Systems analysis documentation: computer-aided data dictionary definition

By P. J. H. King*

This paper discusses the need for formal notations and techniques for use in systems design and analysis. It argues that the data dictionary is a central tool in this work, makes proposals for formalising it and suggests computer based procedures for aiding its construction.

(First received February 1968 and in revised form September 1968)

A need exists for the development of standard techniques and formal notations for use in systems design and analysis. Their value lies in expediting work, facilitating work sharing by aiding communication between team members, assisting supervision, and providing a clear and generally understood design/implementation interface. Canning (1966) states '. . . our present tools are just too cumbersome to undertake the more complex studies in the average business environment'. The need for such methods is referred to in Gibson *et al.* (1967) and has been cogently argued by Grindley (1966) whose suggestions are further discussed in King (1967). An attempt has been made to establish a formal theoretical basis for this type of work described by Bosak *et al.* (1962) and briefly by Clippinger (1962). This, however, is at present far removed from the realm of day to day practical application.

A scheme designed for the documenting and study of existing data systems called AUTOSATE has been developed by Butler *et al.* (1964). Variants of these ideas under other names, e.g. Dataflo, Crowther-Watson (1967), also exist. The basic concepts are those of 'station' and 'event chain'. A data 'system' is divided into a number of separate 'stations', and information flowing between stations and data retention at stations is recorded. This is processed to produce documentation and identify 'event chains' or sequences of derivative connected activity. The basic philosophy is that we record the present system with a view to improving on it. It seems likely that outside the large complex data systems of the military area for which Autosate was designed, the approach will be more appropriate in an industrial context than in business data processing. In an industrial context the data system, though essential, is ancillary to the manufacturing activity, and any data system redesign is constrained by the nature of the existing plant and machinery. In the context of business data processing there is normally more scope for data systems redesign, and it is frequently better to identify basic requirements and design a new system based on these than to attempt to record and relate all details of every existing clerical process, although the essential functioning and purpose of these activities must be well understood.

The Autosate approach is to consider the 'total system' and the idea of an 'application' is rejected. This view is not adopted in the present paper, and we conform to the usual notions of applications and related applications. Once the broad scope of an application is refined and interfaces with continuing manual and other systems roughly specified, a detailed systems study within these limits is required to produce program and other specifications. This activity is defined in Gibson (1967) as systems design and in Canning (1966) as levels 1 and 2 of systems work. It is argued here that part of this activity should be to create a data dictionary for the application. This will detail data required on output documents, information provided by input documents and data for retention on files. The dictionary should define the structuring of data including, where relevant, structures occurring at intermediate stages in processing. Thus if input data is to be edited to a standard internal form, then this form may be specified.

The significant part that a data dictionary can play has been well demonstrated by Fisher (1966) in an attempt to clarify the important parts of satisfactory documentation. The dictionary described by Fisher is fairly primitive. It is only a slight elaboration of the list of data elements involved which one expects to find with any reasonably satisfactory piece of systems documentation. A similar dictionary exists in the documentation produced by Autosate which also shows the relationships of data elements to the defined 'stations'. The dictionary of Systematics (Grindley, 1966; King, 1967) is also a simple list of data elements with their elementary properties, but in addition there is an attempt to categorise elements as 'given' and 'derived' with respect both to the whole system, and to various sub-systems.

This paper makes no attempt to propose a comprehensive methodology for use in systems design and analysis along the lines of the three proposals we have discussed. Rather there is a recognition that any such proposals will include some form of data dictionary, that existing methods use some form of data dictionary and that the data divisions of COBOL and NEBULA (Braunholtz *et al.* 1961) were an early attempt at formalising this concept. It is suggested that the concept of a data dictionary for an application is important and that its content and form should be further developed. Its construction is facilitated if this is computer aided, and suggestions for this are made. The work of constructing a dictionary aids analysis by documenting and clarifying work as it proceeds but its final purpose is to specify in a precise and convenient form the data involved in an application including its grouping and structure.

The major data processing languages (e.g. COBOL, NEBULA) recognise the importance of data definition by having a distinct data division. Recent language development lays even greater emphasis on data structure

* *Computer Unit, University College of Wales, Aberystwyth*

specification and BCL (Hendry, 1966; Hendry and Mohan, 1968) could be described as 'structure oriented'. The systems methodology proposals discussed in the foregoing all present their data dictionary as an alphabetic list of data element names together with field requirements and miscellaneous information. They do not include group or structure names in the dictionary nor provide formal facilities for group naming and specification. The data divisions of COBOL and NEBULA, on the other hand, emphasise structure and display it clearly. BCL does not attempt to display structure but specifies it precisely in a very simple way.

Notation of the BCL type is adopted in the suggestions in this paper. Information conveyed in this notation is processed to provide data dictionary documentation, the self-consistency or otherwise of the information being determined during processing. Amending and correcting facilities are provided to alter and improve the documentation. It is suggested that the processing involved should be a computer function.

Attributes of a data dictionary—programming requirements

As stated above a data dictionary provides working information for implementation. What does this requirement involve?

First, it must specify all data elements involved giving a useful and meaningful name to each for human communication purposes. A field specification for each element or adequate information to determine one is required, together with allowable ranges for numerical items and sets of allowable values for non-numerical items. Secondly, it should provide information on the grouping and structuring of the data, since this is often a natural way of defining records within a programming scheme. As with elements, sensible and meaningful names are required for groups and structures. A third requirement is information on variable and optional occurrence of data. For example, in an input structure groups or elements occurring optionally or a variable number of times must be so specified. With variable occurrence, information is required on variability likely to be expected in practice, since decisions may be necessary on whether to use fixed or variable length records.

Data names which are clearly meaningful frequently prove rather cumbersome for programming. A fairly common technique of avoiding this is to assign simple codes as alternatives to names to give the required brevity. We suggest that a data dictionary should define such codes.

Data dictionary construction during analysis and design

Data definition in programming requires individual data elements to be identified and defined before groups and structures; there is definition 'upwards' from the elements. This is so even where the notation (e.g. COBOL) is apparently 'downwards'. It is necessary for program generation but implies that formalised documentation and use of the computer's checking capacity must wait on all detail being specified.

Data specification is 'from the top down', that is broad groups and structures are defined first and given names, then sub-groups and sub-structures and so on down to the individual elements; this is the natural approach of systems analysis. BCL has a simple notation for such a method of working and groups and structures are defined by statements of the type

$$A \textbf{ is } (B,\ C,\ D)$$
$$B \textbf{ is } (X,\ Y)\text{, etc.}$$

This notation corresponds to the natural approach of systems analysis. In BCL the above definition of A is only valid if B, C and D are already defined. Use of this type of notation for analysis and design requires this restriction to be relaxed, since introduction of data names without a precise definition must be allowed. There will, of course, be a general idea of what such a name signifies but there must be freedom to leave precise specification to later.

In the example above A and B are defined as group names but C, D, X and Y remain undefined. Subsequently they may be defined as group names or specified as data elements by the giving of a field definition, information on permissible values, uniqueness, etc. There is advantage in not requiring field specifications at the time names are introduced as is usually required in programming languages. Names not defined as groups or given a field definition represent data about which further information must be provided.

A large number of definitions of the type discussed, together with field specifications, information on variability, optionality, etc., would not be very readable even though precise in information content. In addition to proposing formalised ways of giving these definitions it is suggested they should be processed by computer, vetted for errors and used to create a data dictionary file. From this the data dictionary documentation will be obtained. Thus it is explicitly recognised that part of systems analysis and design is itself data processing and that a computer can be used advantageously in this. An important feature is that the dictionary file is created at an early stage and information is added continuously as the work proceeds. A display is always available of the 'state of the work so far' with processing and clarification as it proceeds. Much of the formal documentation chore thus becomes a computer function. Work outstanding is indicated in the dictionary documentation and, where a team is involved, there is automatic amalgamation of different members' work, each receiving up to date documentation of the whole project in standard form.

The proposed data dictionary documentation

It is suggested that data dictionary documentation should have four main sections although when complete the fourth will not be present. The four sections are:

(i) An alphabetical list of all data names—the main dictionary.
(ii) A list of data elements only, giving field descriptions and information on permitted variability.
(iii) A document for each major structure displaying the data relationship. This includes linear ordering information where relevant, e.g. for an input structure present on paper tape other than in its natural order.
(iv) A list of undefined data names. This indicates work still to be done.

We do not attempt to give full details of these documents here although some features are apparent from the example which follows. Briefly the alphabetical list, or main dictionary, gives a code for each name which is used in the remainder of the documentation and indicates whether it is an element or group name. If it is part of another structure or structures the reference(s) of the containing structure(s) are given. If the dictionary covers data for more than one procedure area then the relevance of the data to the procedure areas is specified. It is important to emphasise that the main dictionary is a list of *data items* that occur in the system. It is *not* a list of suggestions for identifiers for parts of computer store. The second item of documentation lists all data elements, together with information necessary or desirable if satisfactory programs are to be obtained. The third part gives conventional COBOL type documentation for each major structure including level numbers. The fourth part will contain no information when the documentation is complete. At an intermediate stage it gives all data names which have been introduced either as part of a group definition or individually and have not been specified either as group names or as data element names. Essentially this list is an indication of work requiring to be done provided the major structures in the application were identified initially and there have been no omissions during subsequent breakdown.

An illustrative example

We give a simple illustrative example which is intended to be self-explanatory. Some information supplied during analysis and design is given in (*a*) and the documentation derived from it as a result of processing in (*b*). It should be emphasised that information of the type shown in (*a*) may be in any order and added to a computer-held data dictionary file in piecemeal fashion. Statements are available for defining structures in the way already described, for specifying multiple occurrence of data items or structures, for giving field specifications, ranges, and allowable sets of values.

The four sections of the resulting dictionary documentation are shown in (*b*). In the main dictionary, for each name there is a code and type indicating element or group. The context gives the level at which the name occurs in a group and the code of the immediately containing group or a reference to a major structure display. Also the number of occurrences of the item within its context is given.

(*a*) *Information supplied by the problem analyst*

```
DAILY-TAKINGS is (KIOSK, DAY, DETAIL
                    occurs 0 to 40 times, BANKING)
DAILY-TAKINGS comment FILE RECORD
                                  STRUCTURE
DETAIL is (CODE, AMT)
CODE value unique within DAILY-TAKINGS,
        range 1 thru 40
AMT field 999.99
BANKING field 9999.99
WEEKLY-TAKINGS is (KIOSK, WEEK, DETAIL
        occurs 0 to 40 times, TOTAL-BANKINGS)
DETAIL in WEEKLY-TAKINGS similar DETAIL
                          in DAILY TAKINGS
KIOSK common TO BOTH STRUCTURES
```

(*b*) *Dictionary documentation derived from file*

(i) *Main Dictionary*

Code		*Type*	*Con-text*	*Occur-rence*
A1	AMT*IN*D3	E	3 D3	1
A2	AMT*IN*D4	E	3 D4	1
B1	BANKING	E	2 D1	1
C1	CODE*IN*D3	E	3 D3	1
C2	CODE*IN*D4	E	3 D4	1
D1	DAILY-TAKINGS	G	1 (1)	
D2	DAY		2 D1	1
D3	DETAIL*IN*D1	G	2 D1	0 *to* 40
D4	DETAIL*IN*W2	G	2 W2	0 *to* 40
K1	KIOSK		2 D1 2 W2	1
T1	TOTAL-BANKINGS		2 W2	1
W1	WEEK		2 W2	1
W2	WEEKLY TAKINGS	G	1 (2)	

(ii) *Data element field description*

Code	*Name*	*field*	*value restrictions*
A1	AMT*IN*D3	999.99	
A2	AMT*IN*D4	999.99	
B1	BANKING	9999.99	
C1	CODE*IN*D3	99	range 1 thru 40 value unique within D1
C2	CODE*IN*D4	99	range 1 thru 40 value unique within W2

(iii) *Major structure displays*

```
(1) 01 DAILY-TAKINGS comment FILE RECORD
           STRUCTURE
       02 KIOSK
       02 DAY
       02 DETAIL occurs 0 to 40
          03 CODE
          03 AMT
       02 BANKING
(2) 01 WEEKLY-TAKINGS
       02 KIOSK
       02 WEEK
       02 DETAIL occurs 0 to 40
          03 CODE
          03 AMT
       02 TOTAL BANKINGS
```

(iv) *Undefined data names*

```
       D2 DAY
       K1 KIOSK
       T1 TOTAL BANKING
       W1 WEEK
```

Use of 'on-line' computer facilities in systems analysis and design

The computer function required to aid design and analysis in the way described can be provided using 'on-line' computer facilities. A particular project is assigned a file for its dictionary information which is used for recording this part of the work as it proceeds. Information is added to the file in a piecemeal fashion whenever convenient. Up to date documentation in whole or part is requested as needed. It would be natural for up to date documentation to be obtained quite frequently, particularly where a team effort is involved, since this would provide their working papers.

Also the documentation would serve as an aid to the team leader in the management of the project.

A new set of dictionary documentation may well cause an immediate response. If a data name reads unsatisfactorily when seen in context the on-line facility enables immediate substitution of a more suitable one. A field specification may have been omitted in error and this will be remedied on seeing the list of undefined data names. The project leader may note that a comment indicates misunderstanding within the group or that ascertainable value restrictions have been omitted. In general use of on-line facilities will greatly assist the production of good documentation. They make 'working on one's work' less of a chore than is otherwise possible.

Whilst we suggest that on-line processing is useful in providing the facilities needed, the activity described can also be carried out in a conventional batch processing environment. The benefit of the on-line technique in this context is essentially one of time scale. It allows the user to effect small updatings of his files as the need arises and eliminates the need for interim manual alterations to documentation. In a batch processing context numerous small updating runs would not be practical and additions and amendments would have to be batched into reasonable sized jobs. This means that a certain amount of interim manual updating of documentation is probably necessary. Nevertheless it seems that suitable facilities can be provided on relatively small conventional computers and such implementations should prove valuable, particularly where frequent small updatings can be tolerated and the working dictionary files therefore kept up to date.

References

Barnes, P. G. (1966). The job of a systems analyst, *Computer Bulletin*, Vol. 10, No. 3, pp. 25–29.

Bosak, R., *et al.* (1962). An Information Algebra, *Comm. ACM*, Vol. 5, pp. 190–204.

Braunholtz, T. G. H., *et al.* (1961). NEBULA: A Programming Language for Data Processing, *Computer Journal*, Vol. 4, pp. 197–211.

Butler, D. D., Fairbrother, E. M., and Gatto, O. T. (1964). Data System Design and Control using AUTOSATE—an Automated Data System Analysis Technique, Mem. RM-3976-PR, Rand Corp., Santa Monica. Feb. 1964.

Canning, R. G. (1966). Coming changes in systems analysis and design, *Proc.* 1966 *ACM Nat. Conf.*, pp. 373–377.

Clippinger, R. F. (1962). Information Algebra, *Computer Journal*, Vol. 5, pp. 180–183.

Crowther-Watson, M. (1967). Papers on Dataflo circulated privately. Available from National Computing Centre.

Fisher, D. L. (1966). Data documentation and decision tables, *Comm. ACM*, Vol. 9, pp. 26–31.

Gibson, R. P., *et al.* (1967). Education and Training of Systems Analysts, *Computer Bulletin*, Vol. 11, No. 1, pp. 11–17.

Grindley, C. B. B. (1966). Systematics—a non-programming language for designing and specifying commercial systems for computers, *Computer Journal*, Vol. 9, pp. 124–128.

Hendry, D. F. (1966). *Provisional BCL Manual*, Internal document of Institute of Computer Science, University of London.

Hendry, D. F., and Mohan, B. (1968). *BCL* 1 *Manual*, Internal document ICSI 103 of Institute of Computer Science, University of London.

King, P. J. H. (1967). Some comments on Systematics, *Computer Journal*, Vol. 10, pp. 116–118.

Book Review

Optimisation in Control and Practice, by I Gumowski and C. Mira, 1968; 242 pages. (London: *C.U.P.*, £3 5s. 0d.)

This is an extremely odd book. It is mostly devoted to variational problems of general interest to mathematicians and physicists but such engineering and technology as enter into the text is of a curiously antiquated variety. Furthermore, there is very little that is concerned with practical applications of control theory and consequently one can only conclude that the title is particularly unfortunately chosen. I doubt if mathematicians will find much pleasure in the text, since many difficult mathematical results needed are either quoted without proof or dealt with heuristically.

In fact one wonders where the authors have been in the last twenty years, since they use a notation alien to control people. The whole topic of the use of iterative gradient procedures for solving the difficult non-linear control problems that are worrying us today, and all the numerical difficulties of handling large-scale calculations in these terms seem to have missed their gaze.

However, if one forgives the authors for their title, there is some useful content. The authors favour the method of tackling extremal problems developed by Caratheodory at the turn of the century which leads to a partial differential equation system. This is a point of view which not everybody would subscribe to. The references quoted are almost exclusively European and Russian sources, many of them unfamiliar, and this perhaps helps to give the book its distinctive flavour.

However, in the end one cannot quite forgive the authors their title which promises so much but in the event yields so little.

J. Westcott (London)