A survey of languages for stating requirements for computer-based information systems*

by DANIEL TEICHROEW

The University of Michigan
Ann Arbor, Michigan

BUILDING COMPUTER BASED INFORMATION SYSTEMS

Society depends more and more on the recording, analysis, storage, processing, and transmission of data and information. Practically every activity requires an information system. The larger and more organized the activity, the larger and more organized is the information system which serves it. This paper is concerned with Information Processing Systems (IPS) which are built to aid the management and operation of an organization. In particular, the paper is concerned with the methods by which the information needs of the organization can be communicated effectively to those who are asked to implement systems to satisfy the requirements for planning, control, and operations.

The size and complexity of society makes it impractical for a manager or other user personally to satisfy his own information needs, and therefore several functions have evolved with the growth in the use of the computer:

Analysis: Frequently, this term is used with an adjective such as systems, management, or business. The objective of the analysis is to determine, and record, the information needs of the organization and the individuals in it.

Design: The purpose of design is to select the best method of meeting information needs. Since there are usually a number of alternatives available in hardware, software, and processing organization, and since making changes once construction has begun is difficult, it is crucial to design the system as completely as possible before beginning the construction.

Construction: This function consists of building and assembing the modules selected in the design. It includes programming, file construction, hardware acquisition and development of the necessary non-computerized procedures.

In practice, the number of individuals involved in these functions becomes large and some organization is required. One common method is that of a project team which accomplishes all three functions. Another common method is to assign the three functions to separate departments and pass a particular problem from one department to the next, e.g., from analysis to design to construction. (Detailed discussions of the systems building process in use today are available in many papers and books.[1,2,3])

Regardless of whether the project team or functional organization is used, it is of course desirable to document as completely and precisely as possible at each step. The chain of steps of analysis, design, and construction, is only as strong as its weakest link and in practice the chain falls apart first in the lack of adequate documentation from one step to the next.

PRESENT METHODS OF DOCUMENTING REQUIREMENTS

Overview of present methods

The purpose of an IPS, or any group of them, is to serve the organization, and therefore any discussion of the use of the computer must start from the objectives of the organization and the means that its owners and

* This work was supported in part by the ISDOS Research Project, Department of Industrial and Operations Engineering, University of Michigan, and by the U. S. Army under Research Grant DAHC 19-71-G-0005. An earlier version of this paper was published as "Problem Statement Languages in MIS," in E. Grochla (ed.), *Management-Informations-Systeme*, Band 14, Schriftenreihe Betriebswirtschaftliche Beiträge zur Organisation und Automation, pp. 252-282, Betriebswirtschaftlicher Verlag, Wiesbaden, 1971.

managers have chosen to achieve the objectives. As is well-known, it is quite difficult to bridge the gap between the managers and their chosen methods of operating the organization and the precise statements necessary to get computers to do the data processing. There are several major reasons for this difficulty.

First, the organizations are very large and complex and it is not easy for individuals, or groups of individuals, to comprehend all of the interrelationships to the detailed level required for computer processing. Second, the organization has a number of activities going on in parallel and it is difficult to describe everything in a "serial" fashion as is necessary for today's computers. Furthermore, there is no good language for communicating requirements that is understandable by both management and the computer.

This paper concentrates on the techniques by which needs are documented and transferred from the first to the second step, i.e., from analysis to design. The paper is not specifically concerned with the process in the first step, namely, the determination of what the information requirements should be.

Methods for reducing problems associated with documentation of requirements

There have been various attempts to reduce the documentation problem by "shortening" the distance between the person in the organization who needs the information (the user) and the computer. Some methods are listed here in order of the amount of detailed documentation the user must supply, directly or indirectly, from very little to a great deal.

Turning the problem over to another organization

One intuitively appealing approach is for the organization to contract with another for all of its information needs. This has become known as "installation management" or "facilities management." There is not yet enough experience to indicate how successful this will be but in any case, it merely transfers the problem of documentation to another organization. There is certainly more opportunity for this firm to develop expertise in documentation and in fact the absolute necessity of legal, contractual agreements should lead to formal documentation of requirements.

Generalized software packages

In this approach all that is required of the user is to select the package that is appropriate to his needs and to supply the values of the appropriate parameters. Generalized packages[4] are basically of two kinds—application dependent packages and application independent packages. Application dependent packages are generalized programs for performing specific applications such as billing, payroll, accounting, banking and engineering. Application independent packages include report generation and file maintenance, operating system enhancement, simulators and performance monitors, and programming aids. Generalized packages have had only limited success and account for only a small part of the total software development. A recent report[5] estimates the 1972 revenue to be $90 million for applications packages and $110 million for application independent packages. Major interest currently centers on what is probably the most sophisticated example of this approach, the data base management systems,[6] some of which are controlled by parameter values entered on forms or questionnaires—the most widely used example in this category is MARK IV.[6] (Other data base management systems are controlled by task definition and data definition languages.) An example of where user requirements can be stated on forms and directly translated to object code is the Applications Customizer used for the IBM System/3.[7,8]

User-oriented languages

This approach differs from that of generalized software packages in that the user supplies statements rather than parameter values. A user-oriented language is one in which the statements are intuitive and understandable to the user. In the case where the users are managers, the most frequently proposed languages are subsets of English. A number of such languages are in existence but their use appears to be limited to special situations. An example of a user-oriented language intended for management information systems is MUSE.[9]

"Conceptual" frameworks

In these systems the basic framework is provided by the language and is available in a package. This must be supplemented by additional programs unique to the particular situation. An example of this approach is the SIMSCRIPT system for simulation and MAST[10] for business data processing. The user must select the appropriate system and then state his own unique needs usually at the level that permits a program to be written. In practice this approach requires that the user describe his requirements to an analyst or programmer rather than using the language himself.

"Block" system

"Basic Functions" or "Primitives" are defined and usually implemented as macros or sub-routines. The user must then assemble these blocks to satisfy his needs. An example is the BEST system;[11,12,13] several general descriptions exist.[14,15,16] While in theory this approach permits a user to state his needs without a programmer, in practice these systems are used by a programmer. Even for this use, however, these systems to date have received only limited acceptance.

General purpose programming languages (GPPL)

The general purpose programming languages, COBOL, FORTRAN, and PL/1 currently are the most widely used method for building information systems. This category also includes assembly languages which are used whenever optimum use of hardware capabilities is paramount. These languages, of course, require that the user obtain the services of a programmer to implement his information needs.

Relative importance of different approaches

While the above listing has not been supported by quantitative data on relative usage, there are few who would contest the conclusion that by far the largest amount of effort in system building today is based on the use of general purpose programming languages and that undoubtedly this will continue to be true for the foreseeable future. Packages that accomplish "data processing tasks," particularly those now commonly referred to as data base management systems, will come into wider use, and while their use will reduce the amount of programming that would otherwise have to be done, a very large fraction of the total system building will continue to depend on the use of general purpose programming languages. It is therefore worthwhile to examine the system building process based on the use of general purpose programming languages and particularly the inherent problems of communicating between the persons who need the outputs from the system to be constructed and the first automaton in the sequence, namely the compiler. In order to describe these problems it is necessary to make a basic distinction between requirements that an IPS is to satisfy and the processing procedures that will be used to obtain the desired results.

Distinction between information requirements and processing procedures

At the heart of the problem of system building lies the distinction between stating information needs and developing processing procedures that are to be used to satisfy them using the technology of computer-based information processing systems available today. This is a particular instance of the very general concept of goals-means chains. One starts with a goal, lists the various means that could be used to achieve the goal and selects one which then becomes the goal; then the possible means to achieve this goal are listed, one is selected, and so on.

As an example, suppose one is at point A and has a goal of getting to point B. The possible means may be walking, taking a bus, taking a taxi, etc. Assume the taxi method is selected. The goal of getting to point B is communicated to the taxi driver and he selects the means, e.g., the route, etc. Sometimes the passenger will tell the driver the route rather than the destination and sometimes the driver will question the goal (the passenger should go to C instead of B). In general, these actions will be undesirable; in the first case, because the driver presumably knows more about which route is best and in the second because the passenger knows better where he wants to go. This analogy is relevant to the system building situation because ideally the user, or his analyst, should determine the goals of the computer-based system and the system designer and programmer should then select the best method of implementation. All too often, unfortunately, the analyst worries about the best computer means (e.g., the best file structure and record layout) and the programmer worries about the goal (e.g., is this report really needed?). Consequently, both the analyst and the programmer do poor jobs and the resulting system is not effective.

Satisfying the information needs of organization can be represented by a goals-means chain, usually of several stages. The distinction between requirements and procedures at the level immediately before the physical systems design and programming can be illustrated by a simple payroll processing example. (In this simple example, the statement of requirements is represented by one and only one stage. In more realistic examples, several stages of goals-means analysis may be required). The task of the person specifying the needs, i.e., the problem definer, is to describe the requirements for the "target" system which will produce one output: employee paychecks. Certain input information will be available to the target system and the required output type and format is known. These are shown in Figure 1.

In this example it is assumed that the purpose of the

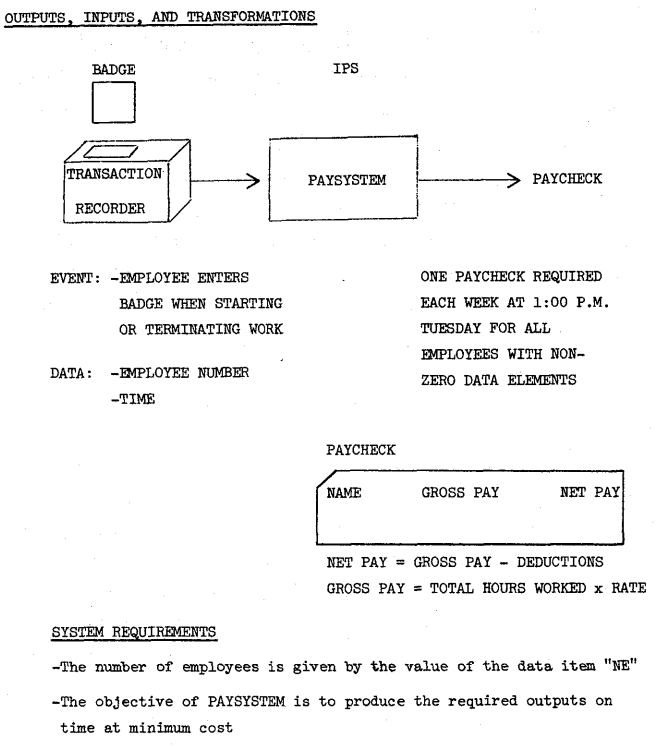

Figure 1—Statement of requirements for the IPS called PAYSYSTEM

target IPS called PAYSYSTEM is to produce one output called PAYCHECK. The time the outputs are to be available is given: each week at 1:00 p.m. on Tuesday for the previous week's work. The number of outputs is specified by saying that one PAYCHECK is required for each employee for whom at least one of the data items included in PAYCHECK other than NAME is different from zero.

The form of the output is stated to be a document containing three data elements: NAME, GROSS PAY, NET PAY. Formulas for computing GROSS PAY and NET PAY are given. PAYSYSTEM must accept one input called EVENT which occurs whenever an employee enters his badge into a transaction recorder. When this occurs the EMPLOYEE NUMBER and TIME are recorded.

Some additional information is given: the number of employees is given by the value of data item NE, and the objective of the IPS is to produce the outputs at minimum cost.

The type of information mentioned above, and shown in Figure 1, is representative of what is necessary to describe the requirements and is sufficient for the purpose of this example, although in a real situation much additional information would have to be specified. For example, Where does the value of NAME come from and how is it associated with the value of EMPLOYEE NUMBER?, How is the value of TOTAL HOURS WORKED determined from TIME?, Where does the value of DEDUCTIONS come from? Additional inputs have to be defined, e.g., to add a new employee to the set of valid EMPLOYEE NUMBERs and to supply changes to the value of RATE and DEDUCTIONS.

All of this merely corroborates what everyone already knows, namely that stating all the requirements for an organization can be a tedious process. Unfortunately tedium is all too frequently avoided by omitting details that are thought to be obvious and leaving them to the programmer to fill in later.

Under any method of system building the type of information illustrated by Figure 1 has to be collected. Usually this is done manually and is recorded using a natural language (English) augmented by tabular or graphic methods such as decision tables and flow charts. Sometimes an attempt is made to follow the company manual that prescribes standards for documentation.

The deficiencies of manual documentation systems based on the use of natural languages have been analyzed in detail elsewhere[17] and it is sufficient to merely summarize the major shortcomings. While English is a good language for communicating qualitative information, it is too ambiguous for quantitative relationships. The addition of tables, flow charts, decision tables helps a little but major difficulties still remain. The methods are too imprecise, e.g., different data names may be used causing confusion and incorrect results. Manual documentation cannot cope with changes. In large systems the documentation becomes too costly and it absorbs too large a share of total resources. Finally, manual documentation methods based on natural languages cannot be automated efficiently.

In the design process the system designer might follow the procedure outlined in Figure 2. First he will determine the hardware that will be used. Sometimes there is only one alternative, in some other situation this step may require considerable effort. Next he will choose the hard software through which the IPS will be constructed and also through which it will be operated. As one aspect of this process the system designer should consider which of the methods outlined in the previous subsection should be used. In most cases today the method of construction will be through general purpose programming languages perhaps supplemented by data base management systems.

Then the system designer decides what files are needed and what information they should contain. In this case, assume that he has decided that there will be a file called EMPFILE and that it will be stored on a

HARDWARE SELECTION

HARD SOFTWARE SELECTION

FILES AND FILE ORGANIZATION

EMPFILE: ONE RECORD FOR EACH EMPLOYEE ON RANDOM ACESS DEVICE

QUEUE: A LIST OF EVENTS WAITING TO BE PROCESSED

PROCESSING PROCEDURE	PROGRAM
1. BUILD UP QUEUE FOR A WEEK	X
SORT	X
UPDATE EMPFILE AND PRODUCE PAYCHECK	X
2. BUILD UP QUEUE FOR DAY	X
SORT	X
MERGE AT END OF WEEK	X
UPDATE EMPFILE AND PRODUCE PAYCHECK	X
3. UPDATE EMPFILE FOR EACH EVENT	X
PRODUCE PAYCHECKS AT END OF WEEK	X

Figure 2—Physical system design

random access unit with one record for each employee and that another file called QUEUE will contain all the events waiting to be processed.

Next, he makes a list of alternative processing procedures (perhaps mentally) and chooses the one which seems to be the best. In this case he might consider letting QUEUE build up for a week (since the output is needed only once a week), sort by EMPLOYEE NUMBER at the end of the week and then update EMPFILE and produce the outputs at the end of the week. In alternative two QUEUE would be built up each day, sorted each day and merged at the end of the week. As alternative three, he might consider updating EMPFILE for each EVENT as it occurred and then producing the output in a weekly run.

The alternative the system designer would choose should be based on the objective stated in requirements. This is frequently a difficult step and may involve much effort if done completely. In this case he may choose alternative three since it is the simplest in concept. However, if he is concerned with processing time he might choose alternative one because it will require less computer time than alternative three. Once the alternative has been chosen the designer then divides the system into parts. Specifications for the various parts of the system are prepared and given to a programmer to write the programs.

The system building process as described above, and as conducted today, depends on manual documentation through the analysis and design phases. Formal documentation begins when the programmer expresses specifications furnished by the system designer in a general purpose programming language. While the basic purpose of this paper is to compare languages for documentation during the analysis phase it is necessary to clearly document why general purpose programming languages are not satisfactory for this purpose, if for no other reason than to dispel the myth, still far too widely believed, that they are.

System building using General Purpose Programming Languages (GPPL's)

While there are methods for causing computers to produce needed output which do not depend directly on GPPL's, it was concluded above that much of the system building in the future will be based on the use of GPPL's or their immediate extensions.

In practice GPPL's are involved in system building only in the construction phase, as shown in Figure 3. The programmer produces source statements which are turned into object code by a compiler. The use of general purpose programming languages causes problems in systems building because by default they frequently are the documentation for the earlier phases. Throughout this discussion COBOL will be used as the example of GPPL since it is now the most widely used in building organizational systems. (The basic arguments, however, are just as valid for the others: FORTRAN, PL/1, and ALGOL, etc.

When COBOL was first developed, it was claimed to have the advantages of being self-documenting and hardware independent. While in most cases it is undoubtedly better to use COBOL than an assembly language, the limitations of COBOL for organizational users of computers are becoming more and more evident: COBOL is a second generation language; it forces the intermixing of business specifications and data processing functions; it results in freezing procedures in the programs; it is not a satisfactory method for communication of information needs; and its use limits the number and size of systems that can be built. The effect of each is discussed further below.

Figure 3—Use of GPPL in systems building

"Second" generation hardware

COBOL was designed to be compiled on second generation hardware. It was developed using experience of another general purpose language, FORTRAN, which was initially designed for first generation hardware; the changes were primarily intended to make COBOL suitable for business data processing, as opposed to numerical calculations. Since COBOL was developed for second generation hardware, it has no facility for dealing with hardware capabilities that are generally available in third generation, but not in second generation, hardware. A program written in COBOL cannot make effective use of random access devices, for example, without some extensions either in the language or in the addition of another language, such as the command language to communicate with operating systems.

COBOL programs are more or less hardware independent of hardware capabilities from one generation to another. The result, however, is performance that is not hardware independent when hardware capabilities change. One immediate consequence of this is that once requirements are implemented in COBOL, the programs must be redone for the next generation, otherwise the result is merely emulation.

As a requirements statement language, COBOL is also limited because much of the information of the type illustrated in Figure 1 cannot be represented. For example, there is no provision for stating that outputs are needed at a certain time or for stating the number of outputs that will be needed.

Intermixing of business specifications and processing procedures

The use of the COBOL language forces the intermixing of the definition of information needs, here called business data specification functions, and the procedures chosen to satisfy the needs, here called data processing functions.

—Business Data Specification Functions (BDSF) define the data manipulation and calculation that must be accomplished to satisfy requirements. Usually these are formulas that define the value for one or more variables, e.g., a BDSF may be "expected sales in a given region, in a given period for a given class of products" or "the total value of inventory at a given time." BDSF are part of the statement of information requirements; in the example in Figure 1, the BDSF are:

NET PAY = GROSS PAY − DEDUCTIONS
GROSS PAY = TOTAL HOURS WORKED * RATE

In many cases, there may be several ways to define a business function. For example, "inventory value" may be defined using the First In-First Out or First In-Last Out method. It is the responsibility of the user to state the exact definitions he wishes to have used. The BDSF are independent of the particular computer implementation that is used to perform the computation.

—Data Processing Functions (DPF) are the operations that must be used in any particular implementation in order to accomplish the actual computation of the values of the business data specification functions at the time they are needed. For example, in order to (eventually) compute "inventory value," data values such as quantity and price must first be stored somewhere. Then they must be retrieved, multiplied, and summed. Other BDSF may use one or both of quantity and price, and hence, it may be better from a data processing point of view to combine several of these requirements. The DPF used are dependent on the particular hardware and processing procedures selected for their implementation. Data processing functions are selected during the design process; in the example in Figure 2 some of the DPF's used are:

SORT, MERGE, UPDATE

It is essential to be very precise in distinguishing between BDSF and DPF. For example, the user may specify that the IPS in Figure 1 is to produce PAYCHECK alphabetically by NAME. This is different from saying, SORT by NAME. SORT is a DPF which may or may not have to be performed depending on other system design decisions.

The use of COBOL results in a program containing both the BDSF's such as

NET PAY = GROSS PAY − DEDUCTIONS

and the DPF's that the systems designer has selected, e.g.,

SORT

Since both BDSF and DPF are intermixed, usually in relatively complicated ways, it is difficult for programmers to separate out the statements which implement the BDSF from those which implement DPF and it is certainly impossible for a computer program or a user to do so. COBOL designers recognized the necessity to separate data descriptions from the statements in the procedure division. It is now necessary to go one step further and separate BDSF from DPF.

Most organizations are now trying to develop data directories and data bases on an integrated basis for as

large a part of the organization as possible in order to avoid duplication and permit comprehensive analysis. In the same way, organizations in the future can be expected to develop directories of BDSF so that they can have standard definitions that can be specified once and used whenever needed.

Freezing processing procedures in programs

One of the consequences of intermixing BDSF and DPF is that the processes are frozen into the program. The programmer expresses the means that the system designer has selected which then become goals to the compiler. The language forces the programmer to specify processing at the level of locating, reading, and writing records and operations on individual data items (PL/1 permits some operations on arrays, i.e., matrices). Therefore, once a program is written units of data cannot be changed without changing the programs. In general, whenever the processing procedures are changed because of changes in hardware, volume of processing, etc., it is necessary to reprogram.

GPPL's as documentation for communicating information needs

When general purpose programming languages were first considered for business problems it was expected that the language being developed, COBOL, could be used for the documentation of information needs. However, this has not happened, as is well stated by Weinberg:[18]

> *"Some years ago, when COBOL was the great white programming hope, one heard much talk of the possibility of executives being able to read programs. With the perspective of time, we can see that this claim was merely intended to attract the funds of executives who hoped to free themselves from bondage to their programmers. Nobody can seriously have believed that executives could read programs."*[18]

COBOL programs are not satisfactory as a communication medium between the user and programmer precisely because they must contain the DPF's. Much of what a user reads when he tries to read a COBOL program is not of interest to him.

Programmer productivity

The amount a programmer can write in COBOL in any given time is limited. Programmer productivity is measured in terms of statements per day—from five to twenty-five. There are not enough programmers to write all the programs that are needed. The reasons for this rate of productivity are partly the difficulties caused by lack of adequate documentation of requirements and partly the fact that the DPFs are programmed many times.

Improvements and extensions to General Purpose Programming Languages

The limitations inherent in the GPPL's listed above are, of course, well-known and a number of attempts to improve or extend COBOL have been made. These need to be listed to examine whether an extended language could eliminate the need for a new requirements language.

"Larger" verbs such as SORT and REPORT WRITER have been embedded in COBOL so that the program is easier to understand and requires less programmer time. Facilities have been added to COBOL to make it possible to use the capabilities of third generation hardware. For example, one manufacturer added IDS to COBOL to make it possible to use random access devices efficiently. Operating Systems and Job Control Languages have been developed to interface the programs and the machines with new capabilities.

These efforts, and the efforts to build data base management systems, are necessary in order to use the present day machines to solve today's problems. However, it is unlikely that such incremental improvements will be sufficient, just as it is doubtful that continued incremental improvement in assembler languages would ever have led to FORTRAN because of the limitations inherent in assembler languages. Similarly, the present effort to solve the problems of adequately documenting information needs by building data base management systems starts by accepting some current features which will, in the long run, limit the effectiveness of the approach.

Need for a requirement statement language

What is required to overcome the difficulties cited above is a formal method of communicating information needs. It must be able to express needs of the type exemplified by Figure 1 without implying any data processing functions of the type selected in the design process exemplified by Figure 2. The analysis in this section has been directed toward showing that general purpose programming languages and their extensions are not suitable for this purpose. The next section will describe a number of languages that have been proposed.

A set of detailed specifications for an ideal "requirements statement language" will then be given.

COMPARISON OF REQUIREMENTS STATEMENT TECHNIQUES

Survey of techniques

The need for a more formal method of documenting requirements for information during the analysis phase has long been recognized. A number of techniques have been proposed. Some of these are listed in Table I.[19–53] Undoubtedly this list is not complete but it includes the known techniques that state as their objective the formalization of statement of requirements or include "analysis" in their title. (Information on omitted techniques is welcome.)

TABLE I—Systems Analysis and Requirements Statement Techniques

Acronym	References	Developer	Status
ADS	19,20	National Cash Register Co.	In use
ASYST	21	Miles Hudson	In development
AUTOSATE	22,23	Rand Corporation	Inactive
CADIS	24,25	J. Bubenko	
CAMIL	26	S. Waters	In development
CASCADE	27,25	Arne Solvberg	In development
CODIL	28	C. F. Reynolds	In development
CORIG		Not known	
DATAFLOW	29,30,31	National Computing Centre	Inactive
IA	32,33	CODASYL Committee	Inactive
	34	H. B. Ladd, W. P. Marcovic	Inactive
LA	35,36,37	B. Langefors	In development
LO	38,39	Lionelle Lombardi	
MINOS		CEGOS	Not known
PSL	40	ISDOS Project	In development
SCOTT	41	SDI Associates	In use
SYMOB	42	Bull, France	Not known
SYNGE		Not known	Not known
SPEC	43	Englic Electric	Inactive
SSP		Robert Brass	In development
SYSTEMATICS	44,45,46,47, 48,49	C. B. B. Grindley	In development
TAG	50,51	IBM	In use
	52	Taggart	Not known
UCS		Phillips	In development
YK	53	Young and Kent	Inactive

The basic criteria used for inclusion in Table I were as follows:

i. The language must be designed to state information needs to the system designer and programmer, i.e., it must not permit Data Processing Function statements.
ii. There must be some attempt to develop a formal syntax sufficient to permit analysis by computer programs if desired.
iii. There must be a reasonably detailed description of the language available in the published literature.

These criteria eliminate a number of languages. In particular, all the languages and techniques mentioned in the second section of this paper are not considered further. The second criterion eliminates the (manual) documentation techniques that are part of most system building procedures.[1,2,3] The language developed by Bosak[54] is not included because it is a file processing language rather than a problem statement language. The output decompositions method (ODM)[55] and simulators[56,57] are not included because they are primarily design techniques though they require a statement of requirements in a structured form as input. Programming languages such as APL,[58] Dataless Programming[59] and BCL[60] are eliminated under the first criterion since they require statements describing data processing functions. Software packages concerned with only parts of the information needs such as LEXICON[61] for data definition will be analyzed separately in a later paper.

Previous surveys of some of this literature (and of other related languages) are given by Young,[62] Shaw,[63] Head[7] and in the discussion and proceedings of two workshops.[64,65,25] Shaw's survey includes Information Algebra,[32] Lombardi's Algebraic Data System,[39] Iverson's language,[58] BEST,[11] as well as Decision Tables, IDS, LUCID, ADAM, COLINGO and ATS, which are not included in this paper. Young[62] surveyed BEST,[11] Decision Tables, Lombardi's Algebraic Data System,[39] Iverson's language,[58] Information Algebra,[32] and Young's and Kent's Abstract Formulation.[53] Information Algebra is also discussed by Sammet[66] in the chapter on "Significant Unimplemented Concepts."

All of these techniques have in common the attempt to bridge the communication gap between the Analysis and the Design phases shown in Figure 3. However, a detailed analysis and comparison of all of these proposals is clearly not feasible in this paper. Therefore, a few techniques have been selected for more detailed

examination. All these techniques satisfy one or more of the following: they are in current use, they represent areas for further improvement and development or they add to the understanding of the historical development.

Most of the analysis in this section is based on seven selected techniques. The earliest is the work by Young and Kent (YK).[53] Information Algebra (IA) is the result of work by the CODASYL Development Committee.[32] Langefors (LA) published several papers in BIT,[35,36] which have been incorporated into a book.[37] This work is being continued by a number of projects in Scandinavia.[25] Lombardi's Algebraic Data System (LO) was published in COMMUNICATIONS OF THE ACM.[32] Accurately Defined Systems (ADS) was developed by the National Cash Register Company.[19,20] TAG was initially developed by Myers[50] and is described in Reference 51. Grindley has published several papers describing SYSTEMATICS (SY).[44–49]

All these seven approaches are concerned with the problem definition phase of IPS building and hence satisfy the first criterion:

IA: "The goal of this work is to arrive at a proper structure for a machine-independent problem-defining language at the systems level of data processing."

LA: "A formal method for performing systems analysis of information systems in business and elsewhere is needed in order to save systems work and programming and to obtain better systems."

YK: "There are three stages in the application of high speed digital computers to data processing problems:

i. Systems analysis—the task of determining what is to be done.
ii. Programming—a statement of how it is done.
iii. Coding—a translation of this statement into machine language.

This paper presents a first step in the direction of automatic programming as well as a tool which should be useful in systems analysis."

LO: "[The language] relies exclusively on non-procedural representation of processes as sets (tables) of relations between data and results (there are no control statements such as GO TO, etc.) instead of procedure descriptions (which are one-to-one translations of flow charts)."

ADS: ADS is specifically intended for complete specifications of problem requirements: "The completion of ADS forms gives the definer a well-documented application that includes all of the information requirements of the problem."

TAG: "The Time Automated Grid (TAG) technique is a computer tool for use in systems definition, analysis, design and program definition."

SY: "SYSTEMATICS is a language for describing information systems without considering the strategy needed to implement them."

YK, ADS and TAG are problem statement techniques that use a practical, straightforward approach without any attempt to develop a "theory" of data processing. They consist of a systematic way of recording the information that an analyst would gather in any case; any experienced analyst could use either ADS or TAG with very little instruction. IA is more concerned with developing a theory. It uses a terminology and develops a notation which is not at all natural to most analysts. LO is more like a non-procedural programming language than a requirements statement technique, since in order to use it, the system design must be completed, i.e., the file processing runs needed must be known. (The language as described in the literature applies to batch processing only.) The approach, however, is relevant because it presents a "non-procedural" technique for stating processing requirements once the runs are determined. LA starts with a precedence relationship among information sets (files) but does not indicate how these are obtained. This technique therefore is more relevant to the analysis of a problem statement and to the design of a system. However, it does suggest some desirable features of a problem statement technique.

Comparison of features of selected languages

In his review, Young states his difficulty in comparing the techniques he considered:

> "I wish that I could fit all of the developments described here into some sort of nice conceptual framework, but I find it difficult to do so. Each of these systems is intended for a somewhat different purpose, and each implementor has had his own ideas on philosophy and language. Perhaps the best I can do is state what I feel are some of their major advantages and leave as an exercise for the reader any sort of generalization."

These seven approaches, on the surface, appear to be very different but upon detailed examination, they have some major similarities.

All of the techniques take essentially the same view of the problem. The purpose is to describe how to

TABLE II—Comparison of Seven Proposed Problem Statement Techniques

PROBLEM FORM	Information Algebra IA	Young & Kent YK	Lombardi LO	Langefors LA	ADS	TAG	Systematics SY
INPUT	INPUT AREAS	INPUT DOCUMENTS	ORDERED INPUT FILES	INITIAL INFORMATION SETS	INPUT	INPUT DOCUMENT AND FILES	not specifically identified
OUTPUT	OUTPUT AREAS	OUTPUT DOCUMENTS	ORDERED OUTPUT FILES	TERMINAL INFORMATION SETS	REPORT	OUTPUT DOCUMENT	
DATA RELATIONSHIPS							
	ENTITY	not used	not used	not used	VARIABLE	DATA NAME	
	PROPERTY	ITEM	FIELD	not used			
	PROPERTY VALUE	not named	not named	not used	VALIDATION RULES		VALUES
	PROPERTY VALUE SET	INFORMATION SET	not used	not used			SETS
	COORDINATE SET						
	DATUM POINT						
	PROPERTY SPACE						
	LINES		RECORDS				
	AREAS		FILES	INFORMATION SET			
	(BUNDLE, GLUMP)		BUNDLE				IDENTIFIERS
COMPUTATIONAL RELATIONSHIPS							
	MAPPINGS	PRODUCING RELATIONSHIPS FOR DOCUMENTS	CONTROL PREDICATES	PRECEDENCE RELATIONSHIPS AMONG INFORMATION SETS	LOGIC	PERIOD AND PRIORITY	not mentioned
	COORDINATE DEFINITION	DEFINING RELATIONSHIPS FOR OUTPUT ITEMS	FIELD DECLARATIONS	not used	COMPUTATION LOGIC	not specifically included	DERIVATION RULES
OTHER INFORMATION	none mentioned	VOLUMES ELAPSED TIME	none mentioned	SIZE OF FILES	VOLUMES	VOLUMES PERIOD	none mentioned
PRESENTATION	FREE FORMAT MATHEMATICAL STATEMENTS	GRAPHICAL NOTATION	FREE FORMAT MATHEMATICAL STATEMENTS	PRECEDENCE GRAPH	FIVE FORMS	INPUT/OUTPUT ANALYSIS FORM	TABLES

produce outputs from inputs. All of the techniques provide some method for describing data relationships as the user views them. All provide some way for stating the computational relationships, i.e., the business data specification function. Several provide some method for stating other data such as time and volume. All are concerned with appropriate methods of recording and presentation of requirements.

The techniques are compared in these five areas of similarity and a summary is given in Table II. Each line of this table gives the different names used by different authors. Throughout this section terms that are used with specific meanings in the techniques are capitalized.

Form of the problem

The first category for comparing the seven approaches deals with their view of the overall problem. The following quotes give the authors' definition of data processing systems and their approach to analysis and design.

IA: "An information system deals with objects and events in the real world that are of interest. These real objects and events, called "entities" are represented in the system by data. The data processing system contains information from which the desired outputs can be extracted through processing. Information about a particular entity is in the form of "values" which describe quantitatively or qualitatively a set of attributes or "properties" that have significance in the system. Data processing is the activity of maintaining and processing data to accomplish certain objectives."

LA: "There are some basic propositions made here in connection with the systematic approach advocated, which appear to be in contradiction to present practices or assumptions. One is the hypothesis that in most cases it is possible to isolate and define the relevant organization functions in a separate operation to be performed before the actual design of the system is attempted. It is thus assumed that these functions are defined from the basic goals of the organization and therefore will not need to await the detailed construction of the system. The other hypothesis is that it is possible to define all input information necessary to produce a desired output. The basic assumption here is that actually any information can only be defined in terms of more elementary information, which will then occur as input parameters. Therefore, once a class of information is defined then it is known what input information is required for its production. The point here is that it should not be necessary to work out formulas, or even computer programs, before it can be determined what input data are needed. In fact, it is well possible to work out formulas or programs for an entity, where important variables are missing, so that starting by programming is no safeguard against ignoring important data."

YK: "The content of our analysis is that the objectives of the data processing system have been stated in terms of the required outputs; these outputs are not considered as subject to revision. On the other hand, although the inputs may be organized in any desired fashion, it appears necessary or at least convenient, to state one of the possible input organizations from which any equivalent one can be derived. It should be noted that the input may supply any one of a number of equivalent pieces of information, e.g., either customer's name to be copied directly onto an output or an identification number from which the name can be looked up."

LO: "The common denominator of file processes is the production of output files as functions of input files."

ADS: "The starting point is the definition of the reports—what output information is required. Once the reports are defined, the next step is to find out what information is immediately available. This is followed by laying out the information system in between the output and input. The origin of all information needs to be specified. The outputs of this system are always looked at in terms of inputs."

TAG: "The technique requires initially only output requirements of a present or future system. These requirements are analyzed automatically [by a computer program] and a definition is provided of what inputs are required at the data level."

SY: "SYSTEMATICS is a language solely concerned with techniques and concepts useful to systems analysts in designing information models to meet user's requirements. . . . It is a tool for specifying solutions to information systems problems. More important, it is also a tool for developing such solutions."

Six of the approaches (all except SYSTEMATICS) assume that the problem statement starts at output, e.g., IA: ". . . from which the desired outputs . . .";

YK: ". . . in terms of desired outputs . . ." Therefore, a necessary part of the problem statement should be the description of the desired output. This requirement is implied by LA in the definition of TERMINAL SETS and in YK by the definition of OUTPUT DOCUMENTS. IA does not mention required output as such and, in fact, in the example given in the paper says, "The problem is to create a new pay file from . . ."

TABLE III—Description of Documents

FOR WHOLE DOCUMENT	YOUNG & KENT	TAG	ADS
1. NAME	√*	√	√
2. TYPES OF DOCUMENTS	INPUT,OUTPUT	INPUT OUTPUT FILE	INPUT REPORT HISTORY
3. WHEN PRODUCED	PRODUCING RELATIONSHIP	PERIOD (S,MI, H;D,W,MO,Q,Y) PRIORITY (NUMERIC) FREQUENCY	SELECTION RULES FOR REPORT
4. MEDIA	NOT MENTIONED	NOT MENTIONED	FOR INPUT
5. SEQUENCING CONTROL MAJOR INTERMEDIATE MINOR	NOT MENTIONED	NOT MENTIONED	SEQUENCES MAJOR INTERMEDIATE MINOR ARBITRARY NUMBER
6. VOLUME AVERAGE (A)	√	√	EXPECTED VOLUME FOR HISTORY AND INPUT AND REPORT
MINIMUM (M)	√	√	
PEAK (P)	√	√	
7. DESIGNED FOR PEAK, AVERAGE OR MINIMUM	NOT MENTIONED	DESIGNER'S CHOICE	NOT MENTIONED
8. OTHER DATA		REFERENCE AUDIT	MEMOS

FOR EACH DATA ITEM	YOUNG & KENT	TAG	ADS
1. NAME	√	√	√
2. HOW USED		FI-FIXED; INFORMATIONAL FF-FIXED; FUNCTIONAL VF-VARIABLE; FACTOR VR-VARIABLE; RESULT	MODIFIED BY — FORMULA — PARTICULAR VARIABLE HOW OFTEN? — NEVER** — PARTICULAR CYCLE** — FIXED TIME** — LOGICAL CONDITION ** by memo only
3. COMPUTATION FORMULAS	DEFINING RELATIONSHIPS FOR VARIABLES ON OUTPUT REPORTS	NOT INCLUDED (MAY BE ADDED AS COMMENTS)	COMPUTATION FORM LOGIC FORM
4. SEQUENCING ROLE	NOT MENTIONED	√	√
5. VALIDATION RULE	NOT MENTIONED	NOT MENTIONED	√
6. FORMAT A or N	IN INFORMATION SET TABLE	√	√
SIZE (NO. of CHAR.) FOR OUTPUT		√ Ordering number of P for presence.	√
7. NO. OF TIMES PER DOC. MINIMUM AVERAGE MAXIMUM	√	RATIO	RATIO

* A checkmark denotes provision for including the information listed in the left hand column.

Data relationships

A requirements statement must have some description of the data that will be produced. The most extensive data description facility is the one used by IA. This starts with the concept of an ENTITY which has a connotation of a physical entity in the real world such as an employee, a paycheck, or an order. Each ENTITY has PROPERTIES which describe that entity, e.g., an employee has an employee number, hourly rate, etc. For any given ENTITY there is a VALUE for each PROPERTY. The PROPERTY VALUE SET is the set of all possible VALUES that a PROPERTY can have in the problem. The COORDINATE SET is the list of all PROPERTIES that appear in the problem. A DATUM POINT is a set of values, one for each PROPERTY in the COORDINATE SET, for a particular ENTITY. The PROPERTY SPACE is the set of all DATUM POINTS, i.e., all possible points obtained by taking the cartesian project of all possible PROPERTIES. Once this PROPERTY SPACE has been defined, further definitions deal with subsets of this space. A LINE is a subset which is roughly equivalent to a record and an AREA is a subset roughly equivalent to a file. Other subsets of the PROPERTY SPACE are BUNDLES and GLUMPS. The basic reason for this choice of data description is to use the concepts of a set theory as the formulation for a theory of data processing. (The authors of IA reject data description by arrays as being too limited.)

In YK, the basic units of data are called ITEMS, which corresponds to PROPERTIES in IA. Their term INFORMATION SET is used for the set of all possible values of a particular item and is, therefore, equivalent to the PROPERTY VALUE SET in IA. The information that can be provided for each INFORMATION SET are: (i) the number of possible values, (ii) the number of characters or digits, and (iii) relationships. The following relationships are defined:

	Description	Symbol	Graphic Symbol
Isomorphism	—one to one correspondence		←-------→ -------
Homomorphic	—many to one correspondence	–	
Cartesian product	—P_jXP_k means a pair of P_j and P_k	×	
	—contained in	ϵ	
Equal to		=	

The relationships may be used to make statements such as, there is one employee number for each employee name and address. YK did not want to make any statements about the file structure and, hence, there are no terms that correspond to records or files. YK also provides a graphical notation for showing relationships.

In LO, the definition of data is more conventional including FIELD (which corresponds to PROPERTY), RECORDS, FILES, etc. The word BUNDLE is used to denote a set of files which is merged on a single input or output unit.

In LA, there is no definition of data corresponding to data items. The problem definition starts with collections of data which are called INFORMATION SETS. This corresponds roughly to the notion of a file in common terminology. LA introduces the concept of an elementary file in which each record contains a data value and enough "keys" to identify it uniquely.

ADS provides three forms on which data is described: REPORT, INPUT, and HISTORY. Each of these forms provides space for some information describing the particular report or input: name, media, volume and sequence and space for each variable. For each variable the forms provide space for name, how the value of the variable is obtained (INPUT, COMPUTATION, HISTORY), a cross-reference, how often the variable appears, and size (number of characters).

TAG provides one form which contains space for data describing the document (or file) and space for each variable. Table III shows a detailed comparison of the data required by YK, TAG and ADS to describe documents and data items.

SYSTEMATICS does not have any rules for specifying structure of data. The major emphasis is on IDENTIFIERS.

Computational relationships—data definition by formula

When general purpose programming languages are used, each program, or sub-program, includes statements which produce output, statements which test the conditions under which the output is produced and statements which compute the values of the variables that appear in the output. It has been argued by Lombardi, in particular, that a "non-procedural" language must separate the statement of *what* output is to be produced when, *from* the statement of the procedure for producing the value of the variables that appear in the output. All seven approaches follow this concept.

In IA the basic operation that the problem definer can use to state his processing requirements is a mapping of one subset of the PROPERTY SPACE into another

subset. Two kinds of mappings are defined. One corresponds to operations within a given file. For example, suppose a tape contains time cards, sorted in order by employee number, one for each day of the week. A mapping could be defined which would take the set of (five) POINTS for each employee into one new POINT which would contain the total for the week. The second type of mapping corresponds to the usual file maintenance operation in which POINTS from a number of input files are processed to produce new output files. These two types of mappings are called GLUMPING and BUNDLING respectively. The actual computation of the PROPERTY VALUES of the new POINTS produced by a mapping is specified by a COORDINATE DEFINITION which must contain a computational formula for each PROPERTY in the COORDINATE SET.

In YK the major unit of processing is a PRODUCING RELATIONSHIP; there must be one PRODUCING RELATIONSHIP for each output document. This PRODUCING RELATIONSHIP gives the conditions under which a document will be produced. This statement may contain conditions (Boolean expressions) that depend on values of data ITEM or on time. For example, a PRODUCING RELATIONSHIP might be a "a monthly statement is produced for a customer each month for all customers with a non-zero balance." A PRODUCING RELATIONSHIP may also state that one output Document D_2 is produced for each input Document D_1. The values of the data ITEMS which appear in the output documents are calculated using a DEFINING RELATIONSHIP. There must be one defining relationship for each data item which appears on an output document.

In LO the statements which control whether or not an output record is produced are called CONTROL PREDICATES. There must be one control predicate for each record for each output file. The CONTROL PREDICATES, in general, are Boolean expressions which may involve the use of INDICATORS. The values of the variables which appear on the output records are produced by FIELD DECLARATIONS which are evaluated at the end of each PULSE in a PHASE.

In LA the relationships are given for production of INFORMATION SETS and, hence, correspond to PRODUCING RELATIONSHIPS. However, they are stated only as precedence relationships, e.g., INFORMATION SETS a, b, and c are necessary to produce d. No computational formulas are given. A problem statement may be represented by a graph as shown in Figure 4 where circles represent elementary INFORMATION SETS and rectangles represent PROCESSES.

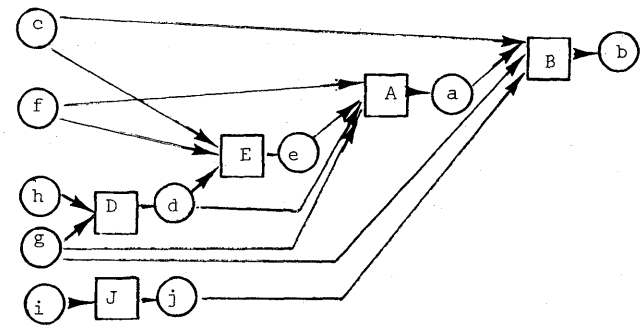

Figure 4—Network representation of problem statement in LA

In ADS some basic information is specified about when reports are to be produced. However, in many cases this is supplied by written notes. This information may be regarded as analogous to the PRODUCING RELATIONSHIPS in YK. ADS requires that each variable be identified as coming from INPUT, COMPUTATION or HISTORY. A form is provided for specifying the computations; this specification is somewhat limited. Another form is used to state logical conditions and these may be used to state when outputs occur and under what conditions computations are performed.

TAG provides for stating how often outputs will be produced by specifying a PERIOD. The available codes are: second, minute, hour, daily, weekly, monthly, quarterly, and yearly. A PRIORITY can be assigned to distinguish a sequence ordering between two documents with the same period. TAG does provide a means for stating which data elements are to be computed but it does not provide for stating the formula for the computation. (The formula can be included in the "Comments" section of the form but it will not be analyzed by the program.)

In SYSTEMATICS there are two types of data items, GIVENS and DERIVED. For each DERIVED item there is a Derivation rule that states the formula by which it is computed. Considerable effort is devoted in SYSTEMATICS to specifying the sets of values over which the rules hold. Consequently data items may be IDENTIFIERS and there are a number of different kinds: PRIMARY, SECONDARY, COMBINED, and COMMON.

Other information

IA and LO do not specify any additional information; LA assumes that the relative size of files is available. YK, ADS and TAG all provide for specifying time and volume requirements.

YK defines two kinds of time: extrinsic (when an event occurs) and intrinsic (the time written on a document). The "operational requirements" consist of a volume for each document (input and output) and a time statement for each output document. Volumes of documents may be expressed in terms of averages over some time period.

ADS permits specification of average and maximum volume in INPUT, REPORT, and HISTORY forms. In addition, each variable in HISTORY is characterized by how long it is to be retained; this may be a fixed number or may depend on a computation.

TAG provides for volume information for documents, size information on data elements and repetition information on data elements within documents. (The information is apparently not used in the programs which process TAG statements.)

Presentation

Both ADS and TAG have well-structured forms for recording the problem statement. They differ in that ADS has five relatively highly structured forms while TAG has a single form. The others do not specify any particular way in which the problem statement must be made. YK describes a graphical method of presentation and LA uses a precedence graph for illustration only.

Analysis, summary and conclusion

Extent of use

Despite extensive recognition of the need for better ways of stating requirements and despite the availability of basic concepts of problem statement languages since 1958 (Young and Kent) and 1962 (Information Algebra) the use in practice of these techniques has not been extensive. Even now ADS and TAG have a limited number of users. Young and Kent's and Lombardi's languages have not been used at all and the development of SYSTEMATICS has not continued after a field trial.[48] Information Algebra has only been used once,[33] and then only for describing requirements for an assembler.

There is no published evaluation of why the techniques are not being used more. There is considerable evidence that many organizations have recognized a need and have attempted to develop their own problem statement techniques but after a while the attempt has usually been abandoned. Comments from a number of such organizations are too subjective to be quoted here, but are incorporated into the following analysis. Specific comments on the use of TAG are that its advantages as a requirement statement language include ease of learning and simplicity in use, its provision of computer processing of requirements data improves ease of modification of the requirement statement, and as a systems design procedure it gives machine-printed copy of program requirements. The disadvantages of TAG as a requirement statement language are that documents cannot be related to each other except through PERIOD, FREQUENCY and PRIORITY and through data elements, that only a two-level data structure is permitted, that repeating groups cannot be handled except through ratios for each variable and formulas cannot be specified. As a systems design procedure, a disadvantage is that it requires manual intervention in the process.

The arguments made against formal problem statement languages can be grouped broadly into two categories: technical problems—the techniques are not satisfactory for stating requirements—and human problems—getting analysts to accept and use them. In practice these two are closely related—an analyst who does not want to change to formal method can usually find some technical reason why the proposed method is not satisfactory. The difficulty here is very similar to that faced in improving other aspects of the system building process.[68,65]

There are a number of reasons that have been suggested for the people problem. One reason is that preparing a rigorous and complete problem statement requires (or at least seems to) more time than the present procedure in which problem statement, systems analysis and programming are collapsed into one indistinguishable process. A second reason is that there has not been any immediate advantage to an analyst or programmer to invest additional time in a more systematic problem statement. Such advantage could come from either or both of the facilities to manipulate the problem statement symbolically and a computer processing of the problem statement itself. TAG currently provides such a software package and one is also available for ADS.[67] (Computer-aided analysis of problem statements will be discussed in a later paper.)

A number of concepts that should be included in a requirements statement language in order to eliminate the technical problem (by ensuring that the formal technique is sufficient to state requirements) are summarized here to provide a basis for the enumeration of desirable objectives of future requirements statement languages.

Form of the problem

There can be no question that the basic purpose of an IPS is to produce outputs. However, it is not clear that

limiting the statement of data processing requirements to outputs only (as advocated by TAG) is desirable. Conceptually, one does not want to prejudice the systems design by stating inputs that may not be needed. Frequently, however, certain inputs must be accepted by the IPS and in that case the problem statement might as well include the facility for specifying them. Also the conditions that must be stated (e.g., the PRODUCING RELATIONSHIPS in YK) in the absence of specification of inputs can become very complicated. The statement of problems will probably be simplified if the problem definer can state his requirements either in terms of "events" which require action in the IPS or in terms of outputs required; whichever is most convenient for him, i.e., either in terms of input or of output. Providing this convenience may complicate the analysis of the problem statement by the computer, but the additional processing time is probably worth it.

One objection expressed against viewing IPS as output producers arises from the belief that in the future IPS will be basically data base storage and retrieval systems in which a data administrator will decide what data are to be stored and users will communicate their requests as inquiries. These two views are not incompatible since a result of an inquiry is an output—an output that can be described in the same way as an output that is produced periodically.

Data description

IA is the only approach that, through the use of the ENTITY concept, attempts to associate data with the real world. It should be noted, however, that the IA language in itself does not depend on how the PROPERTY SPACE is obtained, i.e., whether it is derived from real ENTITIES or from a set of abstract concepts. It is desirable to give the problem definer as much help as possible in defining his data and the analogy to the real world through entities is the best method available. Hence, it might as well be part of a requirement statement language as long as it does not restrict the language in defining data abstractly.

It is important to distinguish between two possible uses of VALUE SETS. (A VALUE SET consists of all possible values of a PROPERTY within PROPERTY SPACE). The first use is for PROPERTIES in which only one value will be in the machine at any one time. For example, the PROPERTY "warehouse number" may have many values in the memory at one time whereas the "quantity" of a particular part number at a particular warehouse will have only a single value at any particular time.

In the first case, the VALUE SETS may be used for validation of input data. ADS, for example, permits validation rules to be given for each data item. In practice, validation is a complex process depending on combination of variables rather than on single variables and such rules are difficult to state on the ADS forms. It may be more desirable to specify validation by defining validation reports as outputs of the system; these then can include any processing specification permitted by the language for specifying data items on output reports.

The second use of VALUE SETS will be in providing information about how much memory space will be required. The basic question is how the problem definer states the role of data items. In COBOL the definition is through the structure definition in the DATA DIVISION and the use of OF and IN; in PL/1 nested qualifiers, separated by periods, are used. In YK the relationships among INFORMATION SETS are used to present this information. In future requirement statement languages it would be desirable to infer as much of the qualifier-identifier relationship of variables from the processing statements themselves and only ask for information that is not included there. It may be possible to obtain all needed information from VALUE SETS and the processing requirements.

The information in a problem statement must be sufficient to infer which data items will have to be stored in the auxiliary memory or in the main memory. A value must be stored in the memory if:

i. It appears in an update statement, e.g., of the form

$$X(,,)=X(,,)+Y$$

Here X might be "gross pay to date" and Y "the pay this week."

ii. It is used in a statement without its value having been computed, e.g., "number of exemptions" in a payroll problem. This data item would appear as input on a new hire transaction and in a change transaction and would be used in pay computation.

ADS permits the problem definer to specify data items to be available in HISTORY. These may be either intermediate data items that are used in a number of places or data items whose values the problem definer believes will have to be stored.

It is immediately clear from the preceding paragraph that one cannot determine what data items fall into these categories unless the problem statement contains information about the time at which processing requirements occur. In the first case (i), there must be some

way of stating that payroll is computed weekly and the "gross pay to date" is cleared (set to zero) at the end of each year. Similarly in the second case (ii) it must be clear that a new hire transaction only occurs once while the pay computations occur regularly.

Time and volume information

None of the problem statement languages have a well-developed syntax for describing the time aspects of requirements, though YK, ADS and TAG provide some capability. Some help in developing an acceptable "time" language might be obtained by studying the master time routines in simulation languages such as SIMSCRIPT or the executive systems for real-time systems. In a very general sense, time is just one of a number of attributes of data items and, hence, could be included in whatever general data description facility is provided by the language.

It may be noted that time specifications are required not only for determining which data items will be stored but also for determining feasible and optimal storage organizations. The criteria used to determine optimality include both memory space and processing time. One important factor to be considered is organization of data to reduce memory space by such techniques as header-trailer organization as used in hierarchical files and IDS. In order to do this, one must be able to infer the header-trailer relationships from such information as qualifiers and identifiers. Another important factor is the question of what data should be stored semi-permanently and which need only be held temporarily. Again, the analogy to simulation may be useful—SIMSCRIPT, for example, distinguishes between PERMANENT and TEMPORARY ENTITIES.

The second part of the criterion is to reduce processing time. One way this can be done is by reducing the number of accesses to external memory. Since a number of different types of processing requirements must be accomplished, the problem statement must contain both the values of each type and the time periods over which they occur so that accesses to auxiliary memories can be grouped whenever possible.

Both ADS and TAG permit some specification of volume data. Since information about volumes is usually the least accurate part of a problem statement, future languages should have considerably extended capabilities, for example, statement of time and volume by symbolic names.

Presentation

Graphical techniques are extremely useful in many areas of stating specifications, e.g., blueprints for construction specification and flowcharts for algorithms. A graphical technique for the problem statement was given in YK and this has since been extended by Young[69] under the acronym GRIST. The problem statement proposed by LA is equivalent to a directed graph. At the present state of development of problem statement languages it appears unlikely that graphical techniques other than flowcharts and graphs will be very useful. Some experimental work with the proposed techniques including GRIST appears justified, however.

Future problem statement languages will undoubtedly depend on forms, probably somewhere between the two extremes of complete specifications by forms and completely free form. Good forms can be extremely useful in acting as questionnaires and check lists.

Top-down approach

How much information about a problem should be collected when? In current practice the analyst will normally start with the general overall and summary data and gradually he will become more and more specific until he has enough detail to be able to write the programs himself. In contrast, ADS attempts to have the analyst specify all the details of the problem statement at one time.

The best procedure may be compromise between current practice and the ADS approach. Description of data, for example, could be divided into two levels:

Composition—how the data is made up of smaller units of data
Representation—hardware related items such as number of bits, precision, etc.

The composition information clearly is needed as part of the problem statement. Representation information, on the other hand, may not really be needed until program construction begins. A similar categorization could be made for processing requirements. Ideally, a problem statement would require specification of necessary data (composition information) for data, for example, and make optional the statement of information which is not needed until later. This is because sometimes it is easier to record all relevant data at one time.

Mathematical manipulation of the problem statement

IA represents an attempt to develop a problem statement notation that might be manipulated symbolically. The use of set notation and the usual set opera-

tions appear a reasonable start for a language in which data processing problems can be expressed. Since to our knowledge IA has only been used once[33] the practical usefulness of IA remains to be demonstrated. It is also not clear how one uses a problem statement expressed in IA in the system design. Both of these questions (the usefulness of IA for problem statement and the derivation of a design from such a statement) provide promising areas for research. Because of the size of data processing requirements it is unlikely that facilities to manipulate the requirements manually will be very helpful. However, there is no reason why such manipulation could not be carried out by computer programs if the language has suitable characteristics.

A REQUIREMENTS STATEMENT LANGUAGE

Objectives of a useful requirements statement language

The discussion in the first two sections has established the need for a better way of stating information needs. The analysis in the previous section has shown that, while there have been attempts to develop such languages, they have not been successful in the sense that they are not in wide use today.

The need for such a language exists even more strongly today and therefore research, development, experimentation and evaluation are needed to develop a satisfactory medium for communicating requirements. A set of objectives for a Requirements Statement Language (RSL) is proposed in this section.

—The language should accommodate the statement of requirements of the kind that are occurring now as well as those that will occur in the future. It is becoming more and more obvious that the cost of changing from one programming language to another is very high. Unfortunately, the present progression from COBOL, to COBOL with extensions, to Data Base Management Systems results in relatively small incremental improvements. The RSL should provide a quantum jump to a completely new generation of capabilities. The characteristics of the situation to be expected in the future that must be accommodated are:

i. Hardware features will increase in quality and reliability. There will be larger hardware with more parallel capabilities—this implies that unnecessary precedence constraints should be avoided whenever possible.
ii. Interrelationship of varying requirements will increase, e.g., jobs with varying priorities, inquiries to be answered, status data to be monitored, outputs required at predetermined times, data to be gathered and results to be distributed over geographically dispersed points, automatic monitoring and control, etc.
iii. The number and type of users with varying interface requirements will increase, e.g., on-line interaction; data entry such as transaction recorder; interrogation, e.g., reservation clerk, users with no programming needed; system builders; analysts and programmers; data administrators; operators; etc.
iv. Systems will become larger and larger and they will become more integrated. This implies: common data bases, any given programmer does not know what else is going on, new functions such as data administrator, etc.
v. Requirements will be more unstructured; immediate response will be required and requirements will be changing rapidly; jobs require more consistency in data and business data function specifications. This implies that the "user" must be able to communicate with the computer system more directly.
vi. The performance of systems will become more important and hence there will be greater emphasis on more explicit recognition and statement of the criteria by which performance is measured and requirements parameters which affect performance.
vii. There will be more need to monitor the system in operation. The systems change over time either in the volume or the capabilities and consequently there must be provision for changing the internal structure of the system without affecting the correct achievement of the requirements.

—The language should be suitable for use by humans in the necessary activity of determining and stating requirements.

i. The language or part of it must be usable by the manager or his assistants. This is necessary to eliminate the (computer) systems analyst as intermediary in order to reduce the chance for misunderstanding and to reduce the implementation time. To some, this specification implies that the language must be a subset of English. However, the fact that a subset of English is not English can severely limit the value of a subset of English as a requirements language. One of the objections sometimes raised against anything other than a natural language as a requirements language is that a

manager will never take the time to use what to him is an unnatural language. It is unlikely that top managers will ever specify detailed requirements. The situation here will be analogous to the current situation in accounting. When a manager first starts out in his career, he is very familiar with the details of accounting and prepares statements for his immediate superior from the reports furnished by the accounting department. As he rises in the organization, he delegates more and more of this to his assistants but he still understands the accounting language and procedures. The career path of the person using the requirements language will be through the management ranks rather than the computer ranks.

ii. The language must be suitable for the top-down approach for problem definition. Most large systems are defined from the top down. The broad, overall outline is developed first and then successively more details are filled in. The language should permit this process and permit checking the problem statement for consistency and unambiguity at each level before proceeding to the succeeding lower levels. The language should, of course, not prohibit the bottom-up approach where this is appropriate.

iii. The language should be suitable for helping in the determination of requirements. It should augment the capabilities of the analysts or teams of analysts who are carrying out the requirements determination.

iv. The language should facilitate the testing and "exercising" of requirements. It is extremely important that statements of requirements be tested before they are implemented. Tests should be made for consistency and completeness. In addition, the person developing the requirements should be able to state data and test conditions that can be used to verify correctness of the requirements statement.

—The language should be suitable for building the system to accomplish the requirements.

i. The language should permit the statement of requirements only and prevent the statement of data processing procedures. This is absolutely necessary in order to make the requirements statement hardware independent and to avoid reconversion costs when the capabilities of the equipment change. It is also necessary to prevent the introduction of restrictions which may limit the efficient use of hardware resources in the later stages of systems building.

ii. The requirements statement must be analyzable by computer programs. The problem statement should not only be readable by a computer program so that the requirements can be stored, but it should also be analyzable so that the problem can be restructured for optimum implementation efficiency without being limited by the sequence used by the problem definers. This is also necessary to permit the automatic construction of the system.

iii. The requirements statement language must permit statement of details necessary for the production of object code. This is necessary if the system is to be constructed automatically. In accordance with the above specifications, however, this detail should not have to be provided all at one time and as much as possible should be available from a library that is built up over time.

iv. The language should permit statements to facilitate the transition process. In most cases, systems already exist with files and programs and it is desirable to be able to move from the present system to the future system in an organized, controlled fashion to reduce inconvenience to the user and reduce cost.

v. The language should be as independent as possible of the particular area of application so that the cost of maintaining separate systems for a number of different applications is eliminated.

Outline of a requirement statement language

A language designed to satisfy the above objectives has been developed and is being tested in the ISDOS Research Project at The University of Michigan, under the acronym PSL (Problem Statement Language). A brief description is given here. A more detailed description is given in the language specifications and user manuals.

The language is used to describe the requirements that refer to the desired target system as a whole, as well as the individual units of the total requirements, i.e., the inputs to and outputs from the target system.

The system requirements include factors such as the parameters that are used in more than one place in the system and whose definition is controlled at the system level; system-wide policies, e.g., the form in which "data" will be used; system constraints that pertain to the system as a whole; resources available to the system,

such as hardware, software, etc.; and the performance criterion that is to be used in evaluating the system and in constructing it.

The description of each input to and output from the system contains five major types of information:

i. Identification information which relates the input or output to the external environment. For example, where the input comes from or where the output goes to, who has prepared the statement, what functional area of the firm it is related to. This section will also contain, where necessary, the interface information such as, for example, if the output has to be accepted both on cards and by teletype.
ii. Timing information which describes the triggering of particular input or output in regard to time and/or other conditions. Time here may be specified as real time or time related to some other calendar which would be defined under system requirements.
iii. Volume parameters which determine the quantity of the input or output required.
iv. Data definition showing how the data groups are related by structure.
v. Data definition by formula which gives the individual computations which have to be performed. Decision tables can be used to specify complex logical conditions.

The language will be processed by computer programs where output will be structured to give the analyst as much aid as possible. An overview of the software system is given in Teichroew and Sayani.[40]

CONCLUSION

Since problem statement languages have not been widely used the comparison and analysis in this paper have been based primarily on "paper" systems. The specifications for an ideal requirements statement language have come from this analysis, personal opinion and limited reports from users. There are signs that the situation is changing.

Head forecasts:

> "Most of the work described here is still in its germinative stages, and consequently has had little impact so far on the day-to-day activities of systems people. But it is likely that today's systems analyst, with his still-primitive analytical tools, will one day become as rare as the machine-oriented programmer who flourished a decade ago."[7]

Sammet holds a similar view:

> "Ideally, the user would state only the definition of his problem and the computer system would develop the solution. While the day of asking the computer to "COMPUTE THE PAYROLL FOR MY COMPANY" is at least one or two decades in the future, I believe we will see a large decrease in the amount of detail a user must provide. More specifically, I expect more statements about *what* is to be done and fewer details on *how* to do it. There will be compilers which can effectively determine which of many alternative algorithms should be used in a given situation."[70]

Hopefully, it will be possible to base the next survey of this kind on much more user experience.

REFERENCES

1 R I BENJAMIN
Control of the information system development cycle
Wiley New York 1971

2 W HARTMAN H MATTHES A PROEME
Management information system handbook
McGraw-Hill New York 1968

3 T B GLANS B GRAD D HOLSTEIN
W E MEYERS R N SCHMIDT
Management systems
Holt Rinehart and Winston Inc 340 pp 1968 [Based on IBM's Study Organization Plan (C20-8075-0) 1963]

4 D H SUNDEEN
General purpose software
Datamation January 1968 pp 22-27

5 Computerworld June 28 1972 p 5

6 CODASYL SYSTEMS COMMITTEE
A survey of generalized data base management systems
May 1969 (Available from ACM)

7 R V HEAD
Automated system analysis
Datamation August 15 1971 pp 22-24

8 R T HARRISON
The IBM application customizer services
In 65

9 G W POTTS
Natural language inquiry to an open-ended data library
AFIPS Conference Proc Vol 36 1970 pp 333-342

10 BOEING CORPORATION
MAST: modular application structuring technique
Commercial Airplane Division Seattle Washington no date 58 pp

11 NATIONAL CASH REGISTER COMPANY
BEST manual
1965

12 J R ZIEGLER
A modular approach to business EDP problem solving
Proceedings ACM 20th National Conference 1965 pp 476-484

13 J R ZIEGLER
Computer generated coding
Datamation Vol 10 No 10 October 1964 pp 59-61

14 R T JONES
Basic commercial data processing functions generalized approach to analysis, programming and implementation
IBM Systems Research Institute New York 1964
15 W STIEGER
Survey of basic data processing functions and design using modules
ISDOS Working Paper No 11 August 1968
16 PROCTER & GAMBLE CO
Basic functions manual
Cincinnati Ohio March 1967
17 D TEICHROEW
Computer-aided documentation of user requirements
F Gruenberger ed Information Systems for Management Prentice-Hall Inc Englewood Cliffs New Jersey 07632 1972 pp 97-112
18 G M WEINBERG
The psychology of computer programming
Van Nostrand Reinhold Company 1971 288 pp
19 NATIONAL CASH REGISTER COMPANY
Accurately defined systems
1967
20 H J LYNCH
ADS: a technique in system documentation
Database Vol 1 No 1 Spring 1969 pp 6-18
21 M H HUDSON
A technique for systems analysis and design
Journal of Systems Management Vol 22 No 5 May 1971 pp 14-19
22 O T GATTO
Autosate
Communications of the ACM Vol 7 No 7 July 1964 pp 425-432
23 D D BUTLER O T GATTO
Event-chain flow charting autosate: a new version
The Rand Corporation Santa Monica California October 1965
24 J BUBENKO O KÄLLHAMMER
CADIS: computer-aided design of information systems
In 25 pp 119-149
25 J BUBENKO JR B LANGEFORS A SOLVBERG
Computer-aided information analysis and design
Studentlitteratur Lund 1971 207 pp
26 S WATERS
The CAM (computer assisted methodology) project
Proc of the NCC Workshop on Approaches to Systems Design April 11-13 1972
27 P AANSTAD G SKYLSTAD A SOLVBERG
CASCADE—a computer-based documentation system
In 25 pp 93-118
28 C F REYNOLDS
The importance of flexibility
The Computer Journal Vol 14 No 3 March 1971 pp 217-220
29 M CROWTHER-WATSON
DATAFLOW—a tool for the analyst
3 pp no date
30 R BOOT
Computer-assisted methods in systems analysis
International Conference on Practical Experience in Systems Analyses 18 pp
31 A S LEWIS
File organization for DATAFLOW
File Organization IAG Occasional Publication No 3 Scolts and Zeitlinger N V Amsterdam 1969 pp 165-177
32 CODASYL DEVELOPMENT COMMITTEE
An information algebra phase I report
Communications of the ACM 5 4 April 1962 pp 190-204
33 J KATZ W C McGEE
An experiement in non-procedural programming
Proceedings Fall Joint Computer Conference 1963 pp 1-13
34 H B LADD W P MARKOVIC
Formalized analysis techniques-aids to computer design and computer use
RCA Camden New Jersey 1957 8 pages and illustrations
35 B LANGEFORS
Some approaches to the theory of information systems
BIT 3 1963 pp 229-254
36 B LANGEFORS
Information systems design computations using generalized matrix algebra
BIT 5 1965 pp 96-121
37 B LANGEFORS
Theoretical analysis of information systems
2 Vol Studentlitteratur Lund 1966 (also available from National Computing Centre Ltd Quay House Quay Street Manchester England)
38 L LOMBARDI
Theory of files
Proc Eastern Joint Computer Conference pp 137-141
39 L LOMBARDI
A general business-oriented language based on decision expressions
Communications of the ACM Vol 7 No 2 February 1964 pp 104-111
40 D TEICHROEW H SAYANI
Automation of system building
Datamation August 15 1971 pp 25-30
41 G F RENFER
SCOT simplifies system design changes and time estimates
Canadian Data Systems June 1970
42 D M CAINER
SYMOB (systeme modulaire bull)
Data Processing March-April 1964 pp 106-108
43 ENGLISH ELECTRIC-LEO-MARCONI COMPUTERS LIMITED
SPEC: a technique for expressing system requirements
October 1966 20 pages plus appendices
44 C B B GRINDLEY
SYSTEMATICS—a non-programming language for designing and specifying commercial systems for computers
Computer Journal Vol 9 August 1966 pp 124-128
45 C B B GRINDLEY W G R STEVENS
Principles of the identification of information
File Organization IAG Occasional Publication No 3 Scolts and Zeitlinger N V Amsterdam 1969 pp 60-68
46 C B B GRINDLEY
Specification and interrogation of formal control systems
Paper presented at TIMS XVII Conference London July 1-3 1970
47 C B B GRINDLEY
The use of decision tables within SYSTEMATICS
Computer Journal Vol 11 No 2 August 1968 pp 128-133
48 C B B GRINDLEY
SYSTEMATICS field trials project
31 December 1968
49 P J H KING
Some comments on SYSTEMATICS
Computer Journal Vol 10 pp 116

50 D H MYERS
A time automated technique for the design of information systems
IBM Systems Research Institute New York 1962 50 pp
51 IBM
The time automated grid system (TAG): sales and systems guide
Publication No Y20-0358-1 (no date approx 1968) 40 pp (Reprinted in J F Kelly Computerized Management Information Systems Macmillan 1970 pp 367-400)
52 W M TAGGART JR
A syntactical approach to management information requirements analysis
PhD Thesis University of Pennsylvania 1971
53 J W YOUNG H KENT
Abstract formulation of data processing problems
Journal of Industrial Engineering November-December 1958 pp 471-479 Reprinted in Ideas for Management International Systems-Procedures Association 1959
54 R BOZAK
A proposed file processing language
System Development Corporation Santa Monica California TM 3392/000/00 1967 15 pp
55 M H GROSZ
Systems generation output decomposition method
Standard Oil Company of New Jersey July 1963
56 J W SUTHERLAND
Tackle system selection systematically
Computer Decisions April 1971 pp 14-19
57 D J HERMAN F C IHRER
The use of a computer to evaluate computers
Proceedings AFIPS 1964 SJCC Vol 25 pp 383-395
58 K E IVERSON
A programming language
John Wiley & Sons New York London 1962 286 pp
59 R M BALZER
Dataless programming
The Rand Corporation Santa Monica California July 1967
60 R HENDRY
BCL, a new data processing language
Datamation January 1968
61 ARTHUR ANDERSEN & CO
LEXICON: automation concept for business information systems general description manual
1972
62 J W YOUNG JR
Non-procedural language: a tutorial
7th Annual Tech Meeting South California Chapter ACM March 23 1965 24 pp
63 C J SHAW
Theory, practice, and trend in business programming
AD625-003 Systems Development Corporation July 1965 18 pp
64 R STAMPER
Computer aids in systems analysis
Computer Weekly International August 12 1971 p 18
65 NATIONAL COMPUTING CENTRE
Approaches to systems design
Proceedings of Workshop 11-13 April 1972
66 J E SAMMET
Programming languages: history and future
Communications of the ACM Vol 15 No 7 pp 601-610
67 R THALL
A manual for PSA/ADS: a machined aided approach to analysis of ADS
ISDOS Working Paper No 35 October 1970
68 EDP ANALYZER
COBOL aid packages
Vol 10 No 5 1972 Canning Publications
69 J W YOUNG
Graphical notation for information system description, GRIST
1967
70 J E SAMMET
Programming languages: history and fundamentals
Prentice-Hall Inc 1969