Towards comprehensive specifications

S. J. Waters

London School of Economics, Houghton Street, London WC2A 2AE

This paper suggests that specifications should be more factually complete so that systems queries can be reduced; this would save much cost and time of analysts, designers, programmers, operators and the computer itself. Technical facts are defined for a specification's data dictionary, messages, data base and procedures and are checklisted so that documentation standards and higher level systems languages can be analysed for completeness—typical examples (e.g. ADS, NCC, PSL) seem to be more incomplete than they are complete.

(Received February 1978)

1. Introduction

This paper is further fallout from the CAM research project, at the London School of Economics, which is investigating computer aided methods of defining, designing, implementing and maintaining computer based information processing systems.

A specification (sometimes termed 'systems spec', 'functional spec' or 'proposal') is probably the most important document of a systems project. It records the *user requirements* for a system so that they can subsequently be designed and implemented—thus, the specification defines '*what* is required' logically rather than '*how* it will be achieved' physically. Usually, systems analysts produce a specification in close consultation with users and, after agreement, pass it on to computer systems designers and programmers; these technicians subsequently raise many *systems queries* when constructing the physical data base and its attendant programs—thus, analysis and design are iterative processes in practice.

A comprehensive specification attempts to minimise the number of systems queries due to facts being omitted from the specification. This is important because incomplete specifications often cause midnight panics in the computer room, missed deadlines, soaring costs and even dismal failures in many systems projects; for example, Grindley (1975) notes one survey that suggested half of programming time was wasted waiting for systems queries to be answered and another IBM survey estimated the cost of resolving a systems query when the system is operational as one hundred times the cost of resolving it during systems analysis. Thus, if analysts make more effort in specifying systems comprehensively then not only should they save much of their own time later but also that of designers, programmers, operators and the computer itself.

Documentation standards are often used in an attempt to achieve comprehensive specifications. Systems analysts enter technical facts on to preprinted forms which eventually can be manually crosschecked for completeness and consistency; ADS (NCR, 1969) and NCC (1969) are two typical examples of this approach. Currently, higher level systems languages are also being developed to input specifications into supporting computer software; PSL (1975) and BDL (Leavenworth, 1977) are advancing on this front. However, such standards and languages can only achieve comprehensive specifications if they are themselves comprehensive and do not omit significant technical facts.

This paper attempts to define the technical facts that may be recorded in a specification. It recognises the main purpose of a specification as enabling the system to be constructed and therefore analyses technical decisions, their alternatives and techniques, to deduce the information a designer or programmer might need in his work. The resulting facts are briefly classified and developed into a feature analysis which suggests some standards and languages are more incomplete than they are complete.

2. Technical facts

Much of the systems analysis literature superficially discusses fact-finding techniques without ever defining the facts that may need to be found. Waters (1979) attempts to remedy this by analysing the problems faced by technicians to establish the information they may need to solve these problems. The aim of this work is to make specifications, documentation standards and higher level systems languages more complete and thereby reduce the high timescales and costs of systems projects.

First, it is important to define what a system is so that analysts can include all parts in the specification. **Fig. 1** extends the work of Martin (1967) into an anatomy of a system which recognises the following logical parts:

Application sub-system containing the user's procedures to transform *transactions* of real-world events into *results* and update the data base history of files of records of elements.

Information retrieval subsystem to interrogate the data base by generating *responses* to *enquiries*, subject to any privacy constraints.

Data base maintenance subsystem to modify the data base by *insertions*, *amendments* and *deletions* and produce *proof lists* of the changes that have been made, subject to any ownership constraints.

Control subsystem to help detect, locate and correct fraudulent and accidental errors by generating *error lists* and *reconciliations*.

Recovery subsystem to prepare for, degrade during and recover from technical failures.

Monitoring subsystem to produce a *log* of operational events.

These last three subsystems are triggered by various *parameters*. Thus, a general system can logically be viewed as six subsystems which collectively transform six types of input messages (which include any time triggers) into six types of output messages. This analysis is not offered as a definitive classification but as a useful checklist of the parts that should comprise a typical system; Waters (1979) explains the analysis in detail.

Now assume that this system is to be designed and imple-

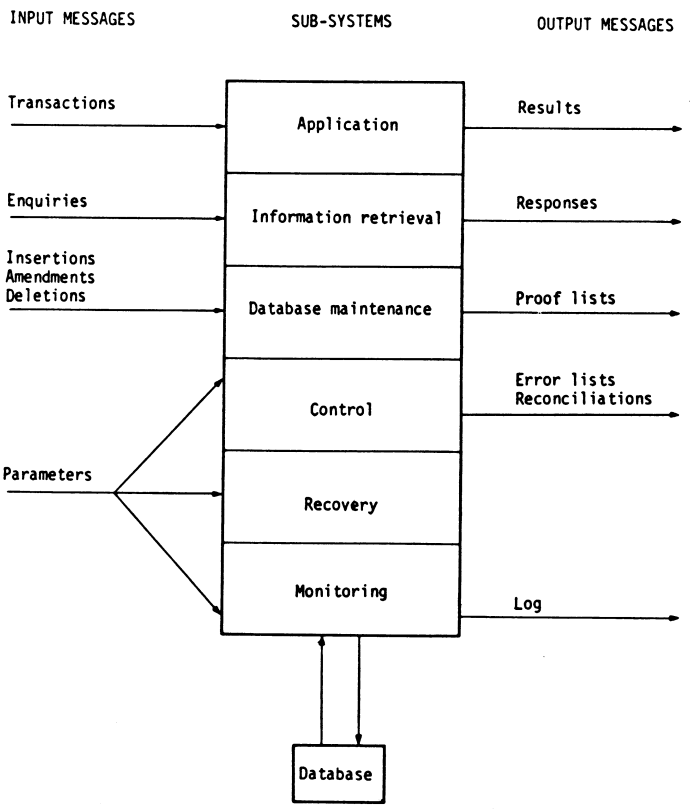

Fig. 1 An anatomy of a system

mented with a free choice of current technology (including batch and real time processing, teleprocessing, centralised and distributed processing, tape file processing and disc data base processing etc.). Then, a specification technique in general, may include the nearly eighty different facts listed in the Appendix; the encircled ones appear to be vital logic which must be defined even if some supercomputer were available which has infinite memory, is infinitely fast and costs nothing. Notice that over forty facts are encircled which means that automatic programming, or self-programming computers, would not eliminate the vast amount of work involved in programming the system (as opposed to programming the computer).

No claim is made here that the list is complete. Teichroew and Hershey (1977) are probably correct in suggesting 'completeness can never be fully guaranteed'—this realisation has no doubt contributed to the ISDOS about-turn of 'design is essentially a creative process and cannot be automated.' Hundreds of computing techniques and applications have contributed to the list but others are bound to have been overlooked.

Even if a comprehensive analysis could be guaranteed, it would not be feasible to specify a system completely anyway because some of the facts yield vast documentation that is only occasionally relevant. For example, fact 14 would require all values of all elements (at least keys) to be specified but this information might only be useful on the rare occasions that algorithmic files are being considered. Thus, some facts would only be documented if designers and programmers request them—Waters (1978) provides an analysis, albeit extremely subjective. Notice therefore that systems queries cannot be totally eliminated but they can probably be significantly reduced.

3. Specification techniques

Probably most organisations use specification techniques, such as documentation standards or occasionally higher level systems languages, to guide analysts. Disappointingly, these are often introduced and then left undeveloped in the light of feedback from their use; for example, a systems query arises on one application (e.g. fact 30 above has been ignored) and that system is revised appropriately but the specification technique is not improved—thus, precisely the same systems query arises on every application that is subsequently specified by that technique. A German proverb suggests 'the devil is in the detail' and those heavily involved in systems maintenance will know the problems caused by important but overlooked details during analysis and design—at least they should be formally given the opportunity to improve weak specification techniques so that future systems do not repeat the mistakes of the old.

A common weakness is that the technique is not comprehensive. Many do not aim to be, but instead concentrate on particular aspects of specifications; for example, Bosak's Information Algebra (1962), Grindley's Systematics (1975) and Pengilly's PROPLAN (1976) mainly confine themselves to the logic of a system so they cannot be criticised for covering less than 20% of the above facts. However, some techniques do aim to be comprehensive; for example,

ADS (1969) 'can assure that there are no loose ends, no omissions and no ambiguities; and that there are no isolated, incoherent, or irrelevant facts. ADS provides all the audits and guidelines needed to assure the definer that definition is complete'.

NCC (1969) helps to 'ensure there are no loose ends, either in fact finding or in system specification', and

PSL (1975) increases the 'preciseness, consistency and completeness' of documentation and its objective is 'to be able to express in syntactically analyzable form as much of the information which commonly appears in System Definition Reports as possible'.

Fig. 2 analyses these three specification techniques as a feature analysis of the facts they cover—none reaches 40% completeness and all omit more than half of the vital, logical facts. Of course, it can be argued that their generous notes and memo facilities allow free-format comments which make them complete, but such informal facilities give no guidance to analysts. Further, it may be the case that they can formally document most of the mass of a specification because a few of their included facts (e.g. contents of messages, data base and procedures) account for the bulk of its pages—however, the three specification techniques examined are more incomplete than they are complete when judged by their coverage of facts.

4. Conclusion

This paper has summarised some recent research (Waters; 1979) which examines the technical facts that may be recorded in logical specifications. The aim is to improve the completeness of this documentation and to help reduce, but not eliminate, the high times and costs of systems queries.

The facts, which are not claimed to be comprehensive, can be checklisted and used to judge the factual completeness of specification techniques, such as documentation standards and higher level systems languages. A feature analysis of ADS (1969), NCC (1969) and PSL (1975) suggests they are more incomplete than they are complete, particularly with respect to the 'devilish details'. Hopefully, user organisations will apply this technique to their own documentation standards—the author would be pleased to hear of extra technical facts that have been overlooked herein.

Finally, the author acknowledges the assistance of his colleagues in the LSE Systems Research Group and at Birkbeck College.

Data Dictionary				*Messages*				*Data base*				*Procedures*			
Fact	ADS	NCC	PSL	Fact	ADS	NCC	PSL	Fact	ADS	NCC	PSL	Fact	ADS	NCC	PSL
①	√	√	√	⑰	√	√	√	㊵	√	√	√	㊽	√	√	√
②	√	√	√	⑱	×	√	√	㊶	×	×	×	㊾	√	√	√
③	×	√	√	⑲	×	√	√	㊷	×	√	×	㊿	√	×	×
④	×	×	√	⑳	×	×	√	㊸	×	√	×	(61)	×	×	×
⑤	×	√	×	㉑	×	×	√	㊹	×	×	×	(62)	√	×	×
⑥	√	√	√	㉒	×	×	×	㊺	×	×	√	63	×	×	√
⑦	√	√	√	㉓	×	×	√	㊻	×	×	×	64	×	×	√
⑧	×	×	√	㉔	×	×	×	㊼	×	×	×	65	×	√	×
9	×	×	√	㉕	×	×	×	48	√	√	×	66	×	√	×
⑩	√	√	×	26	√	√	√	㊾	×	√	×	67	×	×	×
⑪	×	×	×	㉗	×	×	×	50	√	√	√	68	×	×	×
⑫	×	√	√	28	√	√	×	51	×	×	×	69	×	×	×
13	×	×	×	29	×	×	×	52	×	×	×	70	×	×	×
14	×	√	√	㉚	×	×	×	53	×	×	×	71	√	×	√
15	×	×	×	31	×	×	×	54	×	×	×	72	×	×	×
16	×	×	×	㉜	×	×	×	55	×	×	×	73	×	×	×
				㉝	√	×	×	56	×	√	×	74	×	×	×
				34	×	×	×	57	×	×	×	75	×	×	×
				㉟	×	×	×					76	×	×	×
				㊱	×	×	×					㊼	×	×	×
				37	×	×	×								
				㊳	√	√	×								
				㊴	√	√	×								

Fig. 2 Feature analysis of three specification techniques.

Appendix

Data Dictionary

① *Element names*

Each element must be uniquely named so that procedures can specify the variables on which they operate.

② *Set names*

Similarly, convenient groupings of elements into sets must also be uniquely named (e.g. INVOICE, CUSTOMER-RECORD).

③ *Synonyms*

If different people call the same element or set by different names, then synonyms are necessary to identify its unique name with its variations.

④ *States*

It is often convenient to name the states that an element or set can take and define the conditions for which each state applies (e.g. MALE is SEX = 1, FEMALE otherwise).

⑤ *Levels*

Level numbers are used to specify the depth of a set within their hierarchical structures (Waters; 1977).

⑥ *Single keys*

Each element must be uniquely identified (Grindley; 1975) by a key so that procedures can join (Codd; 1970) the occurrences of the variables on which they operate (e.g. PRODUCT-CODE is the key of QUANTITY-IN-STOCK); if the key is itself a single element then it is termed a single key.

⑦ *Composite keys*

Otherwise, the key is termed composite (e.g. PRODUCT-CODE/RESOURCE-CODE is the key of USAGE-QUANTITY).

⑧ *Alternate keys*

An element or set may be uniquely identified by more than one key (e.g. FACTORY-NUMBER/DEPARTMENT-NUMBER/CLOCK-NUMBER and NATIONAL-HEALTH-INSURANCE-NUMBER are both keys of EMPLOYEE-PAY-RATE).

9. *Key relations*

One key may have an *m*:*n* relationship (Davenport; 1977), with another key and these relations dictate access paths through all data (e.g. *m* EMPLOYEE-NUMBERs work in one DEPARTMENT-NUMBER and no other).

⑩ *Pictures*

The format and character set of each element must be specified (e.g. DATE is 99AAA99).

⑪ *Units*

The units of measurement of numeric elements, that are not simply decimal numbers, contribute to calculating logic and must be specified (e.g. SALES-PRICE is measured in pence per metre).

⑫ *Value ranges*

The minimum and maximum values of each numeric element contribute to data validation logic (e.g. SALES-PRICE is between 10p and 99p).

13. *Normal values*

The normal (or nodal) value of an element is often useful in estimating (e.g. SALES-PRICE is usually 15p).

14. *Actual values*

Sometimes, all actual values of an element must be specified (e.g. PRODUCT-CODE is 0007, 0013, . . . , or 9972 may be useful when choosing a data base algorithm).

15. *Value distributions*

Also, the number of occurrences for each actual value may be useful (e.g. SEX = 0 occurs 2,000 times and SEX = 1 occurs 8,000 times may be relevant in data base inversion).

16. *Value relations*

Further, relationships between the values of different elements may be stressed (e.g. if CHAIN-STORE-CODE ≠ 0 then PRIORITY = 1).

Messages

(17) *Contents*

Each message must be defined in terms of the elements and sets it contains.

(18) *Locations*

The source of each input message and destination of each output message must be specified, particularly if they are to be teleprocessed.

(19) *Replications*

The number of copies of each message must be specified, particularly

(20) *Control replications* whereby several versions of the same message may be compared for accuracy, and

(21) *Recovery replications* whereby copies of messages are necessary to recover from failures.

(22) *Replications by locations*

Further, the number of copies of a particular message can vary between locations.

(23) *Frequencies*

The arrival times of input messages and the departure times of output messages must be defined (e.g. ORDERS are daily, STATEMENTS are monthly), noting that these frequencies may vary between

(24) *Frequencies by time*—different periods, and

(25) *Frequencies by locations*—different sources/destinations.

26. *Normal occurrences*

The normal (or nodal) number of messages input/output is necessary for estimating the traffic and workload of a system (e.g. 5,000 ORDERS per day).

(27) *Minimum occurrences*

The minimum number of times a particular message can occur vitally affects the logic of a system—probably most messages have a minimum occurrence of zero to cover extreme circumstances (e.g. no ORDERS per day during a postal strike).

28. *Maximum occurrences*

The maximum number of times each message occurs is useful for estimating peak traffic and workload.

29. *Occurrences by time*

The number of messages may vary between different periods.

(30) *Occurrences by keys*

The number of messages that can occur for a particular key usually has a profound impact on the logic of a system but is often overlooked (e.g. can several ORDERS be input for the same CUSTOMER-CODE and can several ORDER-LINEs refer to the same PRODUCT-CODE? If so, what special procedures are necessary?)

31. *Occurrences by locations*

The number of messages may vary between different sources/destinations.

(32) *Message type sequences*

Different types of input message may arrive in a particular order (e.g. INSERTIONS, AMENDMENTS and DELETIONS precede all TRANSACTIONS); similarly, different types of output messages must usually depart in a defined order (e.g. RECONCILIATIONS precede all RESULTS).

(33) *Message sequences*

Occasionally, all input messages of the same type arrive in strict order (e.g. CLOCK-CARDS are in EMPLOYEE-NUMBER sequence); output messages usually depart in strict order, if only to help locate those that are subsequently queried (e.g. PAYSLIPs are output in FACTORY-NUMBER/DEPARTMENT-NUMBER/EMPLOYEE-NUMBER sequence).

34. *Near sequences*

Input messages sometimes arrive almost in order (e.g. ORDER-LINEs are usually in PRODUCT-CODE sequence for each ORDER but not always).

(35) *Sequences by locations*

The order of messages can vary between different sources/destinations, and

(36) *Sequences by replications* different copies of the messages.

37. *Error rates*

The proportions of incorrect messages are necessary for estimating.

(38) *Media*

The devices supporting input and output messages must be specified.

(39) *Formats*

So must their precise layouts.

Data base

(40) *Contents*

Each logical, data base record must be defined in terms of the elements and sets it contains.

(41) *Locations*

The sites of data base records must be specified, particularly if they are to be distributed.

(42) *Control replications*

Data may be replicated so different versions can be compared for accuracy, or

(43) *Recovery replications* can be quickly recovered from failures.

(44) *Replications by locations*

Further, the number of copies may vary between sites.

(45) *Frequencies*

The times at which the data base is updated must be defined, noting that these may vary between

(46) *Frequencies by time*—different periods, and

(47) *Frequencies by locations*—different sites.

48. *Normal occurrences*

The normal (or nodal) number of logical, data base records is necessary for estimating its size.

(49) *Minimum occurrences*

The minimum number of times a particular type of data base

record can occur affects the logic of a system—often, this is zero for the case of the very first run of a system.

50. *Maximum occurrences*
The maximum number of times each data base record occurs is useful for estimating peak sizes.

51. *Occurrences by time*
The number of data base records may vary between different periods, and

52. *Occurrences by locations*—different sites.

53. *Hit ratios*
The proportion of data base records that are accessed.

54. *Hit groups*
Highly-active areas of the data base.

55. *Fan in/out ratios*
The number of times an accessed data base record is hit by input/output messages.

56. *Volatilities*
Data base growth and/or decay.

57. *Overflow patterns*
Distributions of inserted data base records.

Procedures

(58) *Contents*
Each procedure must be defined in terms of the statements it contains, for making decisions and taking appropriate actions.

(59) *Names*
Procedures and individual statements may be uniquely named (e.g. labels); each statement may perform a function for

(60) *Create verbs*—generating a value of an element (e.g. including $+$, $-$, $\times$, $\div$, Σ, Π, roots, powers, sine, cosine, tangent, etc.) or an occurrence of a set (e.g. a data base record or output message),

(61) *Destroy verbs*—eliminating an occurrence of a set (e.g. an invalid input message or a deleted data base record), or

(62) *Test verbs*—deciding which actions should be taken (e.g. including $<$, $=$, $>$, etc.).

63. *Frequencies*
The times at which each procedure is operated may be stressed.

64. *Keys*
Further, the frequency of amending an element of one data base record which is a key for another record deserves emphasis (e.g. SALESMAN-CODE on CUSTOMER-RECORD can only be changed at the end of a month).

65. *Sequence keys*
The records of a physical data base are usually ordered on sequence keys (which include record keys) and their frequency of change may also be significant (e.g. in preserving the sequence of a large serial file); further, it may also be important to specify if

66. *Sequence key series*—the new values of sequence keys are always greater than their old values, or

67. *Sequence key cycles*—the values of sequence keys may be swapped.

68. *Strategic delays*
Many delays may be incorporated in a physical system purely as technical strategy (e.g. reallocated stocks will not be be returned to free stock until the day after they were originally allocated).

69. *Frequencies by time*
The times at which a procedure is operated may vary between different periods, or

70. *Frequencies by locations*—different sites.

71. *Normal occurrences*
Estimating processing time may require that the number of times a statement or procedure is operated be defined as a normal (or nodal) value,

72. *Minimum occurrences*—a minimum value, and

73. *Maximum occurrences*—a maximum value.

74. *Occurrences by time*
These occurrences may vary between different periods,

75. *Occurrences by keys*—different keys, and

76. *Occurrences by locations*—different sites.

(77) *Sequences*
Sometimes, procedures must be operated in a defined sequence and this can have a profound impact on the logic of a system (e.g. stocks must be allocated first in—first out for outstanding orders before today's orders in customer priority sequence).

References

ADS (1969). *A Study Guide for Accurately Defined Systems*, NCR Ltd.
LEAVENWORTH, B. M. (1977). BDL—Non-procedural Data Processing, *The Computer Journal*, Vol. 20 No. 1, pp. 6-9.
BOSAK, R. *et al* (1962). An Information Algebra, *CACM*, Vol. 5 No. 4.
CODD, E. (1970). A Relational Model of Data for Large Shared Data Banks, *CACM*, Vol. 13 No. 6.
DAVENPORT, R. A. (1977). Data Analysis for Database Design, *Proceeedings of IRIA Conference on Data Analysis and Informatics*, Paris.
GRINDLEY, C. (1975). *Systematics*, McGraw-Hill.
MARTIN, J. (1967). *Design of Real-Time Computer Systems*, Prentice-Hall.
NCC (1969). *NCC Systems Documentation Standard*, National Computing Centre Ltd., Manchester.
PENGILLY, P. J. (1976). An Approach to Systems Design, *The Computer Journal*, Vol. 19 No. 1, pp. 8-12.
PSL (1975). PSL Users' Manual, ISDOS Working Paper 98, University of Michigan, USA.
TEICHROEW, D. and HERSHEY, E. (1977). PSL/PSA—A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems, *IEEE Transactions on Software Engineering*.
WATERS, S. J. (1977). CAMO2: A Structured Precedence Analyser, *The Computer Journal*, Vol. 20 No. 1, pp. 2-5.
WATERS, S. J. (1979). *Systems Specifications; Documentation, Standards and Languages*, National Computing Centre Ltd., Manchester.