DATA BASES, MICROCOMPUTERS AND OFFICE AUTOMATION

by

Amir Bukhari

University of Manitoba

Introduction

The microcomputer technology has now progressed to such a state of development and cost effectiveness that it is inevitable that microcomputers will become as common a fixture in the office as the telephone. This new technology has its own peculiar set of problems which must be carefully resolved before it can be effectively integrated in the office environment. These problems, in large measure, are related to the trend which appears to be developing in this field. The office automation appears to be launched on a trajectory which may be described as an integration of local network, data base and information management technologies. Office automation systems are being designed to locally inter-connect electronic machines by cables using, for example, either net technology. Included among these inter-connected machines are microcomputers which provide shared access to commonly used data organized as a data base. The data from these data bases is extracted and converted into information for use for operational or decision making purposes. This paper focuses on the role and design of data bases in the environment described above.

Data Base Management System

A management information system may be modelled as a system packaged together from various compatible hardware and software components. A simple system framework for an information system is to view it as a set of functional levels which interact with each other by means of suitable interfaces.

. MIS

level 0	. Hardware
interface	
level 1	. Operating System
interface	
level 2	. Data Base Managment System
interface	
level 3	. Model Bank
interface	
level 4	. Application Programs
interface	
level 5	. Users

An operating system is a software component which manages the hardware devices and other resources which are part of the computerized system. These include functions such as memory and device management and providing an environment in which a user can communicate with the system. The essential role of the operating system is to enable the user to operate the system. Operating systems do not in any real sense provide the management of data that reside on the devices. The management of data stored in the system is delegated to another software component, commonly known as the data base management systems (DBMS). The primary role of a DBMS is to manage data that resides in the system. Model banks have resources which convert raw data into input appropriate for application programs which produce information for operational and decision making purposes.

Data Base Design Problems

The development of a data base design must resolve three major problems in any environment.

Problem of Time:	Data base design is a time consuming process even for small applications.
Problem of Detail:	Quantum of detailed work necessary to achieve a workable design is overwhelming.
Problem of Quality:	Quality in data base design is difficult to achieve.

These problems are central to any data base environment, including those which support automated offices using micro-computers. The resolution of these problems is difficult but can be greatly facilitated by adopting a rigorous design approach. The term rigorous is used here in the sense as "scrupulously accurate, precise" rather than in the sense of a mathematical proof. An approach to data base design is illustrated here based on a modified example from Raver and Hubbard [1].

Data Base Design Process

The data base designer rigorously establishes the following:

(a) user specifications
(b) an environment
(c) an architecture for the data base
(d) a strategy for verifying that the architecture will in fact satisfy the user requirements
(e) an implementation plan
(f) a realization plan.

A rigorous approach to these steps uses a structured methodology. All of the so-called structured methodologies establish a formal discipline for the solution of the problem. A form suitable for data base development consists of repeated application of the principle:

"DECOMPOSE AND PROVE CORRECTNESS OF DECOMPOSITION".

This principle requires that the designer decompose a given data base application into precisely specified sub-applications. Solve and prove each sub-application. Finally, fit together the solved sub-applications in a specified way to obtain an optimal solution to the original application. This principle is most effectively elaborated by Hoffmann [2]. Specific examples of this type of methodology are Hoare's pre and post condition analysis [3]; Dijkstra's stepwise refinement method [4] or Blaauw's Architecture - Implemention - Realization method [5].

Illustrative Example

Modifying the example given in [1], assume that a feasibility analysis has identified A, B, C, D, E, F, G, H, J, K, L, and M as distinct pieces of information which are of interest to a user. The user would like to see these items organized in five reports.

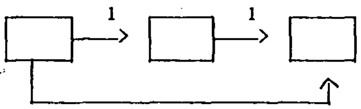
1. A report which uses items B, H, J, K
2. A report which uses items A, B, M
3. A report which uses items A, B, D, G, L
4. A report which uses items A, D, G, M
5. A report which uses items A, B, C, D, E, F, J, K, L, M

In effect, a set of sub-applications of the original applications (based on all the pieces of information) is already available. This, however, may not be always the case. Reports are normally structured entities where the data bears identifiable relationship to one another. These relationships could be functional (1-to-many) or many-many in nature. Suppose further that these relationships have been identified as represented in Figure 1, where "1" denotes a functional relationship and "M" denotes a many-many relationship. The design process would then proceed to integrate all these sub-applications into a single large application and then simplify again, possibly by iterating via a different decomposition, which of course must support the originally desired applications. The rules used to integrate given sub-applications into a single version and the rules used to simplify provide the basis for formal approach. These rules, when stated explicitly, provide a formal discipline -- structured methodology -- which when applied rigorously will yield a number of choices. The designer's job is then to choose the most optimal one of these which supports the intended user requirements in some "best" but verifiable way. Thus, Figure 2 is the integrated version of the five applications. Simplified version is shown in Figure 3. The rules used for integration and simplification are:

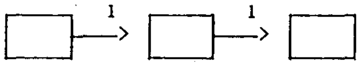
(a) Integrate by using each information item once and showing relationships by labelled arrows.

(b) Simplify by removing all those arrows which are labelled M, but such removal of arrows must not leave any information item totally disconnected.

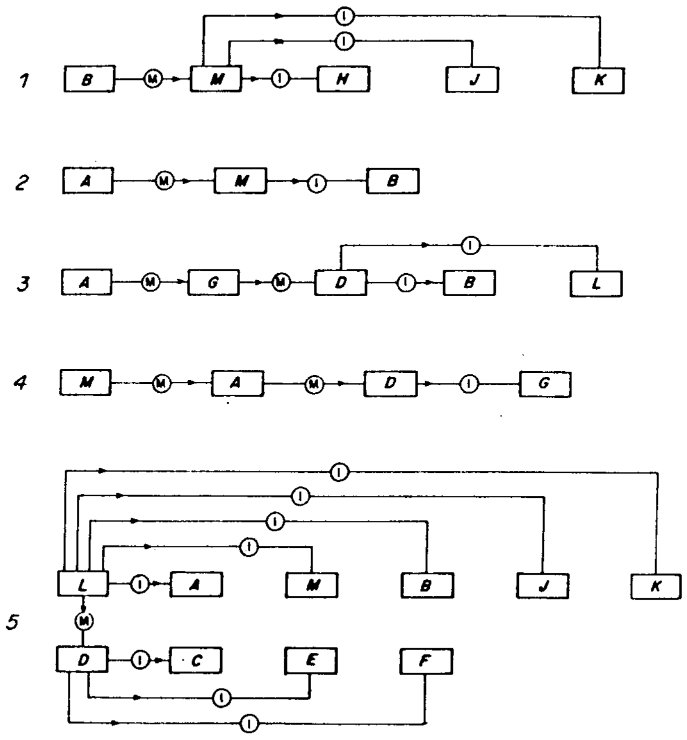
(c) Simplify by removing all transitive dependencies, i.e. if



then simplify to



This is a sample list. There can be different rules and the list can be expanded. For example:


1
B
M
M
I
H
I
J
I
K
2
A
M
M
I
B
3
A
M
G
M
D
I
B
I
L
4
M
M
A
M
D
I
G
5
L
I
A
I
M
I
B
I
J
I
K
M
D
I
C
I
E
I
F
FIGURE 1

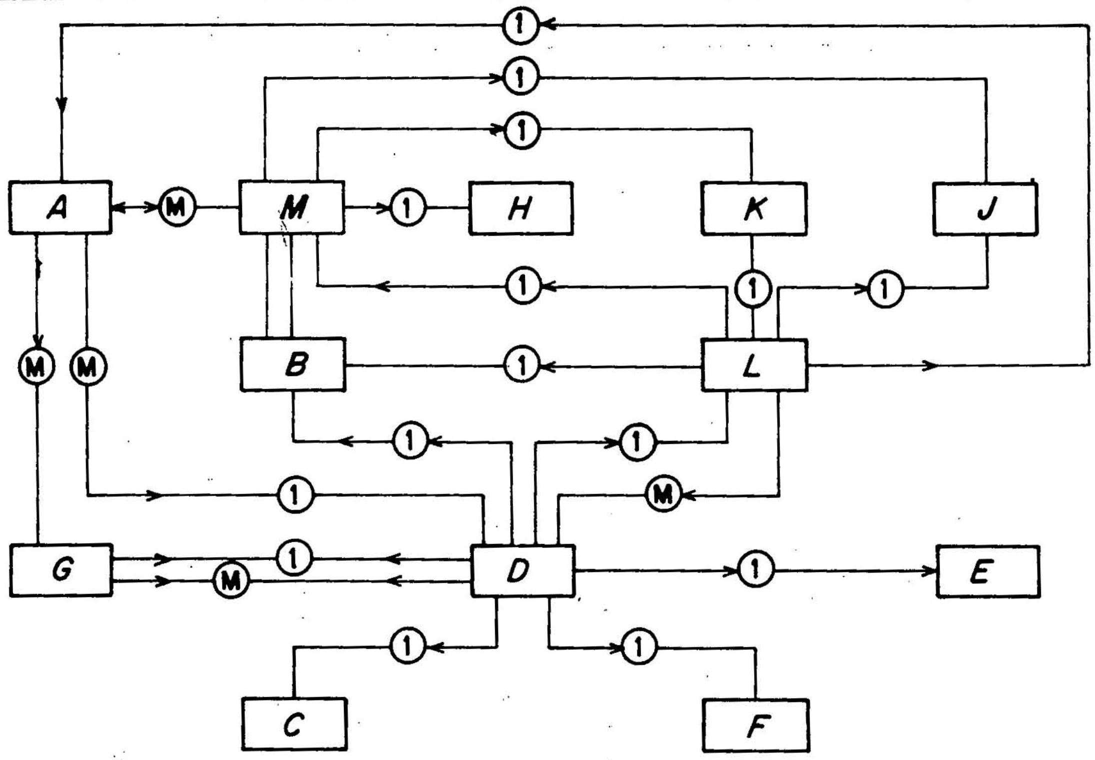


FIGURE 2   INTEGRATION

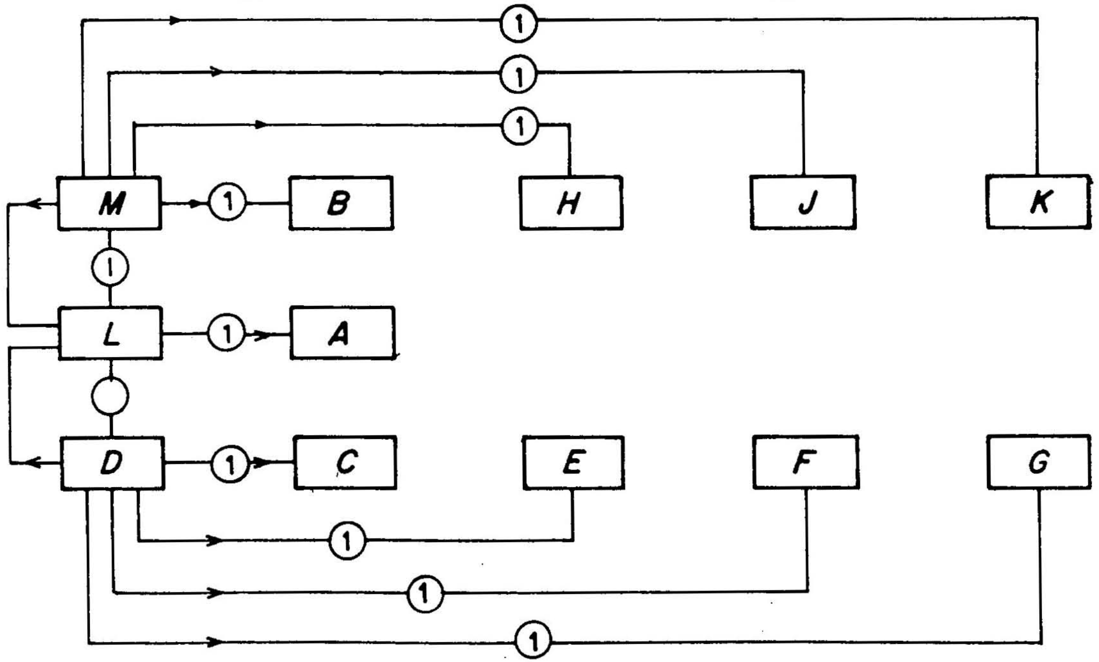


FIGURE 3 DATA BASE ARCHITECTURE

(d) Naming rule for organizing the environment: data elements from which a functional or type 1 relationship emanates is a candidate for a key.

As a result of application of such rules, one is forced into obtaining a hierarchical organization of the data as shown in Figure 4.

Graphical representation followed until now can be transformed into matrix representation which is amenable to formal algebraic analysis. The matrices corresponding to Figures 1 - 4 are shown below, where 1 signifies a functional relationship of a many to many relationship and an empty cell indicates a don't care condition.

The main point of all this discussion is that it is possible to formalize the data base design process and furthermore, it is possible to use a representation of data base problems suitable for algebraic or other formal analysis. Finally, the approach is amenable to verification and validation analysis. Figure 5 illustrates this point. Raver-Hubbard strategy of integration, simplification, and decomposition has lead to a data organization which does not directly support the user application [1]. Additional data indexes must be provided before the application can be efficiently supported. This does not imply that the method is without value, rather it provides a designer a tool for analysis and re-examination. In a manner of speaking, it forces the designer to verify, validate and record reasons for design choices.

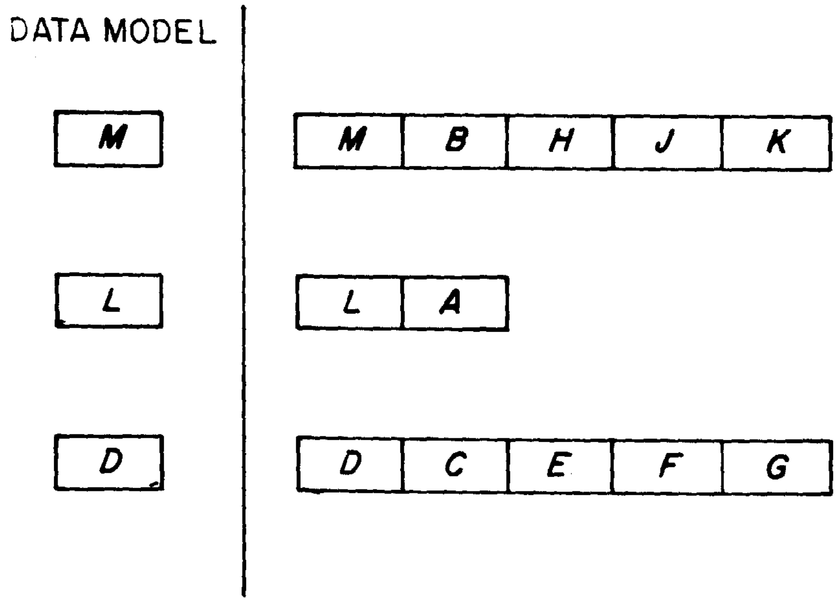


FIGURE 4

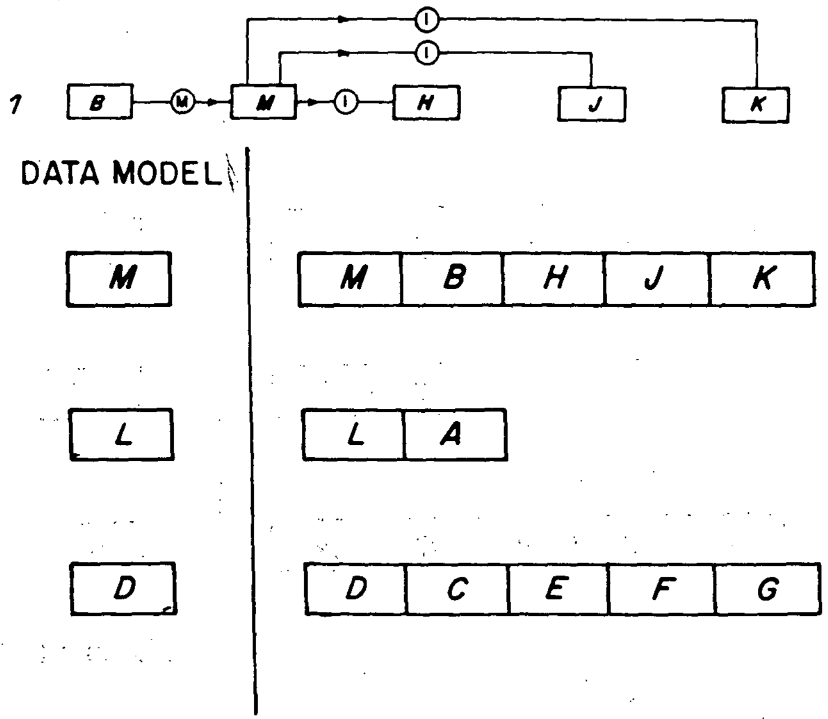


FIGURE 5

GETTING READY FOR ANALYSIS

(1)	M	H	J	K
B	m			
M	m	1	1	1

(2)	B	M
A		m
M	1	

(3)	B	D	G	L
A			m	
M	1			1
G		m		

(4)	A	D	G
A		m	
D			1
M	m		

(5)	A	B	C	D	E	F	M	J	K
D			1		1	1			
L	1	1		1			1	1	1

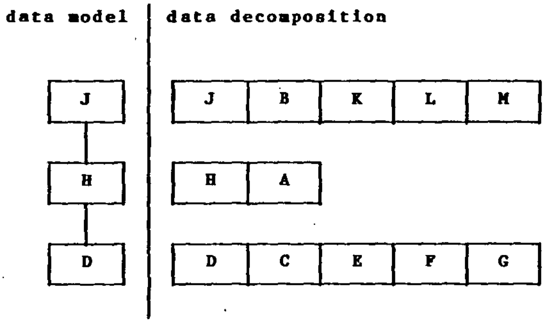
INTEGRATION

	A	B	C	D	E	F	G	H	J	K	L	M
A				m			m		m			
B									m			
D												
G				m								
L	1	1		1					1		1	1
M	1	1								1	1	1

algebraic representation:

	A	B	C	D	E	F	G	H	J	K	L	M
D			1		1	1	1				1	
L	1								1			
M		1								1	1	1

data model/decomposition schema:

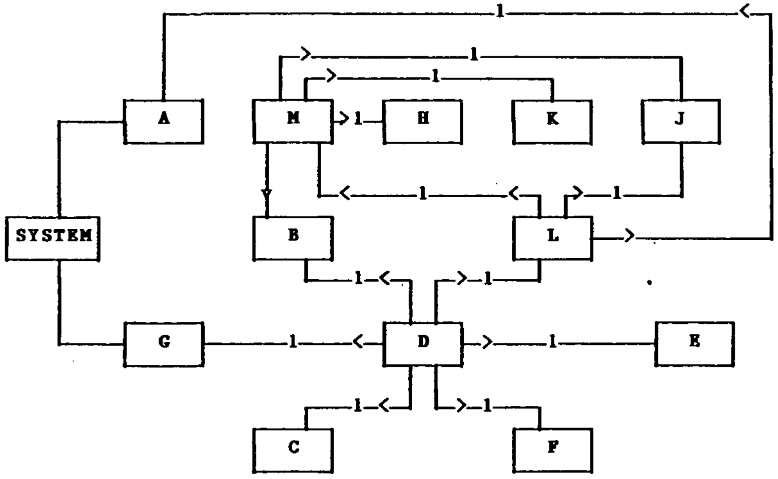


An examination of the data model/data decomposition schema above illustrates an extremely important data base concept. Data model provides the framework within which a given data decomposition is to be synthesized into an integrated whole again. In the above case, the new decomposition consists of J, B, K; K, L, M; H, A; and D, C, E, F, G. These new groupings of information or data items are to hang together as a hierarchical structure organized by using J, H and D. Data models provide the structure or the mosaic into which individual data items are to be organized. The major concern in data base theory is to find data models which are linear in algebraic sense. This is important because the essence of linearity in algebra is the closure property. The closure property interpreted in data base context tells the system whether a data item belongs or does not belong -- in effect, endows the data base structure with the essential property of consistency. Consistency is a major design issue and it can be best handled by adopting a formal approach to it.

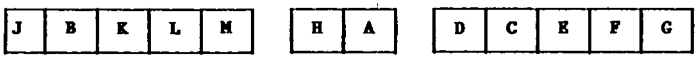
Data Models

Why did one obtain a hierarchical data model in the previous example? Are there other data models which are useful? The answer to the first question is that the simplification rules were designed to yield a hierarchy, if one existed. If these rules were changed, then a network model of data would result. A network model is a richer structure. It maintains more connection among data than those maintained by the hierarchical model. However, both hierachical and network models are based on explicit representation of relationships among data. The relationships among data need not be explicitly recorded. They can be implicit. This is the case with the relational data model which organizes data tables. The implicit relationships encoded in the tables serve to structure the data.

network data model:



relational data model:



In a network data model, a new data entity SYSTEM is introduced which provides an entry point to the system (in the hierarchical model the root serves this purpose) and furthermore the requirement of maintaining a minimal number of relationships is relaxed. Thus, the network model provides a richer and more efficient mechanism for accessing data. However, the trade off is that the user must now specify how to navigate in the system, i.e. specify how to get to one data item from another. This introduces a degree of complexity not present in the hierarchical model.

The relational model presents all data in tabular form. The names of the data items serve as the headings of the columns and entries in the columns are the values of the data items. The early workers in the relational data model field were surprised to find that it is not an easy matter to organize data into tables and yet be able to easily access them and maintain them consistently. This realization lead to the theory of normalization which is an attempt to design tables in such a way that consistency is always guaranteed and yet data is easily accessible without having to specify a lot of navigational information as required in the network model. Normalization principles are in fact applicable to all data models. For example, the rule to remove transitivities (used in obtaining the hierarchical data model) is actually a principle supported by normalization theory of relations.

Comparison of Data Models

There are several ways to compare the data model. For example, hierarchical models are costly to maintain, relational models are costly to operate and network models are costly to bring up. In the office environment, especially on the microcomputers where the many users will be competent but technically naive, the most important consideration is that of user friendliness. On this count the relational model is the winner. Tables are easily understood and do not require elaborate mechanism before they can be used. In fact, this feature of relational model is touted to be one of its most significant recommending features.

Alternate Approaches to Design

The approach recommended by Raver and Hubbard [1] is to:

- specify
- integrate
- simplify
- iterate to optimize

There are other approaches to data base design which are significant in context of implementation on the microcomputer.

The approach based on Hoare's pre- and post-conditions is based on the following procedure:

- establish a set of relationships on or among the input variables and call these the pre-conditions

- establish a set of relationships on or among input and output variables and call them post-conditions

- design algorithmically a procedure to start with the input values subject to the pre-conditions and obtain the values of output variables subject to the post-conditions

- verify that the proposed solution does indeed satisfy the input or the pre-conditions.

An important feature of pre- and post-condition analysis is that it forces the designer to cast the design problem into a mathematical framework of equations and inequalities. As usual the art of such a design procedure is to be able to describe a design problem as a system of linear equations or inequalities, if such a description exists. A solution of such a linear system then produces a design choice. Once a set of design choices has been generated, then a search for an optimal design can be made.

A natural consequence of this approach is that any programs or systems developed using this approach are verifiably correct.

Blaauw's technique [5] is again novel and very powerful. It requires that any system be realized by means of the following steps.

1. Establish the architecture or the functional specifications of WHAT system is required to do.

   Write these specifications in a powerful and expressive language such as APL.

   Since APL is readily available, programs written in it can be readily executed.

2. Establish the algorithms or HOW the architectured system will be implemented. That is describe, again using APL, exactly how the architecture can be made to work. This results in another APL program. Call this the implementation.

3. Test the architecture (step 1) and implementation (step 2) by running the two programs simultaneously with the same data. If the two programs yield the same results then the architecture and its implementation are equivalent in the sense that recipes provided by the algorithms encoded in the implementation do indeed satisfy the architectural requirements.

4. Use the implementation as a model to physically realize the system. The consideration here is what actual components to use and where to locate these components in relation to one another in a physical environment.

The term realization is intended to focus on the actual physical, concrete choices of components, their location and inter-relationships within the system. Thus, steps 1 and 2 may be written in APL but the actual realization of the system may call for using COBOL or PL/I and the language.

The novelty of Blaauw's approach lies in the fact that it can be used to design complex systems and test them before actually realizing them.

The merit of the above approaches is that each outlines in a formal manner what must be done in order to realize a verifiably correct design. Blaauw [5] in particular has applied his technique to the design of a very large electronic system but is equally applicable to other disciplines, especially to data base design in microcomputer environment.

User Friendliness

The importance of user friendliness cannot be over emphasized. Any design which does not take this important system factor into account is not a good design. There is much empirical evidence to support the thesis that a suitable way to achieve user friendliness is to adopt a formal approach, and design it right into the system.

Environment

It is rightly said that for a well designed system, a "good environment is not a luxury but is a necessity". An environment is created in a system when names are given to objects in a system. Names and name spaces play an important role in data base work. The choice of names determines whether a system is user friendly or not. Again, it is careful design of names and naming procedures which contribute to developing a reliable system. Even the single letter names used in the illustrative example can be useful. A name of length greater than one is an error. On the other hand, use of single letters is rather devoid of semantic content and may be unfriendly for use by occasional naive users; but again the technical users of a system may be most comfortable using single letter names as opposed to having to use longer names. In fact, the perilious beginnings of a new system always start with the creation of an environment.

The Future

It is a matter of time before automated offices, electronic fund transfer and other electronic marvels will become as common place and as easy to use as the telephone. The office of the future will be a concrete management information system where a large number of cooperating users will share large amounts of common data. This is going to impose very heavy design demands on organizing common data into data bases, which can be shared by a wide variety of users. The typical problems raised are those of integrity, security and consistency of data. These problems are of sufficient magnitude which warrant formal technique for their solution.

REFERENCES

1. Raver, N. and Hubbard, G.U., Automated Logical File Design, IBM systems Journal 16, 3, pp. 287-312 (1977).

2. Hoffmann, B., On Gabriel Kron's Methods and Achievements, in Gabriel Kron and Systems Theory, edited by H.H. Happ, Union College Press (1973).

3. Hoare, C.A.R., The Axiomatic Basis of Computer Programming, CACM, 12, 10, pp. 576-583 (1969).

4. Dijkstra, E.W., A Discipline of Programming, Prentice Hall (1976).

5. Blaauw, G.A., Digital System Implementation, Prentice Hall (1976).