls or is controlled by the service offered to him.

Therefore the internal structure of software is regarded to be not only the concern of software designers, but also that of users in general. The question of software structure has thus drawn the attention of many software theorists and practitioners who have proposed many models each describing how software should be constructed. Few interesting examples are discussed below within the framework of the three-fold design criteria, i.e. flexibility, independence and impartiality.



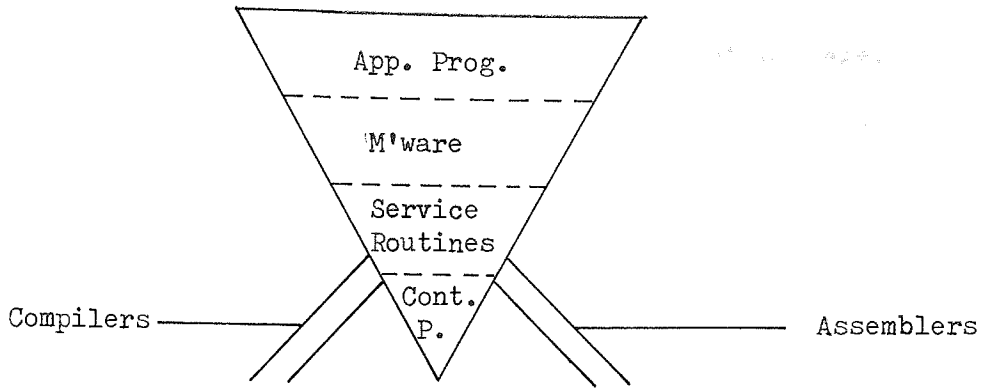
Key:

RO : Rotation of hardware
R1 : Rotation of Op. prog.
R2 : Rotation of Ut. prog.
R3 : Rotation of App.prog.

Dissection of Computer Systems

Figure (15)

1. The Inverted Pyramid



The Inverted Pyramid

Figure (16)

The idea behind the Inverted Pyramid is to show the weaknesses of software regarding the inability to maintain an application and to extend it freely (2). According to the model, software's internal structure does not contain assemblers and compilers as members, but as far as stability is concerned both are indispensable. Also the model shows that extending and, hence, maintaining an application do not depend upon free-will. Rather these functions are constrained by the structure of software as illustrated by the Inverted Pyramid, the idea behind which is that any change in the underlying software does not contain the old version as a subset. As a direct consequence of this change the application programme may not be able to work. Thus demonstrating the strong interdependence between software, as defined conventionally

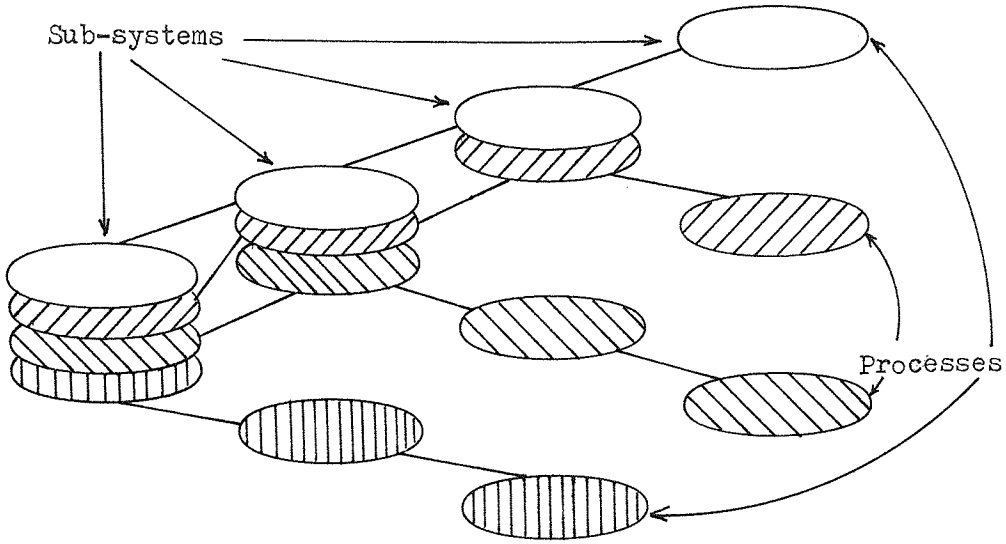
---

(2) d'Agapeyeff, A., Discussion, In P. Naur; B. Randell, (Eds.), "Software Engineering", (Garmisch Repprt, Scientific Division, NATO, Brussels, 1969), pp.22-23.

to cover the bottom two layers of the Pyramid, and applications programmes. In other words, the data processing manager who runs an application package is not free to change or develop his own software because they may affect the workability of that package. That is, the package may not only affect his staff and users, but also it can reduce his ability to change.

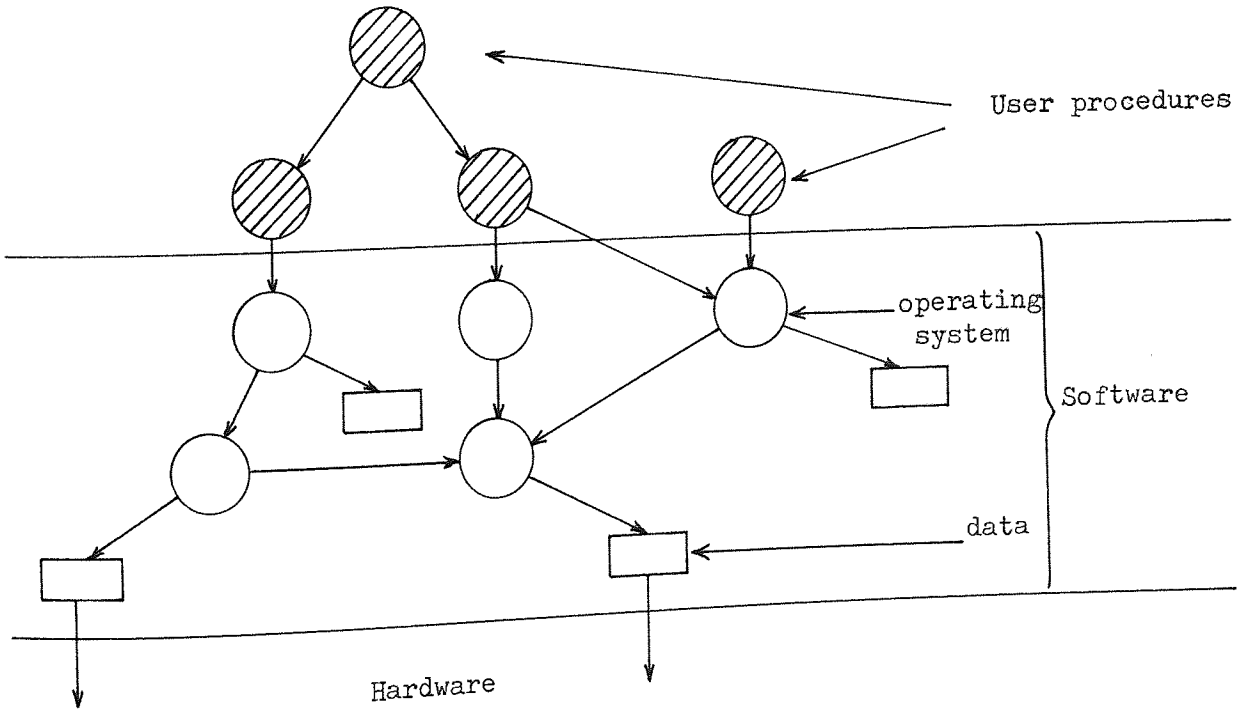
The model also incorporates a layer called middleware. If standard software, i.e. operating programmes, etc., which is supplied by manufacturers is conceived of as an abstract machine of the real one, then middleware is similarly seen as another abstract machine between the user and the software and is provided by the user. Consequently, the principal purpose of the concept of middleware is to reduce the dependence between user's applications and manufacturer's software which may result in more freedom to change. That is, middleware is user oriented and it is tuned for his applications. Thus, middleware covers functions such as file handling procedures, real-time schedules, etc.

2. Operating Procedures



Flexible Sub-systems

Figure (17)



User-Software Interaction

Figure (18)

Another model of software is that shown in Figure (18) in which the media that links users' procedures to hardware is that of operating procedures (3). The message of the model is that of building more flexibility in software's structure. It is suggested that this can be achieved by reducing, and eventually eliminating, the degrees of discreteness normally associated with procedures supplied through some external channels, specifically operating procedures. Thus if users are allowed and provided with the facilities needed to enable them to adapt and/or rewrite whatever procedures they may find unsatisfactory, the users then and as a consequence of this promotion of understanding, may bring software under their control. One form of such control is that of merging both users own-produced procedures and those supplied from outside. Consequently the users can build as many sub-systems as they may need by the use of all that is available to them as illustrated in Figure (17).

It follows from the above two examples that dependence upon external sources of programmes is not at all recommended not only for applications programmes but also for others such as operating procedures. More significantly is the user's ability to build-up new procedures from already existing ones. In this respect current package architecture as comprising a number of highly

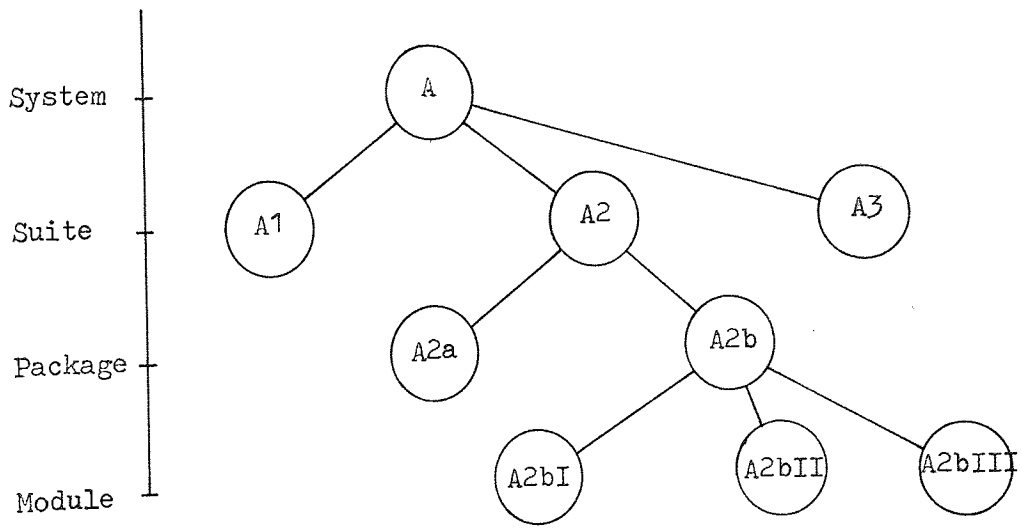
---

(3) Moore, B.J., "Operating Systems for a Range of Computers", In C. Boon, (Ed.), "Operating Systems", Infotech State of the Art Report No.14, Maidenhead, Berks., 1972, pp.225-239.

dependent chunks of machine coded instructions precludes the multi-use of its constituents in other suitable applications as well as it makes the task of adapting it to suit the operating conditions of the user very difficult.

HIERARCHIC STRUCTURE

In the foregoing no particular reference has been made to the type of intra-relationships that may exist within software. To this the concept of holons is of immediate pertinency. A demonstration of the state of the art of software's trends in hierarchical organization as well as the use of terminology may reflect some of the misconceptions and indeterminateness.



Hierarchic Architecture

Figure (19)

Figure (19) shows a typical examples of the architecture of software

or part of it. This model suggests the construction of a matrix to contain a complete system, e.g. an application package (4). The horizontal dimension of the matrix is not to exceed ten columns, i.e. boxes whereas the vertical dimensions entails four levels. The resulting matrix is partially ordered by the relation is contained in. The elements of the matrix, i.e. the boxes, do not, however, lexicographically contain one another, but the execution of one includes, potentially at least, the execution of the boxes below it. Accordingly the top layer is given the name system; below it comes the level of the suite which is followed by that of the package, and finally at the bottom is that of the module. A number of such matrices, it is suggested, can result in a complete corpus of software.

Comparing this model with that of the holons, one immediately recognizes that the above hierarchy is deficient in more than one respect. To start with the boxes, apart from being defined as modules, have had no mention made of their properties and the rules governing their behaviour, such as their modes of couplings. Also, the term box does not carry with itself specific meanings, and, thus, does not reduce any uncertainty regarding what a module is. Moreover, by constructing a ceiling, the model is in effect limiting or even prohibiting the evolution and/or extensibility of software into higher orders. What has been

---

(4) Pyle, I.C., "Hierarchies: An Ordered Approach to Software Design", Infotech State of the Art Report No.11, op.cit., pp. 253-272. Pyle uses the term boxes and modules to refer to the same thing. Also his hierarchy is directly relevant because I.B.M. and I.C.L. use the same hierarchy for their packages, specifically the bottom two levels, viz. package and module.

said about the use of the term box is also applicable to the other terms, i.e. system, suite, etc. As regards the relationship this model disregards the response of the subordinate box to its ordinate. That is what the concept of holons calls the input hierarchy.

Another model which elaborates on the relationship between any two modules is that saying

> "A module that does not call any other module is of height zero. Whereas a module which calls one or more other modules is of height one higher than that of the highest among the ones called by it" (5).

Evidently this rule is not only more concrete than the previous 'is contained in' relationship but also it is better because it offers more flexibility and dynamics.

The concept of holons can be seen to incorporate that rule. In a response to a certain stimulus, the holon in question may select and bring into action a whole hierarchy of holons to carry-out the designated demands. However, the resultant hierarchy i.e. output hierarchy, may change as stimuli change within the course of time. Conversely the same holon, irrespective of its hierarchical level, regards itself as a subordinate part to some other supra-holon. Consequently the holon's role reverses to that of feedback, through filtering and reporting in response to the stimuli it may receive.

---

(5) Dijkstra, E.W., "Complexity Controlled by Hierarchical Ordering of Function and Variability", In, Naur, Randell, (Eds.), op.cit., pp.181-185.

It may have become clear that the matrix model and all similar hierarchics are less flexible than that of holons. For the latter, i.e. the holons, does not only avoid to relate holons to specific hierarchical levels and thus restricting their transferability from one to another; the hierarchy of holons is built, by definition, to promote and cope with the independence of its constituents (6). Both dynamics, i.e. transferability and communication are regarded to be prerequisites of flexibility. Communication is enhanced in the case of a holon by recognising that a holon, regardless of its level, may encounter some unroutinized, i.e. unprogrammed, situations. Thus, when designing a holon, an explicit communication requirements should be identified and catered for including the uncertainties.

Additionally, the inverse relationship between flexibility and mechanization is also identified and stressed by the holons concept. Granting that both complete flexibility and zero uncertainty are unattainable due to the finiteness of involved attributes, e.g. the designer's brain, core-size, etc., then, the confinement and formal allocation of parts/modules to specific different levels, as that of Figure (19), may be taken as faulty in the respect that it may increase the finiteness itself. Permanent association of a part to a particular level imposes restrictions on the use, not only of the part

---

(6) Koestler emphasizes that the only way to cope with independence is through communication. When this is established an integrated whole, i.e. vertical and horizontal structures, may result; c.f. Chapter 2, p.55.

itself, but also of its constituents. Consequently restricting farther the available resources. The architecture of all packages and almost all users' own programmes is a case showing such permanency. All them, specially packages, are signs of wasted resources. This is because all design and programming efforts that may have gone into a package cannot be restored back and used again in any economical way. Instead, a new programme has to be written from scratch under the constraints of the old whether packages or programmes. Thus by the intentional dropping of such permanent allocations the concept of holons strikes a distinguishable balance between economic feasibility and dynamic structuring of even complex programmes. Also this dynamic property implies flexibility which is, once again, essential ot counterbalance the effects of automation.

CONCLUSIONS

This chapter reveals that if the user is to control his software, including applications programmes, he will have to develop his own buffer, i.e. the middleware, between him and the manufacturers software. This is because the supplied software impairs the users efforts to extend or change his applications programmes. Therefore there is little doubt about the user's need for rigorous design method approach and competent designers to write such a buffer. Also the present package architecture as inseparable wholes prohibits the use of its parts in building new applications and consequently leads to duplication, i.e. self contradictory. The concept of holons is seen to provide self-consistent framework for a solution.

CHAPTER 5

AN EXAMPLE OF A TYPICAL APPLICATION AREA AND ITS MATCHING COMPUTER PACKAGE

In the following discussion is directed towards an important application area i.e. production scheduling together with a typical computer application package of that area, viz. CLASS (1).

Production scheduling is reviewed in order to identify its characteristics and uncertainties. This is done by looking at scheduling within the wider context of production control and planning activity. In this pursuit an account of network analysis techniques is provided.

CLASS is examined to reveal the technique it uses and the facilities it provides. Problems of compatability and interaction between the package on one hand and both data processing departments and the larger area of production control on the other are considered.

The discussion terminates by investigating CLASS within the framework of the criteria that have been developed so far in previous chapters.

---

(1) CLASS stands for Capacity Loading and Scheduling System. It is a stand alone package, i.e. it is not part of a larger system. The minimum configuration requirements are: any System /360 or /370 with 32K; 2 x 2311 disks; Line Printer; Card Reader; operating system DOS and OS. It is written in Assembler. It is released in object code. Only the User Manual is available which gives an account of the facilities provided by the package and the way they can be used. It also describes the input requirements and outlines the output.

SCHEDULING AND ITS ORGANIZATIONAL CONTEXT

Scheduling is taken to constitute one part of the larger activity of production control and planning. In this context, planning is mainly concerned with the development of a method for accomplishing a production requirement (2). Consequently planning as such has two types. The first is the long-range planning which deals with decisions such as plant investment, acquisition of resources, etc. whereas, the second covers short-range planning such as the deployment of the given resources to achieve the desired goals. However the process through which resources are used is designated as scheduling, thus it is described as the assignment of specific times for projected operations either of the plant as a whole or of individual work centres. This is supposed to be done by taking into account due-dates, processing time, and a time allowance for the average, and variance in, waiting time of operations at various machine facilities (3). However, scheduling as such must be distinguished from the related activity of sequencing which is described as the order in which jobs are serviced (4)

---

(2) Niland, P., "Production Planning, Scheduling, and Inventory Control: A Text and Cases", (Macmillan Co., London, 1970), pp.51-81.

(3) Sisson, R.L., "Sequencing Theory", In R.L. Ackoff, (Ed.), "Progress in Operations Research", Vol.1, (J. Wiley & Sons, N.Y., 1961), pp.293-326.

(4) Churchman, C.W.; Ackoff, R.L.; Arnoff, E.L., "Introduction to Operations Research", (J. Wiley & Sons, N.Y., 1957), pp.450.

Considering production control activity as a hierarchy of holons makes it possible to identify some of its intra-relationships that exist between the various holons. The hierarchy extends almost over the entire hierarchy of most manufacturing companies. On receiving an order for a product the whole mechanism of the hierarchy may, consequently, be triggered into action. According to the concept of holons the mechanism involves some automated as well as unautomated procedures (5). The mechanism has its specified policies, i.e. its canon, to control its routine or quasi-routine actions along with unroutinezed procedures dictated by, and required to enable it to deal with, the inevitable uncertainties as it continues to interact with other external and/or internal environments. Assuming that the mechanism has been reactivated, hence its basic output hierarchy as shown in Figure (20) may include, beside others, the following branches:

a - Order analysis

b - Inventory control;

c - Work-In-Progress control;

d - Loading and Scheduling;

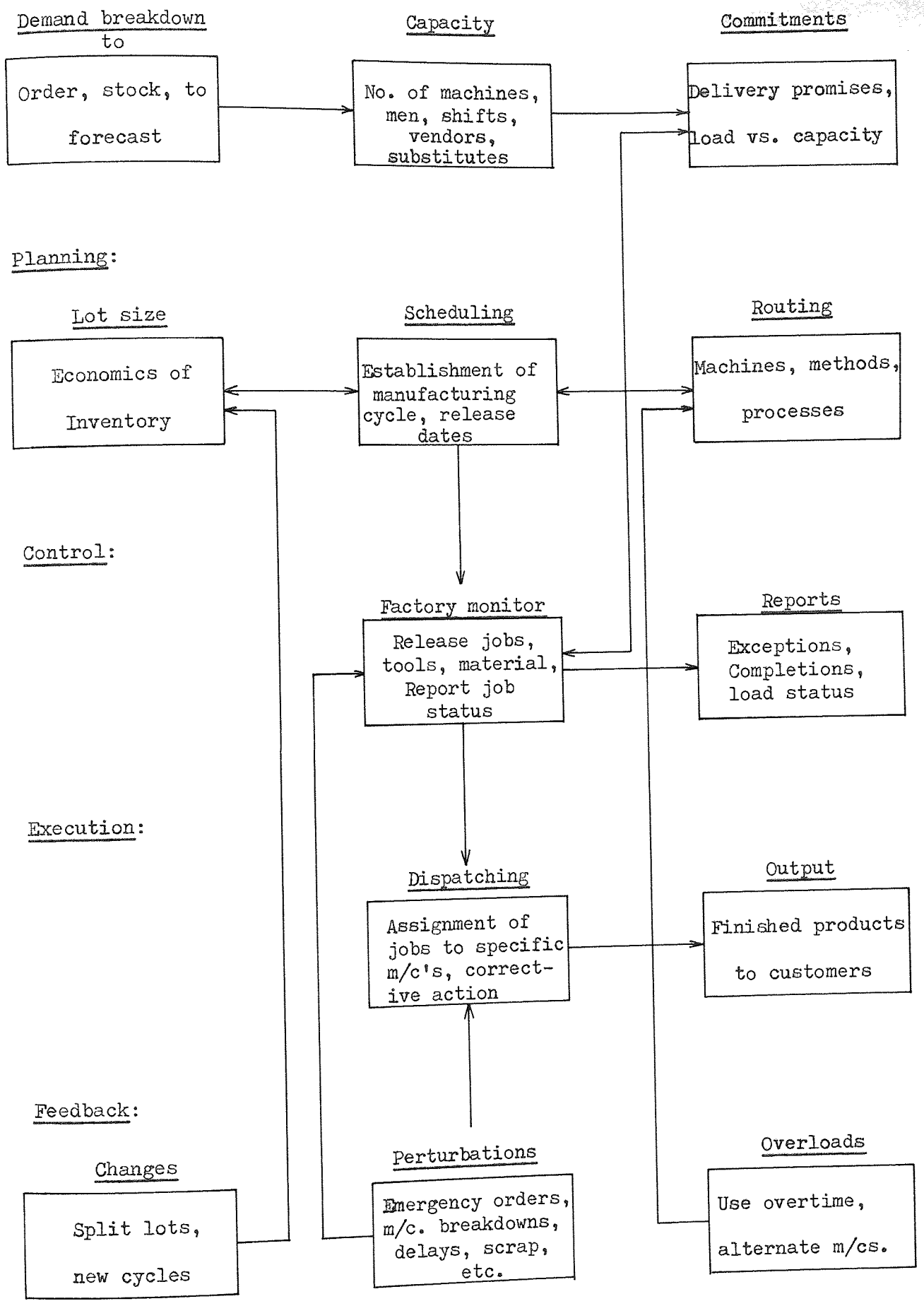
e - Purchasing;

f - Demand Forecasting;

g - Performance analysis; etc.

---

(5) Automation does not only refer to computerized procedures, but also any well-specified actions.



Rowe's Operating Characteristics of a Production System

Figure (20)

An appreciation of the input requirements and output of the scheduling activity shows the entwining nature of these branches. As far as production planning and scheduling are concerned, a statement of production requirements, over a specific time interval, is pre-eminent. Such a statement is either produced on the basis of a demand forecast covering the same time interval or on the basis of received orders, or a combination of both. In whatever case, the statement constitutes the primary document according to which needs are calculated and resources are, thus, allocated. This is done by translating forecasts and/or orders into material requirements and operations. Material requirements cover raw materials in addition to work-in-progress. Accordingly stocks can be adjusted and replenishments are ordered. However, in this respect work-in-progress constitutes a further critical common factor which strengthens the interdependence between demand/order breakdown, inventory control, and scheduling since work-in-progress represents work either previously and/or currently scheduled. When this work slips behind, schedule rescheduling may become necessary and, thus, stocks and delivery dates may be affected as a result.

The second face of forecast/order breakdown is that concerning operations to be performed, i.e. the added value. Here an operation can be broadly described in terms of: (a) men, who may be classified according to some criteria into skilled, semi-skilled, etc.; (b) machines which may be grouped into groups of like machines, etc.; (c) time, i.e. in terms of manufacturing cycle. This is taken

to denote the average elapsed time between the date a job is released to the shop and the date it is shipped (6). The main components of such a manufacturing cycle may include: (i) actual working time; (ii) waiting time; (iii) transportation time. Thus, a manufacturing cycle can play a critical role both in making delivery promises and in planning.

Consequently the entwinings within the hierarchy are reinforced through the manufacturing cycle. Again purchases may be requested and stocks can be adjusted, and work is, sometime, sub-contracted in accordance with the manufacturing lead time, i.e. cycle, and the volume of demand/orders to be scheduled. These schedules take into account other pertinent factors that can affect their outcomes such as: (i) identification of alternate routines, i.e. permitting the use of similar machines or groups of machines, and/or allowing variations in the sequence of operations to be performed, etc.; (ii) sequence control, i.e. to determine the best-possible sequence according to which batches of orders are assimilated with the purpose of reducing as much as possible the set-up times of jobs and tools on the machines (7). These measures are designed to increase the flexibility and workability of schedules in case of uncertain occurrences.

---

(6) Poirier, C.C., "Automated Data Processing for the Corrugated Box Plant", In J.F. Pierce, (Ed.), op.cit., pp.416-432.

(7) Sisson, op.cit., pp.304-308.

One schedule may differ from another with respect to the amount of details included in each. The extent of detail can be regarded as arbitrary only insofar as the manner in which the person responsible views the orders-mix. Regardless, the depth of details is considered as a function of the requirements' mix and how close the current level of operations is to the available capacity, i.e. the degree of tolerance, and the degree of labour' versatility, i.e. the extent to which workers can switch from one type of operation to the next and from one work centre to another (8).

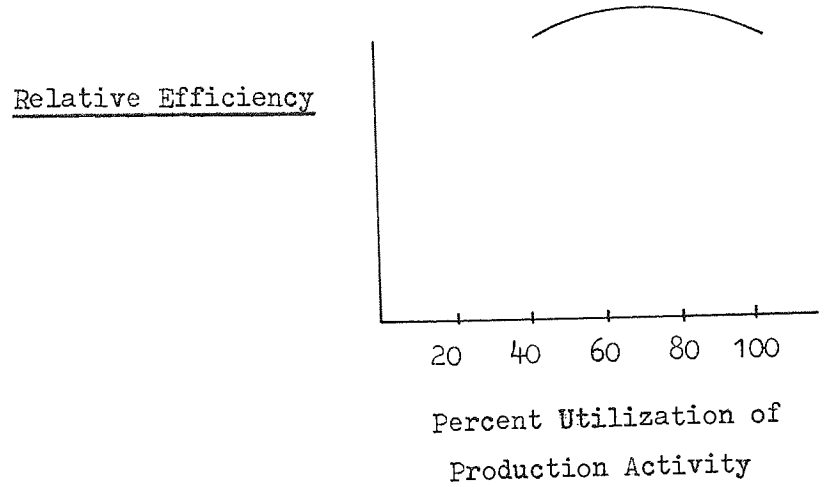
Given a list of the work to be done, i.e. production requirements, a typical scheduling application may, therefore, include the following features. (1) Plant capacity which may be broken-down by individual machines, by groups of the like machines, by department, by work centre, by manpower skill classification, or by any combination of these. (2) Amount of capacity needed to produce the work required. This, for example, can be calculated from the operation sheet of each required part. (3) Allocation of available capacity to the shop orders being scheduled. (4) A cycle for rescheduling to cope with delinquent orders.

It is important to note that plant capacity should not be allocated fully. There ought to be an extra capacity left unallocated to cope with unforeseen difficulties. When there is no

---

(8) Niland, op.cit., pp.66-75.

spare capacity the degrees of freedom of scheduling fall sharply resulting in more rigid schedules and inability to handle change as shown in Figure (21).



Spare-Capacity

Figure (21)

Scheduling Techniques

There are two common approaches to scheduling used in computer applications packages. The first is that of network analysis whereas the other is that of simulation. Network analysis covers both Critical Path Method, i.e. CPM, and Programme Evaluation and Review Techniques, i.e. PERT. The fundamental elements of simulation are: (i) a mathematical model of the process or other phenomenon being investigated; (ii) a sample

of inputs (9). Regarding scheduling and sequencing the appropriate technique has to cater for stochastic situations due to the formation of queues at the various machine facilities (10).

The object of network analysis is to determine the minimum time interval in which a job/order/schedule can be completed, and to identify the critical path. A definition of the critical path for either CPH or PERT is that series of activities having a precedence relationship to one another, the delay of any one of which would result in a comparable delay of the completion of the whole job/order/schedule (11). The term activity is used to designate a task or part of the whole job. Such activities are coupled with duration estimates. The term event denotes either the start or the end of an activity or number of activities. Consequently an activity is regarded to describe what may happen between the event with which it starts and the event with which it ends.

Other network paths that are not critical offer float, or slack, which is the total amount of time that non-critical path activities in the network can be delayed without affecting the overall job completion date.

---

(9) Spur, W.A., Bonini, C.P., "Statistical Analysis For Business Decisions", (Richard D. Irwine, Inc., Homewood, Illinois, 1967), pp.407-426.

(10) Morgenthaler, G.W., "The Theory and Application of Simulation in Operations Research", In R.L. Ackoff, (Ed.), op.cit., pp.363-419.

(11) Langefors, B., "Activity Network for Planning and Scheduling", BIT, 1962, 2, No.1, pp.21-34.

The definition of activities and the choice of events are arbitrary, but there is an obvious advantage in using breakdowns that correspond to the everyday concepts employed by people who will use the network and the analyses based upon it. The level of detail, as outlined before, in the breakdown, depends upon the magnitude and complexity of the project itself and the organizational levels that will utilize the network. Large and complex projects may use one network for over-all project management, and other more detailed networks for use at lower levels of management.

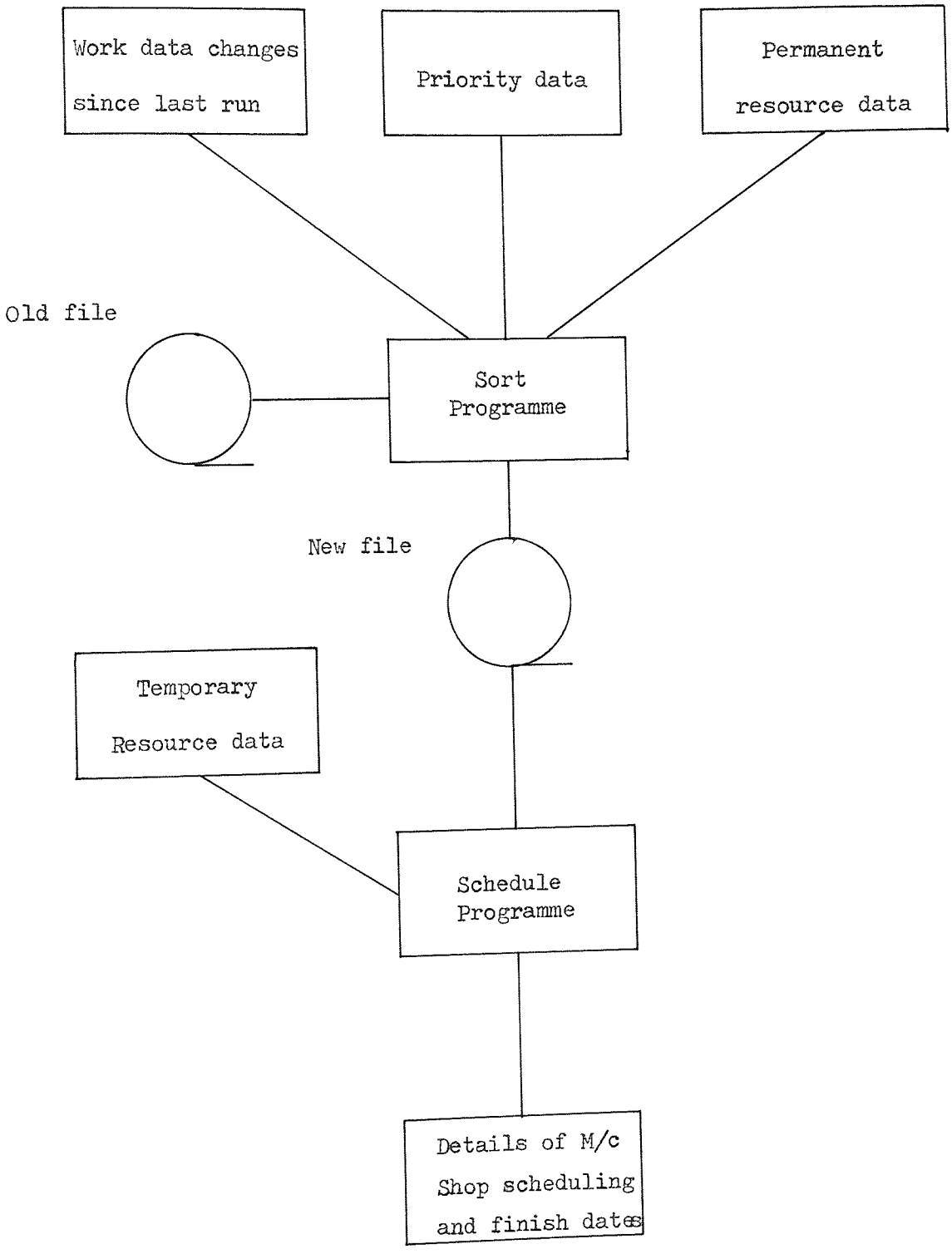
There is a difference between CPM and PERT regarding the use of probability. PERT, on the one hand, associates uncertainty with the estimated activities durations. This is achieved by using three estimates for the duration of each activity; optimistic, pessimistic, and the most likely. Assuming the probabilistic beta-distribution for these estimates, the three durations are used to calculate an average or expected duration of each activity. The expected times for all activities are then used in a network analysis to determine the critical path and the amount of float in the same manner that is done for CPM. Having determined the critical path, the data related to the expected times for the activities on the critical path and their variances may be combined and used to estimate the probability of meeting alternative target completion dates. However, the CLASS package uses CPM and therefore does not make use of probabilities.

The CLASS Model

The model covers the following functional areas: (i) Work-in-progress; (ii) scheduling; (iii) performance analysis, as schematically shown in Figure (22). The package operation involves the following. (1) Creation and/or updating of the W.I.P. order file that contains all production data relevant to the scheduling process. (2) Scheduling to infinite capacity, i.e. unlimited run with no restrictions. (3) Scheduling to finite capacity, i.e. limited run with limited resources. (4) Short term sequencing to finite capacity, i.e. sequencing individual operations in the short term according to priority rules. In all cases time periods are multiples of one day.

A. Scheduling to Infinite Capacity - Forwards or backwards.

Using both the timing and the statement of requirements it becomes relatively easy to perform the infinite capacity loading phase. Each network may then be analysed and the capacity required for the operation is deducted from the particular machine/facility. This deduction is made at the earliest start time, EST, or latest start time, LST, of the activity, i.e. the operation, and a different deduction list is maintained for each facility. The list is divided into time periods whose magnitudes are selected according to the management's plan. For each machine, or group of like machines, a histogram can be printed, comparing both capacities required and



Overall System Flow Chart For Job Shop

Figure (22)

available. These histograms demonstrate the overall state of facility utilization and thus may indicate to management where over and/or under-loadings may have occurred. Management can accordingly perform its control function to smooth the situation. Another possible use of the histograms is to plan man-power utilization.

Thus the main characteristics of this phase are:

i) it is possible to develop long term loading to infinite capacity.

ii) activity times are calculated by CPM for the two passes, i.e. forward and backward.

iii) when an activity's duration exceeds that is available the package automatically tries to:

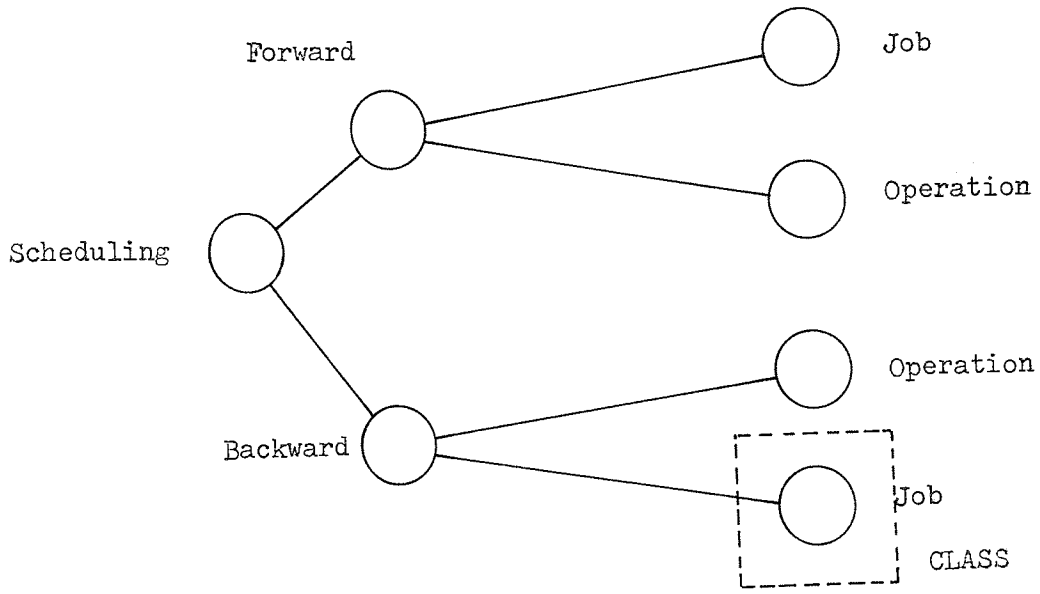
- reduce the transit time, and/or
- split batches, and/or
- overlap batches,

subject to management approval.

B. Scheduling to Finite Capacity

In this context, there are two modes of scheduling, viz. forwards and backwards. Each can be equally performed on individual jobs or operations. However, as far as CLASS is

concerned it restricts its users to backwards scheduling on individual jobs as shown in Figure (23) by the dotted line.



Finite Scheduling Four Techniques

Figure (23)

The purpose of this technique is to prevent jobs from being loaded past today and so the critical path times should be adjusted, if feasible, to fit the network into the time available. Preserving priorities, networks are loaded on to the facilities at their latest short times, LST. Any overloads are avoided by trying to load the operations at earlier times and not sooner than the early start time. If a network, already reduced in time scale, must start earlier than today, then it should be reloaded from 'today' and forward scheduled.

It is envisaged that servicing time can be minimized by not loading operations until late start time is reached. Besides a reduction in W.I.P. costs, delivery date is considered. However, individual operations cannot be chased.

Consequently, the main characteristics of this phase are:

i) in case of overloaded capacity time shippage or delay are offered

ii) alternative machine or group of machines is automatically tried

iii) jobs are loaded in priority order

iv) individual operations are not loaded

v) work cannot be loaded in the forward mode from 'today'

vi) work can be loaded in the backward mode from delivery dates.

The output reports include lists of machines with the operations arranged in chronological or priority order. Overall loading digures under the actual conditions of limited capacity may be available.

W.I.P. status report.

C. Short Term Sequencing

When infinite scheduling is only to be performed it has to be supplemented by a short term sequencing whereby the accumulated load can be sequenced according to priority order and a list of jobs is released to the workshop. When jobs are scheduled, e.g. to particular weeks, then detailed sequencing ought to follow, specifically when machine groups are used. The load allocated to a group has to be arranged in priority order or in a sequence that accounts for set-up conditions, e.g. changeover time can be minimized when jobs with similar set-ups are run in sequence.

In case of back scheduling nothing is done to make sure that there is work available at the beginning of the scheduling stage so that the short term run in this case must take the results of the long term scheduling and pull some of the work forward before sequencing the work at the work centres. It can also 'pull forward' work in order to fill unallocated capacity. Before embarking on the action to 'pull forward' the package checks on the state of W.I.P. in the light of the pre-determined policy of W.I.P.

Consequently short term sequencing of operations is performed by CLASS. Also work can be 'pulled forward' to fill whatever capacity that may be available.

D. Priority Factors

a) Priority factors that are automatically included:

i) management assigned value such as customer or order importance

ii) based upon float in the jobs network, hence

$$\text{Priority} = \frac{1}{\text{delivery date - "today's" date}} \Big/ \text{Sum of the remaining processing times}$$

iii) the package does not pay special attention to either the job's complexity or the number of its constituent operations

iv) when a job's plan is altered the package offers the facility to maintain the job's schedule by, for example, reducing its transit times, etc.

v) the age of a job in progress is considered

b) In case of short term sequencing the package considers the following additional factors:

i) whether an operation is already started

ii) length of operation

iii) the package does not question whether an operation is delaying next operation

iv) the package does not compare loading levels of work centres.

DESIGN IMPLICATIONS

Despite its apparent popularity CLASS as briefly outlined above constitutes the antithesis of the criteria presented here so far. This may become clearer as it is seen within the context of each criterion. Considering definitions/solutions and their inevitable change, CLASS as a ready-to-implement solutions represents a static definition/solution to the scheduling problem. To start with, it is not adequately documented since there is no available literature about its design or structure. Consequently the user's ability to modify it or even adapt it to his own environment is very much reduced. Also, sticking to one technique of scheduling, i.e. to one point of view, without providing facilities to change the technique or add another such as forward scheduling of finite capacity is an extra sign of the package's static philosophy. This type of definition is bound to cause problems to the user particularly when it is fitted into the wider context of production planning and control which are likely to bring uncertainty very closely to the scheduling procedures. This uncertainty can take different forms. The most relevant is that associated with capacity. It is most likely that the actual capacity will be either more or less than that planned for the plant. This can be explained in terms of: (i) machine breakdowns; (ii) shortages of stocks; (iii) absenteeism; (iv) planned maintenance. Also, uncertainty can be introduced through inaccurate or imprecise timing of operations, i.e., durations of activities. The third source of uncertainty is

that characterized by demand/order fluctuations. This can be attributed to: (i) unexpected customers' orders/demand; (ii) excessive spoilage or rework; (iii) lost shop orders. Uncertainty may, therefore, affect the reliability and flexibility of the scheduling technique insofar as the amount of unscheduled orders that can be cancelled. To minimize the effect of uncertainty the scheduling solution should tend to be more dynamic. Conversely, the technique used in CLASS for finite capacity is job oriented, i.e. it treats the job as a whole and does not break it down into its constituent operations. By doing so the package does not offer means to compare progress against schedule to identify bottlenecks. Consequently, the user who is interested in operations may have to wait for twenty six hours, assuming that the data processing schedule allows, needed to run both infinite scheduling and then short term sequencing (12). Thus the package as it stands now represents static solution which becomes worse as it is inextensible.

The rigidity of CLASS becomes explicit when it is subjected to the three inter-related design criteria, viz. bias; flexibility; independence. Besides the designer's own bias in selecting the

---

(12) Because of the incompatability between the requirements of the particular user and the output of the package, large number of production controllers ignore the produced schedules and develop their own, mainly in the form of Gantt chart. A simple example that demonstrates such incompatabilities is that the sequence of lists is by loading sequence, not-part-number, and cross-referencing is thus much reduced.

techniques incorporated into the package, bias is also extended to confine the package to I.B.M.'s hardware and software environment. This is implied by the use of the Assembler language which compels the user to base his software and hardware upon I.B.M. products. Such is a direct consequence of choosing to write the package into a low level language whose role as a principal factor in determining the stability of the user's whole software has been illustrated in the Inverted Pyramid model of software (13). Consequently, having committed himself to such a package, the user's future developments will tend to be biased in favour of I.B.M. products in order not to disturb the stability of his data processing function. This is evident in almost all computing departments by being oriented to specific manufactures.

The rigidity of CLASS is also obvious in its size. As a large programme, 32K, without being truly modular its constituent parts are highly dependent on each other. That is, its parts cannot be separated from each other or used as building blocks in developing other programmes. This, in addition to other factors such as its static definition, low level language and the inavailability of source code listings, makes it very difficult for the user's staff to update the package or to locate errors. As a consequence, the user of CLASS finds himself highly dependent upon I.B.M. to maintain the package leaving his staff primarily concerned

---

(13) Cf. Chapter 4.

with writing CLASS parameters. All these factors lend the package as a source and cause of many problems particularly when it is seen within the framework of its function.

The purpose of the package is to provide data that may help management in its production planning and control functions which are characterized by dynamic interactions and change. Management requirements often change in accordance with change in conditions or situations. To meet adequately such requirements the data processing staff have to have an adequate understanding of the architecture of the package which is denied by I.B.M. Acceptance of such a deprivation can be attributed to failure either of understanding the mechanics of production control activity and/or the inter-dependence between the package and other applications programmes. Concerning the former the current attitude of existing systems analysts is demonstrated by five out of five interviewed analysts who maintained that the technical aspects of scheduling is not their concern (14). Considering the latter the problem involves the attitude towards M.I.S. In this context, larger companies such as car manufacturers tend to develop their M.I.S.'s by implementing

---

(14) The question put to the analysts was: How do you rate your familiarity with scheduling methods and techniques? They rated their familiarities as being poor and justified this on the grounds of specialization. See Appendix 2.

a number of packages covering a number of closely related areas such as production requirements, BOMP; inventory control, ICS; etc. (15). The approach used in coupling the various packages is a typical componentoriented design which is emphasized by the fact that CLASS, as well as other packages, are also stand alone packages. The application of such an approach is intended to reduce manual-automated functions interactions and incompatabilities to hide the limitations of the individual packages. Thus any subsequent assessment pertaining to a particular package becomes increasingly difficult insofar as to dissociate the package from other related packages.

CONCLUSIONS

It is thus concluded that CLASS, as an example of manufacturers' computer applications packages, both reduces the degrees of freedom of its users and degenerates their design abilities when analysed from the viewpoint of:

1. Flexibility: this is much reduced because
   (i) It represents a static definition of scheduling by imposing a particular technique on its users which can be incapable of handling varying requirements as situations change.

---

(15) Approximately 71% of package users have two or more packages. Also 70% of all packages are provided by the manufacturers. In all production control applications the packages are provided by the same hardware manufacturer. See Appendix 1.

(ii) It is a closed-end system and thus inextinsible.

(iii) Its usability is limited to I.B.M. configurations. This is because it is written in Assembler. Thus it is general purpose package only within the sphere of I.B.M. computers.

(iv) It is highly specialized, i.e. its parts cannot be incorporated in other applications designs.

(v) Its design is not properly modular. Therefore its parts are inseparable.

2. Bias is much increased because:

(i) It represents its designer's concept of scheduling that cannot be modified because of lack of data concerning its structure.

(ii) It is totally biased to I.B.M. due to the fact that it is written in low level language.

(iii) It is extremely ordered, in the sense that it can be used in any other application.

3. Independence is much reduced because:

(i) Its parts are highly dependent upon each other because of the built-in strong couplings or order which permanently link these parts to each other.

(ii) It is wholly dependent upon I.B.M. software support, that is it is not transferable.

Consequently, CLASS is a typical ad hoc bottom-up approach which is implied from the foregoing. Also it is a classic example which demonstrates and substantiates the argument concerning the likely effects of packages upon the experiences and abilities of systems analysts/designers whose function under CLASS is reduced to that of preparing the parameter cards of the package.

PROBLEM DEFINITION AND INTEGRATIVE SUMMARY OF PART 1

The activity of problem definition is believed to be unlimited and needs complete concentration in order to derive a reasonable definiton. Thus chapters one through five have been totally devoted to define the problem of the design of applications packages which is conceived of as comprising two classes of questions. The first class considers the definition of the package itself and asks is it a design problem of just a computer programme? Consequently is it confined to practitioners with the exclusion of the users? Or is it a design problem of an organizational activity? If so, does an organizational activity in one environment correspond exactly to a similar activity in another environment? If not, why and what are the factors that can be attributed to such variation? etc. The second class examines the relationships that relate a package to its wider contexts, thus it includes questions like: does a package relate to an M.I.S.? If so, what is the designer's concept of the M.I.S.? How an M.I.S. can influence the design process? Does a package relate to software? If yes, in what manner? How software's structure and definition can influence the package? What is the relationship between a package and both its producer and user? What are the consequences of a package upon its environments, and so on.

In attempting these questions, chapters one through five have aimed towards a number of specific objectives. First to identify as many attributes and dimensions as can be conceived of in order to develop a fundamentally systematic conceptual framework for applications packages. Second, within this framework various drawbacks of current designs are identified and methods of averting them are proposed in the form of design criteria. Third to show that an understanding of the preliminaries of systems concepts provides flexibility in so far as interpretation of these concepts are concerned. That is depending upon the degree of familiarity with, say one concept, it can be interpreted in various ways to suit various conditions. Fourth to demonstrate that the main causes behind the present unsatisfactory state of applications packages' design stems largely from not only the designers and/or users lack of familiarity with the fundamentals of the systems approach, but more serious than that is that of confusing the systems approach with incompatible traditional approaches. Fifth to emphasize that the underlying design approach is bound to be reflected in the behaviour of the package when it is: (i) on its own; (ii) interacting with other programmes, and in its effect on the data processing function.

However the sequence of the chapters does not correspond to the listed order of the purposes. This results from the logic of the process of constructing the framework which is built gradually by adding more constructs as the discussion has advanced from one point to another.

Consequently, the contribution of chapter one has been that of outlining the basic philosophy of the framework. It is argued there that definitions and solutions are incomplete and the former assume their adequate forms only when the latter have been sufficiently developed. Thus, a problem can be said to be truly defined only when it is solved indicating the ceaseless property of the iterative process between analysis and synthesis. Thus defining a package as a definition/solution of a problematic situation implies catering for change which is explicit in the interplay between behaviour and structure. The design criteria which affect this interaction and the scope of the package are: content; bias; flexibility. The relationships between these properties are:

The more content a package has, the more rigid the structure it exhibits.

The more content a package has, the more biased it may become.

Rigidity increases proportionally with bias.

Chapter two further elaborates on the theoretical basis of these criteria and concludes that 1) the designer's own bias and content can be transferred to the model/package through either: (a) copying existing problems into the potential computer version; or (b) inflicting new problems as a result of moulding the situation into his own pre-conceived model. 2) order has been seen to induce rigidity. This applies to both the human activity and the

package. 3) size is directly proportional with rigidity, and inversely proportional with independence. Therefore small modules will ensure both independence and flexibility.

However, the main contribution of chapter two is the adaptation of the concept of holons to offer: (a) a flexible definition of a package as a self-contained, rule-governed stable open system; (b) a dynamic hierarchical design of mudules or holons which implies: an open-end hierarchy; canons or permanent behaviour of holons must be flexible to reduce the effects of automation; catering for arborization or vertical structuring and reticulation or horizontal interlocking of holons.

Chapter three considers one environment, viz. M.I.S. which is defined to include the whole of the company, in which arborization and reticulation of holons may take place. In this respect, the chapter identifies two approaches of achieving this. The first is the top-down approach which is an interpretation of the systems approach and has not gained any significant popularity in practical application. The other is the bottom-up which is often misinterpreted and thus gives way to ad hoc designs, and unfortunately is the most popular. Also, the effect of ready-to-implement solutions upon the expertise and skills of data processing practitioners. The dependence upon packages marks the start of a vicious circle since the more a company opts for packages, the less experience its designers may get, and more dependent upon packages the company will become. This set up becomes less attractive to competent designers to stay

in the company, this, in turn, leads for more packages.

Chapter four discusses the problem of user independence. If the user values independence from the manufacturer, he may have to develop his own software buffer, or middleware, to increase his own flexibility regarding extensibility and modifiability of his software. Also, it is argued that the present package architecture as inseparable wholes prohibits the use of its parts in building new programmes and thus leads to duplication of effort, i.e. self-contradictory to the basic idea behind packaging.

Chapter five, substantiates the argument of the preceding four chapters. This is done by examining CLASS as a typical example of applications packages.

The remainder of this thesis discusses methods and proposes techniques that may lead to a solution. This is attempted from two points of view within a consistent framework, i.e. the concept of holons. Consequently chapter six tackles this issue from the viewpoint of programme design and architecture, i.e. a software context, whereas chapter seven takes up the same issue but from the viewpoint of the process of decision making and its elements and the consequences of indiscriminate mechanization upon the decision maker, i.e. an organizational context.

CHAPTER 6

THE IMPLICATIONS OF COMMUNICATION SYSTEMS UPON THE DESIGN OF APPLICATIONS PACKAGES

This chapter proposes a concrete programme design technique, viz. structured programming, which adheres to the top-down approach and consequently caters for the criteria of flexibility, impartiality and independence. This is to rectify the current misconception of modularity that results in ad hoc and illogical modular designs.

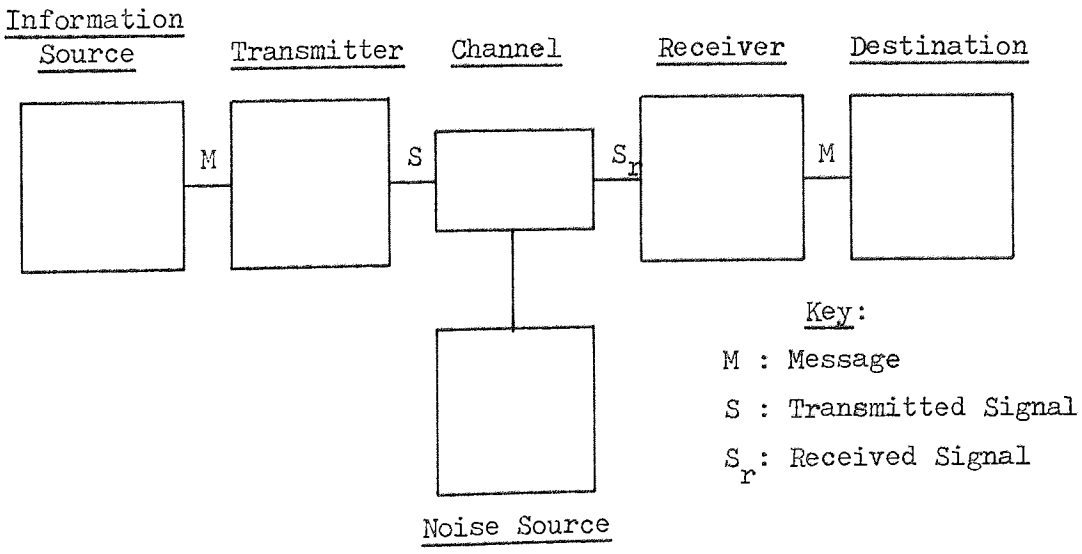
In order to achieve this objective an attempt is made to define applications packages, and software in general, in terms of their communication roles.

To arrive at such a definition, certain theorems of the communication theory and Ashby's concept of requisite variety are considered for their direct contribution in establishing the theoretical foundations of the definition and in deriving further properties for applications packages.

Also viewing a package within the context of a communication system (1) makes it possible to: (a) examine the design implications of the user-practitioner interaction as being components of the encompassing communication system (2); (b) determine the factors that affect the behaviour and the life-span of a package, i.e. the process of gaining entropy.

THE MATHEMATICAL THEORY OF COMMUNICATION - AN EXPOSITION

In the following an overview of the conventional communication model and the relationships between information, choice and entropy are discussed within the framework of selected theorems of the general theory of communications.



Schematic Diagram of a General Communication System

Figure (24)

(1) Shannon, C.E.; Weaver, W., "The Mathemetical Theory of Communication", (Illini Books, Illinois, 1972), p.7.

(2) Epistemologically, communication and understanding are two closely related and dependent phenomena; Dewey, op.cit., p.47.

1. Information and Choice

The particular meaning of the term 'information' as used in the communication theory has neither relation to the way in which it is used in everyday parlance nor to the suggested attitude which considers information as interpreted data. Rather information is treated as a measure of choice available to a person when he selects a message (3). In other words, information is defined in terms of the probability associated with the selection of a particular message.

2. Choice, Uncertainty and Entropy

Given a set of possible events whose probabilities of occurrence are, $p_1, p_2, \ldots, p_n$. It is theorized that in order to measure the amount of choice which is associated with the selection of the event or the uncertainty of the outcome, the measure itself, say $H(p_1, p_2, \ldots, p_n)$, must exhibit the following three properties:

i) H should be continuous in the p_i.

ii) If all the p_i are equal, $p_i = \frac{1}{n}$, then H should be a monotonic increasing function of n. With equally likely events there is more choice, or uncertainty, when there are more possible events.

---

(3) Weaver, In Shannon et al, op.cit., pp.8-9.

iii) If a choice is broken down into two successive choices, the original H should be the weighted sum of the individual values of H.

It is concluded that the only relationship that satisfies the above three conditions is of the form:

$$H = -K \sum_{i=1}^{n} p_i \log_2 p_i \quad \ldots\ldots \quad 1$$

Hence, in the special case where all possible messages/ events are equally likely, the relationship reduces to:

$$H = K \log_2 n \quad \ldots\ldots \quad 2$$

Where K is always a positive constant, and is taken as a unity, i.e. $K \sim 1$.

However, the above expression of H is similar to that of the second law of statistical thermo-dynamics for entropy. Thus, the term entropy is used to define the amount of information or uncertainty.

3. Properties of the Entropy Function H

i) $H = o$ if all the p_i but one are zero, this one having the value unity. Thus only when the outcome is certain does H, i.e. information, vanish, otherwise H is positive.

However, because the amount of information is the negative of the logarithm of a probability it is also considered as a negative entropy, or <u>negentropy</u> (4).

ii) For a given n, H is a maximum and equal to log n when all the p_i are equal, i.e. 1/n. This is also <u>the most uncertain situation</u>.

iii) Suppose there are two events, x and y, with m possibilities for the first and n for the second. Let p(i,j) be the joint probability of the occurrence of i for the first and j for the second. The entropy of the joint event is:

$$H(x,y) = -\sum_{i,j} p(i,j) \log_2 p(i,j) \quad \ldots\ldots \quad 3$$

where

$$H(x) = -\sum_{i,j} p(i,j) \log_2 \sum_{j} p(i,j) \quad \ldots\ldots \quad 4$$

and

$$H(y) = -\sum_{i,j} p(i,j) \log_2 \sum_{i} p(i,j) \quad \ldots\ldots \quad 5$$

from which

$$H(x,y) \leq H(x) + H(y) \quad \ldots\ldots \quad 6$$

expression (6) transforms into <u>equality only when x and y are independent</u>, i.e. $p_{(i,j)} = p_{(i)} \cdot p_{(j)}$.

iv) Any tendency to normalize the probabilities p_1, p_2, ..., p_n increases H. Thus, the processes of averaging or randomizing the probabilities will result in increasing the value of H.

---

(4) Wiener, N., "<u>Cybernetics: or Control and Communication in the Animal and the Machine</u>", (The M.I.T. Press, Cambridge, Mass.,

v) Let x and y be two chance events, as in (iii), not necessarily independent. Then, the conditional entropy, $H_x(y)$, is a measure of uncertainty of y when given x. That is:

$$H_x(y) = H(x,y) - H(x) \quad \ldots\ldots \quad 7$$

or

$$H(x,y) = H(x) + H_x(y) \quad \ldots\ldots \quad 8$$

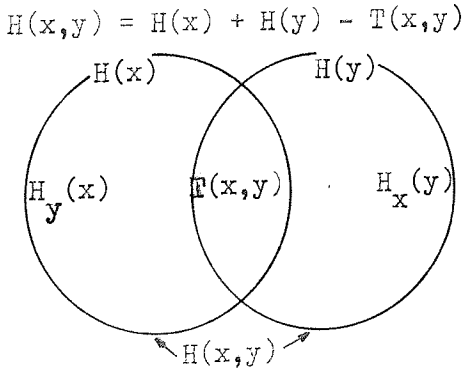
Expression (8) reads, the joint uncertainty, or entropy, of the joint event (x,y), is equal to the uncertainty of x plus that of y when x is known.

vi) From (iii) and (v), then:

$$H(x) + H(y) \geq H(x,y) = H(x) + H_x(y) \quad \ldots \quad 9$$

$$H(y) \geq H_x(y) \quad \ldots\ldots \quad 10$$

That is, the uncertainty of y can <u>never be increased</u> by knowledge of x. Likewise, any other relationships can be



Key:

H(x): uncertainty of x.
H(y): " y.
$H_y(x)$: equivocation.
$H_x(y)$: noise.
T(x,y): transmitted uncertainty.

<u>Two Communicating Holons</u>

Figure (25)

conveniently derived by the use of Venn-diagrams as in Figure (25) to

illustrate the situation at hand (5). In this context, it may be appropriate to note that H(x) and H(y) are partially related sets (6).

4. The Fundamental Theorem for a Noiseless Channel

An analogy is drawn between the source that generates information, say x, and the channel that carries this information, say y, as being another source which generates the output (7). Now since x and y are partially related sources of information, the overlap or the intersection between H(x) and H(y) is the transmitted information. Consequently, and as indicated by Figure (25), the greater the overlap, the more related the input is to the output.

However, the fundamental theorem for a noiseless channel introduces the concept of rates at which information can be transmitted. A source which has an entropy H bits/signal; a channel may have a capacity of C bits/signal, and hence, the average rate of transmission cannot exceed C/H bits/signal (8).

---

(5) Miller, G.A., "What is Information Measurement?", In Buckley, (Ed.), op.cit., pp.123-128.

(6) Edwards, E., "Communication Theory", In W.T. Singleton; R.S. Easterby; D.C. Whitfield, (Eds.), "The Human Operator in Complex Systems", (Taylor & Francis Ltd., London, 1967), pp.37-53.

(7) Miller, op.cit.,pp.126-127.

(8) Shannon, op.cit., pp.58-59.

5. The Fundamental Theorem for a Discrete Channel with Noise

When there is noise in a communication system, transmission errors are liable to occur. Errors may be counteracted by redundancy. It the probability of error is made to approach zero this must be coupled with a large increase in the amount of redundancy so that the rate of useful transmission may effectively tend to zero. However, it is proved that such is not the case (9). The proof says that there is a finite channel capacity C, such that if signals are generated at the rate of H bits/sec, information can be transmitted with an arbitrarily small error provided that $H < C$. When H is greater than C, then information is lost, the rate of information-loss can be made, as required, to almost equal to the amount of excess entropy, i.e. H-C.

This theorem can be used to establish another design criterion. If H is the amount of uncertainty handled by the activity to be computerised and C is that of its computerised version, then problems will arise when and where there is incompatability between H and C. Also this criterion can be used to distinguish between what is programmable and what is non-programmable (10). However, the term uncertainty is used to indicate two types. One is the uncertainty of the occurrence of an event which can be attributed to environmental changes or disturbances. The other is that concerning the programmability of the process of decision making, i.e. the uncertainty associated with decisions such as deciding on objectives, selecting a certain course of action, etc.

One of the implications of the theory of communication is the concept of variety. It is believed that variety can be used as

---

(9) Ibid, p.75.

(10) This aspect is discussed in length in Chapter 7.

a principal design criterion because it provides a framework within which flexibility, bias and independence may all be achieved.

VARIETY IMPLICATIONS OF CURRENT PACKAGE DESIGN

The design of software as the media which links users, e.g. management, at one end with the computing machine at the other, has to demonstrate adequate understanding of the holonic, i.e. openness, dynamics, complexity, etc., nature of such potential applications. One concept which is central to both communication and control is that of variety. Consequently the variety of a set of distinguishable holons, say software, is defined to indicate either: (i) the number of distinct holons, or (ii) the logarithm to the base 2 of the number (11). Thus, the similarity and continuation of variety and communication are clearly identifiable. Closely related to variety is the concept of constraint. It is defined as a relationship between two holons it exists when variety changes, i.e. when variety that exists under one condition is less than the variety that exists under another (12). In this respect, ordering and/or coupling of modules in a specific manner amount to imposing constraint upon the degrees of freedom of the constituent modules. Thus, if all combinations are possible, then the number

---

(11) Ashby, "An Introduction to Cybernetics", op.cit., p.126.

(12) Ibid., pp.129, 134, 206.

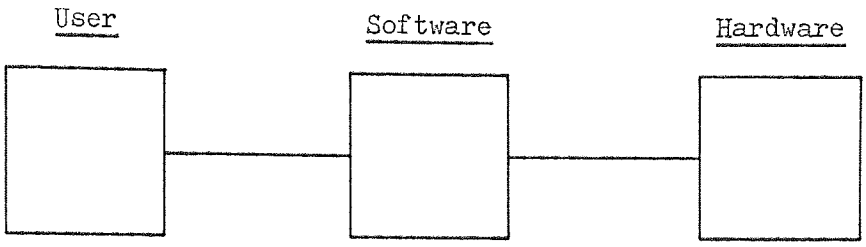
of degrees of freedom is equal to the number of all available modules. Consequently, if only one combination is possible, as in the case of permanently coupled modules, the degrees of freedom are zero. Considering CLASS and most of the available applications packages, and indeed users' own designed programmes, it is immediately recognisable that the degrees of freedom offered by each programme are zero. Generally mechanization or automation is associated with a reduction in the number of the degrees of freedom. By permanent coupling of modules into one and only one sequence variety of the whole is reduced to zero. Considering the amount of variety management may have to interact with, this design approach is regarded to be faulty. First, it imposes a freezing effect upon the modules, and secondly, it leads to uneconomical use of resources by not allowing full use of such modules as, for example, in building new programmes. This, as indicated earlier, constitutes the antithesis of the underlying assumption of applications packages as being fundamentally designed to avoid duplication of programming efforts. More important than that, current design of applications packages ignores one very basic and essential characteristic of the notion of variety, i.e. 'only variety can destroy variety' (13). The design implication of this relationship is that depending upon the variety of decisions that may be handled by a specific managerial level, related software applications must exhibit a matching amount of variety through built-in flexibility to balance the former. In order to do that an appreciation of the relationship between software and its users have to be clearly realized. One way of achieving this is by

---

(13) Ibid., p.207.

investigating the role of software within the communication network of a company.

SOFTWARE AND COMMUNICATION



Schematic Presentation of User/Software/Hardware Interaction

Figure (26)

NATO defines software as "programmes developed to control * the actions of computers"(14). The key word in the definition, for the present purpose, is that of control. Already it is established that control and communication are interrelated. Consequently, if software is to control hardware, then evidently it has to impart messages to activate and direct the behaviour of the component parts of hardware. In view of the previosly discussed holonic structure of software, through the operating programmes, software maintains its communication mode with hardware (15). Again, in the same

---

* Emphasis is not in the original.

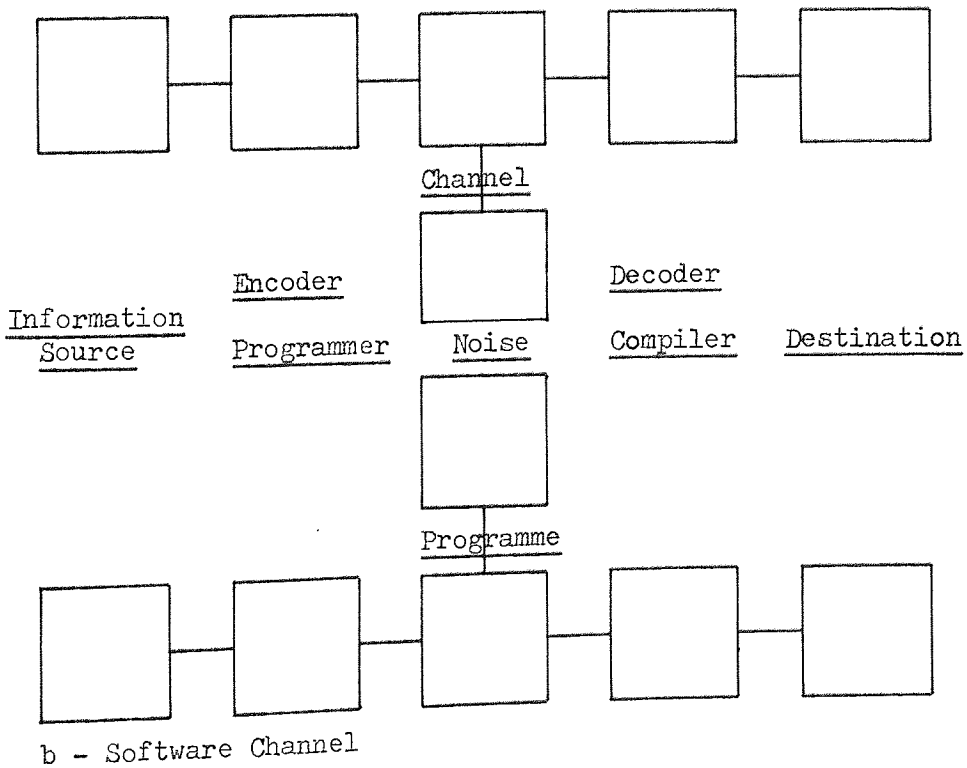
(14) Naur; Randell, (Ed.), op.cit., Preface.

(15) Cf. Chapter 4, p.94.

manner but at a higher level of conceptualization, software in general and applications programmes in particular constitute the media by which users, e.g. managers, control and/or interact with the hardware, i.e. software is the media that allows communication between users and hardware as shown in Figure (26). In this set-up software's role can be depicted essentially to be similar to that of a conventional communication channel.

To establish the validity or otherwise of the proposition that software is analogous to an ordinary communication channel Figure (26) is exploded in the same fashion as Figure (25) to produce that of Figure (27). By successive consideration of the boxes of Figure (27) and starting from left to right, correspondence may be identified. However, it may be appropriate to stress here that the two analogues are not treated as isomorphics.

a - Conventional Channel



b - Software Channel

Schematic Presentation of the Analogy between the Two Channels

Figure (27)

Table (2)

Correspondence Between Conventional and Software Communication

Conventional	Software
1. Information Source Produces a message or series of messages from a set of possible choices.	Information Source The user may inquire or produce data depending upon his needs.
2. Transmitter/Encoder Operates on the message and transforms it into signals suitable for transmission over the channel.	Programmer The human operator who translates the message into a sequence of instructions compatible with, and suitable for transmission over the channel.
3. Channel Carries the signals from the transmitter to the receiver.	Programme Carries the data from the programmer to the receiver.
4. Receiver/Decoder Operates on the received signal and attempts to extract the message and present it in a usable form to the destination.	Compiler Another transformation process. Instructions are translated into machine language instructions comprehensible by the central processing units.

5.	Destination The person or thing for whom the message is intended.	Destination The central processing unit to perform the intended data processing operations.

N.B. Communication in the reverse direction also holds. That is the C.P.U.'s response goes through the same order in steps.

In Figure (27) and Table (2), the information sourse, say x, can take more than one form. Thus it may represent an organizational department which requires the hardware, say y, to process varieties of data. The selection of such data is essentially influenced by two primary factors. First, the type and range of data software can handle, i.e. the capacity of software. Second, the situation which activates the communication process. As the difference between the second and the first increases communication degenerates and interaction eventually ceases (16). An approach to maintain an effective interaction between x and y is by offering software enough variety to cope with that of its user as variety only destroys variety. This variety can be generated by allowing temporary coupling of different, independent modules in various ways in response to varying demands without the need for software to have infinite capacity.

(16) This is based upon an interpretation of the fundamental theorem for a discrete channel with noise.

The encoder is the computer practitioner, i.e. an analyst, programmer, operator, etc., whose function is to receive and translate messages into sets of instructions compatible with the requirements of the decoder, or more specifically, the compiler. It is possible in this regard, to fuse both steps into one (17). Next to the encoder is the channel. The channel is represented by a programme which has, like most other communication channels, a finite capacity. Beyond that limit economic transmission may cease to operate and/or data distortion and overflow may occur.

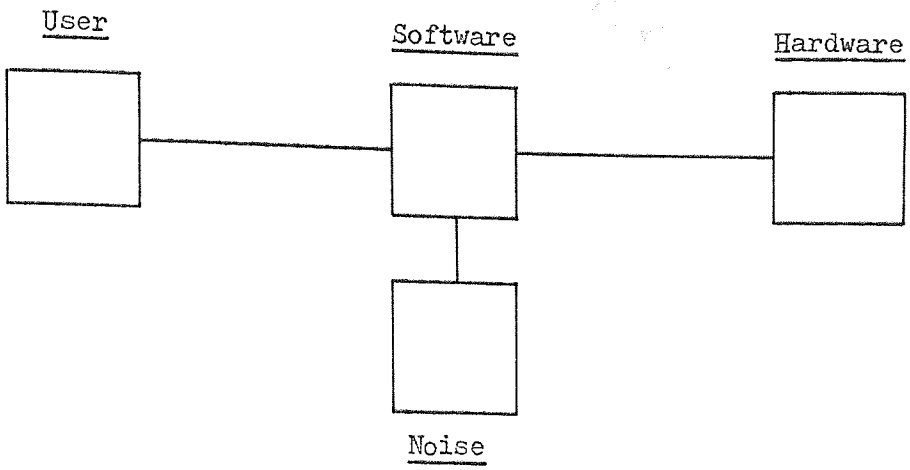
To minimize the chances of the occurrence of data distortion and loss, redundancy has also been catered for in most software languages (18). Noise in the form of instruction errors and circuitry may affect the communication process. Since errors, in one form or the other, are unavoidable, much emphasis has been attached to redundancy in programming languages (19). For, without a degree of redundancy, errors cannot be tolerated. Following the channel is the decoder, i.e. the compiler, which receives the discrete signals, i.e. coded instructions, and further operates on the signals and produces object programmes. Software can therefore be treated as a discrete communication channel with noise as shown in Figure (28).

---

(17) A case in mind is that of a manager who operates a terminal. In this set-up he actually plays the combined role of an information source and encoder.

(18) Boehm, op.cit., p.54.

(19) Wienberg, "Natural Selection as Applied to Computers and Programs", op.cit., p.149.



Schematic Presentation of Software As a Discrete Channel with Noise

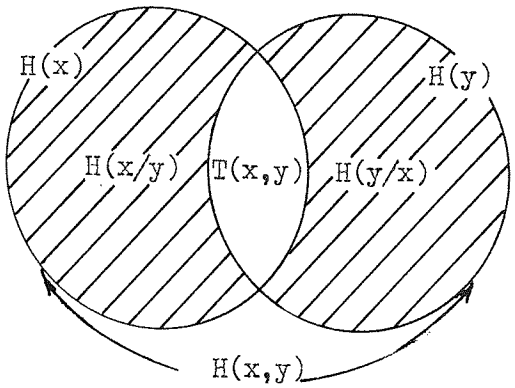
Figure (28)

USER-PRACTITIONER COMMUNICATION MODEL

Among the various interacting components of the above communication model, the user as an information source, and the software practitioner as an encoder, are of special interest. The impact and properties of their relationship can best be demonstrated in the context of communication theory (20).

---

(20) It must be stated that the problem is primarily that of human communication. Therefore subjecting it to the theory of information may involve significant reductionism. Nonetheless, information theory suffices the present purpose of outlining the problem areas.



Source-Encoder Interaction

Figure (29)

In Figure (29), let x be the source, and y be the encoder. If x wishes to impart a message H(x) to y, then according to the theory of communication, the information transmitted is that portion only common to both x and y denoted by T(x,y). The rest, the shaded area, is information lost. In general this may take the form of equivocation and/or noise. When communication is from x to y, then equivocation, i.e. H(x/y), can be attributed to ambiguity or mispresentation by the source, i.e. x, of his requirements. That is, equivocation is information that could have been imparted to the encoder, i.e. y, but it did not materialize. However, noise, i.e. H(y/x), is unintended information added randomly to the message/signal during transmission (21). Noise can be attributed to the encoders inability to predict and/or unfamiliarity with the source's set of choices. Obviously the objective should be to increase the intersection of H(x) and H(y),

---

(21) Wiener, Cybernetics, op.cit., pp.64-69.

i.e., the information common to both x and y, as much as possible. This results in the reduction of the symmetric difference, i.e. equivocation and noise. This can readily be explained by the following equality

$$H(x) \triangle H(y) = H(x) \cup H(y) - H(x) \cap H(y) \quad \ldots . \quad 11$$

in which, as the intersection, i.e. the second term of the right hand side, increases, the difference vanishes.

If either the manager, i.e. the source, or the practitioner, i.e. the encoder, or hopefully both, extends his repertoire of knowledge about the others limitations and requirements, communication between them may be enhanced. This is attributed to the fact that the encoder's output signal, in the form of designs, may become a closer representation of the source's intended message or requirement. In other words, the amount of lost information can, thus, be reduced in accordance with better understanding. However, if one is more willing to spend more effort than the other in this pursuit, the practitioner has to be that one. If he wishes to secure his career then he can convince management with his useful qualities by yielding better designs.

ENTROPY AND PROGRAMME DESIGN

Entropy and communication are fundamentally two closely related concepts. Entropy is defined in the hard sciences as the statistical tendency of matter to move, at random, towards total

disorder and decay (22). It is argued that entropy and information, as defined in the communication theory, are analogues except that one is the negative of the other (23). This is because of the isomorphic correspondence between the mathematical formulae of entropy and information (24).

Entropy, as interpreted here is used to identify and explain some intrinsic programme properties and their effect upon programme design. In the theory of communication, all the components of a communication system are governed by probabilities. The same applies to software as comprising special purpose channels. Accordingly, it is most likely to find some channels that are more probably used than others. As the higher frequency programmes, or parts of a programme, obviously tend to be more up-to-date, lower frequency parts become, relatively out-of-date. The outdating process is caused and speeded by an accumulation of errors and by being static or immobile in a basically mobile environment. In other words, the more a user interacts/communicates with a certain programme, the more likely he is to become aware of the programme's weaknesses, and consequently, the more likely he is to press for such changes.

---

(22) Schrödinger, E., "What is Life", (Cambridge Univ. Press, Cambridge, 1945).

(23) Brillouin, L., "Thermodynamics and Information Theory", In W. Buckley, (Ed.), op.cit., pp.161-165.

(24) The interpretation of information in terms of entropy has ever since it was proposed caused a philosophic, scientific and intellectual controversy. This is because entropy is considered by some as a purely physical quantity, whereas others see it as a universal concept. The latter is held here.

The continuing existence of a programme depends upon two interrelated factors, namely survival time and identification. By survival time is meant the time span during which the programme exhibits a more or less constant behaviour, i.e. the main characteristics of the programme are always maintained allowing only for minor changes to take place. If the behaviour of the programme adheres to these limits, then and only then, the programme may remain identifiable, i.e. operational (25). The constancy, in this respect, does not imply rigidity since it is conceived of as a relative phenomenon, i.e. the programme changes in response to its environment. Once again as the environment is perpetually changing so the programme itself needs to be frequently modified depending upon the rate of change of the environment, in order to exhibit a constant behaviour. Otherwise, and as time lapses, the programme may gain increasing amounts of obsolescence. In other words, when the relationship that binds a programme with its environment loosens, the process of exchange of energy in the form of information between the two, in terms of maintenance, slows down accordingly, and time may come when the programme's behaviour does not preserve its acceptable limits. Worse still is the situation when the energy needed to revitalise the programme is estimated to be nearly as much as that required to develop a new one. Then it can be maintained that the programme has gained enough entropy to destroy its purpose, and hence, its existence.

---

(25) Weinberg, G., "A Computer Approach to General Systems Theory", In G.J. Klir, (Ed.), op.cit., pp.98-142.

DESIGN IMPLICATIONS

Complementary to the holonic, communicative and entropic conceptualization of a programme is the design of the structure of the holon itself. The fundamental principle is that of requisite variety. The high degree of relevance of this law to design methodology can be shown regarding the struggle to control entropy. It has been discussed that as the variety within a system increases, in terms of flexible coupling of its components, the ability to handle uncertainty also increases proportionally (26). Uncertainty can be interpreted as a measure of the degree of randomness within the holon. This relationship, though it may not have been interpreted in a similar fashion can be regarded to constitute the theoretical constructs behind the notion of modularity. Modularity, in the context of software, is defined as the ability to extend a system in increments (27). With the underlying idea of having a set of modules that can be individually developed, debugged, improved, or extended with minimal personnel interaction or system discontinuity (28).

Through modular structuring of programmes, applications of either of the two approaches, i.e. top-down and bottom-up, can be performed. Such has resulted in the development of two programming techniques, i.e. conventional modularity, and structured programming. One central feature to both is coupling. Two

---

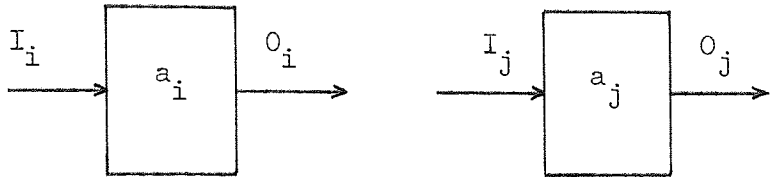
(26) Ashby, An Introduction to Cybernetics, op.cit., p.208.

(27) Martin, J.; Norman, A.R.D., "The Computerized Society", (Prentice-Hall, N.Y., 1970), p.599.

(28) Gillette, H.R., In Naur; Randell, (Eds.), op.cit., p.39.

modules when connected to each other are said to be coupled. Thus, if a_i and a_j are any two modules, as shown in Figure (30), coupled through a specific corresponding aspect, viz. A_i and A_j respectively, then:

$$C_{ij} = A_i \bigcap A_j, \text{ such that } i \neq j \quad \ldots\ldots \quad 12$$



Coupling of Modules

Figure (30)

Again, coupling as such is inadequate, hence for couplings to be purposeful, then if:

I_i : is the set of input quantities for element a_i;

O_i : is the set of output quantities for element a_i;

I_j : is the set of input quantities for element a_j;

O_j : is the set of output quantities for element a_j,

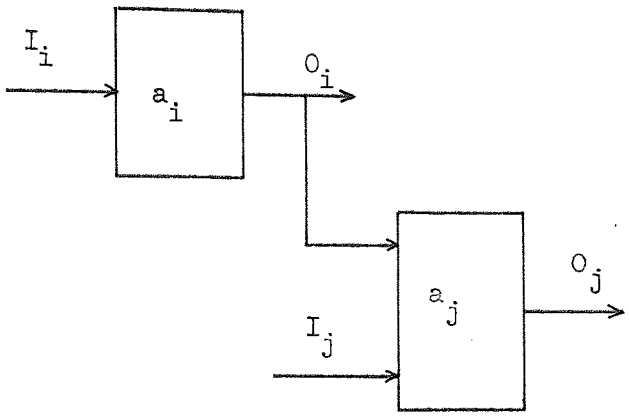
a controlled, or directed, coupling of two elements is the set of all quantities that belong to the output quantities of the first element, e.g. O_i, and to input quantities of the second, e.g. I_j (29). That is, if d_{ij} is the directed coupling of the pair of modules (a_i, a_j), as shown in Figure (31), hence symbolically:

---

(29) Orchard, R.A., "On an Approach to General Systems Theory", In G.J. Klir, (Ed.), op.cit., pp.205-250.

$$d_{ij} = O_i \; I_j \quad \text{such that } d_{ij} \neq d_{ji} \qquad \ldots\ldots \qquad 13$$

then the two modules are said to be dependent (30). This indicates any change in either set I_i or O_i may result in a



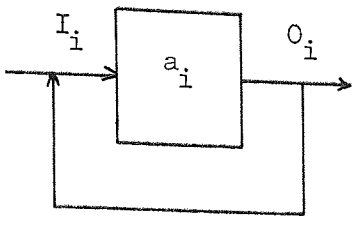
Directed Coupling of Two Elements

Figure (31)

change in O_j. Thus co-ordination and harmony must always be maintained. That is, according to the concept of variety, unless the element of a_j is equipped and built to cope with all possible variations in O_i, i.e. the output of a_i, O_j may not conform to its specifications if any variation occurs in O_i. Therefore, to circumvent this probability and to control the behaviour of individual modules to remain within the stated limits, directed coupling has to be defined for i = j, so as to allow feedback, i.e. for i = j

---

(30) Lange, O., "Wholes and Parts: A General Theory of System Behaviour", trans. by E. Lepa, (Pergamon Press, Oxford, 1965), pp.11-21.



Coupling with Feedback

Figure (32)

$$d_{ii} = O_i \bigcap I_i \quad \ldots\ldots \quad 14$$

Consequently, d_{ii} is a measure of feedback. Thus, an application programme S which comprises a set of connected modules can be defined as a set D of independent modules $a_1, a_2, \ldots, a_n$ with

$$D = \{d_{ij}/d_{ij} \text{ is the directed coupling of } (a_i, a_j) \text{ such that } a_i, a_j \in S\} \quad \ldots\ldots \quad 15$$

directed couplings between them.

However, in practice, modules are as large as complete programmes. Specifically this point, i.e. size, is what differentiates the notion of conventional or pseudo-modular and structured programming.

1. Pseudo-Modularity

The essence of modular programming can be demonstrated as the process of dividing a programme into small parameter-driven

modules, each module being a closed sub-programme with a single entry point which returns to the calling module at the next sequential instruction following the call (31). Each module is assigned some distinct function by performing it the module contributes in fulfilling the requirements of the overall programme. The main advantages of modularity are: (1) a number of programmers may work on different modules of a single programme; (2) testing can be carried out on separate or several modules in the same time; (3) testing is facilitated by not having to create test files (32); a programmer may only be concerned with testing the variables of his module.

However, such advantages cannot materialize unless the three-fold criteria, i.e. flexibility, bias and impartiality, are considered. In this respect, the design difficulty is two-fold, one being the coherence of the module, and the other the coupling of modules, though retaining their individual independence. Two modular properties, as shown in Figure (33), are regarded to affect the independence of one module from the other (33). First, is the module's strength, i.e. a function of the intra-relationships of the module. To maximize module strength is equivalent to

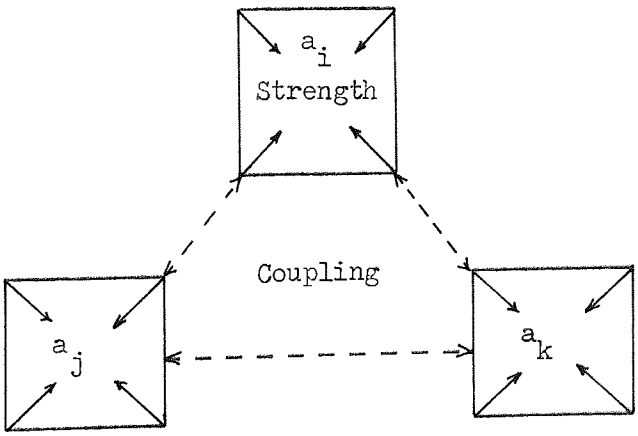
---

(31) Leavenworth, K., "Modular Design of Computer Programs", Data Management, 1974, July, pp.14-19.

(32) There are commercially available test programmes that help programmers to set-up their test data in the core store.

(33) Myers, G.J., "Characteristics of Composite Design", Datamation, 1973, Sept., pp.100-102.

maximizing the relationships among the module's constituents. Accordingly, a module which performs one function is stronger than that which has two or more parts to perform two or more functions. The second property is that of module coupling. As defined above, module coupling is a function of inter-relationships between modules. Independence and flexibility of modules can be maximized by minimizing the couplings. One way of attaining such requirement is by specifying that all input to and output of a module take the form of parameters or arguments. Also pertinent to coupling is the transferability of control data between modules. The idea is that coupling



Module's Strength and Coupling

Figure (33)

between two modules, say a_i and a_j, increases when, e.g. a_i, sends control data to a_j. Thus maximizing the modules independence has to be characterized by minimizing the transferability of control data, and that may also yield more flexible design.

However, modularity, in practical terms, is conceived of as an arbitrary decision concerning the division of a large programme into smaller parts or apparent modules. These modules are frequently not adequately independent of one another due to the ad hoc manner in which modularity is attempted within the context of bottom-up design approach. Modularity when void of its basic conceptual framework, becomes another term referring to monolithic programming or rigid, uninterrupted long sequence of instructions. The principal deficiency of a monolithic programme is that if a part of it fails to work so does the rest of it. Modularity cannot solve this problem if the overall philosophy and design approach are not modular in essence. Consistency is indispensable. There is no point in fragmenting a programme into a set of smaller pieces when the programme is designed originally as monolithic, or, at least, the logical boundaries of the various functions have not been identified (34). The resulted pieces, however, are not modules, rather they can be called pseudo-modules. Such is regarded as a sign of confusion stemming from lack of appreciation of the systems approach, optimization and bottom-up approach. In the presence of such confusion, the problem encountered with pseudo-modular programmes, e.g. CLASS, is that a programme is often comprised of few large modules that cannot be simply subdivided. Hence if programmes are indivisible, they must be treated as wholes. This

---

(34) In the interviews, practitioners were asked about their criteria that may decide the number and size of modules. Almost all of them said this is a matter decided by programmers.

implies that all subsequent activities such as testing, debugging, maintenance, or most important of all, attempting to understand the module's structure and behaviour, are seriously hampered by the necessity to comprehend and digest a large sequence of instructions. The situation is further aggravated by the excessive use of jump/goto statements. So the structure of such programmes can be labelled as monolithic. Therefore, judging from the current practice of software manufacturers, as well as users own expertise, modular application through ad hoc approaches has departed away from the original purpose of modularity, i.e. variety, flexibility and understanding.

2. The Concept of Structured Programming and Top-Down Design

From both the inadequacy and degeneracy of pseudo-modularity, structured programming has evolved as an application of the top-down approach (35). The intrinsic quality of structured programming is that of extending some rigorous mechanisms to ascertain the successive sub-division of a programme into smaller modules through a continuous process until the level of atomic-modules, i.e. consisting of a few statements, has been reached. This is done within the guidelines of top-down design by first developing a general framework which becomes more specific by the use of structured programming at successive lower

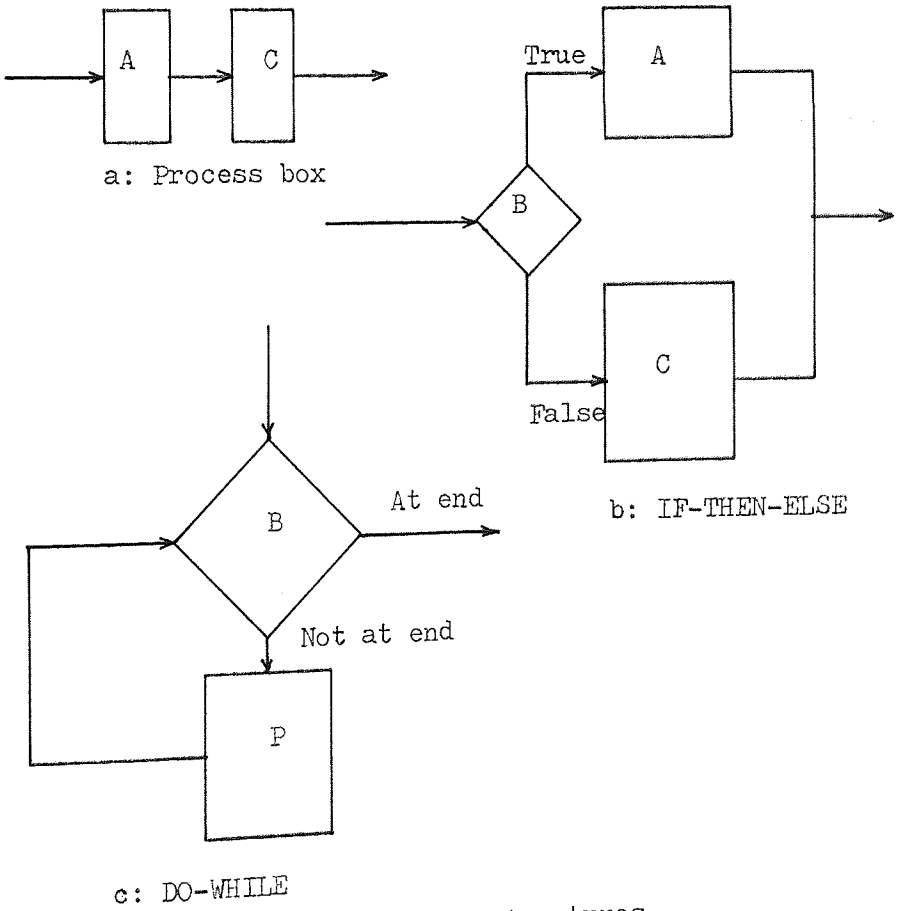
---

(35) Dijkstra, E.W., "Programming Considered as a Human Activity", Proceedings of IFIP Congress 65, (Spartan Books, Washington, 1965).

levels of the hierarchy (36). In other words, structured programming complies with the holonic structure, particularly with regard to production of output hierarchies.

2.1. Elements of Structured Programming

Any programme can be constructed from three basic building-blocks, i.e.: (a) process boxes; (b) IF-THEN-ELSE mechanisms; (c) DO-WHILE mechanisms (37). These are illustrated in Figure (34). Also these three structures are referred to as linear sequence, selection and loop patterns respectively. In linear sequence one



Basic Structures

Figure (34)

(36) Chapin, N.; House, R.; McDaniel, N.; Wachtel, R., "Structured Programming Simplified", Computer Decisions, 1974, June, pp.28-31.

(37) Ibid.

step is done after another. It is analogous to the AND operation since the flow of control is in series, i.e. do first A and then do C, the selection consists of a test followed by two parallel control routes only one of which is taken in any given time depending upon the result of the test. This is analogous to the OR operation since the rule is, for example, if B is ture then do A, else do C. In the loop, control transfers according to: perform P while test B is false. Accordingly, the loop P may never be executed since the outcome of B may cause an exit after control enters the pattern. Alternatively, it may be required to execute P at least once. This can be achieved by the construction of a linear sequence, i.e. the loop body P, followed by the 'do-while' structure. This is known as the 'do-until' or 'repeat-until' structure.

Top-down design facilitates the proof of programme correctness (38). This may be done by: (a) first viewing the prpgramme as a collection of major functions, e.g. as a main programme that may call a set of major modules; (b) assuming that these modules are operationally acceptable; (c) attempting to demonstrate that the overall logic of the programme, e.g. the logic of the main programme itself, is correct and that the interfaces are compatible. Following this is the breaking of each major module into smaller modules and going through the same three-step process. This continues until the lowest level of modules is reached whose proof is regarded as a straightforward task (39).

The author finds it rather important to note that he would have liked to provide a number of structured programming modules or holons as illustrations. However, due to limitations in time and resources this was not possible for it is a multi-year/person effort.

---

(38) Hoare, C.A.R., "Proof of a Structured Program: The Sieve of Eratosthenes", Computer Journal, 1972, Vol.15,No.4, pp.321-325.

(39) Yourdon, E., "A Brief Look at Structured Programming and Top-Down Program Design", Modern Data, 1974, June, pp.30-35.

CONCLUSIONS

Considering applications programmes, and software generally, as communication channels leads to some fruitful results. First it singles out that both the user and the analyst as two components of the communication system have to be compatible regarding understanding and familiarity of each other's requirements and constraints in order to attain adequate communication. Also it constructs a rigorous framework of systems design within which concepts such as requisite variety, coupling and/or control, entropy, etc., become directly applicable in interpreting, developing and validating existing and new design criteria such as flexibility, impartiality and independence. Finally it facilitates the task of identifying and selecting the appropriate techniques that are compatible with the derived criteria. In this regard, the advantages of the top-down design approach can materialize by the use of structured programming whose contributions may cover the following.

1. Higher Resilience: That is, the ability of a programme to recover from errors. This includes the ease and speed with which errors can be detected, located and corrected. The reduction or elimination of the number of alternate paths in a module increases its resilience significantly when it is compared with large, pseudo-modules, with many branches.

2. Better Replacability/Modifiability/Adaptability: The grouping of the three properties is based upon their strong inter-dependence. Since to modify a module, may involve replacing a defective part or the whole of the module. By so doing adapt-ability is maintained. At any rate replacing small modules is logically much less expensive than replacing a large pseudo-module, e.g. NIMMS Stock Control. However, as interactions between the user and his environment on one hand, and the user and the programme on the other, continue programme modification becomes inevitable. Irrespective of the type and degree of change, modifiability depends upon the documentation and compre-hensibility of the modules involved. When a programme/pseudo-module is lengthy and cumbersome, it is frequently reported that the programmer concerned tends to abandon the programme under review and embarks on producing his own new version even if the programme was originally written by him. This coupled with the attitude towards documentation as being boring and barren, reduce the modifiability of the programme whereas small proper modules facilitate and enhance all these properties by the mere size and independence that distinguish them.

3. More Openness/Flexibility/Partial Processing: Ascertaining that modules are highly independent, i.e. weakly coupled to each other, leads to attainment of a comparable degree of flexibility. Also, by designing independent modules, variety increases as modules

can be added, removed and modified according to the requirements. This, in turn, presupposes openness. Evidently this is not the case with pseudo-monolithic modules where rigidity and strong/ permanent coupling are ubiquitous. Thus modularity when properly applied yields flexible, independent and impartial modules.

CHAPTER 7

DECISION MAKING AS A DETERMINANT OF DESIGN CRITERIA

The purpose is to study the compatibility of the top-down design approach and design criteria, for example flexibility and independence, with the needs of decision making. The implications of regarding the mechanics of the process of decision making to be equivalent to that of problem solving opens a new horizon to consider the effectiveness of the package version of problem solving as represented by its current architecture. This involves looking at the following items.

1. An organizational model representing a three-level management framework. This can indicate the functions, properties and scope of each level.
2. Types of decisions, i.e. whether being routine, programmed, non-programmed and the difference between programmed and mechanized decisions.
3. The psychological effect of automation in the form of current packages upon the decision maker.
4. Problem solving methods within the context of the concept of holons and their respective handling capabilities of the decision maker's uncertainty level, the more likely it is to interact with higher uncertainty that is associated with it.

This is concluded by investigating the relationship between uncertainty and programmability.

MODELS OF USERS' HIERARCHIES AND AUTOMATION

In more recent attempts to understand and reveal some characteristics of the two closely related phenomena of management or control (1) and decision-making or problem solving process (2), a number of approaches and models have been proposed. In the following the models considered are those supposedly based upon the systems approach and oriented towards the design and development of management information systems.

It has been suggested earlier that problems, like systems, are hierarchical in nature. The higher the problem's level, the more likely it is to interact with higher uncertainty that is associated with it. Uncertainty on its part implies complexity and vice versa. Consequently an organization as an open system that interacts with its environment, can be observed to import and export uncertainty (3). Thus, ascending the organizational hierarchy, functional boundaries become increasinging fuzzier and their scopes enlarges. The data requirement of each level differs significantly from levels above and below it. This requirement is characterized by the properties of

---

(1) Parsons, T., "Structure and Process in Modern Society", (Glencoe: Free Press, 1960), pp. 59-96.

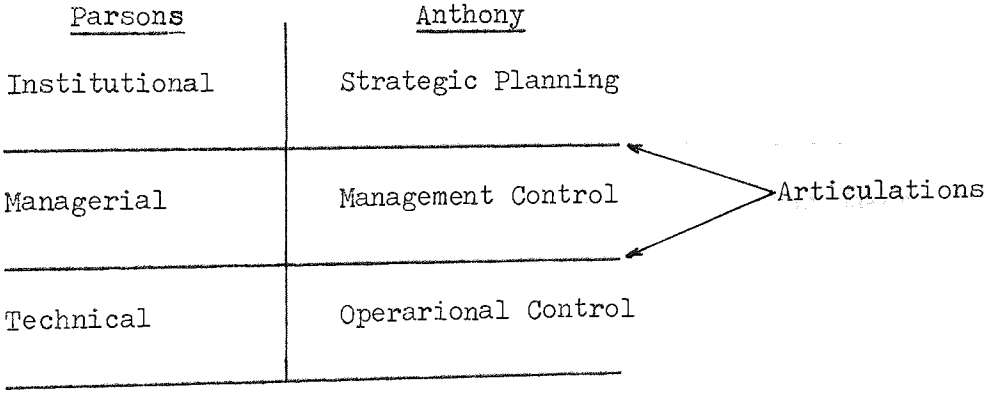
(2) March et al, op.cit., 172-199.

(3) It should be noted that uncertainty, in this context, implies the same meaning as used by Shannon; see Chapter 6, p.133.

the type of decisions being considered in the light of the degree of specificity. Taking the degree of specificity as a classification criterion the organizational hierarchy has been depicted in a number of models of which the following are but some.

1. Three-Level Management Framework

This view divides the hierarchical structure of organizations into three levels, e.g. strategic planning, management control and operational control (4), as depicted in Figure (35). The



Three-level Hierarchy

Figure (35)

dividing lines of the model represent areas of overlapping functions and responsibilities.

---

(4) Anthony, R., "Planning and Control Systems: A Framework for Analysis", (Harvard Univ. Press, 1965). It may be appropriate to note that this model is fundamentally similar to that of Parsons except that the latter is wider in scope. Thus Parson divides the hierarchy into technical system, managerial system and community or institutional system from bottom to top. Also, Parsons discerns between the two dividing lines by maintaining that the one which separates technical and managerial systems is simpler and less obscure than that separating managerial and institutional systems, Parsons, op.cit

1.1 Strategic Planning

This is the process of deciding on: 1) the organizational objectives; 2) changing these objectives; 3) the resources needed to attain these objectives; 4) the policies that govern the acquisition, use and disposition of these resources (5). However, the term strategic is used to refer to the relationship between an organization and its environment (6).

Being mainly concerned with environmental forces the strategic level is characterized by an interaction with unreptitive or, scarcely so, problematic situations. This, in turn, is coupled with a comparable degree of unspecificity of the actions and/or procedures that can be employed to combat such situations. Consequently, being deprived from adequate specification makes it extremely difficult to automate insofar as to cater for all possibilities.

1.2 Management Control

This is mainly

> "... the process by which managers assure that resources are obtained and used effectively and efficiently in the accomplishment of the organization's objectives" (7).

---

(5) Anthony, op.cit., p.24.

(6) Ansoff defines strategy to denote decisions pertaining to the external affairs of the organization rather than its internal problems. An example is that of determining the product mix and the potential market, Ansoff, H.E., "Corporate Strategy", (Penguin Books, 1968), p.18.

(7) Anthony, op.cit., p.27.

In other words, this process is mainly concerned with the problem of structuring resources for optimum utilization. Thus the decisions that are likely to be involved with pertain to: 1) working-out information needs; 2) policies and responsibilities; 3) transforming resources into the required output (8).

This level is more involved with internal organizational affairs. It is responsible for the conversion of strategics into simple and operational form. The conversion process when viewed from the top can be observed to be accompanied with a reduction in uncertainty. Consequently if the conversion is to be successful, management has to ascertain that both strategies communicated from above are realistic and that estimated capabilities and capacities at lower levels are reasonably accurate. In this context, the process of conversion is influenced by the characteristics of the internal environment of the organization regarding the permissible mode of communication, i.e. whether it comprises both vertical, top-down and bottom-up, and horizontal (9). Personal interactions, individual-organization conflict of objectives are some of the potential sources of perturbations introduced from above as well as below. Thus the type of problems of this category are quasi-repetitive and quasi-specified. Depending upon the degree of

---

(8) Ansoff, op.cit., p.19.

(9) Likert, R., "New Patterns of Management", (McGraw-Hill, N.Y., 1961), pp.44-60.

specificity and the frequency of occurrence, decisions may be considered. Thus faced with this type of problems management requires high degree of flexibility which is much affected by automation.

1.3 Operational Control

This is

> "... the process of assuring that specific tasks are carried out effectively and efficiently" (10).

Consequently decisions may pertain to problems such as operating objectives, pricing and output level, operating levels, production schedules, inventory levels, etc., all being enveloped by the overall property of repetitiveness and self-regeneration (11).

1.4 Impact on Automation

In view of the above categorization, automation in the form of current package philosophy and architecture imposes unsurmountable difficulties due to serious design misconceptions. This is because a package as a ready-to-implement solution often comprises one solution to what is essentially a varying problematic situation. In other words, the current philosophy of packaging implicitly forces the users to treat all possible versions of the

---

(10) Anthony, op.cit., p.69.

(11) Ansoff, op.cit., p.21.

situation in the same manner. This reductionism asks the user to perform isomorphic mapping between whatever situation that may be at hand and the package. Such a philosophy obviously gives way to many problems largely due to the inevitable conflict that exists between the package which tries to reduce the managers freedom, and the manager who seeks more flexibility. This is also as true for packages which are designed specifically for strategical decision making (12). These packages while they recognise the characteristic variability of this type of decisions, it is observed that they retain the same architecture of other packages, i.e. of being unchangeable, inextensible and rigid. In other words, the effectiveness of offering more than one technique to choose from is strongly questioned because they are contained within a pre-determined rigid framework dictated by the current package architecture.

Another approach to decision making is that which associates it to programmability. The basic idea of the approach is reviewed below.

2. Decisions and Programmability

The thesis of this idea is based upon the recognition of the similarity between the processes of decision making and problem

---

(12) Examples are I.B.M.'s STRATPLAN and I.C.L.'s PROSPER packages.

solving (13). In this context decisions are classified into routine decisions and others characterized as problem solving. Concerning the latter, when faced with a problematic situation (14), the process to arrive at a decision comprises three main activities: (a) an intelligence activity with the military meaning of the word (b) a design activity which involves inventing, developing, and analysing possible courses of action (c) a choice activity that is mainly concerned with the selection of a course of action from the available set of alternatives (15). These three activities should be seen to be compatible with the systems approach and consequently they are interdependent and iterative in essence (16) and thus emphasize the need for flexible structuring of definitions or solutions.

From this conceptualization it has been possible to distinguish between the various types of decisions on the basis of

---

(13) Simon, "The New Science of Management Decision", op.cit., p.2; also March; Simon, "Organizations", op.cit., pp.172-199.

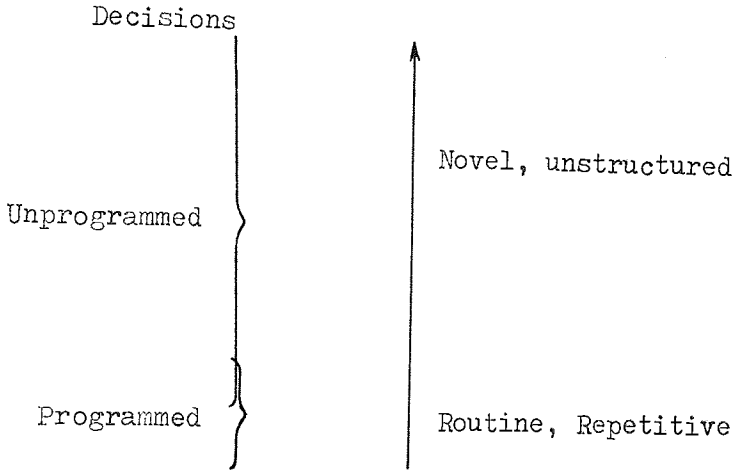
(14) It is assumed that the decision maker develops his decision on the grounds of a conceived model of the real situation. This model, as mentioned previously, is equivalent to his definition. In Simon's terms, the simplified model of the real situation used by the decision maker is that referenced as the "definition of the situation".

(15) Simon, "The New Science of Management Decision", op.cit., pp.2-3.

(16) Ibid.

programmability (17). In this context, programmed decisions are distinguished by being repetitive, routine and well-defined, i.e. they are not novel. Non-programmed decisions

> "... are novel, unstructured, and consequential. There is no cut-and-dried method of handling the problem because it hasn't arisen before, or because its precise nature and structure are elusive or complex, or because it is so important that it deserves a custom-tailored treatment" * (18).



Programmed-Unprogrammed Decisions

Figure (36)

---

(17) This should not be interpreted in the limited stereo-typed computer use, but it does not exclude it. In this regard the degree of programmability is similar to that proposed by Cyert and March and reads "In this continuum (represented by Figure (36),) from great specificity and repetition to extreme vagueness and uniqueness, we will call decisions that lie toward the former extreme programmed and those lying toward the latter end non-programmed". Cyert, R.M.; March, J.G., "Organizational Factors in the Theory of Oligopoly", Quarterly J. of Econ., 1956, 70, pp.44-64. However, Forrester has classified decisions along this continuum depending upon the degree of pre-programming. Thus he discerns between 'Automatic programming', e.g. computer instructions; 'incomplete policies', e.g. instructions to middle management for handling fairly routinized problems; 'judgment and intuition', e.g. less structured decisions; 'randomized' decisions for which no programmes exist at all. Forrester, op.cit.

* Emphasis is not in the original.

(18) Simon, "The New Science of Management Decision", op.cit., pp.5-6.

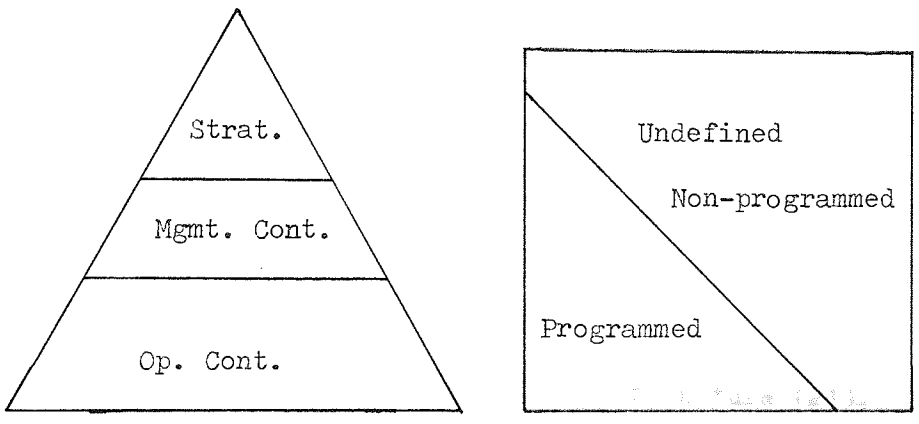
2.1 Impact on Automation

Regarding the distinction between programmed and non-programmed decisions it can be maintained that the former is more mechanizable than the latter. However, there is a subtle difference between being programmable and being mechanizable. Programmability does not automatically imply mechanization, it only indicates specificity which can facilitate mechanization insofar as available resources and complexity of the considered situation allow. In this respect, mechanization implies rigidity since there is an upper limit on coping with complexity and retaining in the same time the required flexibility. Consequently confusing the two notions may result in an over-simplification of the decision which, as was seen earlier, can lead to user conflict and dis-satisfaction (19). This constitutes a major drawback of current applications packages. This problem can be resolved by adhering to the relationship between flexibility and independence which asks for a fundamental change in the architecture of packages. Considering that resources are limited and situations vary continuously with time mechanization can become more flexible by designing

---

(19) This question constitutes one of the main criticisms of management science, thus Churchman et al emphasize that "Much of management science has been conducted under the very naive philosophy that a certain kind of reason must prevail, and that once this reason has been made clear, the manager will either accept it or be charged with gross negligence or, still worse, gross stupidity", Churchman, C.W.; Schainblatt, H., "The Researcher and the Manager: A Dialectic of Implementation", Management Schience, 1965, Vol.11, No.4, pp. B-69/87.

holons that can be coupled and decoupled according to needs (20). This does not only apply to programmed decisions, it is as applicable to non-programmed situations since the resulting architecture is not pre-determined and fixed as exhibited by existing packages, rather it is constructed by the user and therefore the architecture is customer-tailored as emphasized above.



Hierarchy and Programmability of Decisions

Figure (37)

Considering the above two models within the context of the nature of definitions, as shown in Figure (37), it can be concluded that a strategic decision is not wholly unknown or non-programmable otherwise it may be unsolvable, at the other extreme, an operational decision is not completely programmable or specified otherwise there would not have problems such as these encountered

---

(20) This is covered in Chapter 8.

in inventory control and scheduling. Consequently a programmed decision, as implied by the concept of holons, should not be seen as fully mechanizable.

AUTOMATION AND THE DECISION MAKER

It may have become evident from the above that the previously discussed three-fold, viz. bias, independence and flexibility, characterizes the process of decision making. In this respect it has been proposed that flexibility may not only be considered and treated as a relevant property more than that it has to be seen as an overall organizational objective extending over the entire organizational structure (21). The criteria used to discern between programmable and non-programmable decisions are used to distinguish two types of flexibility, viz. external and internal flexibilities. External flexibility assists in the endeavours to combat and minimize external/environmental perturbations that are likely to induce unforeseen problems. Internal flexibility, on the other hand, is complementary to the former, and its fundamental purpose is to provide a 'cushion' for response to disturbances (22). However, in this respect the current package architecture has, as was seen earlier, failed to cater for such requirement.

---

(21) Ansoff, op.cit., p.56.

(22) Ibid.

The psychological effect of failing to recognise the relative importance of flexibility to the decision maker can have serious drawbacks (23). Reducing the decision maker's degrees of freedom leads to complicated feelings of lack of choice, pressure and psychological failure, that may induce other feelings of helplessness and decreasing sense of responsibility. Consequently the decision maker may become more dependent upon what has restricted his freedom. In this regard the decision maker may find himself 1) subjected to increasing amounts of psychological failure, even when the package is successfully operating; if the decision maker adheres to rationality he may succeed in this respect and fail as a human being (24) that may result in a psychological conflict. 2) He may be forced to reduce inter-departmental interactions as concessions for the sake of automation (25).

Thus it follows that a package in the form of ready-to-implement solution which is characterized by being inflexible and static, with respect to the definition/solution it comprises,

---

(23) Argyris, "Management Information Systems: The Challenge to Rationality and Emotionality", op.cit., "Interpersonal Barriers to Decision Making", Harvard Business Review, 1966, 44, 2, pp.84-97.

(24) This is a result of misconceiving management practices which was guarded against by Churchman et al, n.q. (19), "The Researcher and the Manager", op.cit. Also, Mason, R.O.; Mitroff, I.I., "A Program for Research on Management Information Systems", Management Science, 1973, Vol.19, No.5, pp.475-487.

(25) Argyris, C., "Personality and Organization", (Harper & Row, N.Y., 1965).

besides its bad effect upon data processing function as previously discussed, has comparable degenerative consequences upon the abilities of the decision maker or problem-solver/definer. This becomes more significant when both the data processing and the decision makers are considered as interacting components of a larger encompassing context, viz. the M.I.S.

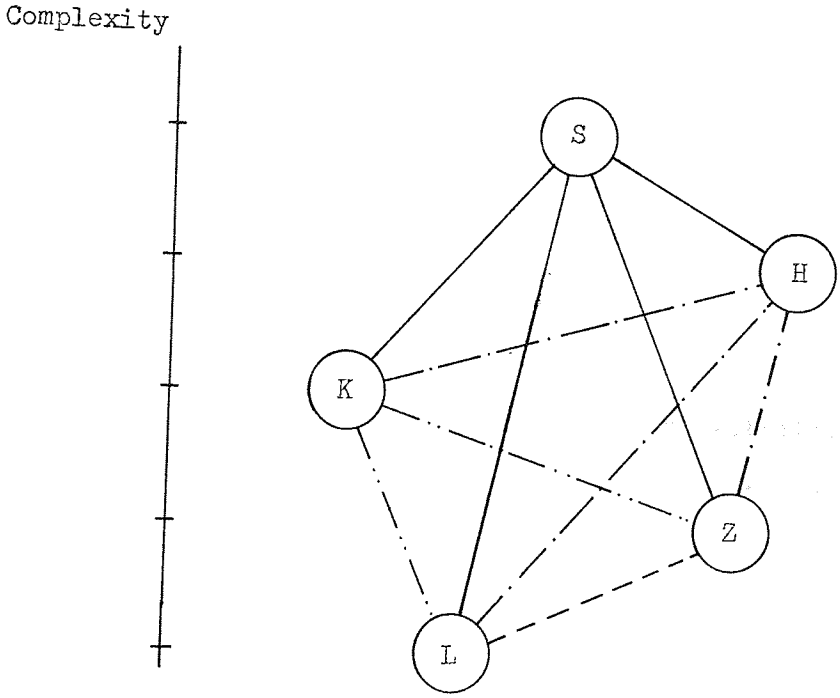
A HOLONIC INTERPRETATION OF CHURCHMAN'S MODEL FOR PROBLEM-SOLVING

Recently a significant advance towards the identification and understanding of the methodological bases of problem-solving has been established around the works of five prominent philosophers with each representing a specific approach (26). Consequently, the potential result that may be obtained through the application of a certain framework, or inquiring system, is bound to be characterized by the fundamental philosophical premises and the methods allowed through that inquiring system. Therefore, Churchman's model can be used to serve the purpose of providing design methods that are first compatible with the top-down design approach suggested earlier and secondly consider rigorously the decision maker's requirements and environment.

---

(26) Churchman, "The Design of Inquiring Systems", op.cit.

In this set up the concept of holons offers a useful hierarchy through which the relationships among the five methodologies and the usability of each can all be identified. Also the usefulness of the concept of holons is rendered to its recognition of automation and its emphasis upon the maintenance of flexible structure.



Key:

S: Singerian I.S. ————

H: Hegelian I.S. — . — .—

K: Kantian I.S. —.—.—

Z: Leibnitzian I.S.— — — —

L: Locke I.S.

Methodologies as Holons

Figure (38)

The functioning of the hierarchy as illustrated in Figure (38) can be explained as follows. Any higher level holon, i.e. Singerian, Hegelian, Kantian or Leibnetzian respectively in descending order, can activate and communicate with any or all lower holons by sending and/or receiving messages directly or indirectly through the appropriate channels which are represented in the diagram by various kinds of lines. Thus, the output hierarchy of the Singerian inquiring system comprises some or all holons that are connected to S by solid lines. Also, the output hierarchy of the Hegelian inquiring system comprises some or all holons that are connected to H by dashed-single dotted lines. Similarly, the output hierarchy of the Kantian inquiring system may include any holons that are linked by dashed-double dotted lines. Finally, the output hierarchy of the Leibnitzian inquiring system consists of the dashed line that links it to Lockean inquiring system. However, such structure demonstrates enough flexibility to activate whatever required mechanism to handle a certain problematic situation. The choice of the mechanisms is partly determined by their conceived respective characteristics as briefly outlined below, and partly by the complexity of the problem.

1. Properties of Lockean Inquiring System

Locke's philosophy attaches much weight and emphasis to sense data. Its basic premise is that all knowledge of material

existence depends upon sensation. However, Locke then qualifies this assertion by pointing out that sensations, themselves, obstruct the build-up of knowledge of objects (27). Consequently, the Lockean methodology can be interpreted as being empirical (28). Being based upon sensations then judgment is bound to contain bias to what constitutes the observed object. Also, the guarantor of Lockean Systems is its generality. Thus, Lockean inquiry can be useful in the context of well structured and least uncertain problematic situations.

2. Properties of Leibnitzian Inquiring System

Formal logic and axioms form the foundations of Leibnitzian thinking. Mathematical or symbolic model building are the most notable characteristics of Leibnitzian inquiring systems. Thus the building blocks of such models are the elementary formal truths. Propositions are often tested within the context of primitive truths by the application of the deductive mode of logic. The difference

---

(27) Locke, J., "An Essay Concerning Human Understanding", (Everyman's Library, 1972), Book IV, Chap.3: "The Extent of Knowledge"; Chap.4: "The Reality of Knowledge".

(28) Locke differentiates between knowledge of matters of fact and of relations between ideas. Accordingly the latter rests upon sheer observation, 'internal' or 'external'. The outcome is a reduction of 'experience' to sensations as the constituents of all observation, and 'thought' to external associations among these elements both sensations and associations being supposed to be merely mental or physical, ibid.

between Leibnitz's philosophy and that of Locke may be equivalent to the difference between deductive and inductive logic respectively. It is maintained that Leibnitz inquiry identifies, besides what is known through experience, certain 'innate ideas' and 'innate principles', which can be known independently of experience (29). It follows that the guarantor of Leibnetzian models may be identified through validity, strength, rigour and consistency. This is because the mathematical orientation of such models which delimits the capacity of Leibnitz's inquiry to that of the available analytical techniques. This is also reflected upon the nature of problematic situations that can be investigated. These situations have to be unambiguously described. Examples are Operational Research models such as inventory control, mathematical programming, etc.

3. Properties of Kantian Inquiring System

Kant's philosophy is distinguished by its idea of a priori knowledge. This type of knowledge is concerned with space and time and causality and comparison which are the fundamental elements of building relationships (30). Also, Kant emphasized that a priori knowledge is not purely 'analytic',

> "i.e. such that the opposite would be self-contradictory" (31).

---

(29) Russel, "The Problems of Philosophy", op.cit., p.41.

(30) Ibid., p.48; Kant, "Critique of Pure Reason", op.cit., p.72.

(31) Russell, "The Problems of Philosophy", op.cit., p.46.

He concluded that pure mathematics, though a priori, is synthetic. Therefore Kant's inquiry is fundamentally concerned with the development of at least two models (32). The type of problems that can be dealt with encompasses these that are covered by Leibnitz and/or Locke. Consequently by activating both the Leibnitzian and Lockean inquiring systems alternative synthetic solutions can be generated to select the best suitable solution for partially specified problems.

4. Properties of Hegelian Inquiring System

Hegel's philosophical thesis can be summarized as that everything less than the whole is fragmentary, and, thus, incapable of existing without the complement supplied by the rest of the world (33). This incompleteness appears, according to Hegel, equally in the world of thought and in the world of things. Thus, considering any abstract or incomplete idea, it is found, on examination, that if its incompleteness is forgotten, the person concerned will become involved in contradictions. These contradictions turn the considered idea into its opposite or antithesis. To escape the person was to search for a new less incomplete idea which is the synthesis of his original idea and its antithesis (34).

---

(32) Mason; Mitroff, op.cit.

(33) Mure, G.R.G., "The Philosophy of Hegel", (Oxford Univ. Press, London, 1965), pp.1-40.

(34) Ibid.; also Russell, "The Problems of Philosophy", op.cit.

The same is repeated again with the new idea, until the person reaches the 'Absolute Idea' which as Hegel holds, has no incompleteness, no opposite, and, therefore, requires no further elucidation.

Now since attainment of the 'Absolute Idea' is practically unachievable, any other idea is, essentially, incomplete (35). Consequently, Hegelian inquiry may involve all preceding types of inquiries. That is, it may activate Kantian inquiry to yield two opposing synthetic Leibnitzian models. These consequently are applied to the same fact set as supplied by the Lockean inquiry. The two opposite Leibnitzian models are supposed to derive different meanings from the data set as each model interprets data differently so emphasizing the relativistic qualities of information. Consequently, higher echelons of an organization have to be provided with the facility to construct such conflicting models as decision makers who belong to these echelons are supposed to deal with ill-structured problems and often conflicting objectives. Thus, the crux of Hegelian inquiring system is the maintenance and enhancement of conflict to allow for decision makers to disagree.

---

(35) The earlier discussion of definitions is also applicable to the notion of an 'idea'. That is, since an idea constitutes a mental image or mapping of the world; and since the mind is finite, according to Descartes, then the idea is bound to be incomplete in some respect; Cf. Chap.1; also, Anscombe; Geach, op.cit.

5. Properties of Singerian Inquiring System

This inquiring system is advocated to encompass all other inquiring systems. That is it can trigger into action any type of inquiry depending upon the complexity of the situation at hand. However, this inquiring system is regarded to be most suitable for problems that are characterized with an exceedingly high degree of uncertainty (36). The main theme of Singer's inquiry is its endlessness. It is maintained that the essential (37) features of this inquiring system may include the following:

1. The purpose of the inquiring system is to develop the ability to choose the right means to achieve the desired goals.
2. Its measure of performance has not been developed.
3. The environment of the inquiring system is a co-operative environment.

   "One sees how fuzzy the boundaries of the inquiring system become because inquiry is evidently needed to create co-operation and co-operation to create inquiry" (38).

4. The decision maker is any one.
5. The designer is any one.

---

(36) Churchman, "The Design of Inquiring Systems", op.cit., pp.186-205; also, Mason et al, op.cit.

(37) Ibid., pp. 200-201.

(38) Ibid.

That is, the usefulness of this methodology is most identifiable with top management whose function is that of interacting with the environment to ensue control over a portion of it.

IMPLICATIONS AND CONCLUSIONS

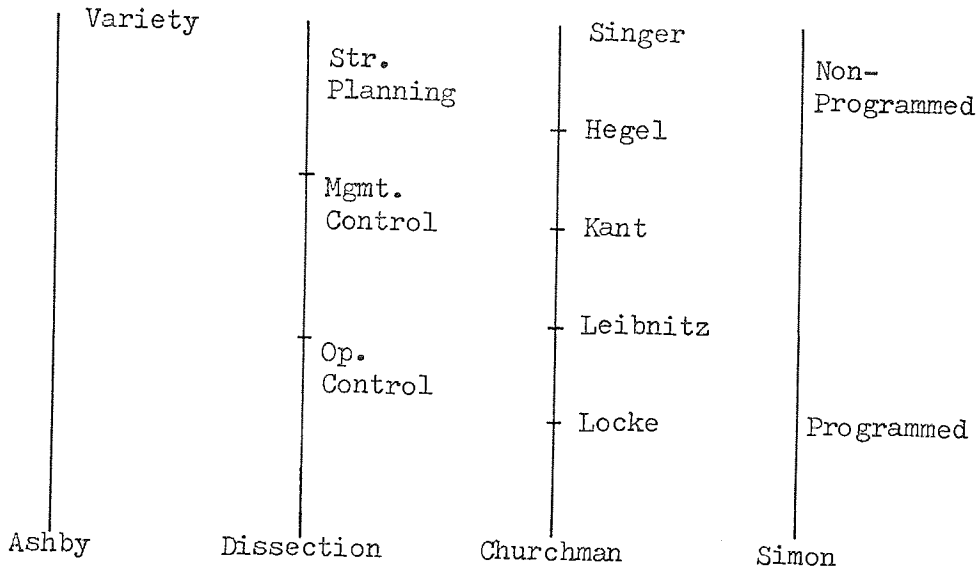
Thus, the above shows that Churchman's model when considered within the framework of the concept of holons meets the specifications of top-down approach. Also the concept by being essentially flexible adds this property to Churchman's model. Consequently, the holonic concept serves as a unifying agency between design methodology and that of problem solving.

In the following the implications of the holonic arborization or vertical structures and reticulation or hierarchical levels on one hand and the decision maker on the other is investigated.

1. Activity Arborization and Hierarchical Reticulation

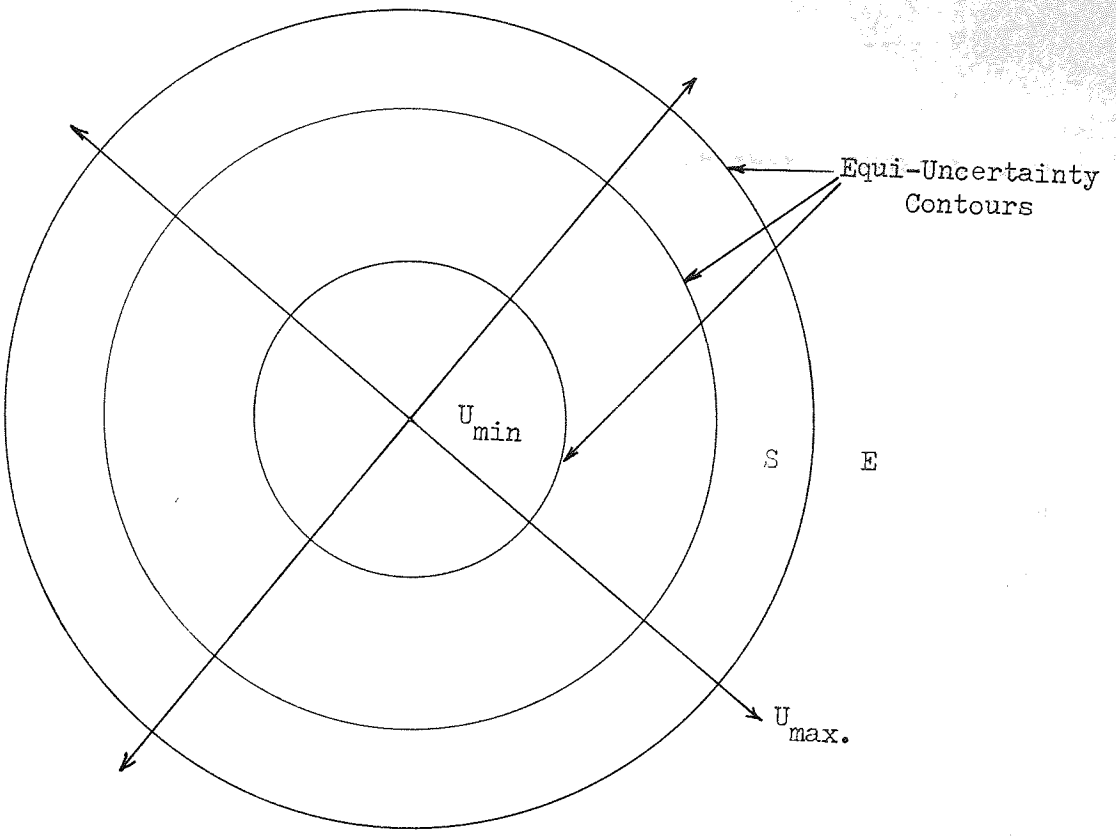
Taking the amount of uncertainty U as a classificatory criterion, and considering Figure (39), that any one organization can be dissected into its main activities such as production, sales, management services, finance, R & D, etc., then the organization as a whole and individual activities can each be

divided into three consecutive, though partially overlapping levels. Namely, top-level management which corresponds to maximum uncertainty, middle-management which corresponds to medium uncertainty, operational control which corresponds to minimum uncertainty.



Schematic Representation of the Various Taxonomics w.r.t. Uncertainty

Figure (39)



Equi-Uncertainty Contours

Figure (40)

In conjunction with Figure (40), let E be the environment, and S be the organization. However, if S is defined as the Cartesian product of the family of activities $(A_i)_{i \in I}$, then it can be denoted by:

$$S \subset \prod_{i=1} A_i$$

Similarly, relative to the organization S, the environment E is the corresponding family of sub-environments $(B_j)_{j \in J}$, such that:

$$E = \bigcup_{j \in J} B_j$$

But for every i there is j, i.e. $i \in J$, such that

$$A_i \cap B_j = A_i$$

then

$$S \cap E = S$$

That is, every activity of S is also an activity of E. But the total amount of uncertainty* within S must be less than the total amount of uncertainty outside it, i.e. $U_S \leq U_e$. Conversely, if the organization's internal uncertainty is equal to that of the environment's, then the organisation's characteristics and qualities that distinguish it may all disappear. In other words, the difference $U_S - U_e$, which is negative, accounts for the organization, pattern and adherence of S.

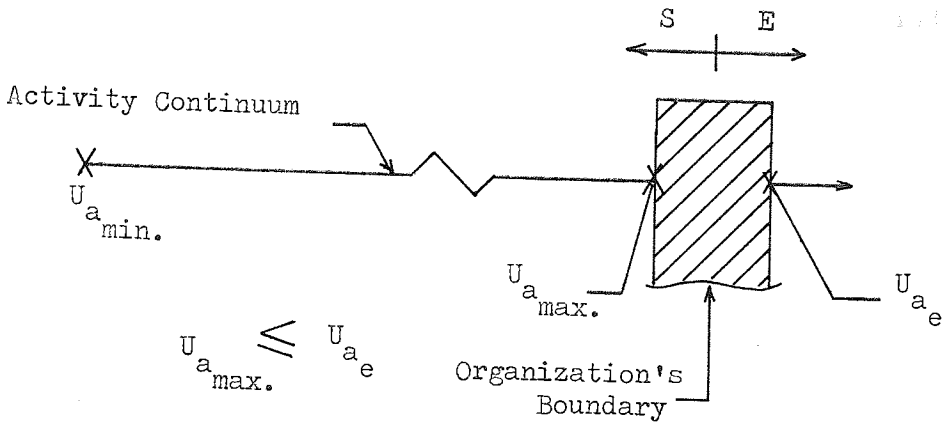
Consequently, the amount of uncertainty that is associated with a certain holon, that belongs to an activity, depends on its allotted level on the continuum. Thus holons that lie near the lowest hierarchical level are presumed to deal with the least amount of uncertainty and a Lockean, i.e. empirico-inductive, inquiry may suffice the corresponding variety since they do not activate subordinate holons and being activated in response to superordinate holons (39). However, as holons traverse the activity's continuum,

---

(39) This is because low-level holons are asked to do precisely what they have been designed to do. This, however, should not forbid them from moving freely within their defined space.

* Uncertainty in terms of disorganization.

they are bound to encounter higher variety and uncertainty as illustrated in Figure (41). Holons that are most near to the



Activity Continuum and Uncertainty

Figure (41)

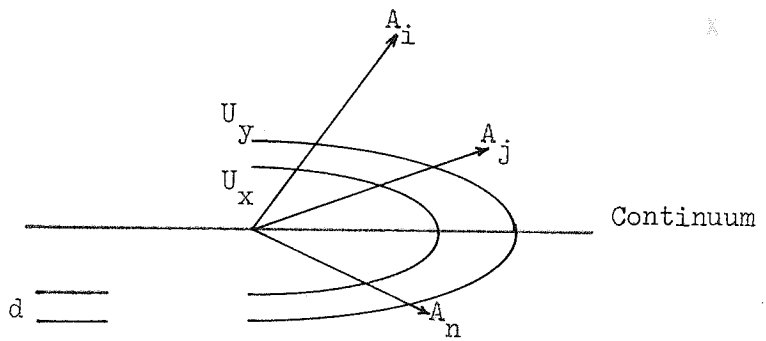
boundary are supposed to interact with the largest uncertainties. More precisely the uncertainty level of the inside surface of the boundary-shell, i.e. $U_{a_{max.}}$ should be less than that of the outer-most surface, i.e. U_{a_e}. The difference between the two, i.e. $U_{a_{max.}} - U_{a_e}$ is attributed to the endeavours of top management (40) and thus a Hegelian or even Singerian inquiry may be used to cope

---

(40) It should be noted that U_{a_e} is equivalent to the environment's uncertainty as referenced to that particular activity. Also, as Dewey maintains, more differentiation leads to enlarging the immediate relevant environment. Consequently, the more differentiated the activity is, the wider is its potential environment. And the wider the environment, the higher is the potential uncertainty, Dewey, op.cit.

with the high variety. However, the managerial structure can be conceived of as concentric contours (41). Starting from the outer surface of the boundary and moving inwards the first identifiable shell is that of top management followed by middle management and ending with the operational level.

Consequently, all holons that exhibit individually an equivalent set of governing rules and strategies to cope with varieties may, thus, all form a contour that is characterized by a particular uncertainty range since each contour is not a line per se, but a shell. Thus the activities of each shell may



$$U_x(A_i) = U_x(A_j) = \ldots \neq U_x(A_n)$$

also $$U_y(A_i) = U_y(A_j) = \ldots = U_y(A_n)$$

such that $$U_y(A_i) - U_x(A_i) \geq 0$$

implying that $U_y \geq U_x$ irrespective of i, j, ..., n

Equi-Uncertainty Contours

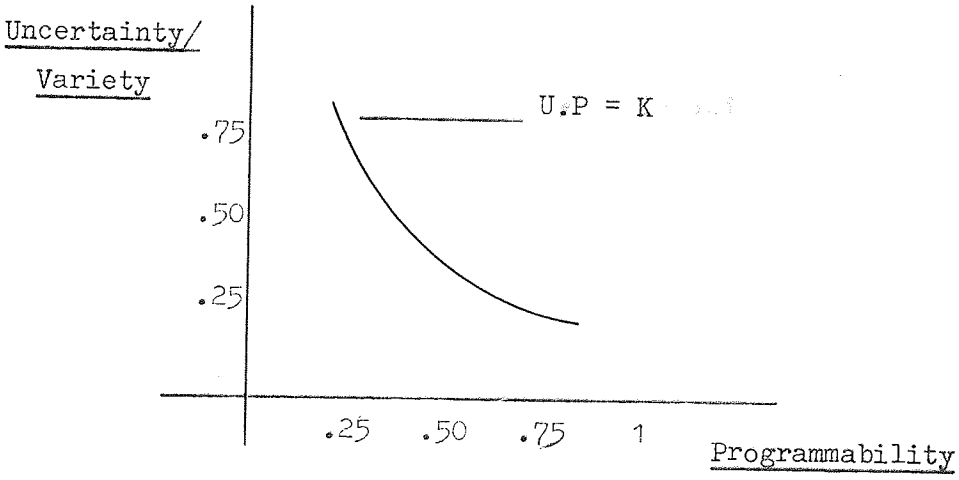
Figure (42)

(41) That is points on the same curve/contour have the same amount of uncertainty. The activity's uncertainty continuum is regarded to be analogous to electric field lines and the equipotential contours which are orthogonal to them are analogous to equi-uncertainty.

assume different uncertainties as their position varies in accordance with pertinent events.

2. Uncertainty and Programmability

As each major activity extends throughout the organizational hierarchy, and as there can be a corresponding scale of uncertainty, then, specificity may couple the two continuums together. However, in order to programme a function, a minimum level of specification must be attained (42). It is



Uncertainty vs. Programmability

Figure (43)

---

(42) Newell, A.; Shaw, J.C.; Simon, H.A., "Chess Playing Programs and the Problem of Complexity", In Feigenbaum; Feldman, (Eds.), op.cit., pp.39-70.

clear from the previously outlined taxonomies that particularly for higher level decision makers, and sometimes lower ones, not every function is specifiable. This is also explainable in terms of the complexity barrier (43). Then, first, from the definition of the barrier, and secondly from the parallelism between complexity and uncertainty, and thirdly from the holonic representation of the decision making process, it is reaffirmed that the design of the independent parts and their associated high flexibility, variety and impartiality will tremendously help to penetrate that barrier which is seen as an upper bound for programmability and thus achieving the required viability. This is so specifically when the non-linear relationship as that of Figure (43) between uncertainty or variety and programmability is considered.

---

(43) Beer points out that for a system to be viable it has to advance towards a minimum complexity and if the system is able to penetrate that complexity barrier it then secures the following properties: 1) The ability to make a response to a stimulus which was not included in the list of anticipated stimuli when the system was designed. 2) It can learn from repeated experience what is the optimal response to that stimulus. 3) It grows. 4) It renews itself. 5) It becomes robust against internal breakdown and error. 6) It continuously adapts itself to a changing environment and so it increases its chances of surviving conditions which had not been foreseen by the designer. Beer, "Decision and Control", op.cit., pp.256-258.

CHAPTER 8

CONCLUSIONS: A PACKAGE AS A HIERARCHY OF HOLONS

This thesis has considered the question of man-machine interface design by concentrating on two objectives. The first is to attempt to develop a systemic design framework which is self consistent, compatible with other concepts, and applicable to real situations. The second objective is to examine the framework and its consistency, compatability and applicability in the theoretical and philosophical context of a real situation. In order to limit the scope to manageable proportions, the situation chosen is that of applications packages design.

This theme has implied the examination of the possible role a package may play from various angles. The result has been the rejection of the present package architecture on the grounds of being rigid and proposes instead the design of independent modules as an essential condition to attain other properties such as flexibility required to match the dynamic properties of the users' environment and also to increase the variety of software.

This has been reached by looking at a package as a ready-to-implement solution. Such has indicated the examination of definitions and solutions in order to establish their mutual dependence and of being open-ended and dynamic. In this respect, a definition, being fundamentally a model of some phenomena as conceived by the designer concerned, is essentially imperfect and

carries a varying degree of bias and content depending upon the designer's conceptual framework and background. Consequently the inherited bias and content are responsible for the rigidity of the package and its prospective. It thus follows that to reduce the effect of bias and content the designer should provide adequate facilities to cater for modifiability and adaptability of the package. Failure to recognise this has been attributed to the poverty of understanding of existing systems designers of the various methodological approaches. This has led to ad hoc inadequate designs where, for example, the so-called modules are as large as complete packages.

Some important design requirements, viz. flexibility, independence and impartiality have been identified and further studied in relation to order, i.e. the sequence or pattern or arrangement of parts or instructions, and size, i.e. the number of instructions within a part. This study has concluded that these design requirements are rendered more difficult to attain as both order and size increase. Also, as far as the user is concerned, order must be only partial so that he may be left with enough freedom to cope with unprogrammed disturbances.

The concept of holons has been found to provide the required conceptual model which is rigorously based upon a true understanding of the systems concept. Within this model the problem of designing a flexible, independent and impartial package structure is largely resolved by defining a package as a self-contained, rule-governed stable open-system. This new look of a package emphasizes the need for openness, flexibility, hierarchical

structuring of its constituent modules or holons which are also self-contained, and the provision for the inevitable subsequent entwining with other packages or programmes. Additionally, the holonic treatment of automation shows a clear understanding of the relationship between a holon and its environment by insisting that each holon must have its canon (1) and strategy. The holon's canon refers to its invariant properties and structure, i.e. its specifications. Thus the invariant behaviour of a holon is explainable in terms of its canon whereas its strategy is responsible for its behaviour in view of environmental perturbations.

Two significant environments of an application package have been considered. The first is the M.I.S. which is defined to contain all automated and non-automated procedures. Consequently a package is contained in an M.I.S. and not vice versa and thus emphasizing the control function of users. In this respect two M.I.S. design approaches have been discussed, viz. top-down and bottom-up. It does not appear possible to use either approach by itself. The ability to identify mutual interdependence between the two approaches is of much importance for an effective use of either one. It is envisaged here that such an ability can be enhanced by clearer understanding of the mechanisms, advantages and limitations of each. Also this knowledge may be easier to acquire when designers - i.e. users of the approaches - develop more rigorously based conceptual frameworks. The absence of such knowledge and frameworks have resulted in indiscriminate use of the bottom-up approach. This has, in turn, resulted in over-complicated and ad hoc designs.

It has been shown that current package architecture causes degeneration of the design skill and expertise of data processing

(1) For more details see p. 56

practitioners by making the users more dependent upon package suppliers. The more a company opts for packages, the less deign experience its designers may get and the less attractive it may become to competent designers to stay thus pushing for more ready-to-implement solutions.

The second influential environment of a package is software. It is noted that the unwary user can lose his independence and freedom by letting the manufacturer control his software. Thus the user who accepts manufacturers' software as given and does nothing to control it is putting himself in a very awkward situation. Alternatively, if the user values independence and recognises his inevitable need for adaptability and modifiability he will have to develop his own buffer, i.e. the middleware (2), to reduce the effect of the built-in bias in the manufacturers supplied software. Also it is indicated that the current package architecture, with its inseparable wholes, prohibits the use of its constituent parts in building new programmes and consequently leads to duplication, i.e. contradicting the main idea behind packaging.

The above deficiencies have been demonstrated through the examination of a typical package, namely CLASS, within the context of the three design criteria, i.e. flexibility, bias and independence. Flexibility is much reduced because the package represents a typical example of static definition which impairs

---

(2) As defined on p. 93

modifiability and adaptability. Also it is a closed-end system that recognises neither the need for extensibility nor the possible use of its parts as building blocks. Bias is much increased mianly because CLASS is written in a low level language that dictates a particular software and hardware configuration, and thus makes CLASS intransferable.

Thenceforth the discussion has concentrated upon finding an alternative package architecture having the three basic properties: (a) flexible structure; (b) impartiality through independence of modules; (c) compatability with the characteristics of the users' process of decision making. In the search for an alternative package architecture, software has been treated as communication channels. This conception has identified the relationship between the user and the designer as being two components of the communication system and consequently they must be compatible with regard to understanding and familiarity of each other's requirements and constraints in order to attain adequate communication. Also it has provided a rigorous theoretical framework compatible with that of the holons and within which concepts such as requisite variety, coupling and/or control, entropy, etc. have all confirmed the inadequacy of the current package architecture and supported that of designing small and independent parts. It is proposed that such an architecture can be designed by the use of the top-down design approach which may be made practicable by the proper application of the technique of structured programming.

The need and suitability of the design of parts instead of the existing package form have also been reaffirmed by examining the various kinds of decision a user may have to consider and the processes of problem solving he has to apply in order to arrive at such decisions. This examination has resulted in the following:

i) the higher the level of uncertainty, the more indefinite and ill-structured is the decision making process;

ii) the more indefinite the decision making process, the greater is the need for flexible software;

iii) more flexibility implies less dependence upon manufacturers or external sources;

iv) the more flexible the structure is, the greater the number of degrees of freedom that is built into it, by increasing the ability to couple or decouple proper modules without incurring serious disturbances and/or modifications. This may enhance the software's capability of producing enough variety to match that introduced to the decision maker by the environment.

Starting from these premises, a new design and implementation architecture are introduced. The main concept of the proposed method is not new. Its basic characteristics are familiar and exactly analogous to methods used in the assembly line in which

design and production are fundamentally concerned with: (a) production of parts; (b) particular emphasis upon compatabilities. When the analogous method is used in conventional assembly lines, the above two characteristics do not pose any serious problems through the establishment of rigorous standards. In the case of software, using present methods, such requirements represent the main sources of problems and headaches; the proposed method should largely eliminate such difficulties. If the typical adopted design strategy is that of applying the top-down approach, and of limiting the size of modules to only few statements - through the use of structured programming which also increases the transferability of the modules since it is a sort of high level language - all within the framework of the concept of holons, the problems can be confined and tackled. However, at present, lack of standards presents limitations to the proposition and precludes its immediate full scale application. For example, the optimum size of a module needs to be calculated before a truly independent holon can be produced. There is at least another Ph.D. project in the software field here which would attempt to devise and test a technique for producing these holons or modules.

The holonic understanding implies the design and production of self-contained, open and stable modules. The condition of stability is essential for any building operation. Once again, being an autonomous whole on its own, a holon, i.e. a module, acquires the required degree of independence and consequently the important property of multi-purposefulness. At

the same time, emphasizing the integrative role the module has to play as a constituent part of a larger super-whole, the module's canon carries a certificate of its specifications along with the range of uncertainty it can cope with as determined by its strategy.

Consequently, an application programme can be conceived of as a dynamic hierarchy of holons in which higher modules communicate and activate lower ones and in other instances can themselves be activated by other holons. Thus, the position of the module within a programme is determined, at a given time, relative to the definition of the situation. Consequently the programme itself becomes synchronized with the problems and their variations. Also it may be used in more than one programme at the same time thus effectively reducing duplication of effort since software replication costs of a prototype are only those of the media on which modules are copied. As a result design of subsequent new programmes can largely be reduced, as time progresses, to that of coupling modules according to requirements. Hence as each new module is designed the variety of software as a whole increases. Also flexibility can be achieved through the arbitrary inclusion and/or exclusion of modules as required. In this manner a user does not have to be constrained by a predetermined structure, but contrarily he may be able to tune his programmes as he needs as a consequence of him acquiring

enough flexibility. Another important consequence is that of reducing bias as a result of increasing independence of modules and by not having to comply with an unalterable architecture. Additionally, programme resilience and modifiability can all be increased.

The design and production of holons can be carried out by the same traditional software sources, e.g. manufacturers, software houses, etc., as well as in-house. The danger of importing rigidity, bias and dependence is minimized since in either case the resulting programmes tend to be more or less tailor-made. Any one source may produce families of modules whose members can be classified according to various criteria such as precision, robustness, time-space performance and generality (1). As far as precision is concerned the choice may include specifications like 'width' of characters and 'size' of address, etc., whereas the choice of robustness means the trade-off between reliability and compactness in space and time. Generality can be arranged according to the degree of freedom entertained by the user to adjust the module's parameters at run time. Other criteria such as choice of algorithm, choice of accessing mode, etc., can also be considered.

---

(1) McIlroy, M.D., "The Outlook for Software Components", In "Software Engineering: State of the Art Report No.11, op.cit., pp.243-252.

This thesis attempts to present a useful and consistent systemic design framework. Its relevance and applicability to real situations has been shown by considering the design problem of applications packages. Through the framework it has been possible to identify the close relationship between programme design and the characteristics of user's practice in problem solving. This is tackled by the identification and extension of the concept of holons to provide a rigorous conceptual model that may resolve the problem of system-subsystem relationships. Thus the definition of a holon is considered to be adequate for a module.

Finally, a deeper consideration of the methodological theme of this thesis will reveal its wider applicability not only to automated applications programmes, but also equally to non-automated procedures of man-machine systems design in general. This is evident in the great emphasis the thesis has placed upon the dynamic properties of systems behaviour and their design requirements by envisaging components, and their characteristics, through their contextual relationships and not vice versa and that is the crux of systems thinking.

It may be appropriate to end this thesis by emphasizing the need for further research. Two specific areas are software design and management information systems. Research in software is needed for the following reasons:

i) to attempt to define the optimum size of a holon in terms of the number of instructions;

ii) to select or devise the appropriate programming language;

iii) to test the practicability of building and rebuilding programmes by coupling and decoupling holons.

This may be started by testing the practical usefulness of the technique of structured programming in achieving the above goals.

Research in management information systems design might examine:

i) the more accurate defining of the relationship between programmability and uncertainty in relation to the process of decision making;

ii) the development of a method by which changing real situations, as identified by design makers, can be matched and be built up into programmes from the available holons.

Another interesting area for research is to study more deeply the interface nature of computer programmes from the viewpoint of communication and its effects on programme design.

REFERENCES

1. Ackoff, R.L., (Ed.), "Progress In Operations Research", J. Wiley & Sons, New York, 1961.

2. _____, "Management Misinformation Systems", Management Science, 1967, Vol.14, No.4, pp. B-147/156.

3. _____; Sasieni, M.W., "Fundamentals of Operations Research", J. Wiley & Sons, New York, 1968.

4. _____, "Scientific Method: Optimizing Applied Research Decisions", J. Wiley & Sons, New York, 1968.

4a. _____, "Towards a System of Systems Concepts", Management Science, Series A - Theory, Vol.17, No.11, July, 1971, pp.661-671.

5. Alexander, M.J., "Information Analysis", Sci. Res. Ass., 1974.

6. Anscombe, E.; Geach, P.T., (Trans.), "Descartes Philosophical Writings", Thomas Nelson & Sons Ltd., London, 1971.

7. Ansoff, H.I., "Corporate Strategy", Penguin Books, 1968.

8. Anthony, R., "Planning and Control Systems: A Framework for Analysis", Harvard Univ. Press, 1965.

9. Argyris, C., "Personality and Organization", Harper & Row, New York, 1965.

10. _____ , "Interpersonal Barriers to Decision Making", Harvard Business Review, 1966, Vol.44, No.2, pp.84-97.

11. _____ , "Management Information Systems: The Challenge to Rationality and Emotionality", Management Science, 1971, Vol.17, No.6, pp. B-275-292.

12. Ashby, W.R., "General Systems Theory as a New Discipline", General Systems, 1958, Vol.3, pp. 1-6.

13. _____ , "An Introduction to Cybernetics", Methuen & Co. Ltd., London, 1971.

14. _____ , "Systems and Their Informational Measures", In G.J. Klir, (Ed.), loc.cit., ref.(58), pp. 78-97.

15. Beer, S., "Cybernetics and Management", English Univ. Press, London, 1959 .

16. _____ , "Decision and Control", J. Wiley & Sons, London, 1966 .

17. Bertalanffy, von L., "Problems of Life", Watts & Co., London, 1952 .

18. _____ , "General Systems Theory: Foundations, Development, Application", Allen Lane The Penguin Press, London, 1971 .

19. _____ , "The History and Status of General Systems Theory", Academy of Management Journal, 1972, Dec., pp.407-426.

20. Boehm, B.W., "Software and its Impact: A Quantitative Assessment", Datamation, 1973, May, pp.48-59.

21. Boon, C., (Ed.), "Operating Systems", Infotech State of the Art Report No.14, 1972, loc.cit., ref (55).

22. Boulding, K.E., "General Systems Theory: The Skeleton of Science", In W. Buckley, (Ed.), loc.cit., ref. (24), pp.3-10.

23. Brillouin, L., "Thermodynamics and Information Theory", In W. Buckley, (Ed.), loc.cit., ref (24), pp.161-165.

24. Buckley, W., (Ed.), "Modern Systems Research for the Behavioural Scientist", Aldine Pub. Co., Chicago, 1968 .

25. Bunge, M., "A General Black Box Theory", Philosophy of Science, 1963, 3, pp.346-358.

26. _____ , "Analogy, Simulation, Representation", General Systems, 1970, Vol.15, pp.27-34.

27. Chapin, N.; House, R.; McDaniel, N.; Wachtel, R., "Structured Programming Simplified", Computer Decisions, 1974, June, pp.28-31.

28. Churchman, C.W.; Ackoff, R.L.; Arnoff, E.L., "Introduction to Operations Research", J. Wiley & Sons, New York, 1957 .

29. _____ ; Schainblatt, H., "The Researcher and the Manager: A Dialectic of Implementation", Management Science, 1965, Vol.11, No.4, pp. B-69/87.

30. Churchman, C.W., "The Systems Approach", Dell Pub. Co., New York, 1968 .

31. _____ ; Ackoff, R.L.; Arnoff, E.L., "Analysis of the Organization", In J.A. Litterer, (Ed.), loc.cit., ref. (68), pp.274-286.

32. _____ , "The Design of Inquiring Systems", Basic Books Inc., London, 1971 .

33. Cyert, R.M.; March, J.G., "Organizational Factors in the Theory of Oligopoly", Qrtly. J. of Econ., 1956, 70, pp.44-64.

34. d'Agapeyeff, A., "Discussion", In P. Naur et al, (Eds.), loc.cit., ref. (80), pp.22-23.

35. Dewey, J., "Logic: The Theory of Inquiry", Henry Holt & Co., New York, 1938 .

36. Dijkstra, E.W., "Programming Considered as a Human Activity", Proceedings of IFIP Congress 65, Spartan Books, Washington, 1965 .

37. _____ , "Complexity Controlled by Hierarchical Ordering of Function and Variability", In Naur et al, (Eds.), loc.cit., ref. (80), pp.181-185.

38. Duncan, "Discussion", In "Software Engineering", Infotech Report No.11, loc.cit., ref. (54), pp.171-172.

39. Edwards, E., "Communication Theory", In W. Singleton, et al, (Eds.), loc.cit., ref. (100), pp.37-53.

40. Eilon, S., "What is a Decision", Management Science, 1969, Vol.16, No.4.

41. Emery, F.E., (Ed.), "Systems Thinking", Penguin Books, 1970 .

42. Feigenbaum, E.A.; Feldman, J., (Eds.), "Computers and Thought", McGraw-Hill, New York, 1963 .

43. Forrester, J.W., "Industrial Dynamics", M.I.T. Press, Cambridge, Mass., 1961 .

44. Gerard, R.W., "Units and Concepts of Biology", Behavioural Science, 1958, 3, pp.197-206.

45. Gillette, R., "Discussion", In Naur et al, (Eds.), loc.cit., ref (80), pp.39.

46. Gorry, A.; Morton Scot, M., "Framework for Management Information Systems", Sloan Management Review, 1971, Vol.13, No.1, pp.55-70.

47. Gray, W.; Rizzo, N.D., (Eds.), "Unity Through Diversity", Two Volumes, Gordon & Breach Sci. Pub., New York, 1973.

48. Gremion, C., "Toward a New Theory of Decision Making", Internal Studies Mgmt. Org., 1972, 2, 2.

49. Hoare, C.A.R., "Proof of a Structured Program: The Sieve of Eratosthenes", Computer Journal, 1972, Vol.15, No.4, pp.321-325.

50. Holden, G.K., "Factfinder 13: Production Control Packages", N.C.C. Pub., 1973 .

51. Infotech State of the Art Report Number:

52. 6, Computer Networks;

53. 8, Application Techniques;

54. 11, Software Engineering;

55. 14, Operating Systems,

Maidenhead, Berks., England.

56. Kant, I., "The Critique of Pure Reason", Great Books of the Western World, Vol.42, 1952 .

57. Katz, D.; Kahn, R.L., "Common Characteristics of Open Systems", In F.E. Emery, (Ed.), op.cit., ref. (41), pp.86-104.

58. Klir, G.J., (Ed.), "Trends in General Systems Theory", J. Wiley & Sons, New York, 1972 .

59. , "An Approach to General Systems Theory", Van Nostrand Reinhold, New York, 1969 .

60. Koestler, A., "The Tree and the Candle", In Gray, et al, (Eds.), loc.cit., ref (47), pp.287-314.

61. Krieble, C.H., "Design of Management Information Systems", In J.F. Pierce, (Ed.), loc.cit., ref. (86), pp.375-390.

62. Lange, O., "Wholes and Parts: A General Theory of System Behaviour", trans. by E. Lepa, Pergamon Press, Oxford, 1965 .

63. Langefors, B., "Activity Network for Planning and Scheduling", BIT, 1962, 2, No.1, pp.21-34.

64. Laszlo, E., (Ed.), "Relevance of General Systems Theory", George Braziller, New York, 1972 .

65. Leavenworth, K., "Modular Design of Computer Programs", Data Management, 1974, July, pp.14-19.

66. Likert, R., "New Patterns of Management", McGraw-Hill, New York, 1961 .

67. Lipschutz, S., "Set Theory and Related Topics", Shaum's Series, McGraw-Hill, New York, 1964 .

68. Litterer, J.A., "Organizations: Systems, Control and Adaptation", J. Wiley & Sons, New York, 1969 .

69. Locke, J., "An Essay Concerning Human Understanding", Everyman's Library, 1972 .

70. McIlroy, M.D., "The Outlook for Software Components", In Infotech Report No.11, loc.cit. ref. (54), pp.243-252.

71. March, J.G.; Simon, H.A., "Organizations", J. Wiley & Sons, New York, 1958 .

72. Martin, J.; Norman, A.R.D., "The Computerized Society", Prentice Hall, New York, 1970 .

73. Mason, R.O.; Mitroff, I.I., "A Program for Research on Management Information Systems", Management Science, 1973, Vol.19, No.5, pp.475-487.

74. Maslow, A.H., "Motivation and Personality", Harper & Row, New York, 1970 .

75. Miller, G.A., "What Is Information Measurement?", In Buckley, Ed., loc.cit., ref. (24), pp.123-128.

76. Moore, B.J., "Operating Systems for a Range of Computers", In C. Boon, (Ed.), loc.cit., ref. (21), pp.225-239.

77. Morgenthaler, G.W., "The Theory and Application of Simulation in Operations Research", In Ackoff, (Ed.), loc.cit., ref. (1), pp.363-419.

78. Mure, G.R.G., "The Philosophy of Hezel", Oxford Univ. Press, London, 1965 .

79. Myers, G.L., "Characteristics of Composite Design", Datamation, 1973, Sept., pp.100-102.

80. Naur, P.; Randell, B., (Eds.), Software Engineering", Garmisch Report, NATO Scientific Division, Brussels, 1969 .

81. Newell, A.; Shaw, J.C.; Simon, H.A., "Chess-Playing Programs and the Problem of Complexity", In Feizenbaum et al, (Eds.), loc.cit., ref. (42), pp.39-70.

82. Niland, P., "Production Planning, Scheduling, and Inventory Control: A Text and Cases", Macmillan Co., London, 1970 .

83. Orchard, R.A., "On an Approach to General Systems Theory", In Klir, Ed., loc.cit., ref. (58), pp.205-250.

84. Parsons, T., "Structure and Process in Modern Society", Glencoe: Free Press, 1960 .

85. Pattee, H.H., "The Evolution of Self-Simplifying Systems", In Laszlo, (Ed.), loc.cit., ref. (64).

86. Pierce, J.F., (Ed.), "Operations Research and the Design of Management Information Systems", Special Ass. Pub., 1967 .

87. Poirier, C.C., "Automated Data Processing for the Corrugated Box Plant", In Pierce, (Ed.), loc.cit., ref. (86), pp.416-432.

88. Popper, K.R., "The Logic of Scientific Discovery", Hutchinson & Ltd., London, 1972 .

89. Pyle, I.C., "Hierarchies: An Ordered Approach to Software Design", In Infotech Report No.11, loc.cit., ref. (54), pp.253-272.

90. Rapoport, A., "The Uses of Mathematical Isomorphism In General Systems Theory", In Klir, (Ed.), loc.cit., ref. (58), pp.42-77.

91. Rapoport, A., "The Search for Simplicity", In Laszlo, (Ed.), loc.cit., ref. (64), pp.15-41.

92. Russell, B., "The Problems of Philosophy", Oxford Univ. Press, London, 1973 .

93. Schoderbek, P.P., (Ed.), "Management Systems", Wiley Series in Management & Admin., 1967 .

94. Schrödinger, E., "What is Life?", Cambridge Univ. Press, Cambridge, 1945 .

95. Shannon, C.E.; Weaver, W., "The Mathematical Theory of Communication", Illini Books, Illinois, 1972 .

96. Simon, H.A., "Administrative Behaviour", Macmillan Co., New York, 1947 .

97. _____ , "A Behavioural Model of Rational Choice", Qrtly. J. of Econ., 1955, Vol.69, pp.99-118.

98. _____ , "The New Science of Management Decision", Harper & Row, New York, 1960 .

99. _____ , "The Architecture of Complexity", In Litterer, (Ed.), loc.cit., ref. (68), pp.98-114, also in General Systems, 1965, Vol.10, pp.63-76.

100. Singleton, W.T.; Easterby, R.S.; Whitfield, D.C., (Eds.), "The Human Operator in Complex Systems", Taylor & Francis Ltd., London, 1967 .

101. Sisson, R.L., "Sequencing Theory", In Ackoff, (Ed.), loc.cit., ref. (1), pp.293-326.

102. Spur, W.A.; Bonini, C.P., "Statistical Analysis for Business Decisions", Richard D. Irwine, Inc., Homewood, Illinois, 1967 .

103. Stern, H., "Information Systems in Management Science", Management Science, 1970, Vol.17, No.2, pp. B-119/123.

104. Stratton, A., "Total Systems Analysis", First Internal Res. Conf. On O.R., 1973, Univ. of Sussex.

105. Valkenburg, M.E. von, "Introduction to Modern Network Synthesis", J. Wiley & Sons, New York, 1965 .

106. Vickers, G., "A Classification of Systems", General Systems, 1970, Vol.15, pp. 3-6.

107. Weaver, W., "Science and Complexity", American Scientist, 1948, 36, pp. 536-544.

108. Weinberg, G., "Natural Selection as Applied to Computers and Programs, General Systems, 1970, Vol.15, pp. 145-150.

109. _____ , "A Computer Approach to General Systems Theory", In Klir, (Ed.), loc.cit., ref. (58), pp. 98-142.

110. Westaway, F.W., "Scientific Method", Hillman-Curl Pub., New York, 1973.

111. Whitehead, A.N., "Process and Reality", Indiana Univ. Press, 1971.

112. Weiner, N., "Cybernetics: or Control and Communication in the Animal and the Machine", M.I.T. Press, Cambridge, Mass., 1971.

113. Wolverton, R.W., "The Cost of Developing Large-Scale Software", I.E.E.E. Trans. On Computers, 1974, Vol. C-23, No.6, pp. 615-636.

114. Wymore, W., "A Wattled Theory of Systems", In Klir, (Ed.), loc.cit., ref. (58), pp. 270-300.

115. Young, O.R., "A Survey of General Systems Theory", General Systems, Vol.IX, 1964, p.61.

116. Yourdon, E., "A Brief Look at Structured Programming and Top-Down Program Design", Modern Data, 1974, June, pp. 30-35.

117. Zander, A., "Resistance to Change - Its Analysis and Prevention", In Schoderbek, (Ed.), loc.cit., ref. (93), pp. 200-203.

APPENDIX 1

SOME CONFIGURATIVE DATA OF THE QUESTIONNAIRE

The questionnaire was limited in its scope to packages that cover a part of an operational activity such as: sales accounting; costing; inventory control; production control; etc. within an organization and perform day-to-day operations. However, all scientific, mathematical and statistical packages are excluded.

The sample of the mail-questionnaire comprised 262 computer users mainly of the Midland area of whom 104 had completed the forms * thus yielding a response of about 40%.

The returned forms included the following:

1. Number of companies who make or had made use of packages

Table 1

No	Yes	Total
39	65	104
38%	62%	100%

2. Number of companies who are making use of packages

Table 2

No	Yes	Total
47	57	104
45%	55%	100%

The difference between the entries of the two tables, i.e. 7%, is accounted for those companies who abandoned their packages. Table (2) is further broken down into the following

a. Number of packages per user

Table 3

1	2	Total
17	40	57
29%	71%	100%

This shows that the majority of package users depend to a large extent upon packages.

b. Future considerations of using packages by those who are presently using them

Table 4

No	Yes	Total
32	25	57
56%	44%	100%

c. Future considerations of using packages by those who are not presently using them

Table 5

No	Yes	Total
39	8	47
83%	17%	100%

3. Number and function of financial packages

Table 6

	Number	%
Payroll	36	25%
Ledger	15	11%
Costing	8	6%
Others	11	8%
Total	70	50%

4. Manufacturers supplied financial packages

Table 7

Payroll	Ledger	Costing	Others
64%	38%	85%	4%

5. Manufacturers share of production packages

Table 8

Production Control	Stock Control	Scheduling	Parts B/D
94%	93%	79%	76%

6. Overall manufacturers share of packages

Table 9

	Manuf.	Others	Total
Number	97	43	140
Per cent	69%	31%	100%

Table (9) is broken down by the major manufacturers as follows

Table 10

I.C.L.	I.B.M.	Other Manuf.	Total
48	28	21	97
34%	20%	15%	69%

APPENDIX 2

AN EXTRACT OF INTERVIEWS QUESTIONS TO ILLUSTRATE CURRENT USERS THINKING

Introduction

Fourteen manufacturing and business companies who use packages were selected from those who had completed and returned questionnaire forms.* This was for the purpose of developing a realistic image of the practical state of applications packages. In almost all cases the attitude of the interviewees was that of co-operation and encouragement.≠ This has been substantiated by their expressed willingness to always provide more time as needed to discuss the various aspects of the problem. Such discussions have helped not only in developing that image, but more important is their constructive influence upon the orientation of the research throughout its various phases.

In the following some key questions and the common attitude towards them are described.

---

* Cf. attached list of visited companies.

≠ The interviewees were in the main either data processing/management services managers, project leaders, package specialists, or managers who are supplied with the output of packages.

Trends and Attitudes

Interviewees were asked:

1) What is the degree of package compatability with your definition of the problem concerned?

The answer to this question varied between fair and good. Interestingly the latter was maintained by those who use more than one package and the former by those who use only one package. However it should be noted that the d.p. practitioners definition of the problem was very much dissimilar to that of the real package users, i.e. the production managers. The latter were more critical and less satisfied. This must be qualified by recording that it was not possible to conduct as many visits to production departments as was desired.

2) What did make you decide upon the package and what are your criteria of selection?

The trend was that they did not have much choice since they are short of required skills and expertise to undertake project design. Also as far as selection was concerned packages were chosen in view of two considerations: 1) compatability with existing hardware and, 2) costs. In other words, and as expected there does not exist a set of criteria according to which a user can either decide upon whether to go for a package or not, or what aspects of the considered package should be investigated.

3) What is your conception of the package-user relationship?

The general attitude of d.p. practitioners did not regard such a relationship as a strong one. Accordingly, the user's role starts and ends with the print-out. This is an obvious serious misunderstanding of the user-package interaction which is further aggrevated by the popular view d.p. practitioners regard themselves as being more knowledgeable about user's requirements and needs than the user himself.

4) Concerning the specific problem the package addresses, e.g. scheduling, how do you rate your familiarity with the methods and techniques of this area?

Almost invariably practitioners displayed inadequate understanding of the characteristics and relative effectiveness of the various existing techniques. Their individual knowledge is limited to their experience, and void of basic theoretical background.

5) Who is responsible for software and package maintenance?

A package is considered to be the responsibility of its supplier in the same way software is seen. The actual design and structure of either software or a package induce little interest.

6) If the package is modular, is each unit of it identified and classified for direct correspondence to:

i) its actual source listings?

ii) its actual object form?

iii) its corresponding elements of user documentation?

iv) its corresponding elements of technical documentation?

In all cases only actual object form is available for a package as a whole. This has invoked a host of questions involving:

a) How are modules define? In terms of size, relationships, etc.

This has proved to be an area of confusion.

b) How modules are used and in what frequencies?

In most cases a package is run as a whole. Also because of strong connection between its various parts, such parts cannot be separated/isolated from the package and used eith on their own or in connection with other modules to perform another application.

c) How do the practitioners cope with the variability/ changeability of requirements?

He relies on the supplier either to adapt the package to suit the new requirements or to replace it with a new varsion or a larger package.

d) Is it possible to add and/or delete modules?

In some cases the user can delete modules but be cannot add modules. Otherwise, the supplier will withdraw his support which is a risk no one has been ready to take.

e) How are software errors made?

i)	Unexpected side effects to changes	30%
ii)	Logical flows in the design:	
	- original design	-
	- changes	35%
iii)	Inconsistencies between design and implementation	20
iv)	Clerical errors	15%
v)	Inconsistencies in hardware	-

Some other questions are quoted as appropriate in the footnotes throughout the thesis.

COMPANIES VISITED

The following are the names of those companies who have been visited more than two times.

1. W. & T. Avery Ltd., Data Processing Dept., Systems Designer.
2. Cadbury Schweppes, Computer Project Development Manager.
3. Chrysler U.K. Ltd., Pre-Production and Material Systems, Manager.
4. Duport Computer Services Ltd., Technical Manager.
5. Forward Trust Ltd., Development Manager.
6. Foster Menswear Ltd., Data Processing Manager.
7. G.K.N. Transmissions Ltd., Computer Manager.
8. G.K.N. Group Computer Centre, Management Services Manager.
9. Lucas Aerospace Ltd., Engine Management Group, Data Processing Dept., Systems Manager.
10. Newey Goodman Ltd., Data Processing Manager.
11. Richard & Wallington Industries Ltd., Data Processing Manager.
12. Rover-British Leyland U.K. Ltd., Data Processing Manager.
13. Triumph-British Leyland U.K. Ltd., Systems Manager.
14. Wilmot Breeden Ltd., Data Processing Manager.

THE UNIVERSITY
OF ASTON
MANAGEMENT CENTRE

APPLICATION PACKAGES QUESTIONNAIRE

Q1/373

SECTION I.

HAVE YOU EVER USED APPLICATION PACKAGES IN THE PAST AND ABANDONED THEM? YES NO IF NOT PLEASE GO TO SECTION II

IF YES? PLEASE GIVE THE FOLLOWING INFORMATION:

	PACKAGE NAME	SOURCE/ORIGIN	DOCUMENTATION:			AMOUNT OF MODIFICATION NEEDED			AREA OF APPLICATION	HOW LONG WAS IT USED (MONTHS)?	ASSESSMENT		
			POOR	ADEQUATE	GOOD	LITTLE	MODERATE	MUCH			NOT USEFUL	USEFUL	VERY USEFU
1.													
2.													
3.													

SECTION II.

ARE YOU PRESENTLY USING APPLICATION PACKAGES? YES NO IF NO, PLEASE GO TO SECTION III.

IF YES, PLEASE GIVE THE FOLLOWING INFORMATION:

	PACKAGE NAME	SOURCE/ORIGIN	DOCUMENTATION			AMOUNT OF MODIFICATION NEEDED			AREA OF APPLICATION	HOW LONG HAS IT BEEN IN USE (MONTHS)?	ASSESSMENT		
			POOR	ADEQUATE	GOOD	LITTLE	MODERATE	MUCH			NOT USEFUL	USEFUL	VERY USEFU
1.													
2.													
3.													

SECTION III.

ARE YOU IN THE PROCESS OF CONSIDERING AND/OR IMPLEMENTING NEW PACKAGE(S) YES NO IF NO, PLEASE GO TO SECTION IV.

IF YES, PLEASE GIVE THE FOLLOWING INFORMATION:

PACKAGE NAME	SOURCE/ORIGIN	AREA OF APPLICATION	FEASIBILITY	STAGE REACHED: PLANNING FOR IMP.	IMPLEMENTING

SECTION IV.

COMMENTS (ON PACKAGES, OR QUESTIONNAIRE, ETC.)

SECTION V.

WOULD YOU SUPPORT ANY FURTHER ENQUIRY?

IF YES, PLEASE GIVE THE FOLLOWING ENFORMATION:

NAME OF THE COMPANY: ADDRESS:

NAME OF PERSON WE SHOULD CONTACT IN FUTURE:

PLEASE FOLD SO THAT RETURN ADDRESS SHOWS IN WINDOW ENVELOPE SUPPLIED