DATABASE SYSTEMS ANALYSIS AND DESIGN

E. E. Tozer, Software Sciences Limited

April 1976

ABSTRACT

Developments in the database field have tended to emphasise programming technology, with a dearth of accompanying progress in systems analysis and design methods. This paper puts forward an overall view of system design which is intended to act as a constraining framework. It is based upon a pragmatic approach and is presented in a form which could be (and is being) used on large scale implementation projects.

Orderly analysis and design procedures are encouraged. The taking of premature design decisions is discouraged, especially through the recognition of three distinct views of data: conceptual, implementation and storage, and through recognition of distinctions between design of each of these, specification of mappings between them, and design of programs and run sequences. It is envisaged that specific procedures developed elsewhere (see references) could be incorporated into the methodology described here.

Software Sciences Limited
Abbey House
282/292 Farnborough Road
Farnborough, Hampshire
Telephone: 44321
Telex: 858228

1. INTRODUCTION

As the computer systems of an organisation develop, there is an increase in the degree of overlap between originally distinct systems. Further, increased confidence and familiarity leads to the individual systems being enriched in power and sophistication. The combined result is a vast increase in the complexity of the system design task, at that very point in time when greater reliance is being placed upon the performance, reliability and accuracy of the computer systems.

The programming aspects of this situation have been recognised for some time, and have been tackled with partial success in the form of "Database Management" systems. Traditional systems analysis and design procedures are being outstripped both by the complexity of requirements, and also by the developing programming technology. This paper proposes an overall view of the processes involved in analysis and design, with particular emphasis upon the data, as opposed to the processing aspects. Different, partially overlapping, aspects of this field have been tackled in a theoretical manner by Bramhill & Taylor (1975), Brown (1974), Robinson (1974), and many others. Whether or not a theoretical basis exists for the taking of a design decision, a system designer "in the field" has to take that decision, and he had to live with the results.

Rather than offering precise formulae for each and every stage of the progress, this paper puts forward an overall framework, which identifies important stages, and draws the necessary distinctions between them. This framework is most important, as it ensures that premature design decisions are avoided, and gives direct guidance as to the actual sequence and purpose of particular analysis and design procedures. There is no intention to over-constrain the particular techniques or their variants which individual practitioners would embed in this framework; the only criterion is that the technique should be adequate for the purpose, and consistent with the overall methdology. However, in the interest of being as specific and as helpful as possible, descriptions of, or

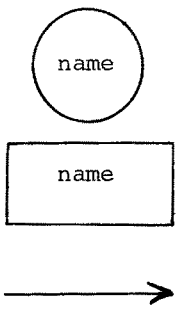
references to, suitable techniques are included where appropriate.

The analysis and design approach proposed is seen as being applicable to DP systems in general. Only in the latter stages does it become dependent upon use of a particular DBMS. Projects using conventional files or non-Codasyl DBMS would benefit considerably from adoption of the design approach put forward, with appropriate variants in the latter stages.

Section 2 explains the overall viewpoint taken, which is that of viewing systems analysis and design themselves as a system.

This system is explained in detail in section 3, and each of the main processing functions is examined in turn in more detail in sections 4 to 11.

Except where it is explicitly stated otherwise, flowcharts in sections 2 and 3 show data flow:-



Signifies a data container (file, document etc).

Signifies a process.

Signifies data flow, which may only be from container to process or process to container.

2. VIEWS OF DATA

There are several distinct views of data which are relevant to the process of analysis and design. This view is not coincident with that adopted by ANSI SPARC/DBMS Study Group (1975).

The areas of classification chosen are:-

(i) Level of abstraction

a) Conceptual

The view of data held by the organisation. This view is purely a function of the mode of operation of the organisation and the policies of its managers. It is independent of the existence of any computer systems, and should be expressed in a form most suitable to end-users in the organisation.

b) Implementation

All or some subset of the conceptual data view may be used by computer systems. The implementation view is a representation of that subset. It is designed and encoded into machine-readable form for this purpose. The implementation view remains as independent as possible from c, the storage view.

c) Storage

Actual data, described to programs by b, must be held on backing store in some form. The storage view is a complete description of such data on backing store.

(ii) Breadth of view

a) Global, or corporate.

The complete view of data, relevant to all processes carried out in the organisation.

b) Specific, or functional.

The view of data relevant to a particular function of the organisation.

(iii) Existence

Whether or not it is necessary for occurrences of data items to exist in some particular form.

VIEWS OF DATA

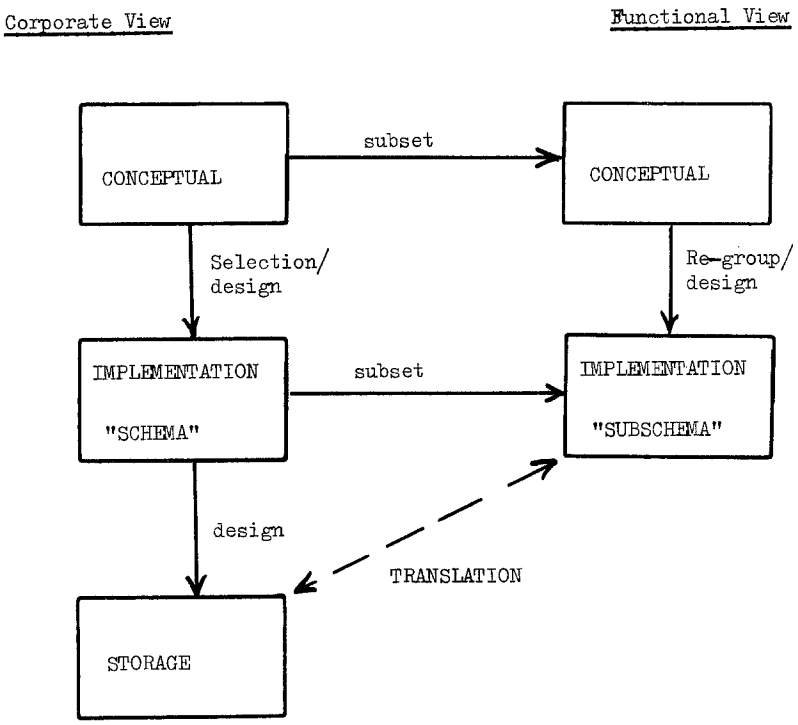


Fig.2.1

The combination of these views of data is shown in Figure 2.1.

The terms SCHEMA and SUBSCHEMA are defined by CODASYL DDLC (1973). They are used subsequently in this paper in the sense shown in 2.1.

All of the data described by the Storage view must actually exist on backing storage. The data described by the SUBSCHEMA view must exist on an as-required basis. There is no requirement that data described by the other 3 views should physically exist at all. Database access is expressed by the "translation" arrow between the subschema and storage views. The DBMS makes use of specified mappings between storage and schema and between schema and subschema, to carry out (hopefully) optimised translation.

Design should commence with the conceptual view, and progress to the storage view. For any given conceptual view it should be possible to choose many implementation views. For any given implementation view it should be possible to choose many storage representations.

A SUBSCHEMA view is unaffected by any SCHEMA change which results in a new SCHEMA of which the SUBSCHEMA remains a subset.

Current developments by the Codasyl DDLC include division of SCHEMA DDL into a number of categories. Some of these categories are relevant to the distinction between implementation and storage views of data. The measurement, tuning, resource allocation and storage categories between them relate to the storage view of data.

The CODASYL DDLC DBAWG (1975) have proposed a more complete separation of the storage view into a Storage Definition Language (DSDL).

3. THE ANALYSIS AND DESIGN PROCESS

3.1 Introduction

This section outlines the method of analysis and design which is discussed in more detail in the remainder of the paper. Figure 3.1 shows the process in flow-diagram form, and can be regarded as a guide to the paper.

Analysis and design is iterative. The need for iteration pervades almost every data path shown in 3.1. It is not shown explicitly because to do so would make the diagram excessively complicated.

3.2 Relationship between Analysis and Design

Traditionally, where system design is recognised as taking place at all, it is regarded as a process which occurs after analysis is complete, and before programming commences. It is more useful to regard both analysis and design as each taking place at a number of different levels of refinement. At each level, the order is analysis (or hypothesis or some blend of the two) leading to definition of functional requirements, leading to design of a process to meet those requirements.

During the design process, at each successive stage of refinement, additional opportunities may be realised, and the unpleasant truth may emerge concerning the severity of some constraints. Either at one stage, or through a number of stages, iteration may take place, by means of presenting the originator of the requirements with the highlights of the results of the design process, and offering the opportunity to modify the requirements in the light of these.

3.3 The Nature of Design

The design process consists of two distinct stages: generation of a range of possible solutions to a problem, and selection from the range of a best-fit, according to a pre-defined set

of criteria. Failure to achieve a good enough solution may result in iteration involving alteration of some of the initial conditions and repetition of both stages.

3.4 Development Stages

The processes described in 3.2 and 3.3 take place at each of a number of stages, representing successive levels of refinement. Some of these stages are related to the differing levels of abstraction of views of data.

THE ANALYSIS & DESIGN PROCESS

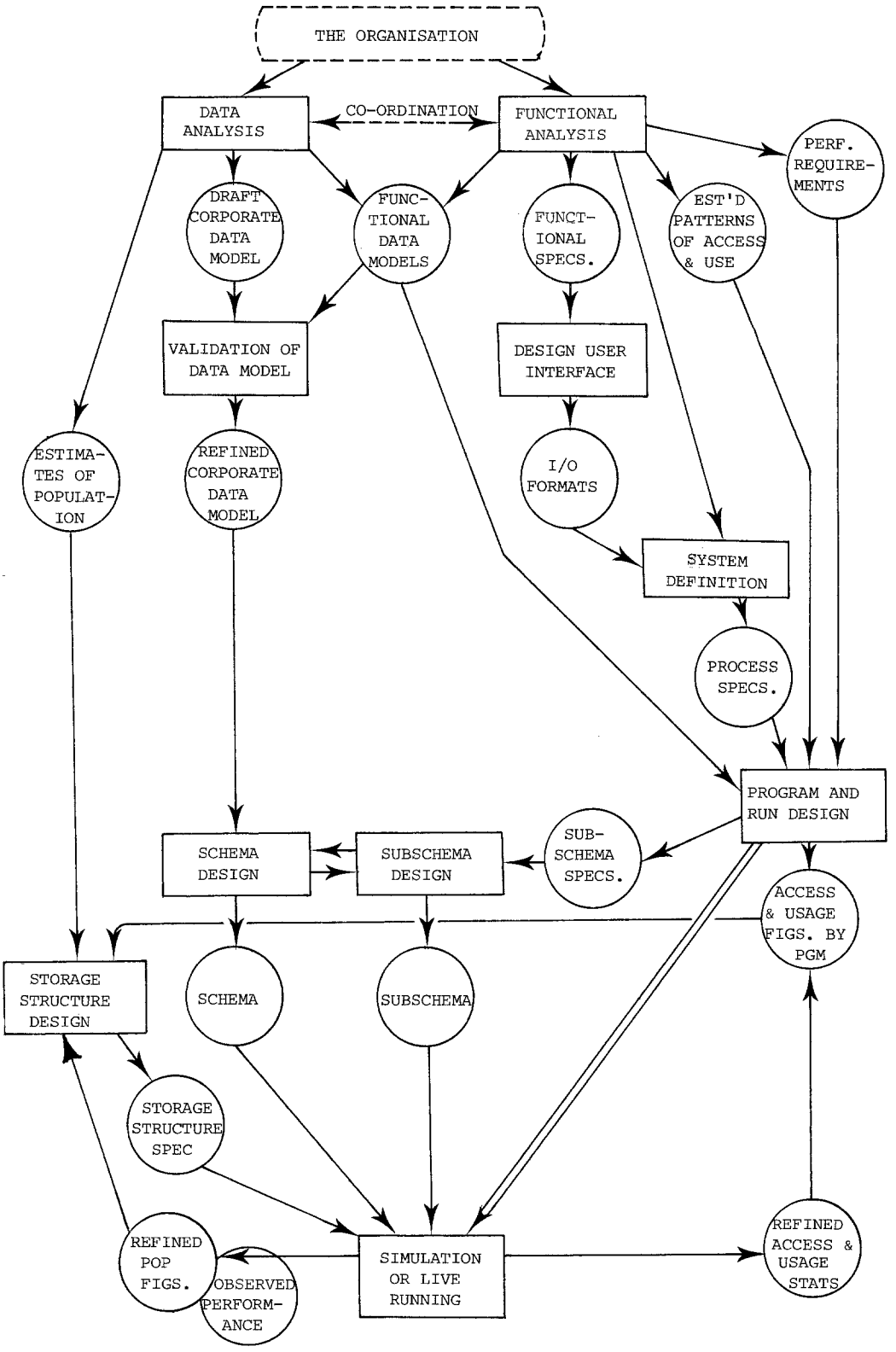


FIG. 3-1

4. DATA ANALYSIS

4.1 Objectives

- To ascertain the objects (concrete or abstract), which are meaningful to the systems of an organisation.
- To discover the nature and relevant of all relationships between such objects.
- To define the private language of the organisation. Thus to provide a commonly accepted terminology which can be used to specify systems and procedures.
- To identify each data item, and to distinguish between different manifestations of the same item (e.g. different coded forms).
- To identify and eliminate synonyms.
- To fully define, unify and rationalise coding systems.
- To build a "where used" index, so that the ramifications of an alterantion to a procedure or to the role of a piece of data can be easily explored.

4.2 Terminology and Conventions

The method of working proposed takes the viewpoint that systems exist to serve the end-user. Thus what the end-user wants, and how he sees his environment is the most important source of information. His perception of his role, and his working environment is necessarily partly subjective. It follows that the approach to data modelling needs to cope with this subjectivity.

The corporate data model is framed in terms of "things" and relationships between things. The terms entity and relationship have been adopted as being widely current for describing the conceptual view of data.

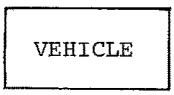
Entity is defined as a person, place, thing or event of interest to the enterprise. As such, its identification must be partly a subjective process. Selection of entities must be primarily directed by end-users of the proposed systems, to whom they must be meangful. It is important to write clear definitions of entities, and to ensure that these definitions are generally accepted.

It is necessary to distinguish between entity types and entity occurrences. An example of an entity type is "Vehicle". Occurrences of this type are:

"Blue Ford Cortina XYZ 132K"

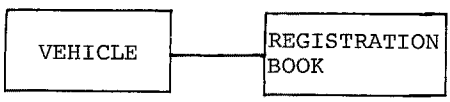
"Red Honda, Motorbike ABC 789J"

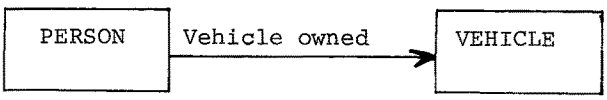
Entity types may be shown on data relationship diagrams as named rectangular boxes.

e.g. 

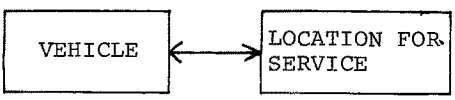
Relationships exist between entities. (Use of the term relationship does not imply any connection with the specialised terminology of the relational data model). Relationship may be 1 to 1, 1 to many, or many to many where 1 or "many" refers to the number of occurrences of the entity type which may be involved in the relationship.

They can be shown in the following way:

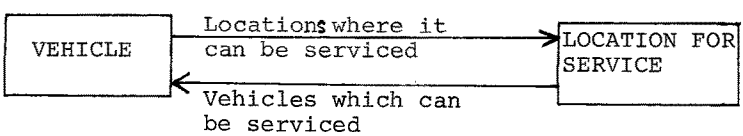
1 to 1 

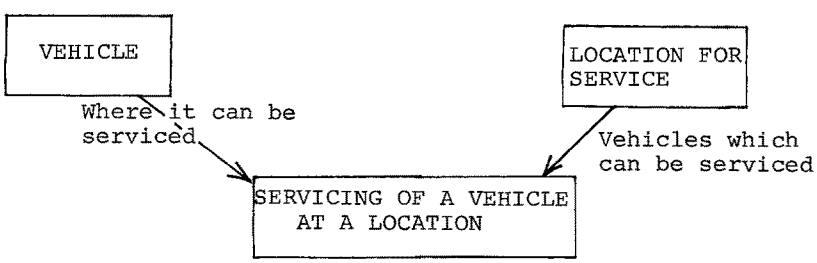
1 to many 

(i.e. with the arrow-head on the many end)

many to many 

Very often, many to many relationships can be analysed into two one-to-many relationships.

e.g. 

or 

may be more appropriate representations for the purpose of a particular application system.

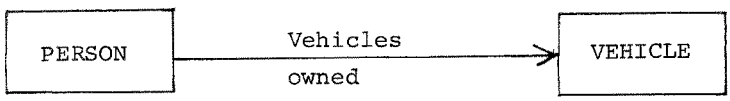
An entity must have at least one type of data value associated with it, and will usually have several. These data items types are termed attributes.

For example the entity type vehicle may have the attribute types colour, chassis-no, engine-no, date of purchase, registration-no, seating capacity, weight, and possibly many others.

One or more of the attributes of an entity may have unique values which can be used to distinguish between different occurrences of the entity. This attribute or collection of attribures is called the identity of the entity. For example, vehicle may be identified by chassis-no, or by registration-no.

Participation of an entity in a specified relationship may be optional.

For example, in:



a person may own no vehicles at all;

or in:

a vehicle may be new and unregistered.

In general, an entity has a condition associated with each relationship in which it may participate.

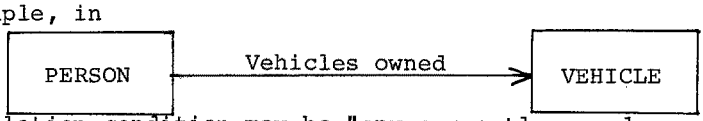
For a particular entity occurrence this condition evaluates to true or false according to whether or not the occurrence participates in the relationship.

In one-to-many or many-to-many relationships, it is necessary to attempt to quantify the "many". Thus for each relationship in which it participates in a "many" role, an entity has associated with it a population. Because data analysis itself proceeds in a number of stages of refinement, it is desirable to permit the expression of this population in several forms, e.g.

*	- many
m-n	- range
m,n	- average
n	- absolute value

The population may depend upon a specified population condition.

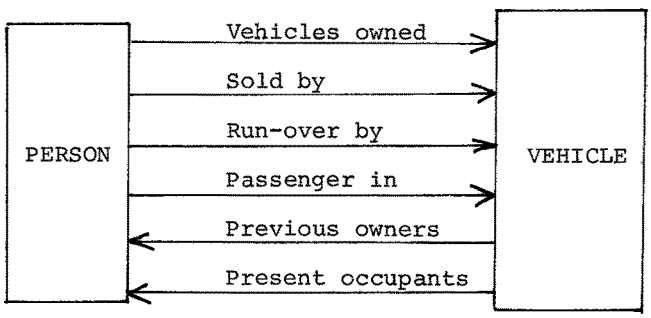
For example, in



the population condition may be "any currently owned, or owned during the past 5 years".

There may be any number of relationships between entities.

For example:

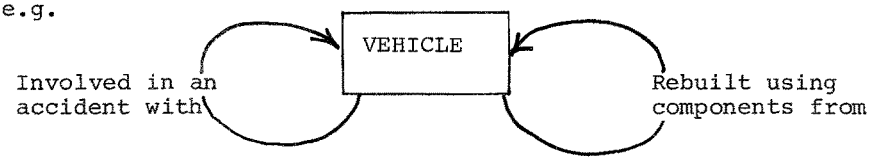


Shows only a small range of the known relationships between person and vehicle.

For this reason, it is always important to write a clear definition of each relationship when it is identified.

Relationships may exist between entities of the same type.

e.g.



4.3 Method of Working

4.3.1 Develop and discuss a series of draft entity diagrams, showing only:

Entity, Relationship/description.

4.3.2 Maintain in parallel a working set of entity and relationship descriptions.

4.3.3 Commence for each entity a list of important attributes.

4.3.4 Develop this material by examining in turn each of a series of the most important application systems. Check-out the interaction of these by discussing the overall diagram with line management at a sufficiently high level for there to be a broad understanding of all the features.

4.3.5 Document what should happen, rather than what actually takes place.

4.3.6 When entities and relationships seem fairly stable and well-identified, refine and augment the data model by showing in diagram form:

- Entities, identifiers
- Relationships, description, populations and conditions.

Such a digram can be called an entity diagram or entity model.

4.3.7 Supporting documentation should now include:

- Entity Definitions.
- Attribute lists for entities.
- Attribute definitions, including meaning of coding systems.
- Relationship descriptions, populations and conditions.
- An overall name-directory, showing entities, attributes and relationships, organised for easy reference.

4.3.8 The complete set of information required to document the corporate data model is:

Entity Data

Definition	Name, description, synonyms, existence rules
Ownership	Where appropriate, that division of the organisation responsible for all occurrences. NB Ownership is more often appropriate to specific occurrences; this is an application-dependent requirement.
Statistics	Expected population Overall growth rate
Relationships	Degree, sequence, nature/meaning
Integrity requirements	Privacy Availability
Archival	Number of versions or time-span to be covered

Of these elements, only definition, relationships and approximate statistics are relevant to initial formulation of the entity model.

Attribute Data

Definition	Name, description, synonyms, existence rules, data type and characteristics, coding structure, permissible time-lag between event and updating of the value.
Ownership	(Where different from the entity).
Statistics	Expected population Overall growth rate
Relationships	Cross-reference to entities Derivation/consistency rules
External formats	Inputs, outputs
Integrity Controls	Privacy, Availability, Validation rules, Default value, Tolerance on accuracy.

Of this data, only definition is relevant to formulation of the entity model.

Relationship Data
Populations if 1:n or m:n.
Conditions under which entity occurrrences particpate.
Conditions governing populations.
Description of the relationship.

5. FUNCTIONAL ANALYSIS

5.1 Objectives

To identify and define the requirements for particular application systems.

To produce for each system identified:

i) A functional requirements definition.
ii) A definition of the data model appropriate to the application function.
iii) A definition of likely ranges for system performance and usage traffic.

This section concentrates on items (ii) and (iii).

5.2 Relation to Other Processes

This consists of definition of the processing requirements of each application function, and definition of the associated conceptual data model. This process proceeds in parallel with data analysis, and close co-ordination is necessary.

It is taken as read in this paper that initial selection of application areas is subject to rigorous examination by senior management. A system is only developed if its value outweighs the cost of having it; in the (normal) situation of making optimum use of scarce resources, the systems having highest urgency, and the most favourable value/cost ratios are of course those developed.

5.3 Describing the Functional Data Model

Each functional data model will be a subset of the corporate data model, with additional information added, concerning:

- patterns of usage;
- access paths;
- attribute subsets actually used;
- sequencing and selection criteria.

In detail, for each entity, and for each function in which it is accessed, information required is:

Attribute subset accessed

Frequency	How often, hit distribution
Turnaround	Time, tolerance
Access paths	Relationships used, key attributes, ordering, selection/search criteria
Traffic by access path	Retrieval rate, Creation rate, Modification rate, Deletion rate.
Integrity Controls	(over and above those defined in data analysis).

Of this data, attribute subset and access paths are relevant to refinement and validation of the entity model; the remainder have more bearing upon database and storage design.

For each attribute, and for each function in which the attribute is used as a data item:

Usage	Source, destination, usage mode
Traffic	Retrieval rate, Creation rate, Modification rate, Deletion rate.

Input format	Decoding
Output format	Report heading, editing, encoding
Internal format	Base, scale, precision
Integrity Controls	(Over and above those defined in basic data analysis).
Relationships	Derivation/Consistency rules.

Format information is relevant to potential generation of text or code from the information if it is stored in a data dictionary system.

5.4 Functional Data Model Diagrams

The entity diagram for the relevant subset of the corporate data model should be augmented to show:

Access paths, sequencing and selection criteria, keys, approximate populations, approximate access traffic.

The remainder of the information specified in 5.2 should be documented in narrative or tabular form.

6. VALIDATION AND REFINEMENT OF THE DATA MODEL

6.1 Purpose

Although the specification of corporate and functional data models is co-ordinated, their features are present for different reasons, and they may be inconsistent in several ways. For this reason they should be cross-checked rigorously.

It is also desirable, having developed these models on a necessarily partly subjective basis, to apply relevant formal methods of analysis to the results. In this way it is possible to find the simplest versions of the data structures.

6.2 Stages of Validation and Refinement

i) Completeness

Does the corporate data model contain features to accommodate all required functional data models?

ii) Consistency

Is the c.d.m. self-consistent?
Is the c.d.m. consistent with each f.d.m.?
Is each f.d.m. consistent with all the others?

iii) Access paths

Access paths and their relative importance should be represented in the c.d.m. Relevant features of an access path are:

- Traffic for - add, modify, retrieve, delete.
- Search keys
- Selection criteria
- Population from which selection is to take place
- Sequencing requirements

iv) Normalisation

Reduction of data structures to 3rd normal form is a powerful tool for elimination of unnecessary complexity, and for finding the simplest possible form of those structures.

However, such a form is not necessarily the clearest or the most appropriate either for end-users or for programming. Hence after normalisation has been carried out, it may be appropriate to "de-normalise" the structures by recreation of hierarchies and repeating groups, where this can be firmly justified on the grounds of usefulness and clarity.

7. SYSTEM DEFINITION

7.1 Objectives

This is the stage at which a particular system is designed at an overall level. Constitutent work units are identified, and are embedded in a control framework which constrains the system to operate in a meaningful manner.

7.2 Identification of Processes

"Natural" units of work are identified from the requirements specification. These units, as yet, bear no relation to computer programs, but instead represent the user's view of specific integral jobs which the system is performing for him.

Examples of processes are:

- production of a report;
- the application of a specific transaction to the database;
- the computation, e.g. of a sales forecast, according to a specific algorithm.

Characteristics of processes are:

- they operate upon data; thus process inputs and outputs are "logical records", which may be:
  - transactions, reports, units of the database, transient data item groupings in memory.

Thus also, the operation of the system may be represented by a data flow chart showing related processes and their connecting data paths. (Figure 3.1 in Section 3 is an example of such a flow chart.)

Where there is value in doing so, processes may be themselves subdivided into processes; (e.g. considering the example above of processes, production of a report may consist of computation of a number of forecasts, and their ranking in some order).

The subdivision of processes in a data-flow sense loses its values at a point where:

- the units of work lose meaning for the user;
- the relationships between the units of work is more strongly that of a procedural or scheduling nature.

7.3 Design of System Behaviour and Controls

Knowledge of the user's expectations of the system permits the design of a logical framework into which the system processes can be fitted, and which will control their operation in a secure manner.

Particular attention should be paid to useability of those system control parameters provided for users.

7.4 Specification of Processes

English plus decision tables is the most appropriate specification medium. Data flow diagrams are also appropriate, but procedural diagrams and charts should be eschewed except where vital, because they tend to impose premature design decisions.

Essential scheduling requirements between and within processes must be identified and fully defined at this stage. A strong mandatory scheduling relationship between work units indicates that they should be regarded as part of a process.

Care should be exercised to avoid jumping the gun on detailed design work through inclusion at too early a stage of "how" decisions on mechanisms for achieving the results required of the processes. However, certain "how" decisions are appropriate at this stage: e.g. selection of computational methods of performing forecasting or optimisation calculations which are integral parts of the system. Such decisions are necessary at this stage because they affect the system's behaviour towards the user, and have an impact upon the data models.

7.5 Specification of Constraints

The value of each process to the system should be ascertained; some processes are optional, and inclusion of these has to be justified. For each function performed, there may be alternative options, e.g. simple or sophisticated.

Some form of ranking based on priority, cost and flexibility should be carried out, in order to select the actual constituents of the first version of the system, and to lay the guidelines for subsequent development phases.

Tolerances of ranges should be defined for:
accuracy of results, performance, consumption of resources (both development and operating), scheduling.

8. PROGRAM AND RUN DESIGN

8.1 Objectives

The primary aim of this stage is to select suitable groupings of system processes, to be formed into programs and program sequences. The resulting work-units are intended to make effective use of the resources of the computer by, for example, avoiding unnecessary repeated transfer of data, whilst at the same time meeting the performance requirements specified.

8.2 Method

Each system process should have defined:

i) Usage frequency;
ii) Desired turnaround time;
iii) Data access requirements, including sequencing;
iv) Scheduling relationship to - date/time,
- other processes.

There is a trade-off to be exercised between keeping processes separate, which makes inefficient use of the computer, but which retains flexibility, and the binding together of processes into programs which can make more effective use of the computer, but which will need to be redesigned if new processes are introduced.

Stages of the procedure are:

i) Choose program groupings according to:
   - I/O - Database access
     - Non database "batch" I/O; TP message handling
   - Scheduling Functions
   - Common service functions
   - Liability to be invoked in a co-ordinated manner (i.e. function occurrences part of the same process).
   - Close similarity of associated functional data models
   - Similarity of scheduling requirements - e.g. end-month.

ii) Specify inter-program scheduling relationships;

iii) Specify overall frequency and scheduling requirements;

iv) Specify the data-access needs of each program in terms of:
   - subset of the schema data model,
   - access paths,
   - sequencing,
   - selection criteria,
   - frequency of access.

9. SCHEMA DESIGN

9.1 Objective

The aim is to design an effective implementation view of the corporate data model. The effectiveness is judged in terms of:

- Clarity and appropriateness
- The provision of facilities to enable the definition of effective forms of the necessary sub-schemas.
- The schema's suitability as a basis for the definition of an efficient storage structure.

9.2 Relationship Between Design of SCHEMA and SUBSCHEMA

The details of the method for SCHEMA design are dependent upon the nature of the file-handler or DBMS in use. In particular, if a conventional file-handler is used, the drawing of meaningful distinctions between

- SUBSCHEMA
- SCHEMA
- Storage

may prove difficult.

The simulation of sub-schema using a conventional file handler is described in 10.2.

In this section, design methods are proposed which are appropriate to use in Codasyl-type DBMS. (CODASYL DDLC 1973).

The Schema must be designed in principle before subschemas can be designed. This is because the choise of structural facilities in a Codasyl SUBSCHEMA is limited to those already present in the SCHEMA. At the same time, one of the main inputs to SCHEMA design is a definition of the needs of those SUBSCHEMAS. Hence it is most likely that there will be several cycles of iteration between schema and subschema design before satisfactory designs are achieved.

When modifying the SCHEMA in response to altered or new SUBSCHEMA requirements, care must be exercised to avoid changes which would invalidate other existing SUBSCHEMAS. In particular, it is necessary to take great care to avoid subtle changes, which would leave the SUBSCHEMAS and DML syntactically valid, but which would alter the semantics.

9.3 Method for Design of a CODASYL SCHEMA

9.3.1 Translation of the entity representation of the corporate data model involves exercise of the following choices:

- Representation of entities:
  - as record-types,
  - as several record types, associated in some way,
  - as group items within a record type.
- Representation of relationships:
  - as set types,
  - as several set types, possibly with additional record types, e.g. as is required for many-many relationships,
  - as repeating groups,
  - as loose associations of records, e.g. by common search keys.
- Association of attributes with entities.

9.3.2 The information produced in the design process can be classified by:

a) To be encoded in schema DDL
   - i) Relevant to structure e.g. RECORD, SET, AREA
   - ii) Relevant to mode of use e.g. ORDER, KEY
   - iii) Relevant to storage design e.g. LOCATION MODE, ACTUAL, VIRTUAL

b) To be documented for reference by application programmers.
   e.g. Record occurrence rules
   Record retrieval modes
   Record selection rules
   Processing necessary to maintain consistency when each record is STORED, MODIFYed or DELETEd.

A subset of the type a) information can usefully be shown in diagram form. Successive, not necessary complete, forms of such a diagram can be used as working notes during the design process.

9.3.3 Choice of Records and Sets

The most basic choice in SCHEMA design is "what are to be the record types and set types".

Stages:

i) Make each entity-type a record type, including all attributes of the entity data items within the record. Represent relationships between entities as set types.

ii) Many-to-many relationships, treated as in (i) above, will yield a record/set structure which is invalid in a CODASYL DBMS. This is overcome by breaking any such relationship into two one-to-many relationships, possibly making use of an additional "intersection" record, as indicated in Section 4.2.

It is possible that several alternative structures will result from stages (i) and (ii). There is no harm in this; the alternatives should be carried forward until there are grounds for choosing between them. The design (or designs) represent a "first draft" and are now subject to a number of stages of refinement.

iii) Where different subsets of the attributes of an entity have different access needs (i.e. are used in different subschemas, and/or have different access paths), consider their division into 2 or more record types.

NB Data items common to these record types are redundant, and must be kept consistent in some way.

iv) Where, within a record type, there is a repeating group which has a large, variable population, it may be better to make the elements of the repeating group the member records of a set, and thus make the DBMS responsible for the space management problem.

v) Where record types are very small, and several record occurrences are commondly accessed together, then it may be worthwhile to consolidate record types, so that the structure is represented by group items within a record type.

vi) Where a relationship is used rarely or not at all it may be that the overheads involved in sets are not justifiable. In this case, the set-type can be eliminated. Instead, the record occurrences can be associated by a selection process based on like data item values.

vii) Grouping of several record types by making them members of the same set type is appropriate where this reflects the common patterns of access.

9.3.4 Choice of declared record keys, and record sequences in sets, is governed by the most popular sub-schema usage.

9.4 Information from Data Analysis

Choice of record types, and of the access paths to them, is governed by the relative frequency with which a group of items is accessed in a particular manner.

Thus an analysis procedure is required which:

i) For each item, determines the relative popularity of different access paths to it.

ii) For each access path, lists, ordered by popularity, the data items to which access is requested.

Storage structure design (Section 11) requires that these figures should be dissected by:

i) Type of access -
   batch, where economy of resources is paramount
   online, where response time is paramount

ii) Insert/delete/modify/retrieve operation.

10. SUBSCHEMA DESIGN

10.1 Objectives

The aim of this stage is to achieve a program-view of data which is:

i) Appropriate to the needs of the program.

ii) Unlikely to be sensitive to alterations in other systems, the schema, or the storage structure.

iii) Consistent with the schema, and derivable from it.

10.2 Method

10.2.1 Using conventional data management

By "conventional" is mean any system in which programs normally access directly real files resident on backing storage.

i) For each record-type in each file on backing storage, write a subroutine to perform each of four functions:
   - create;
   - delete;
   - modify;
   - read.

   These subroutines should embody any processing necessary to maintain consistency.

ii) Design the program view of data, including selection and search procedures.

iii) Define the data mainpulation operations to be carried out by the program.

iv) Write subroutines for (ii), using (i) as primitive operations.

10.2.2 Using a Codasyl style DBMS

The problem is to translate a sub-schema requirement, framed in terms of access to the entities in possibly several functional data models, into a record/set subschema, with associated access path details.

As stated in Section 9, Schema and Subschema design are closely interrelated.

Select a subset of the schema which most closely meets the subschema needs. (This is probably best shown in diagram form, based upon the conventions shown in Figure 9.1).

If this is unsatisfactory, determine whether the discrepancy is to be met by modification of the schema, or by special purpose programming to derive the desired data from a feasible subschema.

Any modification of the schema should be subject to the safeguards mentioned in Section 9.2.

There will always be a certain amount of processing which cannot be expressed in schema or subschema DDL.

This falls into two classes:

i) Processing which is generally applicable to operations carried out on a particular part of the database; e.g. maintenance of consistency when a particular record is created or deleted. This is best embodied in general purpose subroutines, which are made available as part of the database documentation.

ii) Processing which is specific to a particular program; e.g. selection criteria applied on a particular access path.

Both sorts of special purpose processing should be clearly documented; the latter may be shown on the subschema structure diagram.

11. STORAGE STRUCTURE DESIGN

11.1 Objectives

The aim of this stage is to design a representation of the database on backing storage which efficiently meets the specified pattern of access and usage, and which also keeps within specified constraints upon use of resources.

11.2 Statistics Required

The performance of the overall combination of database, DBMS and application programs is expected to be "optimal" in some way.

Some combined function of:

- backing storage;
- main storage;
- channel time;
- processor time;

is to be minimised.

Of these only backing storage is not dependent upon usage traffic.

The two necessary sets of information for the design process are:

. The relative weightings of the four main resources to be conserved.

. Statistics defining:
  - for each database record frequency and hit distribution of:
    . add;
    . delete;
    . modify;
    . retrieve operations
  - for each access path used in the database, frequency of use in each of batch and on-line modes.

11.3 Method of support for access paths

Access paths should be ranked according to their level of usage. The most heavily used paths should then be given highest priority for efficient access.

On-line access paths require rapid response, possibly at some cost in consumption of resources. Batch-mode access paths require support at a minimal consumption of resources, at the expense of elapsed time if necessary.

Heavily used access paths should be supported by appropriate indexes and pointer mechanisms where the accompanying overheads can be justified.

11.4 Summary of Choices

For a Codasyl-style DBMS, the range of choices which must be exercised at the stage of storage design are:

Database Access Strategy

- Record placement mode - CALC or VIA.
- Record retrieval modes to support, and how efficiently.
- Real or virtual representation of derived data.
- Set representation mode and linkage options.
- Indexing.
- Search keys.

Storage Usage

- AREA placement.
- Schema record to storage mapping;
  - data item representation,
  - data item distribution.

Storage Characteristics

- Device type.
- Page size.
- Space allocation;
  - amount of overflow,
  - growth rate.

ACKNOWLEDGEMENTS

Although the impetus for this work has come from several years experience gained on a variety of projects, the encouragement to document it in its present form occurred while the author was under contract to the Ministry of Defence. Thanks are due to my MOD colleagues for this encouragement.

The author is also especially indebted to Bill Waghorn of SCICON and Ian Palmer of CACI, who have both contributed significantly to the ideas through extensive discussions. For comments upon drafts of this paper, thanks are due to many colleagues within the Ministry of Defence, and ICL.

REFERENCES

1. ANSI X3 SPARC/DBMS Study Group Interim Report, February 1975.
2. CODASYL DDLC Journal of Development 1973. Revised 1975.
3. CODASYL COBOL JOD as modified by the DBLTG Database Facility Proposal, March 1975.
4. Brown, A.P.G. "Entity Modelling" IFIP TC/2 Conference 1975 (Pub. North Holland Book Co.)
5. Robinson, K.A. "Description of stored data, using modified NCC Standards", Private communication, 1974.
6. Bramhill, P.S., and Taylor G. "Database Design. From Codd to Codasyl". To be published in "Database Journal", 1976.
7. BSC/CODASYL DDLC DBAWG June 1975 Report.