1974

Requirements Statement Language Principles for Automatic Programming

Thomas I. M. Ho

J. F. Nunamaker

Report Number:

74-125

REQUIREMENTS STATEMENT LANGUAGE PRINCIPLES FOR AUTOMATIC PROGRAMMING*

Thomas I. M. Ho**
J. F. Nunamaker, Jr. ***

CSD TR 125

* Presented at 1974 ACM National Conference
** Department of Computer Science & Krannert School of Industrial Administration - Purdue University - Lafayette, Indiana 47907
*** Department of Computer Science & Department of Accounting & Information Systems - University of Arizona - University of Arizona Tucson, Arizona 85721

REQUIREMENTS STATEMENT LANGUAGE PRINCIPLES
FOR AUTOMATIC PROGRAMMING

Thomas I. M. Ho
Department of Computer Sciences
and
Krannert School of Industrial Administration
Purdue University
Lafayette, Indiana 47907

J. F. Nunamaker, Jr.
Department of Computer Sciences
and
Department of Accounting & Information Systems
University of Arizona
Tucson, Arizona 85721

KEY WORDS & PHRASES: automatic programming, Requirements Statement Language

ABSTRACT

The first step in automatic programming is the statement of information requirements in a Requirements Statement Language (RSL), a language for stating system requirements without needing to state the procedures implementing the system. The objective of this paper is development of language design principles for an RSL offering extensive requirements statement facilities. This objective is achieved through the formulation of a formal description of an information processing system. The formal description provides the criteria for requirements statement facilities of an RSL and for the capabilities of software for requirements statement analysis.

INTRODUCTION

Widespread concern with the quality of software is evidenced by the proliferation of methods to improve the maintainability, extensibility, and reliability of software. Indicative of this trend, interest in structured programming is particularly high.

However, the widespread expansion of computer applications coupled with the less spectacular growth in sources of programming manpower makes even structured programming only a short-term solution to improving programmer productivity. Hence, this need for a long-term solution motivates the development of tools for automatic programming or computer-aided production of software. To a great extent, the ultimate success of automatic programming hinges upon the facilities made available to the user for specification of his requirements for the information system he desires. A Requirements Statement Language (RSL) is a non-procedural high-level language that permits the statement of requirements for an information system without stating the procedures necessary for implementation of the system. The effective use of an RSL is aided by a Requirements Statement Analyzer (RSA), a program that performs syntactical and logical analysis of an RSL statement. Then an RSA produces a coded statement to be used by other software components that perform physical system design and that automatically produce source language code implementing the information system described by the RSL statement.

As a proposed solution to a recognized need, the RSL concept is now receiving increased attention in the computing community. Teichroew [1] surveys seven proposed languages and describes several desirable features of an RSL. Recent references include Couger [2], Leavenworth and Sammet [3], Benjamin [4], Merten and Teichroew [5], and McGee [6]. Earlier references include Pridmore [7].

It is interesting to note the similarity between the philosophies of requirements statement and of structured programming. Since requirements statement involves only the definition of system requirements without specifying the procedures for meeting those requirements, crucial design decisions are postponed until later in the system development cycle. In the same way, structured programming [8] advocates the postponement of design decisions, particularly regarding data representation, until after the algorithm has first been abstractly specified.

This paper develops a formal model of an information system. This formal model provides the framework for statement of the criteria for requirements statement facilities of an RSL and for the logical capabilities of an RSA.

HISTORICAL AND TECHNICAL BACKGROUND

To meet the needs outlined above, the Information Systems Design and Optimization System (ISDOS) Project at the University of Michigan has been studying the systems building process with the objective of developing a methodology for computer-aided design and construction of information systems. A description of the ISDOS Project can be found in Teichroew and Sayani [9].

ISDOS was born at Case Institute of Technology (now Case-Western Reserve University) in 1967 and was moved to the University of Michigan in 1968. Affiliation with Purdue University is also maintained through the efforts of Dr. Jay F. Nunamaker, an original member of the ISDOS Project at Case.

The work at ISDOS has involved both the study of existing techniques for requirements statement and the development of new Requirements Statement Languages. All techniques view the problem in essentially the same way. They describe how to produce outputs from inputs. All techniques provide some method for describing data relationships as the user views them. They provide some facility for stating the requirements of the problem. Several provide some facility for stating other data such as time and volume.

Young and Kent [10] represent the earliest work. Information Algebra is the work of the CODASYL Development Committee [11]. Two other efforts have been reported by Langefors [12 and 13] and Lombardi [14]. Accurately Defined Systems (ADS) is a product of the National Cash Register Company [15] and is described by Lynch [16]. The Time Automated Grid (TAG) system, a product of IBM, was developed by Myers [17] and is described by Kelly [18].

ADS and TAG use a practical, straightforward approach without attempting to develop any "theory" of data processing. ADS and TAG are systematic ways of recording the information that a systems analyst would gather. ADS or TAG could be used by any experienced systems analyst with very little instruction.

Young and Kent and Information Algebra represent a problem definition approach that is more concerned with developing a theory. Both use a terminology and develop a notation that is not at all natural to most analysts.

Lombardi's approach requires the completion of the system design before it can be used and resembles a non-procedural programming language rather than an RSL. However, Lombardi's work is relevant because it presents a non-procedural technique for stating requirements once the file processing runs have been determined. Langefors' technique uses the concept of precedence relationships among processes and files without indicating how these relationships are obtained and is relevant to the analysis of a problem statement rather than to the design of a system. However, it does suggest a number of desirable features of a requirements statement technique.

Despite the availability of these RSL techniques, their use has not been extensive. To the best of our knowledge, the languages of Young and Kent and of Lombardi have not been used except in an experimental way. Information Algebra has been used only once by Katz and McGee [19]. It appears that the development and use of TAG has been discontinued by IBM. ADS appears to be gaining in user acceptance. The U. S. Navy [20], in the process of designing a financial system, and a number of other firms [21] have used ADS as a requirements statement technique.

This current work is the result of an evolutionary process involving several different RSL's. The first development SSL/I (SODA Statement Language/I) is the work of Nunamaker [22]. SODA (Systems Optimization and Design Algorithm) is an ISDOS software component that produces specifications for program module and storage structure and for hardware selection from the requirements analyzed by an RSA. Extension of SSL/I resulted in the development of PSL/I (Problem Statement Language/I) described by Koch, Krohn, McGrew, and Sibley [23]. Experience with PSL/I indicated its shortcomings and led to PSL/II possessing improvements suggested by Hershey, Rataj, and Teichroew [24]. Simultaneous with the development of PSL/II, experience with ADS demonstrated the value of a forms-oriented RSL for ease of requirements statement.

OVERVIEW OF THREE REQUIREMENTS STATEMENT LANGUAGES

Past experience with requirements statement techniques has indicated that no existing requirements statement technique is adequate for the complete expression of user requirements relevant to all aspects of systems design and optimization. This deficiency motivated the initial development of SSL/I, the subsequent development of PSL/II, and examination of ADS for desirable features.

ADS is forms-oriented, making it easy to use and still capable of specifying much of the basic requirements statement. SSL/I possesses additional capabilities, particularly in the specification of operational requirements consisting of information on volumes, frequency of output, and timing of input and output. Finally, PSL/II exhibits more powerful generalized facilities for data description, processing requirements, and operational requirements.

MOTIVATION

Cougar [2] acknowledges a lag in the development of systems analysis techniques. Our need for increased systems analysis capabilities stems from our failure to define systems analysis rigorously so that the discipline can be better characterized as a science rather than an art. In general terms, systems analysis is defined as the collection, organization, and evaluation of facts about a system and its environment. In terms of information systems, systems analysis occupies the first two steps of the system development cycle [2]:

1. Documentation of the existing system
2. Analysis of the system requirements to produce the logical design (program module and file structures) of an improved system.

However, we have never developed a formal theory of systems analysis, particularly with regard to the analysis of system requirements.

Young and Kent [10] made a most remarkable first step toward developing such a theory by identifying the basic components of a data processing problem:

1. Information sets
2. Documents
3. Relationships among information sets
4. Operational requirements.

However, their effort stopped short of specifying what analysis must be performed upon the system requirements in order to verify their consistency, completeness, and accuracy. Furthermore, as in the case of Information Algebra [11], no approach to the production of a logical system design is apparent.

Other investigators, Teichroew [25] in particular, have suggested specifications for the analysis of system requirements. However, these specifications are described in a narrative manner and therefore, they lack the rigor that has been claimed to be necessary for the specification of system requirements. In other words, we should comply with our recognized standards and impose the discipline of requirements statement upon requirements statement analysis itself.

Therefore, the development of a formal theory of systems analysis is imperative. The need for formalism is recognized by Wrigley [26]. Such a theory should provide the criteria for requirements statement facilities and for the logical capabilities of requirements statement analysis. Furthermore, the theory should include a formal model of an information system that represents a canonical form of all alternative designs that satisfy system requirements. Although this theory is motivated by the needs of automatic programming, the theory should be equally applicable to software development by traditional manual methods.

The criteria for requirements statement facilities should address several central issues. In particular, problems in requirements statement are still evident in the expression of:

1. Logical data structures and their identifiers
2. Time and volume parameters
3. Processing and logic.

The criteria for the logical capabilities of requirements statement analysis should specify the logical conditions that the requirements statement should satisfy and the reports that can be compiled from the requirements statement to aid system design. Furthermore, the model can also be the vehicle for the description of algorithms for generating logical design specifications for program modules and files.

Therefore, we develop a formal model of an information system. In particular, note that the formal model provides the underlying structure for specification of a Requirements Statement Language and its analysis, and is not the RSL itself. Although the algebraic notation of the model may be lengthy and detailed, it is essential to apply the rigor of mathematics to the definition of requirements statement and its analysis. The set-theoretic approach has the advantage of notation and concepts that are either already understood or can be easily grasped by most individuals. Most important, the preciseness of the mathematical approach insures the desired explicitness achieved by the model.

PRISM: PROPERTIES OF AN INFORMATION SYSTEM MODEL

Definitions:

Let R be the set of all number and character representations. Let $U = \{d_i\}$ be the data names in the information system being modeled by PRISM. Let a data item be the ordered pair $<d_i, r>$, where $d_i \in U$ and $r \in R$, designating an occurence of the data name d_i with value r.

A relational structure D_h is a set of data names. Let $D = \{D_h\}$ be the set of relational structures of the information system being modeled by PRISM. Let F_h be the set of subscripts of the data names in D_h. Then, $D_h = \{d_i : i \in F_h\}$. An occurrence of a relational structure D_h is therefore $\{<d_i, r_i> : i \in F_h \text{ and } r_i \in R\}$.

A data base DB is a set of occurrences of relational structures. Let $\theta = \{<d_i, r_i> : d_i \in D_h$ and $r_i \in R\}$ be an occurrence of the relational structure D_h. Then,

$$OC_h = \{\theta \in DB: \theta \text{ is an occurrence of } D_h\}$$
is the set of occurrences of D_h in DB
$$= \{oc_{h,j}\} \text{ where } j \in \{1, \ldots, c_h\} \text{ and } c_h = |OC_h|.$$

The subscripts h,j correspond to a data-base-key that enables the location of any occurrence in DB.

The identifier set ID_h of a relational structure D_h is a subset of the data names in D_h with the following properties:

1. $ID_h(\theta) = \{<d_i, r_i> \in \theta: d_i \in ID_h\}$ is the occurrence of ID_h for θ.
2. $K_h = \{ID_h(\theta): \theta \in OC_h\}$ is the set of all occurrences of ID_h in DB.
3. If $ID_{h'} \subseteq ID_h$, F is a mapping from OC_h to $OC_{h'}$ which to every element $oc_{h,j} \in OC_h$ associates an element $oc_{h',j'} \in OC_{h'} \ni ID_{h'}(oc_{h',j'}) \subseteq ID_h(oc_{h,j})$.
4. F is one-one if whenever $oc_{h,i}$ and $oc_{h,j}$ are elements of OC_h, and $oc_{h,i} \neq oc_{h,j}$, then $F(oc_{h,i}) \neq F(oc_{h,j})$.
5. F maps OC_h onto $OC_{h'}$ if $\forall oc_{h',j'} \in OC_{h'}, \exists\, oc_{h,j} \in OC_h \ni F(oc_{h,j}) = oc_{h',j'}$.

I. If $D_{h'}$ is a history relational structure, F maps $OC_{h'}$ one-one onto $K_{h'}$. $|OC_{h'}| = |K_{h'}|$, i.e. two different occurrences of $D_{h'}$ always contain different occurrences of $ID_{h'}$.

II. For all other relational structures D_h such that $ID_{h'} \subseteq ID_h, \forall oc_{h,j} \in OC_h \exists$ at most one $oc_{h',j'} \in OC_{h'} \ni F(oc_{h,j}) = oc_{h',j'}$.

The data names that belong to the identifier set are underlined in the definition of the relational structure. Let $G_h \subseteq F_h$ be the set of subscripts of the data names in the identifier set of D_h. Then, $ID_h = \{d_i : i \in G_h\}$ is the identifier set of D_h. Note that ID_h is also a relational structure and could be its own identifier set.

These definitions are illustrated in an example presented later in this paper.

Let $k \in K_h$ and $k' \in K_{h'}$. Identifier set projected equivalence: k is projected equivalent to k' ($k \doteq k'$) if and only if $k' \subseteq k$. Identifier set function equivalence: k is function equivalent to k' ($k \stackrel{f}{=} k'$) if and only if there exists a function f from K_h to $K_{h'}$ which to every element $k \in K_h$ associates only one element $k' \in K_{h'}$. For example, f(January) = 1st quarter. Identifier set equivalence: k is equivalent to k' if and only if $k = k'$. Observe that $h \neq h'$ enables matching of occurrences of two different relational structures with (projected or function) equivalent occurrences of identifier sets.

Let $A = \{N, I, C, H, O_i$ for $1 \leq i \leq p\}$ where p is the number of outputs produced by the information system being modeled. The elements of A represent the various states: non-existence (N), input (I), computational result (C), history (H), and output (O_i) that the data items of the information system may assume. All data items are initially in state N.

The output state is partitioned into p substates, each corresponding to a physical output document specified by the problem definer. A physical output document may contain occurrences of more than one relational structure if the data items occurring on the document belong to relational structures containing different identifier sets. For example, a report may contain both a listing of individual employee wages identified by employee number and of departmental wage totals identified by department number.

The states I and H are not partitioned because designation of the contents of physical input documents and consolidation of history data items into files assigned to physical devices are activities of system design (whether manual or computer-aided) and not of requirements statement. Logical definition of a single input or history data set (set of data names) is accomplished by including logically-related data names in one relational structure. Hence, membership of an input or history data name in a relational structure indicates the logical data set to which the name belongs.

Each computational result data item, i.e. an occurrence in state C, belongs to only one relational structure. Hence, membership of a computational result data name in a relational structure indicates the computation to which the name belongs, thereby making it unnecessary to partition the state C. A relational structure containing a computational result data name is not initially defined by the problem definer, but is instead created during requirements statement as an implicit structure containing the data name of the computational result as the only member of the structure that is not in the identifier set.

Let $T = \{1,2,\ldots\}$ be the set of days corresponding to the dates beginning with January 1, 1970. Processing is cyclical. Therefore, time is measured in terms of cycles:

1. Time of initiation of cycle: t_1
2. Length of cycle (in days): ℓ
3. Cycle number: n

   $t_{n+1} = t_n + \ell$: cycle number $n \geq 1$ happens at time $t_n \in T$. Cycle number 0 represents creation of history relational structures.

Define $T(t_1,\ell,n)$: $n \rightarrow T$ as the function which calculates the time of happening of any cycle iteration: $T(t_1,\ell,n) = t_1+(n-1)*\ell$ for $n \geq 1$. Define $p_{h,h'}$ as the proportion of happenings of the cycle of relational structure D_h to happenings of the cycle of $D_{h'}$. For example, if D_h is an input relational structure for a daily time card and $D_{h'}$ is an output relational structure for a biweekly pay check, $p_{h,h'} = 14$ (ratio of 14 happenings of a daily cycle in the two-week period of a biweekly cycle.)

The cycle of a relational structure is characterized by the cycle of the data item in that structure with the smallest cycle length. The happening of the cycle of a history relational structure represents the reading or writing of occurrences of that structure. The happening of the cycle of an input/output relational structure represents the reading/writing of occurrences of that structure. All items in a single input/output relational structure must have the same cycle. The happening of the cycle of a computational result relational structure represents the execution of the corresponding computation.

Let $s_{i,h,j}(t_1,\ell,n) \in A$ represent the state of data name $d_i \in D_h$ in the jth occurrence of D_h in the nth happening of the cycle of length ℓ beginning at time $t_1 \in T$. $s_{i,h,j}(t_1,\ell,n) \leftarrow I$ represents input of the value of the data name d_i in the jth occurrence of D_h in the nth happening of the cycle of length ℓ beginning at time t_1.

$s_{i,h,j}(t_1,\ell,n) \leftarrow C$ represents computation of the value of data name d_i in the jth occurrence of D_h in the nth happening of the cycle of length ℓ beginning at time t_1.

$s_{i,h,j}(t_1,\ell,n) \leftarrow H$ represents storage of the value of data name d_i in the jth occurrence of D_h in the nth happening of the cycle of length ℓ beginning at time t_1.

$s_{i,h,j}(t_1,\ell,n) \leftarrow O_k$ represents output of the value of data name d_i in the jth occurrence of D_h in the nth happening of the cycle of length ℓ beginning at time t_1.

The state of a data item in any particular cycle implies nothing about the state of that data item in any other cycle.

Let $v_{i,h,j}(t_1,\ell,n) \in R$ represent the value of data name $d_i \in D_h$ in the jth occurrence of D_h in the nth happening of the cycle of length ℓ beginning at time $t_1 \in T$.

The value of a data item in any particular cycle implies nothing about the value of that data item in any other cycle.

Let $OC_h(t_1,\ell,n) = \{oc_{h,j} \in OC_h : s_{i,h,j}(t_1,\ell,n) \neq N$ where $d_i \in D_h\}$. Then, $c_h(t_1,\ell,n) = |OC_h(t_1,\ell,n)|$ represents the volume of occurrences of relational structure D_h in the nth happening of the cycle of length ℓ beginning at time t_1. Let $C_h(t_1,\ell,n) = \{j \in \{1,\ldots, c_h\} : oc_{h,j} \in OC_h(t_1,\ell,n)\}$. Let $K_h(t_1,\ell,n) = \{ID_h(\theta): \theta \in OC_h(t_1,\ell,n)\}$.

Correspondence: $OC_h(t_1,\ell,n) \sim OC_{h'}(t_1',\ell',n')$. $OC_h(t_1,\ell,n)$ corresponds to $OC_{h'}(t_1',\ell',n')$ if and only if $\forall oc_{h,j} \in OC_h(t_1,\ell,n)$ $\exists$ at most one $oc_{h',j'} \in OC_{h'}(t_1',\ell',n') \ni F(oc_{h,j}) = oc_{h',j'}$. Correspondence is a transitive relation.

Unique correspondence: $OC_h(t_1,\ell,n) \simeq OC_{h'}(t_1',\ell',n')$. $OC_h(t_1,\ell,n)$ uniquely corresponds to $OC_{h'}(t_1',\ell',n')$ if and only if $OC_h(t_1,\ell,n)$ corresponds to $OC_{h'}(t_1',\ell',n')$ and $OC_{h'}(t_1',\ell',n')$ corresponds to $OC_h(t_1,\ell,n)$. Unique correspondence is both symmetric and transitive.

In an operational sense, when an input relational structure D_h is matched with a history relational structure $D_{h'}$ (as in updating a master file), the occurrences of the two relational structures involved can be characterized in the following way:

1. Erroneous transactions
   $\{oc_{h,j} \in OC_h: ID_h(oc_{h,j}) \in K_h(t_1,\ell,n) - K_{h'}(t_1,\ell,n)\}$ is the set of occurrences of D_h for which there exist no equivalent identifier set occurrences in $OC_{h'}(t_1,\ell,n)$.
2. Valid transactions
   $\{oc_{h,j} \in OC_h: ID_h(oc_{h,j}) \in K_h(t_1,\ell,n) \cap K_{h'}(t_1,\ell,n)\}$ is the set of occurrences of D_h for which there exist equivalent identifier set occurrences in $OC_{h'}(t_1,\ell,n)$.
3. Inactive master records
   $\{oc_{h',j'} \in OC_{h'}: ID_{h'}(oc_{h',j'}) \in K_{h'}(t_1,\ell,n) - K_h(t_1,\ell,n)\}$ is the set of occurrences of $D_{h'}$ for which there exist no equivalent identifier set occurrences in $OC_h(t_1,\ell,n)$.
4. Active master records
   $\{oc_{h',j'} \in OC_{h'}: ID_{h'}(oc_{h',j'}) \in K_{h'}(t_1,\ell,n) \cap K_h(t_1,\ell,n)\}$ is the set of occurrences of $D_{h'}$ for which there exist equivalent identifier set occurrences in $OC_h(t_1,\ell,n)$.

At first, it may appear that the relational structure defined in PRISM does not possess sufficient generality for the expression of more complex data structures. However, a description of the data structure class for requirements statement and a demonstration of the relational structure's capability to represent the data structure class should serve to establish the generality of the relational structure. The advantage of the relational structure lies in its ability to enable the simple specification of the relations among data without necessitating specification of the complex data structures representing these relations. Therefore, the structure of the data base is not fixed during requirements statement, but instead the automatic programming software constructs the data base by aggregating the relational structures.

THE DATA STRUCTURE CLASS

The types of structures available to the problem definer and the manner in which structures of each type are constructed from other structures describe the data structure class of the requirements statement technique. The available structure types and their component structures include:

Structure	Component Structure
item	none
group	item, group
group relation	group
record	group, group relation
file	record, group relation
data base	file

Item:

The elementary data structure is the item. The item is the smallest structural unit from which all available structure types are ultimately constructed.

Group:

A group is a collection of items or other groups. A simple group is a collection of items only while a compound group is a collection of both items and groups.

A simple group can be used in two ways. One, it can be defined as a collection of items in order to give the collection a name and other attributes of its own. An example is the group EMPLOYER composed of the items NAME, SOCIAL-SECURITY-NUMBER, WAGE-STATUS and RATE. Also, the items CHILD-NAME and AGE form the simple group OFFSPRING. Second, a simple group can be defined as a collection of string-valued items having a "collective value" formed by concatenating the string-valued item components. For example, the items MONTH, DAY, and YEAR form the group DATE-OF-HIRE.

A compound group is a collection of a set of items, called principal items, and a set of groups, called principal groups, with this new collection having a name and other attributes of its own. For example, if the groups DATE-OF-HIRE and OFFSPRING are added to the simple group EMPLOYEE, the result is a new compound group EMPLOYEE consisting of the items NAME, SOCIAL-SECURITY-NUMBER, WAGE-STATUS, and RATE and the simple groups DATE-OF-HIRE and OFFSPRING.

A group may be either repeating or non-repeating. A repeating group may have an arbitrary number of occurrences for each occurrence of the compound group containing the repeating group. A non-repeating group has only one occurrence for each occurrence of the containing compound group. For example, OFFSPRING is a repeating group because the number of children can vary from employee to employee. However, DATE-OF-HIRE is a non-repeating group because each employee has only one hiring date.

Group Relation:

A group relation is a mapping between two sets of groups. The groups belonging to the first set are called parent groups and those belonging to the second set are called dependent groups.

The group relation provides a way of relating groups. For example, with a set of parent PERSON group occurrences:

{PERSON(JOHN DOE), PERSON(J. SMITH)}

and with a set of dependent SKILL group occurrences:

{SKILL(1000), SKILL(2000), SKILL(3000), SKILL(4000)}

a group relation can be created to relate each person to the skill(s) he possesses:

{ <PERSON(JOHN DOE), SKILL(3000)>,
<PERSON(J. SMITH), SKILL(2000)>,
<PERSON(J. SMITH), SKILL(3000)> }.

Also, the group relation provides a way to establish a hierarchic relation between two sets of items. In a hierarchic group relation, each occurrence of a dependent group must be subordinate to one and only one occurrence of a parent group; the dependent group occurrence cannot stand alone. An example of a hierarchic group relation associates a parent group occurrence representing a person with a set of dependent group occurrences representing the academic degrees he holds:

PERSON(JOHN DOE), DEGREE(BS,1970,PURDUE)
PERSON(JOHN DOE), DEGREE(MS,1971,PURDUE) .

Generally, a hierarchic group relation is equivalent to a compound group. However, in a compound group, the principal items do not have a collective name. The compound group name refers to the entire collection of principal items and principal groups. In a group relation, each parent group has its own name.

An occurrence of a group relation consists of one or more occurrences of each parent and dependent group, with each parent group occurrence associated with one or more dependent group occurrences. If the group relation is non-hierarchic, each dependent group occurrence may be optionally associated with one or more parent group occurrences. If the group relation is hierarchic, each dependent group occurrence must be associated with one and only one parent group occurrence.

In a manner analogous to compound groups, a dependent group in a group relation may be repeating or non-repeating. A repeating dependent group has a variable number of occurrences for each occurrence of its parent group; a non-repeating dependent group has only one occurrence for each occurrence of its parent group.

Record:

A record is a collection of groups and group relations in which one and only one group, the record-defining group, is not subordinate to any other group. The record is used to define the major entities of an application. For a given class of entities, e.g. the employees of a firm, the principal items in the record-defining group correspond to fixed entity attributes common to all entities in the given class. The items in the principal group contained in the record-defining group or in the dependent group subordinate to the record-defining group correspond to variable entity attributes. Variable entity attributes either have multiple values or are not necessarily common to all entities in the given class.

The record-defining group may not be the dependent group in a hierarchic group relation contained in the record. However, the record-defining group may be the dependent group in a non-hierarchic group relation that relates records in the same file. A later discussion of the file describes inter-record relations.

There are three record types: the group record, the tree record, and the plex record. Each record type is a generalization of the former type so that a special case of each type is identical to the former type.

A group record is a single compound group. The compound group is the record-defining group.

A tree record is a set of hierarchic group relations arranged as a tree so that each group has at most one parent, and that one and only one group, the record-defining group, has no parent.

A plex record is a set of group relations in which each group except the record-defining group is the dependent group in a hierarchic group relation. In addition, all groups in a plex record may occur in any number of non-hierarchic group relations.

File:

A file is a collection of records. Hence, a file represents a collection of application entities, e.g. employees, projects, or parts. The entities represented by a file may belong to the same class, e.g. employees of a firm, or to different classes, e.g. projects and the parts used in each project.

In the sense that one record of a file can be processed without referencing another record in the same file, the records of a file are independent of one another. However, the records in a file may be explicitly inter-related in a manner apparent to the system. For example, the records in a file may be ordered on the value of the record sequencer, a set of items contained in the record. A file with unrelated records or with records related only by ordering is called an unlinked file. In addition, more general relations are possible by permitting non-hierarchic group relations between groups in different records. A file containing records participating in these more general explicit relations is called a linked file. Of course, the records in a linked file may also be ordered.

Data Base:

A data base is a set of files.

A Relational Model of Data:

A relational model of data for large data bases is described by Codd [27 and 28]. The relational model uses tables for representing the logical data base structure.

A table is a rectangular array with the following properties:

- P1: a table is column-homogeneous, i.e. all the items in any single column are of the same type, but items in different columns are not necessarily of the same type.
- P2: each item in a table is either a number or a character string.
- P3: all rows of a table are distinct.
- P4: the ordering of rows in a table is immaterial.
- P5: the columns in a table are assigned distinct names and the ordering of columns in a table is immaterial.

As a result, a table represents a relation of degree n, where n is the number of columns in the table. An example of a relation of degree 3 is the relation COMPONENT. The triple (x,y,z) belongs to this relation if the part with part number x is a component of the part with part number y and if z units of part x are needed to construct one unit of part y:

COMPONENT
(SUB-PART-NO, SUP-PART-NO, QUANTITY)

2010	6020	4
2015	6020	2
2025	6020	1
3010	6030	3
3025	6030	5

We now demonstrate that the relational model possesses capability to represent the three types of records described earlier. Hence, in addition to its comprehensibility, the relational model also possesses flexibility.

Tabular Representation of Group Records:

Consider the example of a group record, called EMPLOYEE described earlier. The elimination of the principal groups DATE-OF-HIRE (non-repeating) and OFFSPRING (repeating) is accomplished with three separate relations. These three relations convey all the information contained in the group record because the item SOCIAL-SECURITY-NUMBER uniquely identifies each EMPLOYEE and the item CHILD-NAME uniquely identifies the children of each EMPLOYEE:

EMPLOYEE(NAME, SOCIAL-SECURITY-NUMBER, WAGE-STATUS, RATE)
DATE-OF-HIRE(SOCIAL-SECURITY-NUMBER, MONTH, DAY, YEAR)
OFFSPRING(SOCIAL-SECURITY-NUMBER, CHILD-NAME, AGE).

Tabular Representation of Tree Records:

Consider the example of a tree record called PERSON:

```
01 PERSON (parent of SKILL group and
           CHILD group)
   05 NAME
   05 NUMBER
   05 SALARY
   05 SKILL
      10 CODE
      10 TITLE
   05 CHILD (parent of PET group)
      10 CHILD-NAME
      10 AGE
      10 PET
         15 TYPE
         15 PET-NAME
```

Although identical to a group record, the tree record PERSON differs from the group record in addressability: an occurrence of the group record must be retrieved as a complete unit, but certain portions, e.g. the SKILL group, of an occurrence of the tree record can be retrieved without retrieving the entire tree record occurrence. Assuming that the following items uniquely identify the corresponding groups:

Item	Group
NUMBER	PERSON
CODE	SKILL
CHILD-NAME	CHILD
PET-NAME	PET

elimination of the group structures yields the following relations:

PERSON(NAME, NUMBER, SALARY)
SKILL(NUMBER, CODE, TITLE)
CHILD(NUMBER, CHILD-NAME, AGE)
PET(NUMBER, CHILD-NAME, PET-NAME, TYPE).

Tabular Representation of Plex Records:

Consider an example of a plex record describing relationships between suppliers and parts. The parent group SUPPLIER consists of two items: SUPPLIER-NO and SUPPLIER-DESC. The dependent group PART consists of three items: PART-NO, PART-DESC, and QUANTITY. Two relationships between SUPPLIER x and PART y exist: the CANDIDATE relationship holds if x is capable of supplying y; the ACTUAL relationship holds if x actually supplies y. Assuming that SUPPLIER-NO uniquely identifies each SUPPLIER and that PART-NO uniquely identifies each PART, four relations represent both groups and the two relationships between the groups:

SUPPLIER(SUPPLIER-NO, SUPPLIER-DESC)
PART(PART-NO, PART-DESC, QUANTITY)
CANDIDATE(SUPPLIER-NO, PART-NO)
ACTUAL(SUPPLIER-NO, PART-NO).

PRISM Language Constructs:

PRISM statements are assignment statements that set the value of either a state variable $s_{i,h,j}(t_1,\ell,n)$ or a value variable $v_{i,h,j}(t_1,\ell,n)$. A state variable is set by either another state variable or one of the states defined in PRISM. A value variable is set by a value expression consisting of decimal numbers, other value variables, arithmetic operators, and logical conditions. A logical condition, enclosed in parentheses and appearing in a value expression, has the following semantics:

$$(\text{logical-condition}) = \begin{Bmatrix} 1 & \text{if true} \\ 0 & \text{otherwise} \end{Bmatrix}.$$

Such a logical condition is used for the specification of logic in the statement of computational requirements.

In addition to the use of logical conditions within value expressions, logical conditions may appear in the context of conditional assignment statements. A logical condition, enclosed in braces, is called a defining condition and has the following semantics:

$$\begin{Bmatrix} \text{logical-} \\ \text{condition} \end{Bmatrix} = \begin{Bmatrix} \text{defined} & \text{if true} \\ \text{undefined} & \text{otherwise} \end{Bmatrix}.$$

Then, the defining condition is applied to an assignment statement with the following semantics

<assignment> · defined means
<assignment> is performed.
<assignment> · undefined means
<assignment> is not performed.

The defining condition is generally used for the specification of the state of data names appearing in an assignment and for the matching of identifier sets whose relational structures are involved in an assignment.

AN EXAMPLE:

For purposes of illustration, we describe a payroll application which will illustrate the principles to be presented in the remainder of this paper. Biweekly, a PAY-REPORT listing the Social Security number, name, and wages of each employee is produced. Biweekly, each employee submits a TIME-CARD containing his Social Security number, date of pay, and hours worked. An employee history EMPL is maintained to store the Social Security number, name, wage status, rate of pay, and year-to-date wages of each employee. Modifications to the history are entered on a maintenance document called EMPL-UPDATE.

Initial definition of the example consists of the following PRISM definitions:

PAY-REPORT = {SSN NAME WAGES}
TIME-CARD = {SSN DATE HOURS}

$\forall\, d \in \text{TIME-CARD} \;\; \forall\, n \ni n \geq 1$
$\forall\, j \in C_{\text{TIME-CARD}}(4,14,n)$
$s_{d,\text{TIME-CARD},j}(4,14,n) = I$

EMPL = {SSN NAME WAGE-STATUS RATE YTD-WAGES}

File creation:

$\forall\, d \in \text{EMPL} \;\; \forall_{j} \in C_{\text{EMPL}}(4,14,0)$
$s_{d,\text{EMPL},j}(4,14,0) = H$

EMPL-UPDATE = {SSN CODE NAME WAGE-STATUS RATE}

$\forall\, d \in \text{EMPL-UPDATE} \;\; \forall\, n \ni n \geq 1$
$\forall\, j \in C_{\text{EMPL-UPDATE}}(4,14,n)$
$s_{d,\text{EMPL-UPDATE},j}(4,14,n) = I$

FUNCTIONAL SPECIFICATIONS FOR REQUIREMENTS STATEMENT ANALYSIS:

Functional specifications for requirements statement analysis describe the logical capabilities of an RSA. These capabilities include the performance of two functions:

1. Verification of logical conditions in the RSL statement
2. Production of reports from the RSL statement to aid system design.

Logical conditions that must be satisfied by the requirements statement include:

A. Data definition and static analysis
  1. Identifier set membership:
     a. Equivalent identifier sets: matching two relational structures with equivalent occurrences of identifier sets requires that the two relational structures have equivalent identifier sets.
        For any assignment containing defining conditions of the form:
        $\{ID_h(oc_{h,j}) = ID_{h'}(oc_{h',j'})\}$,
        it must be true that $ID_h = ID_{h'}$.

Example:

$\{ID_{EMPL}(oc_{EMPL,j}) = ID_{TIME\text{-}CARD}(oc_{TIME\text{-}CARD,j'})\}$

$ID_{EMPL} = \{SSN\} = ID_{TIME\text{-}CARD}$

     b. Projected equivalent identifier sets: matching two relational structures with projected equivalent occurrences of identifier sets requires that the identifier set of the second relational structure be contained in that of the first.
        For any assignment containing defining conditions of the form:
        $\{ID_h(oc_{h,j}) \doteq ID_{h'}(oc_{h',j'})\}$,
        it must be true that $ID_{h'} \subseteq ID_h$.
     c. Function equivalent identifier sets: matching two relational structures with function equivalent occurrences of identifier sets requires that there exists a function that maps the occurrences of the identifier set of the first relational structure to those of the second.
        For any assignment containing defining conditions of the form:
        $(ID_h(oc_{h,j}) \stackrel{f}{=} ID_{h'}(oc_{h',j'}))$,
        there must exist a function f from K_h to $K_{h'}$ which to every element $k \in K_h$ associates only one element $k' \in K_{h'}$.
  2. History update: each non-identifier history data item must be updated by some input or computational data item.
     $\forall s_{i,h,j}(t_1,\ell,n) = H \ni d_i \not\in ID_h$,
     there must exist an assignment of the form:
     $[v_{i,h,j}(t_1,\ell,n) \leftarrow v_{i,h',j'}(t_1,\ell,n)]$
     $\cdot \{s_{i,h',j'}(t_1,\ell,n) = a\}$
     $\cdot \{ID_h(oc_{h,j}) = ID_{h'}(oc_{h',j'})\}$
     where $a \in \{I,C\}$.

Example:

$\forall d \in \{NAME, WAGE\text{-}STATUS, RATE\}\ \forall n \geq 0$

$\forall_j \ni 1 \leq j \leq c_{EMPL}$

$[s_{d,EMPL,j}(4,14,n+1) \leftarrow s_{d,EMPL,j}(4,14,n)]$

$\cdot \{s_{d,EMPL\text{-}UPDATE,k}(4,14,n+1) = I\}$

$\cdot \{v_{CODE,EMPL\text{-}UPDATE,k}(4,14,n+1) = \text{'C'}\}$

$\cdot \{ID_{EMPL}(oc_{EMPL,j}) = ID_{EMPL\text{-}UPDATE}(oc_{EMPL\text{-}UPDATE,k})\}$

$[v_{d,EMPL,j}(4,14,n+1) \leftarrow v_{d,EMPL\text{-}UPDATE,k}(4,14,n+1)]$

$\cdot \{s_{d,EMPL\text{-}UPDATE,k}(4,14,n+1) = I\}$

$\cdot \{v_{CODE,EMPL\text{-}UPDATE,k}(4,14,n+1) = \text{'C'}\}$

$\cdot \{v_{d,EMPL\text{-}UPDATE,k}(4,14,n+1) \neq \text{' '}\}$

$\cdot \{ID_{EMPL}(oc_{EMPL,j}) = ID_{EMPL\text{-}UPDATE}(oc_{EMPL\text{-}UPDATE,k})\}$

  3. Output source of information: each output data item must have a source of information which is either an input, computation, or history data item whose time of happening is no later than that of the output data item.
     $\forall s_{i,h,j}(t_1,\ell,n) = O_m$
     $\exists s_{i,h',j'}(t_1',\ell',n') \in \{I,C,H\} \ni$
     $T(t_1',\ell',p_{h',h}*n) \leq T(t_1,\ell,n)$

Example:

$\forall n \geq 1 \quad \forall i \in C_{TIME\text{-}CARD}(4,14,n)$

$[s_{SSN,PAY\text{-}REPORT,h}(4,14,n) \leftarrow O_1]$

$\cdot \{s_{SSN,EMPL,j}(4,14,n) = H\}$

$\cdot \{s_{SSN,TIME\text{-}CARD,i}(4,14,n) = I\}$

$\cdot \{ID_{TIME\text{-}CARD}(oc_{TIME\text{-}CARD,i}) = ID_{EMPL}(oc_{EMPL,j})\}$

$\cdot \{ID_{TIME\text{-}CARD}(oc_{TIME\text{-}CARD,i}) = ID_{PAY\text{-}REPORT}(oc_{PAY\text{-}REPORT,h})\}$

$\forall n \geq 1 \quad \forall i \in C_{TIME\text{-}CARD}(4,14,n)$

$s_{SSN,TIME\text{-}CARD,i}(4,14,n) = I \ni$

$T(4,14,n) = T(4,14,n)$

  4. No unnecessary input: each input data item must ultimately be used as the source of information either for an output or history data item or for an operand of a computation data item whose time of happening is no sooner than that of the input data item.
     $\forall s_{i,h,j}(t_1,\ell,n) = I$
     $[\exists s_{i,h',j'}(t_1',\ell',n') \in \{H,O_m\}]$
     or $[\exists s_{i',h',j'}(t_1',\ell',n') \in C$
     $\ni v_{i,h,j}(t_1,\ell,n)$ appears in <value-expression> for $v_{i',h',j'}(t_1',\ell',n')] \ni$
     $T(t_1,\ell,p_{h,h'}*n') \leq T(t_1',\ell',n')$

Example:

$\forall n \geq 1 \quad \forall i \in C_{\text{TIME-CARD}}(4,14,n)$

$s_{\text{HOURS,TIME-CARD},i}(4,14,n) = I$

$\forall n \geq 1 \quad \forall i \in C_{\text{TIME-CARD}}(4,14,n)$

COMPUTE-WAGES = {<u>SSN</u> WAGES}

$[s_{\text{WAGES,COMPUTE-WAGES},k}(4,14,n) \leftarrow C]$

$\cdot \{s_{\text{HOURS,TIME-CARD},i}(4,14,n) = I\}$

$\cdot \{ID_{\text{TIME-CARD}}(oc_{\text{TIME-CARD},i}) = ID_{\text{COMPUTE-WAGES}}(oc_{\text{COMPUTE-WAGES},k})\}$

$[v_{\text{WAGES,COMPUTE-WAGES},k}(4,14,n) \leftarrow v_{\text{RATE,EMPL},j}(4,14,n) * (v_{\text{WAGE-STATUS,EMPL},j}(4,14,n) = 2) + v_{\text{RATE,EMPL},j}(4,14,n) * v_{\text{HOURS,TIME-CARD},i}(4,14,n) * (v_{\text{WAGE-STATUS,EMPL},j}(4,14,n) = 1) * (v_{\text{HOURS,TIME-CARD},i}(4,14,n) \leq 40) + v_{\text{RATE,EMPL},j}(4,14,n) * (40 + (v_{\text{HOURS,TIME-CARD},i}(4,14,n) - 40) * 1.5) * (v_{\text{WAGE-STATUS,EMPL},j}(4,14,n) = 1) * (v_{\text{HOURS,TIME-CARD},i}(4,14,n) > 40)]$

$\cdot \{s_{\text{HOURS,TIME-CARD},i}(4,14,n) = I\}$

$\cdot \{s_{\text{RATE,EMPL},j}(4,14,n) = H\}$

$\cdot \{ID_{\text{TIME-CARD}}(oc_{\text{TIME-CARD},i}) = ID_{\text{EMPL}}(oc_{\text{EMPL},j})\}$

$\cdot \{ID_{\text{TIME-CARD}}(oc_{\text{TIME-CARD},i}) = ID_{\text{COMPUTE-WAGES}}(oc_{\text{COMPUTE-WAGES},k})\} \ni$

$T(4,14,n) = T(4,14,n)$

5. No redundant input: in any particular processing cycle, each non-identifier input data item should be defined only once.

   $\forall s_{i,h,j}(t_1,\ell,n) = I \ni d_i \notin ID_h$

   $\nexists s_{i,h',j'}(t_1,\ell,n) = I \ni h \neq h'$ and $d_i \notin ID_{h'}$.

6. No redundant history: in any particular processing cycle, each non-identifier history data item should be defined only once.

   $\forall s_{i,h,j}(t_1,\ell,n) = H \ni d_i \notin ID_h$

   $\nexists s_{i,h',j'}(t_1,\ell,n) = H \ni h \neq h'$ and $d_i \notin ID_{h'}$.

7. Output data item cannot be used as information source: When $v_{i,h,j}(t_1,\ell,n)$ appears on the right-hand side of an assignment, there does <u>not</u> exist a defining condition in the same assignment of the form: $\{s_{i,h,j}(t_1,\ell,n) = O_m\}$.

B. Dynamic analysis

1. In any input or output relational structure, all data names in the same structure must have the same cycle.
2. For each output data item, the cycle of each input or computation source of information must be equal to the cycle of the output data item.
3. For each history data item, the cycle of each input or computation source of information must be equal to the cycle of the history data item.
4. For each computation data item, the cycle of each input or computation source of information must be equal to the cycle of the computation data item.

Reports produced during requirements statement analysis include the following:

A. Data definition and static analysis

1. Specification of correspondence and unique correspondence relations between occurrences of relational structures.

   Example:

   $OC_{\text{EMPL-UPDATE}}(4,14,n+1) \sim OC_{\text{EMPL}}(4,14,n) \quad \forall n \geq 0$

   $OC_{\text{TIME-CARD}}(4,14,n) \stackrel{=}{\sim} OC_{\text{PAY-REPORT}}(4,14,n) \quad \forall n \geq 1$

2. Incidence matrix: (I_{ih}) identifies the data names d_i used to define each computational relational structure D_h. For $\{i_h,(t_1,\ell,n): d_i \in D_h,$ at $T(t_1,\ell,n)\}$ and $\{h(t_1,\ell,n): s_{k,h,j}(t_1,\ell,n) = C$ where $d_k \in D_h$ and $d_k \notin ID_h\}$:

$$I_{ih} = \begin{cases} 1 \text{ if } d_i \text{ is an operand in the computation of } d_k \\ -1 \text{ if } d_i \text{ is the result of the computation of } d_k (i=k) \\ 0 \text{ otherwise} \end{cases}$$

Example: COMPUTE-WAGE = {<u>SSN</u> WAGES}
UPDATE = {<u>SSN</u> YTD-WAGES}

	COMPUTE-WAGES(4,14,n)	UPDATE (4,14,n)
$\text{HOURS}_{\text{TIME-CARD}}(4,14,n)$	1	0
$\text{RATE}_{\text{EMPL}}(4,14,n)$	1	0
$\text{WAGES}_{\text{COMPUTE-WAGES}}(4,14,n)$	-1	1
$\text{YTD-WAGES}_{\text{UPDATE}}(4,14,n-1)$	0	1
$\text{YTD-WAGES}_{\text{UPDATE}}(4,14,n)$	0	-1

3. Precedence matrix: (P_{ik}) indicates the data names d_i that must be available before each computational data name d_k can be computed. For $\{i_h,(t_1,\ell,n): d_i \in D_h,$ at $T(t_1,\ell,n)\}$ and $\{k(t_1,\ell,n): s_{k,h,j}(t_1,\ell,n) = C$ where $d_k \in D_h$ and $d_k \notin ID_h\}$:

$$P_{ik} = \begin{cases} 1 \text{ if } d_i \text{ is an operand in the computation of } d_k \\ 0 \text{ otherwise} \end{cases}$$

Example:

	WAGES (4,14,n)	YTD-WAGES (4,14,n)
$\text{HOURS}_{\text{TIME-CARD}}(4,14,n)$	1	0
$\text{RATE}_{\text{EMPL}}(4,14,n)$	1	0
$\text{WAGES}_{\text{COMPUTE-WAGES}}(4,14,n)$	0	1
$\text{YTD-WAGES}_{\text{UPDATE}}(4,14,n-1)$	0	1

. Dynamic analysis
Calendar indicating times of happening of:
1. Writing of output relational structures
2. Reading of input relational structures
3. Reading/writing of history relational structures
4. Execution of computation relational structures.

NCLUSION

The criteria for language facilities of an RSL nd for logical capabilities of an RSA are speciied via the vehicle of a formal model of an inormation system. Most of the criteria for logcal capabilities of an RSA are fulfilled by an nplementation of an RSA for ADS developed at the niversity of Michigan [29] and extended at Purdue niversity [20].

KNOWLEDGEMENT

This work was supported in part by the ISDOS roject under the direction of Professor Daniel eichroew, University of Michigan and in part by rant GJ31572 of the Office of Computing Activities, ational Science Foundation.

FERENCES

. Teichroew, D. 1972. A survey of languages for stating requirements for computer-based information systems. Proc. AFIPS 1972 FJCC 41, Montvale, N.J.: AFIPS Press, 1203-1224.
. Couger, J. D. and Knapp, R. W. 1974. System Analysis Techniques. New York: John Wiley and Sons.
. Leavenworth, B. M. and Sammet, J. E. 1974. An overview of nonprocedural languages. SIGPLAN Notices 9, 4 (April 1974), 1-12.
. Benjamin, R. I. 1972. A generational perspective of information system development. Comm. ACM 15, 7 (July 1972), 640-643.
. Merten, A. and Teichroew, D. 1972. The impact of problem statement languages on evaluating and improving software performance. Proc. AFIPS 1972 FJCC 41, Montvale, NJ: AFIPS Press, 849-857.
. McGee, W. C. 1974. Review of: The programmer as navigator. Computing Reviews 15, 3 (March 1974), 107.
. Pridmore, H. D. 1967. The abstract information system concept and the problem of optimum design. Proc. Third Australian Computer Conf., Chippendale, N.S.W., Australia: Australian Trade Publications, 149-170.
. Dijkstra, E. W. 1970. Notes on structured programming. T. H.-Report 70-WSK-03, Department of Mathematics, Technological University, Eindhoven, The Netherlands (April 1970).
. Teichroew, D. and Sayani, H. 1971. Automation of system building. Datamation 17, 16 (August 15, 1971), 25-30.
Young, J. W. Jr. and Kent, H. K. 1958. Abstract formulation of data processing problems. Journal of Industrial Engineering (Nov.-Dec. 1958), 471-479.
CODASYL Development Committee. 1962. An Information algebra phase I report. Comm. ACM 5, 4 (April 1962), 190-204.
Langefors, B. 1963. Some approaches to the theory of information systems. BIT 3 (1963), 229-254.
13. Langefors, B. 1965. Information system design computations using generalized matrix algebra. BIT 5 (1965), 96-121.
14. Lombardi, L. A. 1964. A general business-oriented language based on decision expressions. Comm. ACM 7, 2(Feb. 1964), 104-111.
15. National Cash Register Company, 1968. A Study Guide for Accurately Defined Systems. Dayton, Ohio.
16. Lynch, H. J. 1969. ADS: a technique in system documentation. Database 1, 1 (Spring 1969), 6-18.
17. Myers, D. H. 1962. A time automated technique for the design of information systems. IBM Systems Research Institute.New York (1962).
18. Kelly, J. F. 1970. Computerized Management Information Systems. New York: Macmillan, Ch. 8.
19. Katz, J. H. and McGee, W. C. 1963. An Experiment in non-procedural programming. Proc. AFIPS 1963 FJCC 23, NY: Spartan Books, 1-13.
20. Nunamaker, J. F. Jr.; Ho, T.; Konsynski, B.; and Singer, C. 1973. Specification and design of an information system using computer-aided analysis. CSD TR,CS Dept., Purdue University, West Lafayette, Ind. (September 1973).
21. Cattell, R. Duncan Electric Company, Lafayette, Indiana. Personal communication.
22. Nunamaker, J. F. Jr. 1971. A methodology for the design and optimization of information processing systems. Proc. AFIPS 1971 SJCC 38, Montvale: AFIPS Press, 283-294.
23. Koch, R. F.; Krohn, M. J.; McGrew, P. W.; and Sibley, E. H. 1970. PSL Version 2, Release 1: A PSL Language Primer. ISDOS Working Paper No. 33, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan (August 1970).
24. Hershey, E. A.; Rataj, W. J.; and Teichroew, D. 1973. PSL/II Language Specifications Version 1.0. ISDOS Working Paper No. 68, Dept. of Industrial & Operations Engineering, Univ. of Mich., Ann Arbor, Mich. (February 1973).
25. Teichroew, D. 1971. Problem statement analysis: requirements for the Problem Statement Analyzer (PSA). ISDOS Working Paper No. 43, Dept. of Industrial & Operations Engineering, Univ. of Mich., Ann Arbor, Mich. (April 1971).
26. Wrigley, B. H. Chairman's remarks to the Wharton Conference on Research on Computers in Organizations. Data Base 5, 2-4 (Winter 1973), i-ii.
27. Codd, E. F. 1970. A relational model of data for large shared data banks. Comm. ACM 13, 6 (June 1970), 377-387.
28. Codd, E. F. 1971. Normalized data base structure: a tutorial. In Proc. 1971 ACM SIGFIDET Workshop: Data Description, Access & Control, New York: ACM (November 1971), 1-17.
29. Thall, R. M. 1973. A manual for PSA/ADS: a machine-aided approach to analysis of ADS. ISDOS Working Paper No. 35, Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan (February 1973).