1973

Specification and Design of an Information System Using Computer Aided Analysis

J. F. Nunamaker

Report Number:

74-126

SPECIFICATION AND DESIGN OF AN INFORMATION SYSTEM USING COMPUTER AIDED ANALYSIS

J. F. Nunamaker, Jr.
Thomas Ho
Benn Konsynski
Carl Singer

August 30, 1973

CSD-TR 126

SPECIFICATION AND DESIGN OF AN INFORMATION SYSTEM USING COMPUTER AIDED ANALYSIS*

by

J. F. Nunamaker, Jr., Thomas Ho, Benn Konsynski and Carl Singer

ABSTRACT

This paper describes the use of computer aided analysis for the design and development of an integrated financial management system by the Navy Material Command Support Activity.

Computer aided analysis consists of a set of procedures and computer programs specifically designed to aid in the process of applications software design, computer selection and performance evaluation. Computer aided analysis consists of four major components: Problem Statement Language, Problem Statement Analyzer, Generator of Alternative Designs, and Performance Evaluator.

The statement of requirements was written in ADS (Accurately Defined Systems) and analyzed by a Problem Statement Analyzer for ADS developed at the University of Michigan and extended at Purdue University. The ADS problem definition was supplemented with additional information in order to create a complete problem definition.

The analyzed problem statement was then translated to the form necessary for use by the SODA (Systems Optimization and Design Algorithm) Program for the generation of alternatives and performance evaluation. Fundamental to the SODA approach is the automatic generation of designs of program structure and data structure. This is the point at which SODA differs from the commercially available simulation packages.

It was necessary to supplement the ADS definition with additional information needed to perform the SODA analysis. This additional information was defined in SODA Statement Language.

The paper focuses on the use of ADS as a problem definition technique and the use of SODA as a design aid for the specification of program modules and logical data base structure.

The procedures and programs described are presently being incorporated into a framework that facilitates man-machine interaction for problem definition and information systems design.

*Computer Sciences Department
Purdue University
West Lafayette, Indiana 47906

SPECIFICATION AND DESIGN OF AN INFORMATION SYSTEM USING COMPUTER AIDED ANALYSIS*

by

J. F. Nunamaker, Jr. , Thomas Ho, Benn Konsynski and Carl Singer

INTRODUCTION

The problems inherent with the increasing use of the computer for information systems applications have provided the motivation for the development of tools for automating the production of applications software. The current methods for building information systems contain numerous deficiencies [1] and computer-aided analysis of user requirements is proposed as the first step toward automated systems building [2].

OVERVIEW OF COMPUTER AIDED ANALYSIS

In this paper we describe: (1) a number of software systems for computer aided analysis and (2) how they were used to aid the Navy Material Command Support Activity with the design of a large information system.

The activities performed by the systems for computer aided analysis consisted of:

1. Procedures for stating processing requirements.
2. Automatic analysis of processing requirements.
3. The design of program structure; i.e., determining how many modules must be generated and the size of each module.
4. The design of logical file structures and logical data base.
5. Selection of hardware, including:
   - Central processing unit
   - Core memory size
   - Auxiliary memory
   - Input/output configuration

* Computer Sciences Department
Purdue University
West Lafayette, Indiana 47906

6. The allocation of files to storage devices.
7. The optimal selection of blocking factors for each file.

An overview of the software system for computer aided analysis is shown in Figure 1. Each of the major components is described below.

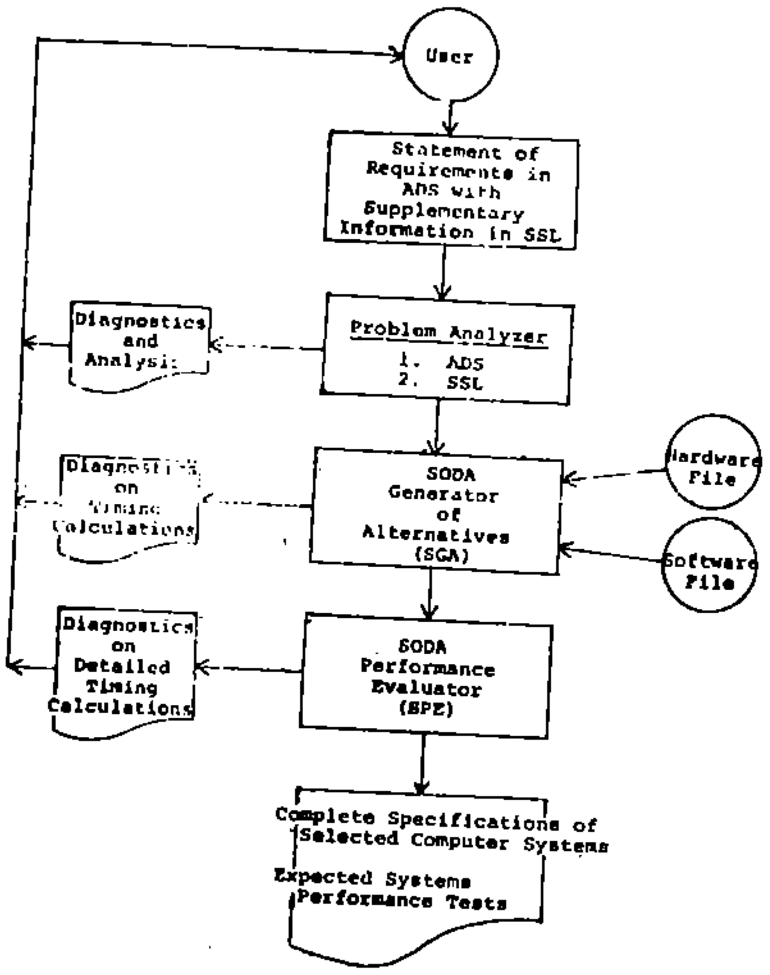
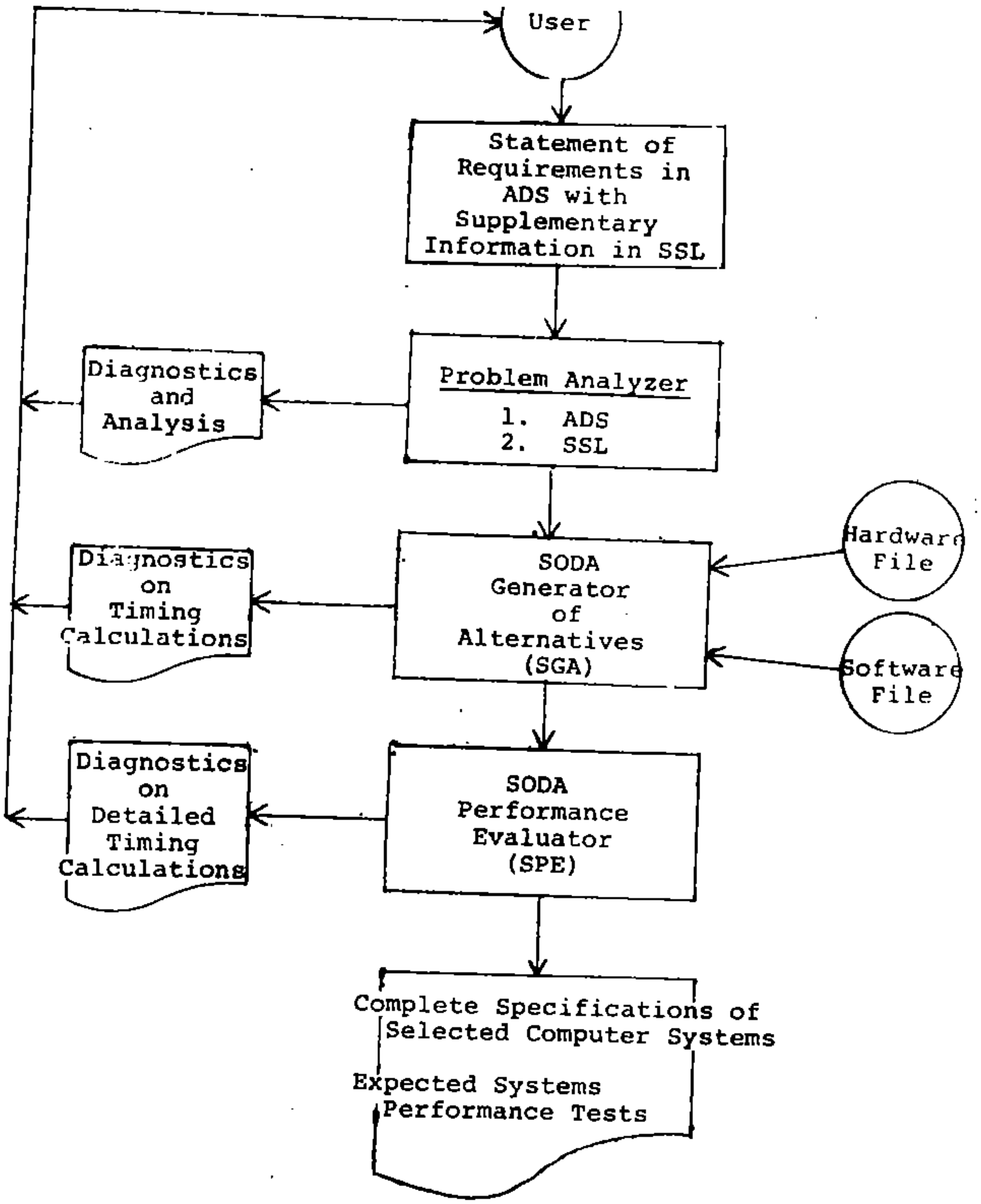

User
Statement of Requirements in ADS with Supplementary Information in SSL
Problem Analyzer
1. ADS
2. SSL
Diagnostics and Analysis
SODA Generator of Alternatives (SGA)
Diagnostics on Timing Calculations
Hardware File
Software File
SODA Performance Evaluator (SPE)
Diagnostics on Detailed Timing Calculations
Complete Specifications of Selected Computer Systems
Expected Systems Performance Tests

Figure 1: SODA (System Optimization and Design Algorithm)

PROBLEM STATEMENT TECHNIQUES

Past experience with problem statement techniques indicated that no existing problem statement technique was adequate for the complete expression of user requirements relevant to all aspects of systems design and optimization. Hence, two techniques: (1) Accurately Defined Systems (ADS) and (2) SODA Statement Language (SSL), were used.

The combined problem statement consists of information on data volumes and frequency of input and output items. The data


User
Statement of Requirements in ADS with Supplementary Information in SSL
Diagnostics and Analysis
Problem Analyzer
1. ADS
2. SSL
Diagnostics on Timing Calculations
SODA Generator of Alternatives (SGA)
Hardware File
Software File
Diagnostics on Detailed Timing Calculations
SODA Performance Evaluator (SPE)
Complete Specifications of Selected Computer Systems
Expected Systems Performance Tests

description, processing requirements, operational requirements, and time and volume information are expressed in units specified by the problem definer. The problem statement must contain sufficient detail so that systems analysts and programmers could use it to design and implement the information system with no additional information.

ADS is form oriented and is used to obtain much of the basic problem definition. SSL is used to express system design parameters and performance requirements, e.g., I/O volumes and frequencies for system design and performance optimization.

Accurately Defined Systems

Accurately Defined Systems (ADS) is a product of the National Cash Register Company (1968) and is described by Lynch [6] and NCR [7]. ADS consists of a set of forms and procedures for systematically recording the information that a systems analyst would gather during compilation of the user requirements for the information system to be implemented. The essential elements of an ADS requirements statement include descriptions of:

1. Inputs to the information system.
2. Historical data stored by the information system.
3. Outputs produced by the information system.
4. Actions required to produce these outputs and the conditions under which each action is performed.

SODA Statement Language

SODA Statement Language (SSL) consists of a set of forms for systematically gathering data on the volumes and frequencies of system inputs and outputs described in the ADS statement. The essential elements of an SSL statement include requirements data for:

1. Inputs to the batch processing subsystem.
2. Queries to the teleprocessing subsystem.
3. Reports produced by the information system.

PROBLEM STATEMENT ANALYZER

Both the ADS analyzer and the SODA Statement Analyzer accept the requirements stated in the respective languages, analyze them, and provide the problem definer with diagnostics for debugging his problem statements and reports.

ADS Analyzer

Computer-aided analysis of an ADS statement performs a number of checks and prepares a series of summaries of the statement of user requirements. The simplest kind of check performed involves the validation of ADS source statements to uncover any violations of the syntax rules of ADS problem statement. Rules relating to naming conventions, numbering conventions, information linking, and the like are specified to guide the user during problem definition.

More complex checks of logical consistency and completeness indicate errors in data element definition and in linking of information sources. Major errors of a logical nature include the use of data elements not defined elsewhere in the ADS statement and the redundant definition of data elements with multiple occurrences in the ADS statement. Less serious errors involve historical data elements for which no update procedures have been specified and definition of data elements not used elsewhere in the ADS statement.

Summary reports produced by computer-aided analysis include a directory of all data element occurrences, indexes to all data elements and processes, matrices indicating the data elements required by each process and the precedence relationships among data elements, and graphical displays of the ADS forms submitted for analysis. The data element directory consists of an alphabetical list of the data elements defined in the ADS statement, the places of occurrence of each element, and the information source of each occurrence. The indexes assign a unique number to each data element and process for identifying row and column positions in the matrices indicating incidence and precedence relationships. The incidence matrix uses process numbers as row indexes and data element numbers as column indexes to identify the data elements used in each computational process. The precedence matrix uses data element numbers as both row and column indexes to indicate, for each data element, the data elements that must be computed before the first data element can be calculated. Finally, the graphical reports display the five kinds of ADS forms in the tabular manner as they would appear in input or manual use of ADS.

SSA (SODA Statement Analyzer)

SSA produces a number of networks which record the interrelationships of processes and data and passes the networks on to the SODA program concerned with the generation of alternative designs.

Each type of input and output is specified in terms of the data involved, the transformation needed to produce output from input, and stored data. Time and volume requirements are also stated. SSA analyzes the statement of the problem to determine whether the required output can be produced from the available inputs. The problem statement stored in machine-readable form is processed by SSA which:

1. Checks for consistency in the problem statement and checks syntax in accordance with SSL; i.e., verifies that the problem statement satisfies SSL rules and is consistent, unambiguous, and complete.
2. Prepares summary analyses and error comments to aid the problem definer in correcting, modifying, and extending his problem statement.
3. Prepares data to pass the problem statement on to SGA.
4. Prepares a number of matrices that express the interrelationship of processes and data.

SGA (SODA Generator of Alternatives)

The process of designand selection begins after the requirements have been stated, verified, and analyzed in SSA. SGA accepts, as input, the output of SSA and a statement of the available computing resources, hardware, and utility programs. The hardware and software file consists of data for the computer systems under consideration. Extensive data exists for the following computer systems.

UNIVAC	1100 Series
Control Data	6000 Series
IBM	360/370 Series

Fundamental to the SODA approach is the automatic generation of designs of Program Structure and File Structure. This is the point at which SODA differs from other techniques such as SCERT and CASE. In order to use either SCERT or CASE it is necessary that a system already be designed to obtain answers regarding feasibility. However, SODA will generate the alternative designs or evaluate a set of manual designs.

The user has three options with respect to the generation of alternative system designs:

- Consider only SODA generated designs
- Consider only designs generated manually
- Consider both sources of designs.

It should be emphasized that SODA will automatically generate designs; the option is available, however, for the user to input some as well.

If the user decides to input designs, he must do so according to a pre-determined format.

SGA is concerned with the problem of producing an operationally feasible design. SGA takes the information from SSA, analyzes the alternative hardware and software information with respect to a specific design, and generates the specifications for the necessary CPU, core size, program structure, and data structure. SGA essentially computes the expected processing time required for alternative designs for each period of time.

SPE (SODA Performance Evaluator)

SPE involves examining the operationally feasible design in an attempt to improve system performance.

The optimization and performance evaluation phase generates a storage structure and schedule, selects auxiliary memory devices, and searches for ways to improve the IPS design. SPE may return control to SGA to select another CPU or core size or to select another set of Program Modules and Files.

SPE selects the minimum cost hardware configuration that is capable of processing the stated requirements in the time available. This phase consists of a number of mathematical programming models and timing routines that are used to (1) optimize the blocking factors for all Files, (2) evaluate alternative designs; i.e., specify the number and type of auxiliary memory devices, (3) assign Files to memory devices, and (4) generate an operating schedule for running Program Modules.

In SPE, the performance criterion is optimized within the constraint set by the capability of the hardware and by the processing requirements. SPE produces reports describing the system and stating its predicted performance. On the basis of these reports, the user may decide to change his problem statement or accept the design; SPE then provides detailed specifications for the construction of the system or the selection of a specific computer system. These reports include:

1. A list specifying which of the available computing resources will be used; i.e., which computer system is required to do the job.
2. A list of the Program Modules specifying the input, output, and computations to be performed in each.
3. A list of Files to be maintained specifying their format and manner in which they will be stored, i.e., an assignment of Files to memory devices.
4. A statement of the sequence and manner in which the Program Modules must be run to accomplish all the requirements.

The magnitude of the information required to describe large information systems motivated the need for a man-machine interactive systems design package which leads to the development of GPLAN/SODA described in the last section.

PROBLEM DEFINITION

As stated previously no single problem statement technique presently available is adequate for complete description of the system as required for computer aided analysis. As a result, two problem statement languages were selected for expression of user requirements relevant to all aspects of system design and optimization:

1. Accurately Defined Systems (ADS) to provide a non-procedural description of user requirements to all SODA packages performing system design, e.g. SGA.
2. A sublanguage of SODA Statement Language (SSL) to express performance requirements, e.g. I/O volumes and frequencies to all SODA packages for system performance optimization, e.g. SPE.

The ADS requirements statement begins with the definition of all system outputs. Then definition continues with the identification of information that enters the system in order to describe inputs to the system. Finally, the requirements statement is completed with the definition of historical data retained in the system for a period of time and with the specification of computations and accompanying logic that subsequently use the input and historical data to produce the system outputs.

Linking of information elements among the various ADS definitions is accomplished in two ways. First, each element of data is assigned a unique name that is always used whenever that element appears in any ADS definition. Second, each use of a data element in a report, history, or computation definition is linked back to its information source elsewhere in the ADS description. Hence, all data elements are chained from output to input and each output can ultimately be expressed in terms of inputs to the system. Chaining is accomplished by assigning page and line numbers to all ADS forms so that each use of a data element can be uniquely identified by the form, page, and line on which the element appears.

An example of an ADS requirements statement will demonstrate the effectiveness of the concepts described above. The ADS example describes the requirements of an application for payroll calculation.

The application produces an output report listing social security number, name, and current pay period wages for each employee. Also, the application includes a master file containing the following information in each employee record:

1. Social security number.
2. Name.
3. Wage status.
4. Hourly rate or pay period salary.
5. Year-to-date wages.

Input to the application is a set of time cards containing the pay period date, employee social security number, and number of hours worked during the pay period.

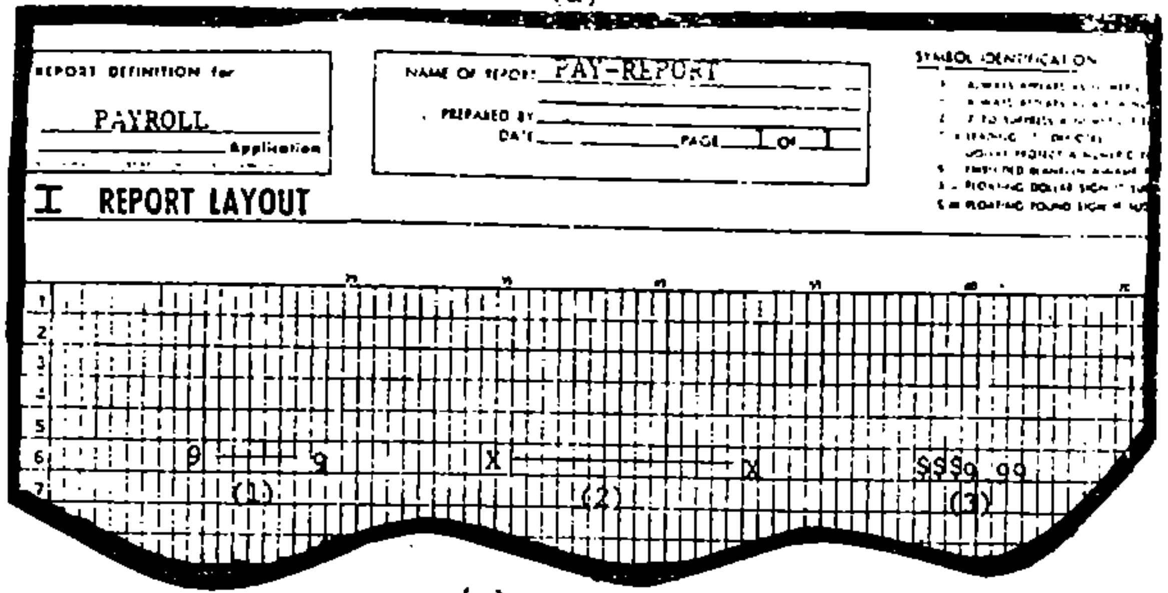
Computations include two types: current wage calculation and year-to-date wage calculation. Current wage calculation is performed for both salaried and hourly paid employees. Hourly calculations are further subdivided into straight-time calculation and overtime calculation. Finally, the logic definition form presents a decision table specifying the conditions under which each computation is performed.

Note the facility for cross-referencing data elements among the various forms. For example, Section III of the report definition form in Figure 2a specifies the source of each element on the report. Similarly, each entry in the history and computation definition forms in Figure 3 includes an indication of the source of the data element specified. Since this example includes only wage calculation and not master file maintenance, the source of all history data elements cannot be specified here. Furthermore, the forms may be incomplete in other respects due to the omission of non-essential details, e.g. report headings, in this example.

In Figure 2a, the Report Definition Form describes the printed output produced by the application. Section I documents the layout of the report by using the symbols identified in the upper right-hand corner to describe the printed fields. The number in parentheses below each field refers to the numbered items in Section III. Section III identifies the source of each data item appearing on the report. Cross-referencing is achieved by specifying H, C, or I for history, computation, or input respectively and by specifying page and line numbers that appear on every form. Section IV shows the sequence in which the output data is listed on the report.

Figure 2b is the Input Definition Form, a description of the input to the source program. Section I describes the format of the input record and is linked to the complete description of each field in Section II. Section II identifies the alphabetic, numeric, or alphanumeric character of each field and its size in number of characters.

(a)



REPORT DEFINITION for PAYROLL Application

NAME OF REPORT PAY-REPORT
PREPARED BY
DATE PAGE 1 OF 1

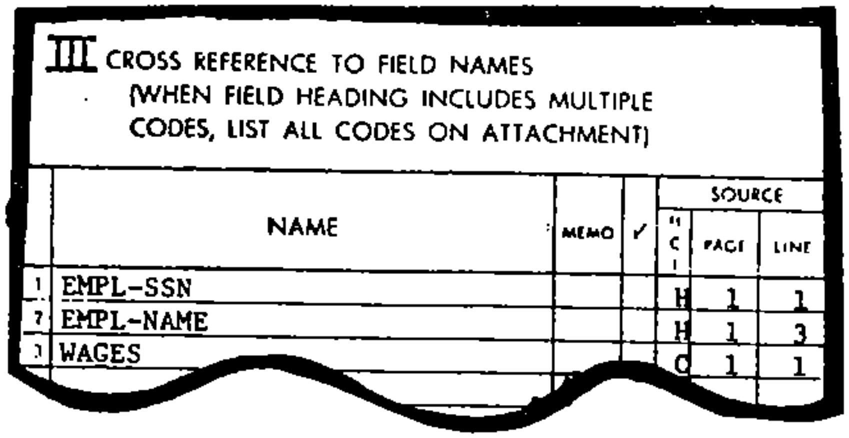
SYMBOL IDENTIFICATION

I REPORT LAYOUT

6	9 ——— 9	X ——— X	$$$9.99
7	(1)	(2)	(3)

Figure 2: Report and Input Definition Forms

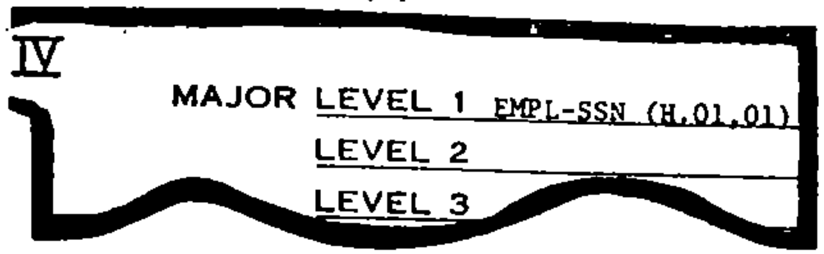
(a)



III CROSS REFERENCE TO FIELD NAMES (WHEN FIELD HEADING INCLUDES MULTIPLE CODES, LIST ALL CODES ON ATTACHMENT)

	NAME	MEMO	✓	SOURCE		
				H C	PAGE	LINE
1	EMPL-SSN			H	1	1
2	EMPL-NAME			H	1	3
3	WAGES			C	1	1

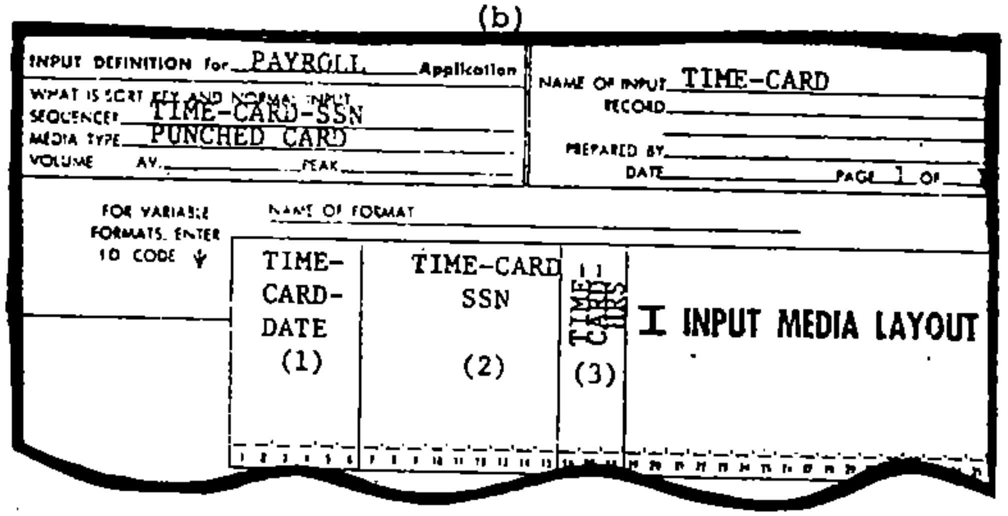
(a)



IV

MAJOR LEVEL 1 EMPL-SSN (H,01,01)
LEVEL 2
LEVEL 3

(b)



INPUT DEFINITION for PAYROLL Application
WHAT IS SORT KEY AND NORMAL INPUT SEQUENCE? TIME-CARD-SSN
MEDIA TYPE PUNCHED CARD
VOLUME AV. PEAK

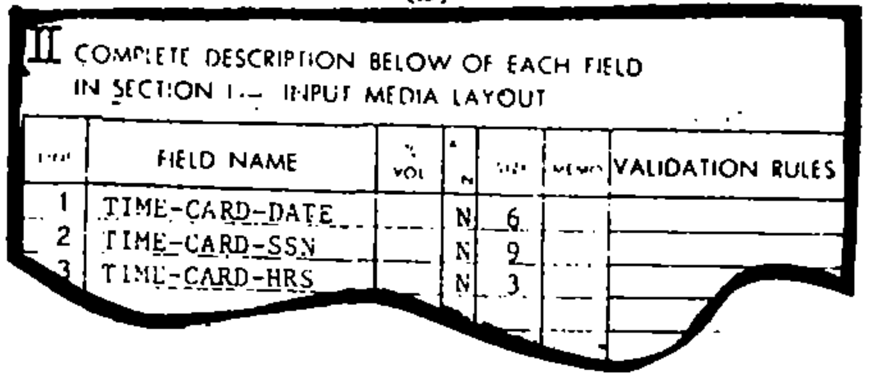
NAME OF INPUT TIME-CARD
RECORD
PREPARED BY
DATE PAGE 1 OF

FOR VARIABLE FORMATS, ENTER ID CODE
NAME OF FORMAT

TIME-CARD-DATE	TIME-CARD SSN	TIME CARD HRS
(1)	(2)	(3)

I INPUT MEDIA LAYOUT

(b)



II COMPLETE DESCRIPTION BELOW OF EACH FIELD IN SECTION I — INPUT MEDIA LAYOUT

	FIELD NAME	% VOL	A N	SIZE	MEMO	VALIDATION RULES
1	TIME-CARD-DATE		N	6		
2	TIME-CARD-SSN		N	9		
3	TIME-CARD-HRS		N	3		

The History Definition Form, a description of the master file maintained by the application, appears in Figure 3a. Again, each field is completely described. In addition, the memo entry in line 5 refers to an explanation of the wage status code in the memo list that actually appears on the Input Definition Form.

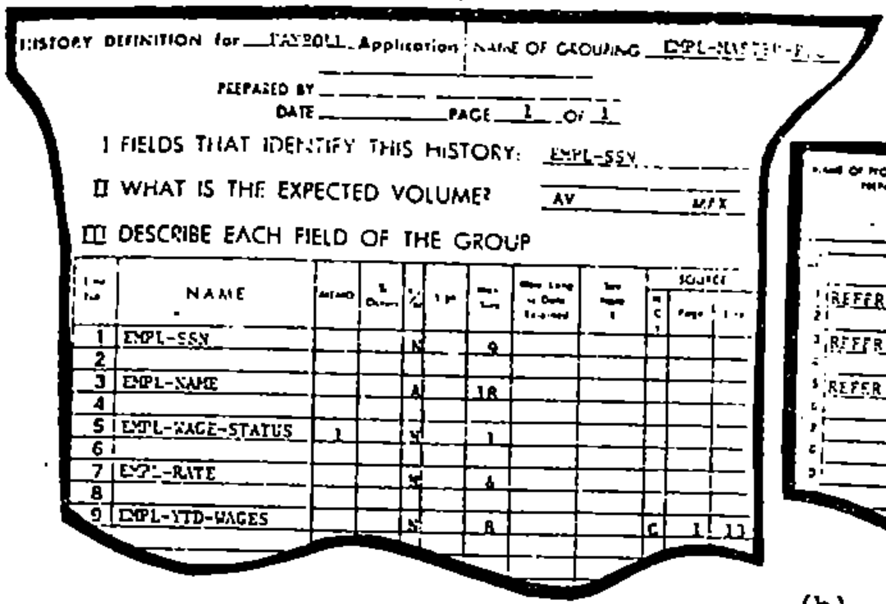
The Computation and Logic Definition Forms are displayed in Figure 3b. The Computation Definition Form lists the variables to be computed and the factors needed to perform the computations. Again, the source of each factor is specified. The entry in the sign column identifies the arithmetic operation to be performed. Since only binary operators are allowed, temporary variables must be generated for intermediate results and are given mnemonic names here for clarity. The Logic Definition Form represents a decision table that specifies the conditions under which each computation is performed. The computations are listed across the top and linked to the Computation Definition Form while the conditions are specified down the righthand side.

ADS possesses obvious advantages over the traditional narrative requirements statement technique. Narrative statements are ambiguous and often incomplete while ADS provides a standardized and systematic approach to system definition. Still, ADS is both exact and precise while remaining hardware independent. ADS promotes effective communication among systems personnel by imposing a discipline that enables the efficient use of human and machine resources. Development time is reduced while software quality is enhanced because the ADS technique enables checking for accuracy, consistency, and completeness of the requirements statement. Above all, dollar savings are realized with the use of ADS for problem definition.

To determine the computer resource demands of the information system for which the design process was performed, additional data supplementary to the ADS statement was necessary. This need required the following data for each input and report described in the ADS statement:

1. ADS page number.
2. Frequency of occurrence.
3. Volume.
4. Brief description.

(a)



HISTORY DEFINITION for __PAYROLL__ Application | NAME OF GROUPING __EMPL-MASTER-F[illegible]__

PREPARED BY ____________

DATE ____________ PAGE __1__ OF __1__

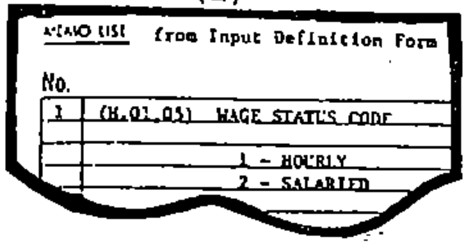
I FIELDS THAT IDENTIFY THIS HISTORY: __EMPL-SSN__

II WHAT IS THE EXPECTED VOLUME? AV ______ MAX ______

III DESCRIBE EACH FIELD OF THE GROUP

Line No.	NAME	MEMO	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	SOURCE		
									[illegible]	Page	Line
1	EMPL-SSN			N		9					
2											
3	EMPL-NAME			A		18					
4											
5	EMPL-WAGE-STATUS	1		N		1					
6											
7	EMPL-RATE			N		4					
8											
9	EMPL-YTD-WAGES			N		8			C	1	11

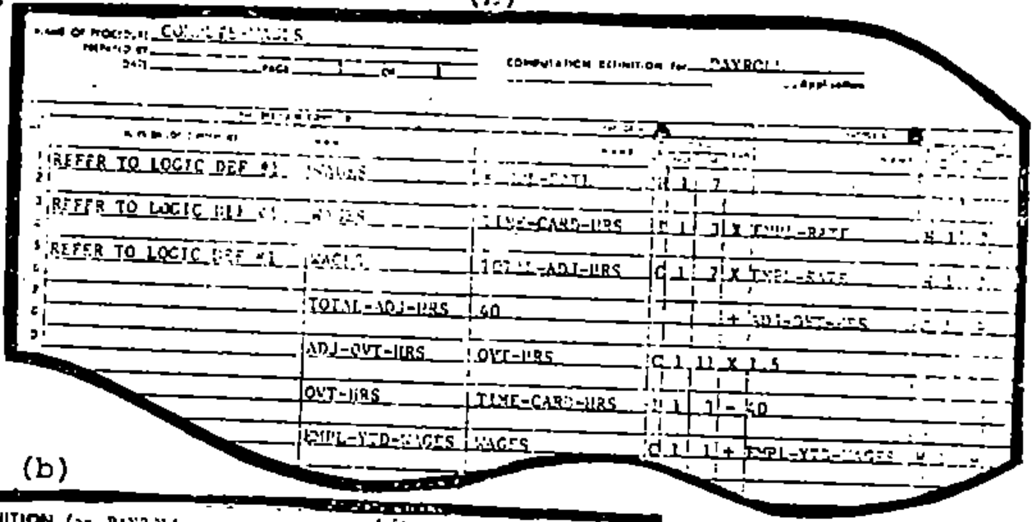
(a)



MEMO LIST from Input Definition Form

No.	
1	(H.01.05) WAGE STATUS CODE
	1 - HOURLY
	2 - SALARIED

(b)



NAME OF PROCEDURE __COMPUTE-WAGES__

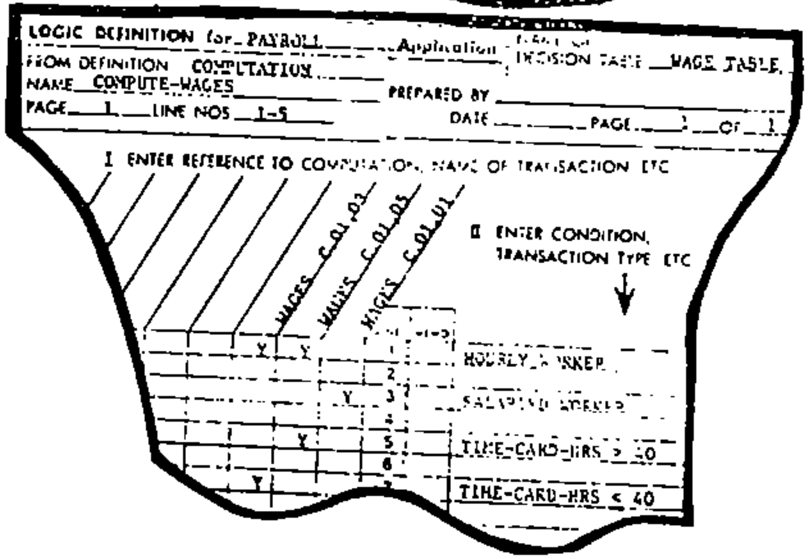
PREPARED BY ____________

DATE ____________ PAGE __1__ OF __1__

COMPUTATION DEFINITION for __PAYROLL__ Application

REFER TO LOGIC DEF #1	WAGES	[illegible]	H 1	7			
REFER TO LOGIC DEF #1	WAGES	TIME-CARD-HRS	H 1	1	X	EMPL-RATE	H 1 [illegible]
REFER TO LOGIC DEF #1	WAGES	TOTAL-ADJ-HRS	C 1	7	X	EMPL-RATE	[illegible]
	TOTAL-ADJ-HRS	40			+	ADJ-OVT-HRS	[illegible]
	ADJ-OVT-HRS	OVT-HRS	C 1	11	X	1.5	
	OVT-HRS	TIME-CARD-HRS	[illegible] 1	1	-	40	
	EMPL-YTD-WAGES	WAGES	C 1	1	+	EMPL-YTD-WAGES	[illegible]

(b)



LOGIC DEFINITION for __PAYROLL__ Application | DECISION TABLE __WAGE TABLE__

FROM DEFINITION __COMPUTATION__

NAME __COMPUTE-WAGES__ PREPARED BY ____________

PAGE __1__ LINE NOS __1-5__ DATE ____________ PAGE __1__ OF __1__

I ENTER REFERENCE TO COMPUTATION, NAME OF TRANSACTION ETC

II ENTER CONDITION, TRANSACTION TYPE ETC

				WAGES C.01.03	WAGES C.01.05	WAGES C.01.01		
			Y	Y			1	HOURLY WORKER
							2	
					Y		3	SALARIED WORKER
							4	
				Y			5	TIME-CARD-HRS > 40
							6	
			Y				7	TIME-CARD-HRS < 40

Figure 3: History, Computation and Logic Definition Forms

This information was represented on SSL forms along with Query profile information which included frequency, size, source, and file reference information.

ADS ANALYZER

The first module of the Problem Statement Analyzer for ADS (PSA/ADS) performs source deck validation, lists the input cards, creates a file containing all valid card images, and constructs a dictionary table to be used by other PSA/ADS modules. Source deck validation checks compliance with ADS syntax rules and detects errors that include:

1. Specification of an illegal form type, i.e., neither Report, Input, History, Computation, nor Logic.
2. Improper form format.
3. Illegal data element name.
4. Invalid page or line numbering.

For each valid ADS entry, the dictionary table records:

1. Place of occurrence.
   a. Form type.
   b. Page number.
   c. Line number.
2. Data element name.
3. Information source.
   a. Form type.
   b. Page number.
   c. Line number.

Then, the dictionary is sorted, in ascending order, according to the following keys listed in major to minor order:

1. Data element name.
2. Place of occurrence.
   a. Form and entry type.
   b. Page number.
   c. Line number.

The second module of PSA/ADS prints the data element directory and constructs a symbol table containing all data element names in alphabetical order. Obtained from the sorted dictionary table, the data element directory lists the data elements in alphabetical order and provides the following information for each data element:

1. Place(s) of occurrence.
    a. Form type.
    b. Page number.
    c. Line number.
2. Information source(s).
    a. Form type.
    b. Page number.
    c. Line number.

During directory printing, the second module performs logical checks to detect the following errors and warnings:

1. ERROR: NO SOURCE OF INFORMATION.
   A data element has been used, but it has never been defined as an input or as the result of a computation.
2. ERROR: ID IS NOT IN BODY OF FORM.
   A data element has been defined as an identifier, usually for sequencing purposes, of a data grouping that appears on a History or Input Definition Form, but the identifier does not appear as one of the data elements defined in the body of the form.
3. WARNING: NO UPDATE FOR HISTORY.
   A data element has been defined in a History Definition Form, but the element has not been defined as a result of a computation. This situation is an error only if the data element represents cumulative data, e.g., year-to-date total. If the data element represents relatively constant data, e.g., employee address, that is updated from input elements, this situation is not an error.
4. WARNING: NOT USED.
   A data element has been defined as an input or as a result of a computation, but it is not subsequently used as an operand in a computation, as a report or history item, or as a decision variable in a Logic Definition Form.
5. WARNING: REDUNDANT INPUTS.
   A data element appears on more than one Input Definition Form in which the element is not used as an identifier, e.g., for sequencing purposes Hence, only those input definitions using that data element as an identifier are probably necessary.
6. WARNING: REDUNDANT HISTORIES.
   A data element appears on more than one History Definition Form in which the element is not used as an identifier, e.g., for sequencing purposes Hence, only those history definitions using that data element as an identifier are probably necessary.

7. WARNING: BOTH INPUT AND COMPUTED.
   A data element has been defined as both an input and the result of a computation, but it does not appear as an operand in a computation. Unless the input data element is being used to verify the computed data element, either the input or computation definition is unnecessary.

8. ERROR: INVALID BACK REFERENCE.
   A data element has been defined with an information source that is not valid. Possible causes include specification of a report definition item as an information source, specification of a non-existent page or line number, and reference to an ADS entry (as an information source) where the desired data element does not exist.

9. ERROR: NO SOURCE OF INFORMATION.
   A data element has been defined for which no information source can be found, i.e., no other definition of that element can be found on any Input, History, or Computation Definition Form.

Also, the second module assigns a unique number to each data element and prints an alphabetical list of the data elements used in the ADS statement. Then, the sorted dictionary table is again sorted, in ascending order, according to the following keys, listed in major to minor order:

1. Form type (numeric)
   a. Report: form type = 1
   b. Input: form type = 2
   c. Computation: form type = 3
   d. Logic: form type = 4
   e. History: form type = 5
2. Page number.
3. Line number.
4. Entry type (each form consists of different entry types).

The third module of PSA/ADS creates a file containing records of the computational processes defined in the ADS statement, prints a list of the computational processes, and generates matrices displaying the incidence and precedence relationships among the data elements and processes defined in the ADS statement. The third module reads entries from the twice-sorted dictionary table and for each computation entry, the module writes one or more (depending on the number of operands in the computation) records on the file of computational processes. Each record has the form:

1. Symbol table pointer of the data element that appears as the result of the computation entry.
2. Symbol table pointer of the data element that appears as an operand of the computation for which the first pointer identifies the result.

At the same time, the third module inserts ADS form page delimiters into the card image file produced by the first module for forms printing by the fourth module. The process file is then sorted in ascending order. Since the data elements were placed in the symbol table in alphabetical order by the second module, this sort lists the processes in alphabetical order and the operands in alphabetical order within each process. Then, the third module generates the incidence matrix indicating the data elements that serve as result and as operands for each process. These relationships are easily derived from the result-operand pairs in the sorted process file. Also, an alphabetical list of the processes is generated with the operands of each process listed alphabetically. Again, the sorted process file is sorted in ascending order according to the following keys in major to minor order:

1. Symbol table pointer of operand.
2. Symbol table pointer of result.

Finally, the twice-sorted process file is used to generate the precedence matrix indicating the direct precedents of each process. Data element I is said to be a precedent of data element J if I must be computed before J can be computed. A <u>direct</u> precedent of J is a precedent of J that is not also a precedent of any other precedents of J. To generate the precedence matrix, the module reads each record in the twice-sorted process file and identifies the operand data element indicated in the second field of the record as a direct precedent of the process result data element indicated in the first field of the same record.

Finally, the card image file created by the first module is sorted, in ascending order, according to the following keys in major to minor order:

1. Form type (numeric, see keys of dictionary sort for legend).
2. Page number.
3. Line number.

4. Entry type.

The fourth and final module reads the sorted card image file and prints the input in a tabular format similar to that of the ADS forms developed by NCR.

PROCESS GENERATION AND PROGRAM MODULE SPECIFICATIONS FROM ADS DEFINITION

The ADS problem statement contains the basic information required to generate program module specifications from processes that may be grouped into program modules to eliminate unnecessary transport of data from history files to program modules. For example, if it is determined that two processes require the same inputs and occur in the same processing cycle, e.g. daily, then the two processes become candidates for grouping into a single program module.

SODA Generator of Alternatives (SGA) performs process generation by compiling four comprehensive summaries for each ADS-described report:

1. Input summary.
2. History input summary.
3. Computation summary.
4. History output summary.

Since the source of each report item is specified in the ADS statement, all sources that are either input items or history items are included in the input and history input summaries, respectively. For report items whose sources are computation items, the input and history input items that are used as operand factors in the computations are placed into the input and history input summaries since the sources of all computation operand factors are specified. Also, the computations required to produce the report items are placed into the computation summary. Finally, the history output summary is compiled by listing all history items whose sources are items listed in either the input, history input, or computation summaries. Therefore, the history output summary indicates those history items that might be updated by the elementary module being specified.

After generating a process for each ADS-specified report, SGA searches for candidates for program module grouping in two ways. First, if some process requires history inputs either identical to or forming a subset of the history inputs required by another process, the two processes are identified as candidates for grouping. Second, if a predominant (approximately 90%) subset of the history inputs of some process is identical to a predominant subset of some other process, the two processes are identified as candidates for grouping. Finally, if the two candidates for grouping occur in the same processing cycle, grouping into a single program module is recommended by SGA.

SODA MACRO SIMULATION

Overview

In most cases, it is not appropriate to perform a micro performance evaluation, when many system design factors have not been specified. Therefore, a macro simulation can be useful as an aid in the specification of the complete systems design.

The SODA macro simulation model is used to evaluate the performance of the alternative computer systems under various simulated workload conditions. The Macro Simulator is used as an aid in design and performance evaluation of alternative design factors. The macro simulation is used to test the sensitivity of the performance considerations on various hardware and software design parameters. Among the system factors simulated at the macro level are:

- The number and capabilities of various devices.
- Specification of system software organization.
- Distribution of teleprocessing arrivals during various periods in the day.
- Query profiles.
- Scheduling of I/O devices to channels.
- Resource queue characteristics.
- Batch scheduling and job profiles.

The macro simulation is used to isolate potential problem areas and determine bottlenecks. This analysis is used to evaluate resource utilization, system throughput, batch turnaround, query response time, overall system behavior, etc.

Job Generation

In order to simulate a workload at the macro level, several characteristics of each job in the workload must be specified:

1. Job interarrival time.
2. Central memory space requirement.
3. Data base access interrequest time.
4. Number of data base access requests by type and complexity.
5. Data base access record length.

The values of these parameters can be derived from a study of the requirements of the information system under scrutiny and the performance capabilities of the computer system being considered.

Simulator Structure

The SODA Macro Simulator is an event oriented simulator not unlike that presented by MacDougall [8]. The major structures maintained by the model during simulation are a job status table, a list of events, various queues, resource status tables and job and resource utilization statistic tables.

The movement of jobs throughout the simulation was accomplished by a progression of events. Each event caused a change in the state of the affected job. For example, an event that assigned a job to the central processor caused a change in the job state from waiting to execution.

The simulation model itself basically consists of a collection of event routines. Each event routine applies the corresponding change in state to the affected job and predicts the type and time of occurrence of the next event that will affect the job based on the values of the parammeters contained in the job's entry in the job table. Event sequencing was co-ordinated by the simulator scheduling routine.

The event sequence was maintained as a list of events ordered by ascending value of time of occurrence. Each list entry consists of four fields:

1. Event identifier.
2. Event time.
3. Pointer to job table entry of affected job.
4. Link to the next entry in the event list.

However, whenever an event routine attempted to allocate a resource that was already busy, the job seeking the resource entered a queue of other jobs seeking the same resource. Eventually, when the resource became available, the scheduler selected a job from the corresponding queue and initiated the corresponding event routine. Job selection from the queue was performed according to a prescribed queue discipline.

Simulation Of Data Base Access

The manner in which the model simulates job interaction with the data base is of particular interest due to the sensitivity of the data management system with respect to particular applications. The procedure differs from that used for ordinary I/O requests to and from disk.

Data base accesses are classified by function and complexity. The primary functions are updating and retrieval. In addition, each function is further subdivided into various categories of access complexity. Complexity is characterized by factors such as the number of keys specified in a retrieval request and the number of indexes that must be searched in order to find the desired data base record. Overall, data base performance is determined by the size of the data base and by the type of physical storage structures employed to represent logical data structures.

The characteristics, e.g. size and physical storage structure, of the data base are specified initially in the simulation model. Each type of data base request, e.g. update or retrieval, is characterized by the various levels of complexity permissible. Based on this data, the model is able to predict the processing time of each type and complexity of data base request made by the jobs in the simulated mix.

GENERATION OF CODE FROM ADS AND SUPPLEMENTARY INFORMATION

Figure 4 illustrates a COBOL program that conceptually might be generated by SODA to fulfill the requirements described in the ADS statement of Figures 2 and 3. The program reads TIME-CARD-FILE, an input file of time cards, and performs the ADS-specified computations and logic to update EMPL-MASTER-FILE-IN and to produce PAY-REPORT.

The ADS description primarily provides information for generating the DATA DIVISION (part A of Figure 4) and the COMPUTE-WAGES paragraph (part C of Figure 4) of the PROCEDURE DIVISION. The remainder (part B of Figure 4) of the PROCEDURE DIVISION contains the procedures and processing logic needed for the application of the ADS logical definition to the physical implementation of the ADS-specified report generation and history file maintenance. Automatic production of this code necessary for physical implementation can be fulfilled in various ways. One software company has incorporated an additional form called an execution definition into its use of an ADS description for code generation. The execution definition form details the processing logic necessary for driving the execution of the logic and computations described in the ADS forms. Another approach to code generation might involve the incorporation of code skeletons for common data processing functions, e.g. transaction processing for master files. Then, the code skeleton is completed during code generation by providing the missing record sequencing identifiers and program termination conditions. Finally, automatic generation of code for report generation features such as positioning of heading and output lines might be accomplished by incorporating another feature into the computer-aided ADS report definition form for specification of report layouts and headings as in the original manual ADS system.

Still, computer-aided analysis involves much more than the rudimentary approach to code generation previously described. Current approaches to code generation from a non-procedural requirements statement merely translate logical descriptions into highly inefficient code, regardless of the quality of the original logical description. For example, deficiencies include the

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PAYROLL-CALCULATION.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SOURCE-COMPUTER.  6500.
OBJECT-COMPUTER.  6500.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
    SELECT TIME-CARD-FILE ASSIGN TO INPUT.
    SELECT PAY-REPORT ASSIGN TO OUTPUT.
    SELECT EMPL-MASTER-FILE-IN ASSIGN TO TAPE01.
    SELECT EMPL-MASTER-FILE-OUT ASSIGN TO TAPE02.
```

A

```
DATA DIVISION.
FILE SECTION.
FD TIME-CARD-FILE
    DATA RECORD IS TIME-CARD
    LABEL RECORDS OMITTED
    RECORD CONTAINS 18 CHARACTERS.
01 TIME-CARD.
    02 TIME-CARD-DATE              PICTURE 9(6).
    02 TIME-CARD-SSN               PICTURE 9(9).
    02 TIME-CARD-HRS               PICTURE 9(3).
FD PAY-REPORT
    DATA RECORD IS PAY-REPORT-REC
    LABEL RECORDS OMITTED
    RECORD CONTAINS 136 CHARACTERS.
01 PAY-REPORT-REC.
    02 FILLER                      PICTURE X(9).
    02 PAY-REPORT-SSN              PICTURE 9(9).
    02 FILLER                      PICTURE X(10).
    02 PAY-REPORT-NAME             PICTURE X(18).
    02 FILLER                      PICTURE X(10).
    02 PAY-REPORT-WAGES            PICTURE $$$9.99.
    02 FILLER                      PICTURE X(73).
FD EMPL-MASTER-FILE-IN
    DATA RECORD IS EMPL-MASTER-REC-IN
    LABEL RECORDS OMITTED
    RECORD CONTAINS 40 CHARACTERS.
01 EMPL-MASTER-REC-IN.
    02 EMPL-SSN-IN                 PICTURE 9(9).
    02 EMPL-NAME-IN                PICTURE X(18).
    02 EMPL-WAGE-STATUS-IN         PICTURE 9.
       88 HOURLY   VALUE 1.
       88 SALARIED VALUE 2.
    02 EMPL-RATE-IN                PICTURE 99V99.
    02 EMPL-YTD-WAGES-IN           PICTURE 9(6)V99.
FD EMPL-MASTER-FILE-OUT
    DATA RECORD IS EMPL-MASTER-REC-OUT
    LABEL RECORDS OMITTED
    RECORD CONTAINS 40 CHARACTERS.
01 EMPL-MASTER-REC-OUT.
    02 EMPL-SSN-OUT                PICTURE 9(9).
    02 EMPL-NAME-OUT               PICTURE X(18).
    02 EMPL-WAGE-STATUS-OUT        PICTURE 9.
    02 EMPL-RATE-OUT               PICTURE 99V99.
    02 EMPL-YTD-WAGES-OUT          PICTURE 9(6)V99.
WORKING-STORAGE SECTION.
01 WAGES                           PICTURE 9(4)V99.
```

B

```
PROCEDURE DIVISION.
OPEN-FILES.
    OPEN INPUT TIME-CARD-FILE, EMPL-MASTER-FILE-IN.
    OPEN OUTPUT PAY-REPORT, EMPL-MASTER-FILE-OUT.
    MOVE SPACES TO PAY-REPORT-REC.
READ-TIME-CARD.
    READ TIME-CARD-FILE AT END GO TO END-TIME-CARD-FILE.
READ-EMPL-MASTER.
    READ EMPL-MASTER-FILE-IN AT END GO TO NO-MATCH.
    IF TIME-CARD-SSN EQUALS EMPL-SSN-IN THEN
       PERFORM PROCESS-TIME-CARD
       GO TO READ-TIME-CARD.
    IF EMPL-SSN-IN LESS TIME-CARD-SSN THEN
       GO TO READ-EMPL-MASTER.
    IF EMPL-SSN-IN GREATER TIME-CARD-SSN THEN
       GO TO NO-MATCH ELSE
          MOVE ≠SYSTEM ERROR≠ TO PAY-REPORT-NAME
          WRITE PAY-REPORT-REC
          GO TO CLOSE-FILES.
PROCESS-TIME-CARD.
    MOVE EMPL-MASTER-REC-IN TO EMPL-MASTER-REC-OUT.
    MOVE EMPL-SSN-IN TO PAY-REPORT-SSN.
    MOVE EMPL-NAME-IN TO PAY-REPORT-NAME.
    PERFORM COMPUTE-WAGES THRU COMPUTE-WAGES-EXIT.
    ADD WAGES TO EMPL-YTD-WAGES-IN GIVING EMPL-YTD-WAGES-OUT.
    MOVE WAGES TO PAY-REPORT-WAGES.
    WRITE EMPL-MASTER-REC-OUT.
    WRITE PAY-REPORT-REC.
    MOVE SPACES TO PAY-REPORT-REC.
END-TIME-CARD-FILE.
    CLOSE TIME-CARD-FILE.
COPY-EMPL-MASTER.
    READ EMPL-MASTER-FILE-IN INTO EMPL-MASTER-REC-OUT
       AT END GO TO CLOSE-MASTER.
    GO TO COPY-EMPL-MASTER.
NO-MATCH.
    MOVE ≠NO MATCH TIME CARD≠ TO PAY-REPORT-NAME.
    WRITE PAY-REPORT-REC.
CLOSE-FILES.
    CLOSE TIME-CARD-FILE.
CLOSE-MASTER.
    CLOSE EMPL-MASTER-FILE-IN, EMPL-MASTER-FILE-OUT, PAY-REPORT.
    STOP RUN.
```

C

```
COMPUTE-WAGES.
    IF SALARIED THEN
       MOVE EMPL-RATE-IN TO WAGES
       GO TO COMPUTE-WAGES-EXIT.
    IF HOURLY THEN
       IF TIME-CARD-HRS LESS OR EQUAL 40 THEN
          COMPUTE WAGES = TIME-CARD-HRS * EMPL-RATE-IN
          GO TO COMPUTE-WAGES-EXIT ELSE
             COMPUTE WAGES = (40 + (TIME-CARD-HRS - 40) * 1.5) *
                                   EMPL-RATE-IN
             GO TO COMPUTE-WAGES-EXIT ELSE
                MOVE ≠INVALID WAGE STATUS≠ TO PAY-REPORT-NAME
                MOVE ZERO TO WAGES.
COMPUTE-WAGES-EXIT.
    EXIT.
```

Figure 4: COBOL Program Example

restriction of a one-to-one correspondence of reports and program modules with absolutely no consideration for module grouping. In addition, failure to consider volume and frequency of access with regard to the various history data items defined eliminates any possibility for generating optimal file structures. Capabilities of computer-aided analysis should include specification of optimal file designs and grouping of single report generating modules in order to eliminate excessive transport of data from files to programs. These capabilities can only be achieved by extension of forms-oriented programming specification techniques like ADS to true requirements statement techniques providing supplementary volume and timing data to the optimization software. Then, the potential of the computer to aid the problem definer during the system design cycle can be fulfilled.

EXPERIENCE WITH A LARGE APPLICATION

The work specifically reported in this paper was done for the United States Navy Material Command Support Activity (NMCSA). The statement of requirements for a financial management system was expressed in ADS by a large Accounting Firm. An ADS analyzer, developed at the University of Michigan [9, 10] was used to check the ADS statement of requirements for completeness, consistency and logical accuracy. The ADS analyzer produced information and reports that were used by the SODA Statement Analyzer. SODA was then used to (1) generate preliminary designs of program structure and logical data base structure for the batch application part of the system and (2) to recommend a computer system for the entire financial management system.

Experience With ADS

NMCSA personnel, engaged in financial manage ent, are currently using ADS to state the requirements of a large information system. This integrated financial management system is a large-scale design and implementation effort for more effective financial management, particularly procurement accounting, within the agency. The systems design effort commenced in May, 1971, and is expected to continue for 4 to 5 years at a cost of 12 million dollars.

A systems design effort of this magnitude has an impact upon many different offices within the complex organization of the agency. Financial managers, the end-users of the system, are scattered among many offices engaged in complicated communication of varied information requirements.

The purpose of the system is to centralize information flow and satisfy the information requirements of a variety of decision makers. The principal communicators in the system are the requiring manager or financial manager, participating manager, and procurement manager. The decision makers observe a hierarchy of authority delegation and reverse order for accountability.

Initial authority is received by the Comptroller of the Navy, from the office of the Secretary of the Navy. In the case of a fund allocation, the comptroller issues a program fund allocation with program values at the line item budget level. The funding then proceeds through the responsible office to the administering office. Here funding proceeds to the requiring/financial manager level where responsibility lies for execution, modification, and delivery of the system within cost and schedule objectives. At this level monthly approval is considered for interim actions. The requiring/financial manager issues project directives which delegate specific authority to the participating manager level operatives.

Fund status inquiries are used to maintain fund control at the project directive line item level. Contract status inquiries are used to control delivery dates and funds for contract items.

Fund status inquiries are characterized by funded project directives and planning project directives. The planning project directive allows for procurement of long-lead-time items. The funded project directive serves a shorter cycle time and therefore has a high activity level of inquiry and update.

Contract status inquiries are primarily identified with the procurement manager. Once a funded project directive is established, the information updates revolve around completion status of tasks in line with procurement. The procurement manager updates the contract status which is observed by the participating manager and the requiring/financial manager.

The basic objective is to significantly improve the timeliness of accounting/financial management information reported by the agency's accounting system to reduce input, processing, and reporting time. This facilitates elimination of most memorandum record systems.

The first module of the system to be implemented is the procurement accounting and reporting subsystem. This subsystem is intended to normalize information flow concerning procurement accounting. This subsystem includes:

1. Funds Accounting
2. Planning Documents
3. Contract Accounting
4. Fiscal and Program Status
5. Other items related to appropriations.

ADS was selected as the requirements statement technique for system documentation. This phase has now been completed and experience has indicated that ADS served effectively as a tool for systems documentation. The ADS statement for the system includes descriptions for 79 reports and for the accompanying history files, computations, and inputs which define 791 data elements.

BEHAVIORAL EXPERIENCE WITH ADS

The first objective of the introduction of ADS into any environment is gaining user acceptance. ADS represents deviation from the established practices and initial resistance to change often occurs. As a result, many questions regarding ADS and its impact upon the organization are raised.

In response to this initial user reaction, an ADS training program is advisable. However, ADS is simple and straightforward so less than one day of intensive training is all that is necessary to adequately prepare individuals to begin using ADS. Then, further training is required only to deal with the specific restrictions imposed upon the use of ADS by the ADS Analyzer software. For example, the Analyzer restricts the length of data element names to forty characters.

The use of a form-oriented procedure such as ADS still requires a significant investment of time and effort to realize the return of a complete and consistent logical systems design. Still, a number of users with ADS experience agree that ADS has saved them considerable time during the specification of logical system design.

This savings is realized by the capability of the ADS Analyzer to provide feedback information to the user. The user should be able to do a better job of specifying his requirements because he receives feedback much sooner in the system design cycle utilizing computer analysis of ADS. Ordinarily, in a completely manual narrative system, ambiguities and omissions in the logical system description are not discovered until physical design or even coding is well underway. By then, many aspects of the system design have been specified so that resolution of difficulties may be impossible.

Physical system design is not the responsibility of the ADS user. Completion of the ADS logical description is followed by the physical system design process that provides the specifications for programming.

Performance of ADS

Experience has demonstrated that ADS is adequate for specification of the logical system. However, an ADS description does not provide sufficient information for optimization of physical system design. Data on system performance requirements was collected to supplement the ADS description in SODA Statement Language. Relevant data includes specification of the frequency of occurrence of each ADS - described input and report and of the volume of each input, report,and history.

Other needed enhancements to computer-aided ADS include facilities for describing data structures and look-up tables and for decision tables expressing processing logic and input validation rules. Finally, additional software for generating report layouts and program test data would add significantly to computer-aided ADS capabilities. Many of these enhancements are included in the SODA Statement Analyzer.

PROGRAM MODULE GROUPING FOR THE NAVY EXAMPLE

For the information system under consideration SGA generated 62 program modules to produce the 79 ADS-specified reports. For each program module, SGA provides the following information to the SODA Performance Evaluator (SPE).

- Brief program module title.
- Frequency of occurrence.
- Program module size, in K bytes.
- History files required for processing.
- File device type.
- Size, in bytes, of each history record input.
- Number of history records input for processing.
- Volume, in number of lines, of printed output.

For each program module, module size and number of arithmetic operations are derived from the quantity and complexity, e.g. alternative logic paths, of computations in the summary produced by SGA. Volume and size of history records input are derived from the history input summary produced by SGA. SGA performs summary analysis on all ADS-specified inputs required to produce each history item. User-provided data on input requirements was then used to derive the volume of the history item under scrutiny. The size of the history item is provided in the ADS description. Finally, twenty record groups were generated with each group containing history items that are used together in a fashion that implies logical connectivity. Each group of records forms the basis for defining history file structures. An overview of the program module specifications for fiscal reporting tasks is presented in Table 1: Batch Program Module Workload Summary.

Note that process grouping into modules and history record grouping into files were performed in a manner that spreads the workload equally among the modules to the greatest extent possible. Workload sharing is made possible by minimizing the variance in the number of computations in each module and by minimizing the variance in the number of records in each file grouping.

BATCH PROGRAM MODULE WORKLOAD SUMMARY

Application/ Program ID	Task Type	Freq/ Month	Memory Required (K bytes)	Language	File ID	Medium Code	Avg. Record Length (char.)	Record Volume	Avg. Output Length (Line
A. Fiscal Reporting									
1. Program Budget Status	Print	1	150	COBOL	H1 H4 H5 H11	Disk	861	2200	340
2. Appn. Status by FY and Acct.	Print	1	50	COBOL	H1 H5 H11	Disk	243	2200	23000
3. Report on Reimbursables	Print	1	35	COBOL	H1 H4	Disk	232	225	24000
4. Report on Obligations	Print	1	100	COBOL	H1 H4 H5	Disk	437	2000	1000
5. Analysis of Appropriations and Fund Balances	Print	1/year June	50	COBOL	H1 H4 H5 H11	Disk	476	2200	800
6. Line Item Report	Print	1	35	COBOL	H1 H4	Disk	232	225	24000
7. Summary Line Item Report	Print	1	50	COBOL	H1 H5 H11	Disk	263	2200	4700
8. Procurement Program Progress Report	Print	1	35	COBOL	H1 H5 H11	Disk	268	2200	4000
9. Worksheet	Print	2/year June, Dec.	25	COBOL	H1 H5	Disk	141	2000	4000

Table 1: Batch Program Module Workload Summary

FUTURE RESEARCH

Extensions to the computer-aided analysis techniques described in this paper are presently being developed at Purdue University. The many software packages for aiding in the design process are being incorporated into a system called GPLAN/SODA. GPLAN: Generalized Data Base Planning System [4, 5].

The GPLAN approach provides an effective framework for the solution of problems involving the design and optimization of large information processing systems. GPLAN/SODA, a Generalized Data Base Planning System containing System Optimization and Design Algorithms, provides capabilities ranging from the construction of an information system described by SSL problem statement to the optimization of individual system performance factors such as input and output.

Following the GPLAN outline, the major components of GPLAN/SODA include:

1. A data base containing general-purpose programming language (GPPL) and SSL descriptions of the information system being designed. COBOL is the GPPL for implementation of the type of information system designed by GPLAN/SODA: business data processing systems composed of small (in relation to the size of data files) programs manipulating large data files.
2. A collection of software packages including:
   a. SODA Statement Analyzer (SSA), a Requirements Statement Analyzer for SSL.
   b. SODA Generator of Alternatives (SGA), a procedure to generate alternative hardware (CPU, core, auxiliary memory devices) and software (program module and file structure) configurations.
   c. SODA Performance Evaluator (SPE), a collection of models that produce cost and performance projection reports in order to evaluate the alternative designs generated by SGA. SPE includes:
      - Simulators for batch and teleprocessing systems.
      - Blocking factor selection model.
      - Model for file assignment to physical devices.
      - Program module scheduling model.
   d. Code generator to map selected program module and file structures to programming language (GPPL) representation for selected hardware configuration.

e. Program re-organizer including capabilities such as [3]:

  - Combination of similar data passes on the same file to minimize transport volume.
  - Merging of loops to enable elimination of code and of intermediate data files.

f. COBOL-to-ADS translator for existing system (see the discussion of SSL for a description of ADS).

g. ADS-to-COBOL generator for a new system.

3. A query language to enable man-machine interaction during all phases of GPLAN/SODA operation.

GPLAN/SODA facilitates communication among the users, designers, and implementers of an information system throughout the life of that system by providing a central clearinghouse of data relevant to any operational aspect of information processing. Problem definition facilities are provided for the user to express his requirements to the designers and implementers. Current manual project management methods often prove inadequate for insuring the integrity of up-to-date information concerning system implementation and for distributing this information among the designers and implementers. GPLAN/SODA maintains a single, up-to-date copy of all relevant data in a central location easily accessible to all designers and implementers. Data administration and standards enforcement are effective because all data name usages and processing specifications must pass through the GPLAN/SODA control software before being accepted and stored.

Second, the potential for expansion and improvement is available through the definition of an interface between each software package and the other components of GPLAN/SODA. The addition of a new software package requires knowledge of the input/output characteristics and the operational capabilities of the package so that the Generalized Data Management System (GDMS), query language, and extraction file definitions can be modified to make the new package available to the user community. The input/output characteristics must be added to the data definitions of the GDMS to enable the new package to use the data base. The query language and its analyzer must be extended to include the query components made possible by the added capabilities of the package. The extraction file structure may require modification to accommodate the needs of the new package.

Finally, GPLAN/SODA provides the man-machine interaction so necessary for the convenient use of any computer-aided tool while isolating the non-technical user from the intricacies of the tools applied and the data manipulation that would be required by these tools in a more conventional environment.

Refer to Figure 5 for an overview of GPLAN/SODA.

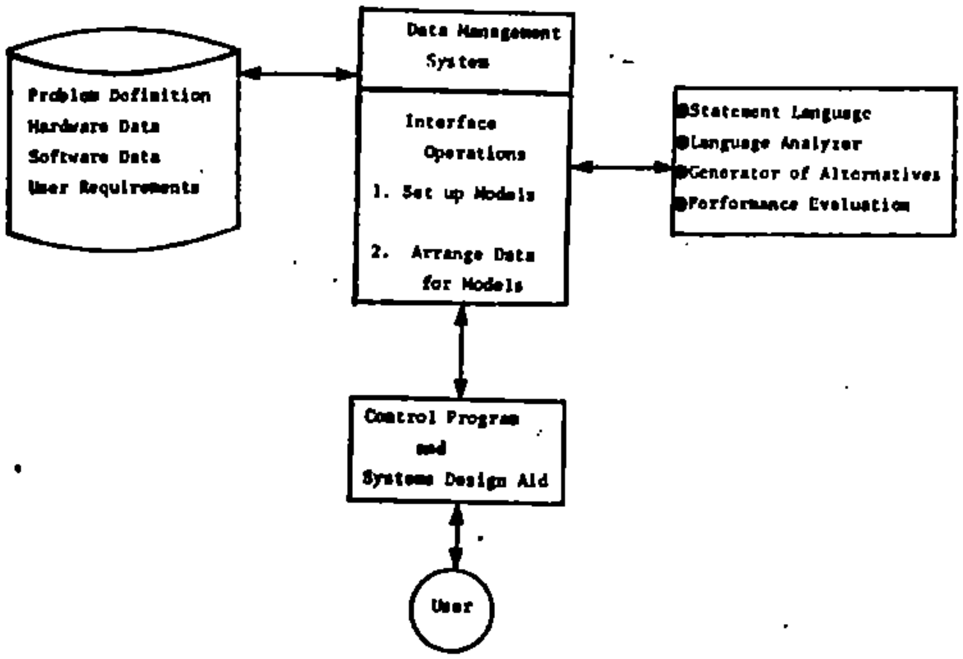


Figure 5: Overview of GPLAN/SODA

CONCLUSION

SODA system models were utilized in arriving at the software module design for the procurement accounting module of the financial system of a large governmental agency. In addition, SODA performed selection of hardware suitable for implementation of the proposed design.

All constraints and system objectives were incorporated into an ADS/SSL problem statement. Program modules were created based on an analysis of the data and information flow relationships.

REFERENCES

1. Nunamaker, J. F. Jr. 1971. A methodology for the design and optimization of information processing systems. Proc. 1971 SJCC 38, May 1971, AFIPS Press, 283-294.

2. Nunamaker, J. F. Jr. and Whinston, A. A macro approach to the planning and cost allocation of computer services. Management Informatics 2, 4, August 1973. (To appear)

3. Nunamaker, J. F. Jr., Nylin, W. C. and Konsynski, B., Processing Systems Optimization through Automatic Design and Reorganization of program Modules, Proceedings of 4th COINS Conference, December 1972, to be published in 1973, Academic Press.

4. Nunamaker, J. F. Jr., Swenson, D. and Whinston, A. B., Specifications for the Design of a Generalized Data Base Planning System, Proceedings of the National Computer Conference, AFIPS Press, June 1973.

5. Nunamaker, J. F. Jr., Pomeranz, John, and Whinston, A. B., Automatic Interfacing of Application Software in the GPLAN Framework, CSDTR 95, Computer Science Department, Purdue University, West Lafayette, Indiana, May 1973.

6. Lynch, H. J. ADS: A technique in system documentation. Database 1, 1, Spring 1969, 6-18.

7. National Cash Register Company. A Study Guide for Accurately Defined Systems. 1968. Dayton, Ohio.

8. MacDougall, M. H. Computer System Simulation: An Introduction. Computing Surveys 2, 3, (September 1970), 191-210.

9. Mertem, A., and Teichroew, D. The Impact of Problem Statement Languages in Software Evaluation, Proceedings of FJCC 1972, AFIPS Press, November 1972, 849-858.

10. Thall, R. 1970. A manual for PSA/ADS: a machine-aided approach to analysis of ADS. ISDOS Working Paper No. 35 (October 1970), Department of Industrial and Operations Engineering, University of Michigan, Ann Arbor, Michigan.