TECHNICAL MEMORANDUM

(TM Series)

Distribution of this Document is unlimited

This document was produced by SDC in performance of contract AF 19(628)-1648, System 465L--SACCS, for Electronic Systems Division, AFSC (562.02)

MANAGEMENT REPORT: CONTROLLING PRODUCTION OF COMPLEX SOFTWARE

By

J. J. Connelly and Y. R. Osajima

Development Branch

17 May 1963

SYSTEM DEVELOPMENT CORPORATION

2500 COLORADO AVE.

SANTA MONICA

CALIFORNIA

SDC

(Page 2 Blank)

ACKNOWLEDGMENTS

The following persons have been associated in the development of the production process described in this paper:

M. I. Bolsky
M. D. Campbell
J. J. Connelly
W. L. Landaeta
M. A. Levine
Y. R. Osajima
J. J. Pavese

Preparation of this document and the briefing script that preceded this document was materially assisted by J. H. Green and D. E. Wolgamuth.

Method of Presentation of This Paper

The text of this paper was originally presented as a briefing, with accompanying flow diagrams and charts. In order to achieve maximum clarity, we have placed each illustration used in the briefing on a left-hand page, with the accompanying text on the facing right-hand page. While the text is complete in itself, the illustrations will undoubtedly be of interest and usefulness to readers.

PREFACE

In order to do well in a position, a person must bring two factors to that position: native intelligence, and experience. Combined, the two will equal that person's skill in fulfilling his responsibilities most effectively.

Native intelligence is more or less a fixed item, but experience is not. Experience can be acquired in two ways, each of which supplements the other. The first way is actual on-the-job work. The second way is by reading about how others have done similar tasks.

The persons who have been associated in the development of the production process described in this paper have accumulated a total of over forty man-years of experience in the management of large-scale computer program system production. Nearly half of this has been with the largest computer program system being produced - the 465L Planning Subsystem.

(The 465L Planning Subsystem is one of the two operational components that make up the 465L System. The other component is the 465L Control Subsystem. As their names imply, these will perform planning and control functions, respectively, for the Strategic Air Command. Each of these subsystems, alone, is in itself a large integrated program system. The Planning Subsystem, in itself, is comprised of approximately 90 programs, 300,000 machine instructions, and a data base of 6,000,000 words. They are called subsystems only because they are components of the total 465L System.)

We hope that our experience can be of help to the managers of other current and future computer programming projects. The record of how we set up and controlled the production process for the Planning Subsystem provides insights into the nature of the problems involved, and into some of the ways by which these problems may be overcome.

We recognize that this paper is not a generalized description of the program production process. However, we do feel it to be a significant milestone in that it is a positive attempt at defining or analyzing computer programming in terms of activities, products, needed resources, and management controls.

This paper, then, is dedicated to those who take on the heavy burden of managing the production of a large computer program system. This task is not easy, but it is a challenge that brings a most rewarding feeling when it is accomplished successfully.

LIST OF ILLUSTRATIONS

TABLE OF CONTENTS

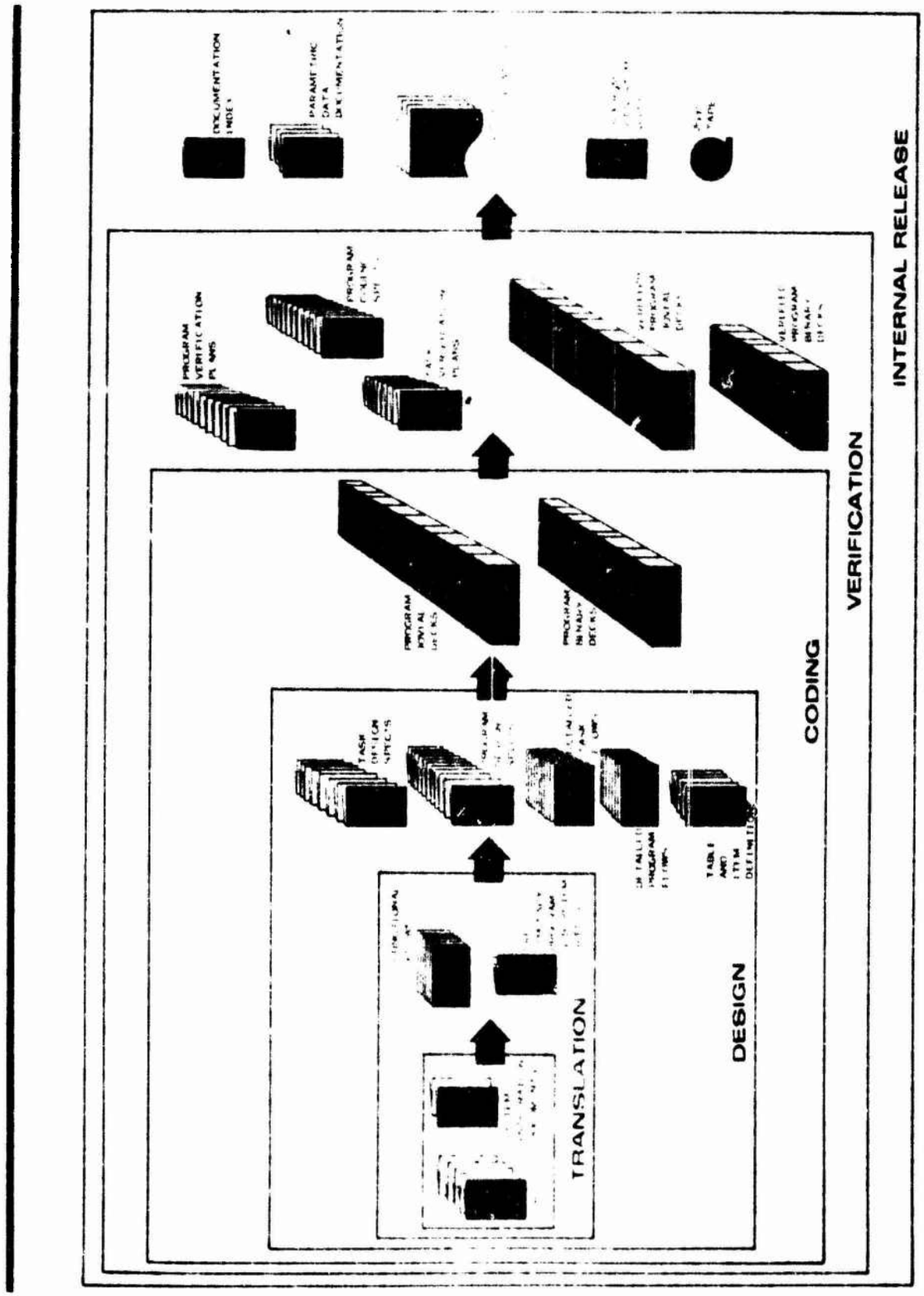

PRODUCTION PROCESS OVERVIEW
TRANSLATION
DESIGN
CODING
VERIFICATION
INTERNAL RELEASE

Figure 1.

I. INTRODUCTION

In order to effectively manage the production process of a large-scale computer program subsystem, one must first have a clear definition of the process. This paper defines the process used in producing the 465L Planning Subsystem. The total production process, needless to say, requires a manager's attention to a great many functions, e.g., processing and controlling system changes, effective utilization of manpower resources, effective EAM/EDPM utilization, etc. Although these are all integral parts of the total process, in this paper we are restricting ourselves to a definition of the basic functions of the production process. Furthermore, the production process, as it is defined in this paper, assumes certain functions that precede the process and thus produce inputs to it, and certain functions that follow the production process and thus utilize its outputs.

Inputs To Production Process - This process starts on receipt of (1) a System Integration Document, specifying the formats of the user input messages and output displays; and (2) Operational Design Requirements (ODR's) containing the following sections: general statement and description of the requirements; logical designs, including assumptions, in both prose and diagram form; specific requirements indicating the areas of human interaction with the program; specific operational program requirements.

The Phases Of The Production Process - The production process, as defined in this paper, is divided into five basic phases--Translation, Design, Coding, Verification and Internal Release. Each of these phases is essentially a building block; the outputs of one phase are the inputs to the next. Interim documents serve as bench-marks to signal the completion of each phase. These documents serve four functions:

1. To insure that programmers perform each step in a rigorous fashion;

2. To enable technical supervisors to inspect intermediate steps so as to insure a high quality product;

3. To enable managers to more accurately assess where the development of the subsystem stands in relation to where it should be, and consequently to better assess future manpower needs and delivery dates;

4. Minimize the impact of manpower turnover, since most of the development is recorded in documentation.

Outputs From The Production Process - This definition assumes that the production process is completed after the Internal Release Phase. It is assumed that there will be some other agency which will then take the product and: perform subsystem testing; install the subsystem in a computer at a given location; formally release the product to the customer.

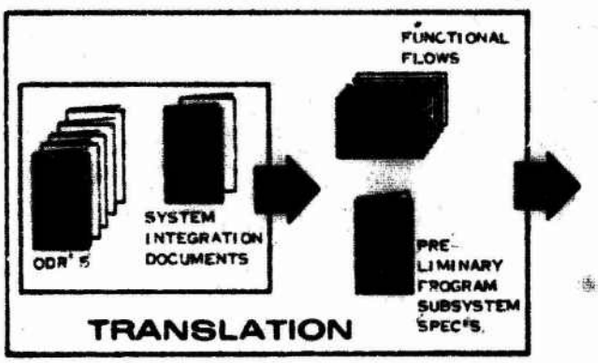

FUNCTIONAL FLOWS
SYSTEM INTEGRATION DOCUMENTS
ODR'S
PRE-LIMINARY PROGRAM SUBSYSTEM SPEC'S.
TRANSLATION

Figure 2

II. TRANSLATION PHASE

Purpose Of The Translation Phase

The Translation Phase has three purposes:

1. To give programmers a clear understanding of the content of the Operational Design Requirements (ODR's);

2. To identify and resolve inconsistencies in the ODR's;

3. To re-group the functions stated in the ODR's so that they are logically grouped from a programming point of view.

Problems In Translation And Their Solution

Translation of the ODR's is by no means easy, when thousands of pages of requirements and seventy or more programmers are involved. Experience has shown that there is a diversity in the understanding of the ODR's because people emphasize different things in their reading, and since there is frequently a varying degree of detail in different ODR's.

But in this phase, uniformity is important. This can be achieved by:

1. The procedurizing of the Translation Phase into steps with definite objectives at the end of each step, and

2. The establishment of two documentation points to signify the mid-point and the end of the Translation Phase.

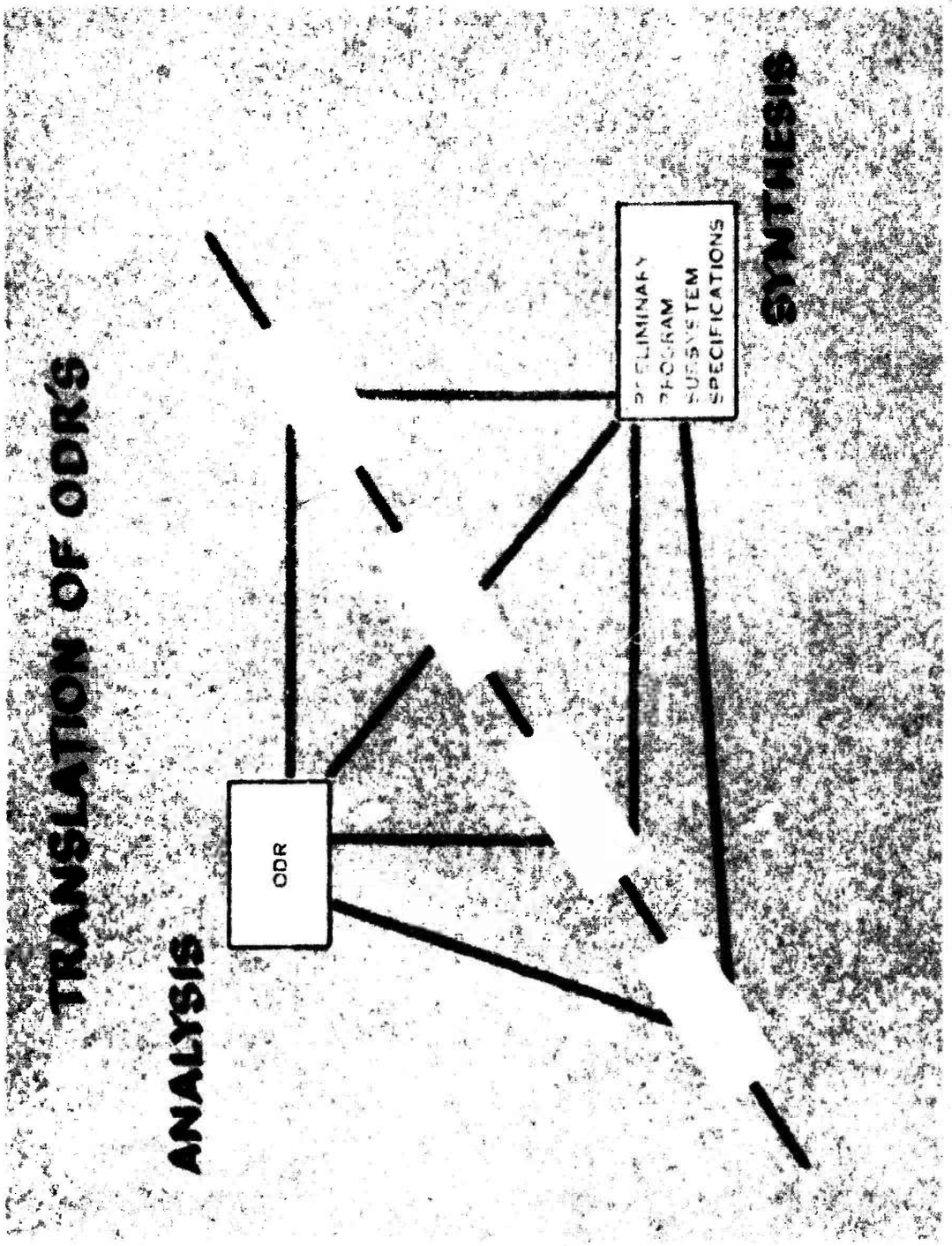

TRANSLATION OF ODRS
ANALYSIS
ODR
SYNTHESIS
PRELIMINARY
PROGRAM
SUBSYSTEM
SPECIFICATIONS

Figure 3

Methodology Of Translation

The Translation Phase is comprised of both analysis (i.e., the breaking up of a whole into its component parts to ascertain their nature) and synthesis (i.e., the combining of parts to form a whole.) Analysis consists of three steps (which satisfy the first two purposes of the Translation Phase--to obtain a clear understanding of the ODR's, and to identify and resolve inconsistencies in the ODR's.)

1. Segmentation of ODR's into functional areas;

2. Identification of ODR subfunctions (OSF's);

3. Performance of detailed analysis of each ODR subfunction.

Synthesis consists of two steps (which satisfy the third purpose of the Translation Phase--to re-group the functions stated in the ODR's so that they are logically related from a programming point of view.)

4. The synthesis of logical tasks from ODR subfunctions;

5. The synthesis of logical jobs from logical tasks, and the documentation of the Preliminary Program Subsystem Specification.

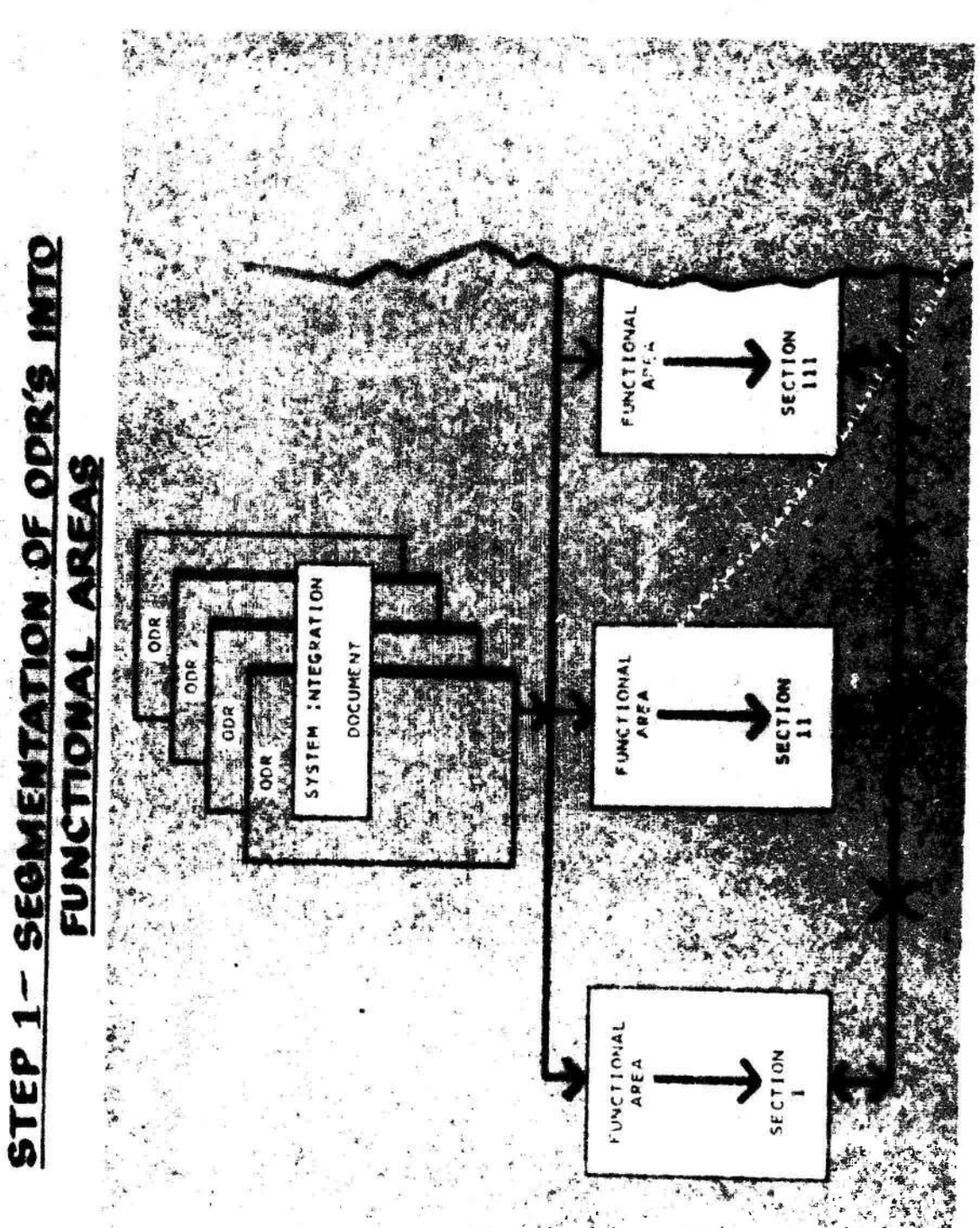

STEP 1- SEGMENTATION OF ODR'S INTO FUNCTIONAL AREAS
ODR
ODR
ODR
ODR
SYSTEM INTEGRATION DOCUMENT
FUNCTIONAL AREA
SECTION I
FUNCTIONAL AREA
SECTION II
FUNCTIONAL AREA
SECTION III

Figure 4

Step 1--Segmentation Of ODR's Into Functional Areas

As stated above, the inputs to the Translation Phase for the Planning Subsystem are ODR's and the System Integration Document.

The first step, then, involves the analysis of the ODR's, in order to define logical subdivisions of the subsystem. Each subdivision consists of all, or parts of, one or more ODR's, and is called a "functional area." The basic criterion employed in defining these logical subdivisions is functional interdependency. For example, in the Planning Subsystem, the Flight Plan Analysis functional area consists of all or parts of the ODR's for Flight Simulation, Airborne Alert, Mating and Routing. The functions defined in these ODR's are all involved in the development of flight plans. The intent of this subdivision of the subsystem into functional areas is to logically distribute the work load among the organizational sections comprising the group of programmers who are to produce the subsystem.

A gross estimate of the scope (i.e., the number of programming instructions) of each functional area is made, and then one or more of these areas is assigned to each section in the group.

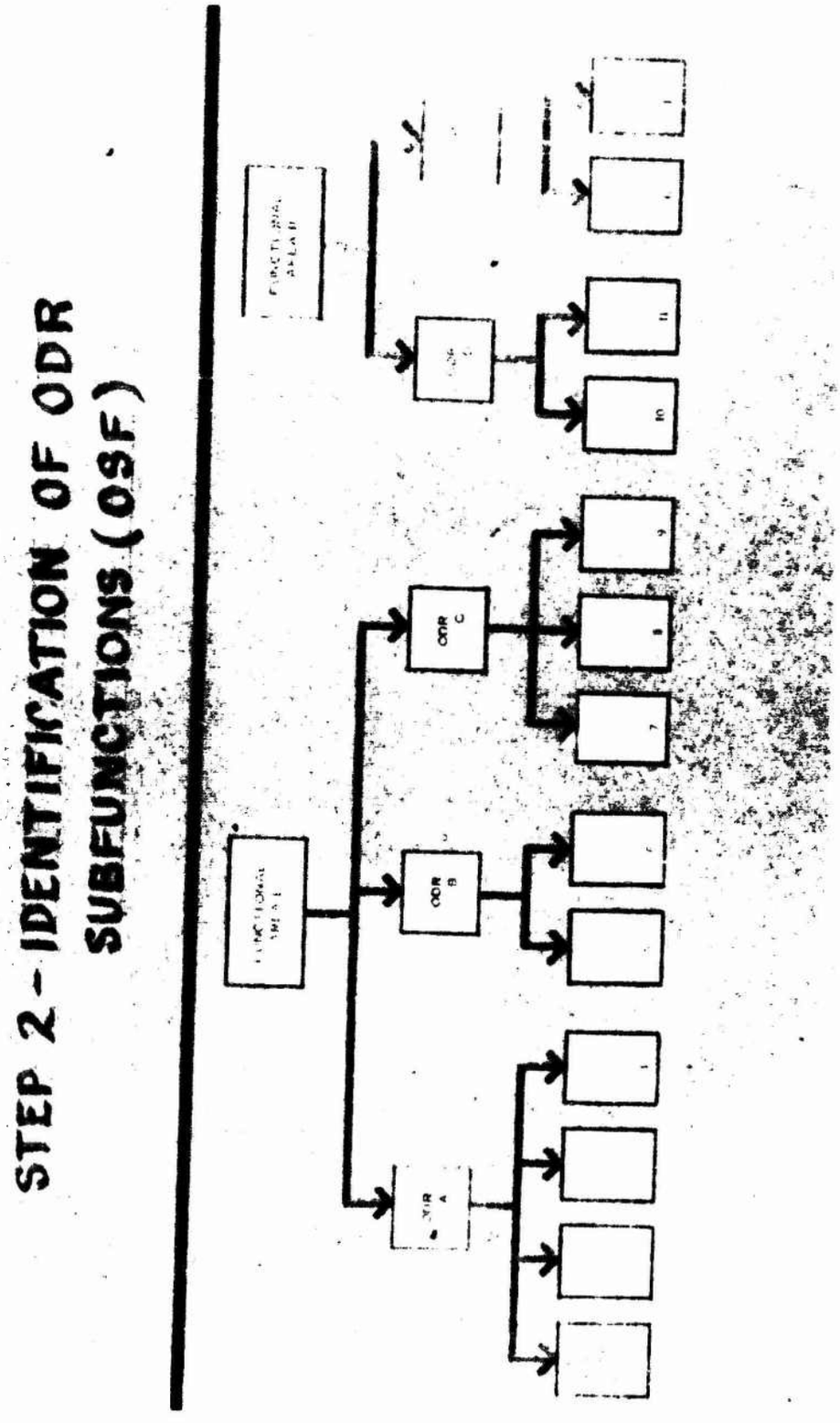


Figure 5

Step 2--Identification Of ODR Subfunctions (OSF)

The programmers - who are to later analyze portions of an ODR in detail - read all ODR's in their particular functional area, in order to obtain a total overview. They then participate in meetings to further sub-divide each of the ODR's. Each division of an ODR is identified as an ODR sub-function (OSF). The purpose of this step is to break down each functional area into manageable parts (OSF's) so that they can be distributed to programmers for subsequent detailed analysis. Questions and/or inconsistencies identified during the preliminary readings are coordinated with the ODR author.

The ODR subfunctions are then assigned to programmers for detailed analysis. The assignments are made as a function of programmer capability, and the complexity and size of the OSF. Generally, each programmer is assigned one or two OSF's.

STEP THREE - PERFORMANCE OF DETAILED ANALYSIS OF EACH CDR SUB-FUNCTION
7
YES
8
IS THIS A KC-97 AND R C AT CONSTANT ALT
NO
11
SEE CHART 2
YES
9
IS FLIGHT LEVEL TEMP-ERATURE GIVEN
YES
10
SEE CHART 2
NO
12
SEE CHART 2

Figure 6

Step 3--Performance Of Detailed Analysis of Each ODR Subfunction

The programmers then begin detailed analysis of their assigned OSF's. Each such analysis consists of a careful reading of the pertinent operational program requirements portion of the ODR, to obtain a clearer understanding of the processing therein described and to identify the data required for the performance of the OSF. Each piece of data is defined in terms of its nature (i.e., fixed or variable), form, range, and functional grouping. For example, aircraft total fuel capacity might be defined as fixed, integer, 50,000 to 300,000, and a function of aircraft type and model.

In performing this detailed analysis, each programmer is responsible for coordinating with the ODR author to validate the analysis. New OSF's are sometimes created by consolidating, redefining or splitting up old ones. New assignments of OSF's are made when necessary.

A form of documentation helps insure that the analysis is performed correctly. It signifies completion of analysis, the first part of the Translation Phase. This form of documentation is the functional flow chart and is essentially a graphic presentation of the prose statement of requirements. Programmers are required to produce functional flow charts for each OSF. These charts portray the OSF processing without explicitly relating it to machine processing. (This explicit relationship is made in the Design Phase.) Each chart is reviewed by the technical supervisor for accuracy and uniformity. The (corrected) chart is then reproduced and copies are given to each programmer working on the functional area to enable him to review it as it relates to his own subfunction(s).

STEP 4 – THE SYNTHESIS OF LOGICAL TASKS FROM ODR SUB-FUNCTIONS

Figure 7

Step 4--The Synthesis Of Logical Tasks From ODR Subfunctions

Synthesis begins with a grouping together and identification of OSF's as logical tasks. A logical task consists of one or more OSF's which collectively accomplish a specific SAC function. An example of a logical task is the Flight Simulation Task. OSF's comprising a given logical task might very well have originally been parts of different ODR's. For example, the cruise mode of flight (a Flight Simulation ODR subfunction) and the segmentation of sortie routes (a Routing ODR subfunction) both comprise part of the same logical task - Flight Simulation. In this step, calculations common throughout the OSF's, such as sine and cosine calculations, are identified for subsequent handling as subsystem routines.

A concurrent activity is the identification of logical groupings of the data required for performance of a given logical task. The data defined during the previous step for each OSF are collected into sets by their functional groupings, (e.g., all data which are a function of aircraft type and model are collected into a single set.)

LOGICAL TASKS FROM THE DOCUMENTATION OF THE PRELIMINARY PROGRAM SUBSYSTEM SPECIFICATION

Figure 3

Step 5--The Synthesis Of Logical Jobs From Logical Tasks, And The Documentation Of The Preliminary Program Subsystem Specification

All logical tasks developed within the subsystem are then analyzed and grouped into logical jobs. A logical job is composed of one or more logical tasks which must operate together to fulfill a class of related user input messages.

The basic criterion for establishing logical jobs, then, is man-machine interaction.[1] An example of a logical job is the Input Processor Logical Task and the Flight Simulation Logical Task mentioned above. (For example, the Input Processor Logical Task would be identified at this step in the development since analysis of all previously developed logical tasks would indicate that each is performing an input processing function and that it would be reasonable to centralize and generalize this function.)

A parallel effort, one closely related to the development of jobs, is the further development of the data base. The data sets previously identified for the logical tasks are now merged to form larger data sets for the entire subsystem. To illustrate, let us assume that two functional areas require data associated with aircraft units. The first requires the units' locations and configurations; the second requires the units' locations and vulnerabilities. At this time, the common data requirement would be recognized and one data set for units would be established, consisting of units' locations, configurations, and vulnerabilities.

The results of the Translation Phase are documented in the "Preliminary Program Subsystem Specification." This document identifies the logical tasks and jobs, the data sets, and the input and output requirements of the subsystem. It also contains prose and graphic descriptions of the manner in which the various logical tasks and jobs relate. Publication of this document signals the completion of the Translation Phase.

1 The human action requirements portion of the ODR's, and the System Integration Document, assist in defining this man-machine interaction.

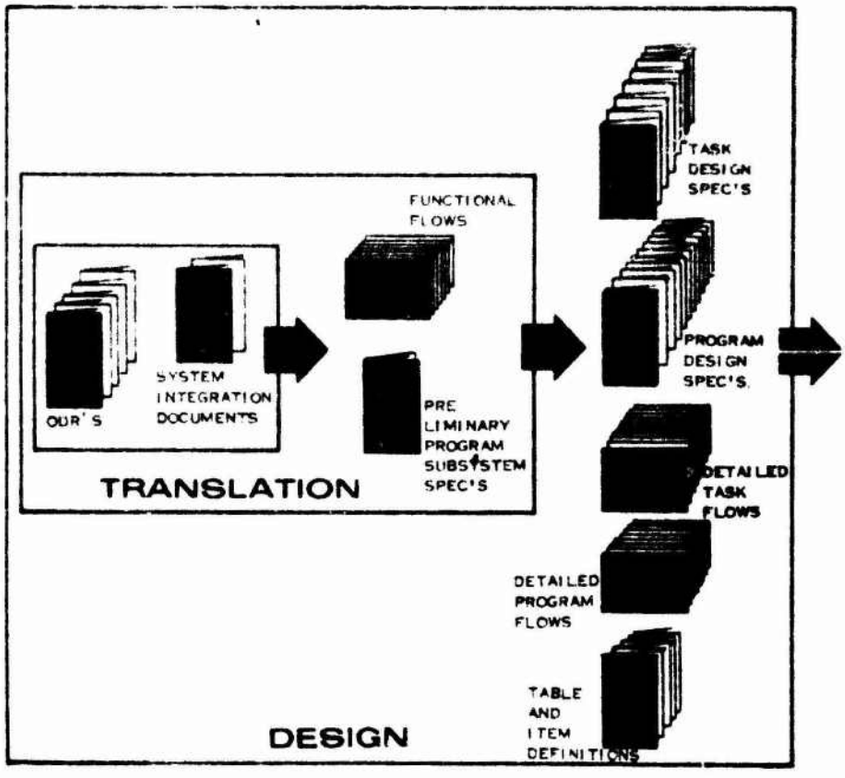

TASK
DESIGN
SPEC'S
FUNCTIONAL
FLOWS
PROGRAM
DESIGN
SPEC'S
OUR'S
SYSTEM
INTEGRATION
DOCUMENTS
PRE
LIMINARY
PROGRAM
SUBSYSTEM
SPEC'S
TRANSLATION
DETAILED
TASK
FLOWS
DETAILED
PROGRAM
FLOWS
TABLE
AND
ITEM
DEFINITIONS
DESIGN

Figure 9

III. DESIGN PHASE

Purpose Of The Design Phase

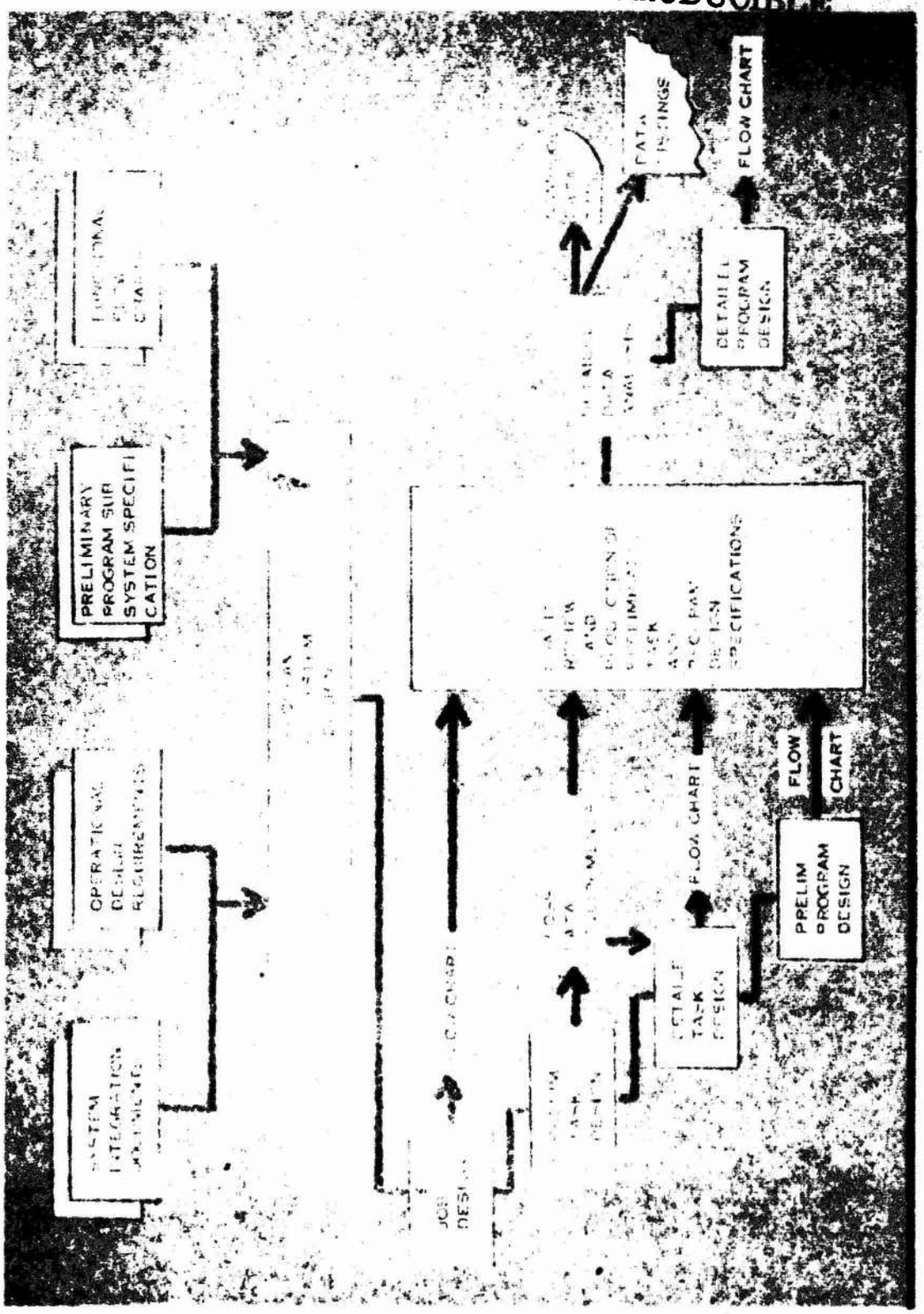
The Design Phase, though simple in definition, is perhaps the most complex and significant phase of programming. Its purpose is to structure the actual jobs, tasks, and programs that comprise the subsystem, in order to produce the most efficient and least costly subsystem possible.

Problems In Design And Their Solution

The orderly and generally accepted approach to program system design is to work from the general to the specific. In our subsystem, this means first designing jobs, then tasks, then programs. The problem, however, is that in actual practice, programmers tend to concentrate effort on the design of the most specific components (i.e., programs) since these are the easiest to grasp and also seem to most directly affect the progress of subsystem development. This means, then, that the design of jobs and tasks may be left for last, and thus be hurried and, consequently, inefficient. Experience has demonstrated that inefficient job and task design results in redundant efforts in design, coding and verification, extensive rework in coding and verification, and attendant low programmer morale.

A solution to this problem is to procedurize the Design Phase and to establish interim bench-marks insuring that each step of the process is performed adequately.

GRAPHIC NOT REPRODUCIBLE



METHODOLOGY OF DESIGN

Figure 10

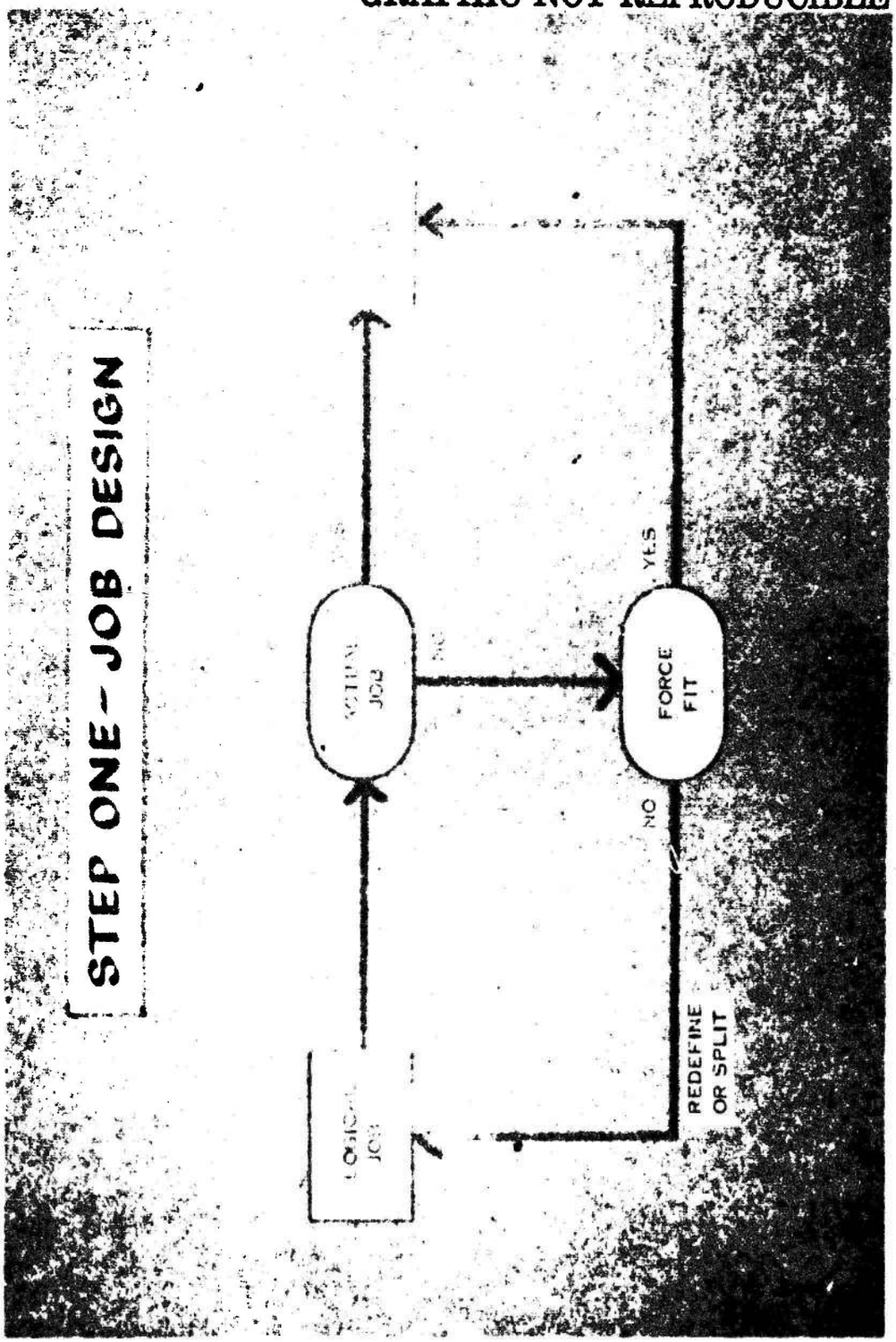
Methodology Of Design

Inputs to the Design Phase, in addition to the outputs from the Translation Phase (i.e., Preliminary Program Subsystem Specification, and Functional Flow Charts) include the System Integration Document and Operational Design Requirements (ODR's) that were also input to the Translation Phase.

The following steps comprise the Design Phase:

1. Job Design
2. Preliminary Task Design
3. Detailed Task Design
4. Preliminary Program Design
5. Quality Review and Production of Preliminary Task and Program Design Specifications
6. Detailed Data Analysis
7. Detailed Program Design

GRAPHIC NOT REPRODUCIBLE



Step One--Job Design

Given the logical jobs produced in the Translation Phase and documented in the Preliminary Program Subsystem Specification, the function of Job Design is to determine whether the logical jobs can be actual jobs. (In the Translation Phase, machine constraints were not explicitly considered. This would have been an additional factor of complexity at that initial phase. It was decided that for the sake of maximum efficiency, the consideration of these factors should be left to the Design Phase.)

The primary machine constraint considered in Job Design is the amount and type of auxiliary storage (i.e., tapes, disc, drums) available to the subsystem. Since, by definition, no human intervention is permitted during the operation of a job, the required auxiliary storage configuration cannot exceed that which is available. For example, if a maximum of ten tape drives is available to the subsystem, no desired auxiliary storage configuration can exceed ten tapes.

The first thing, then, that must be done for each logical job is to determine the required auxiliary storage configuration. The data sets defined in the Preliminary Program Subsystem Specification are re-analyzed for pertinency to the given job. The maximum size of each pertinent data set (e.g., the maximum number of missile units) is used to determine the amount of auxiliary storage required for that data set. The data sets are organized into tape files, drum files, and disc data units. The tape files are grouped so as to form logical tapes.

If the desired auxiliary storage does not exceed that which is available, the logical job can be an actual job. If it does exceed, an attempt is made to change the desired auxiliary storage - by re-allocating files - and thereby forcing a fit. If this attempt fails, the logical job is either redefined or split into two or more logical jobs and the process of Job Design starts over again. Since the criterion for establishing jobs was human interaction, whenever new jobs are created, they must be coordinated with the ODR authors to insure feasibility from the point of view of the user.

The completion of Job Design is marked by the production of job flows which identify the gross auxiliary storage configuration required for each job.

GRAPHIC NOT REPRODUCIBLE

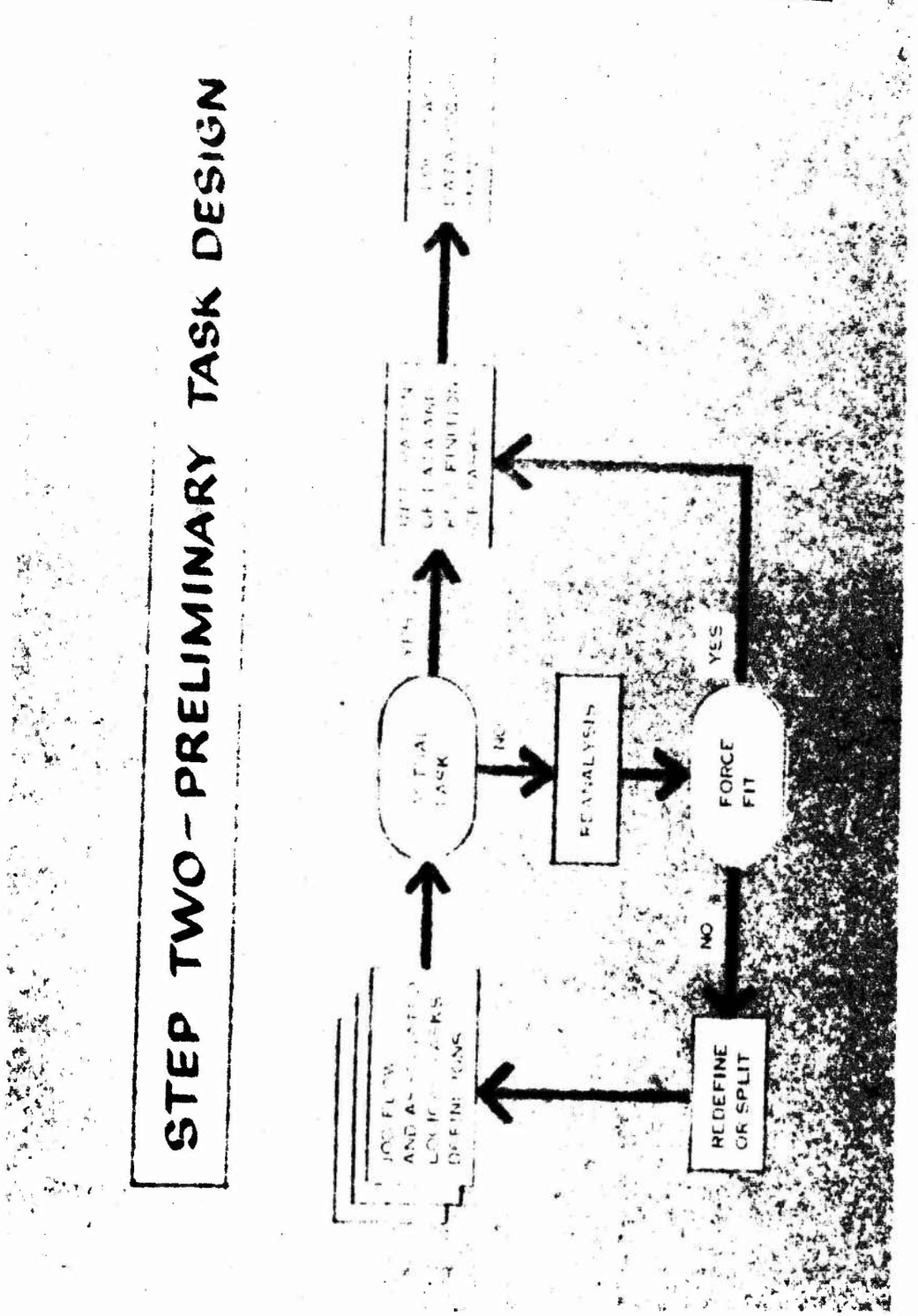


Figure 12

Step Two--Preliminary Task Design

The function of Preliminary Task Design is to design actual tasks from the logical tasks developed in the Translation Phase. The primary programming constraint is available core memory space (to be distinguished from available auxiliary storage - i.e., tapes, disc, drums - space, which was the constraint in the preceding step.)

A preliminary analysis of the job flow for the actual job and of the associated logical task definitions is performed to determine which of the data now defined on tape, disc and drum are required for a given task's operation.

The maximum size of the required data in core is then determined. This maximum size can differ from the size of files since not all the data contained in the files are necessarily required at the same time in core for calculations. For example, a file might contain data defining all missile units, but the logical task would require, at any given time, only the data defining one unit.

The total required amount of core is then computed. This is done by adding the amount of core required for the data, to an estimated amount required for the instructions which will accomplish the data functions. If this total required amount does not exceed the amount of core available, then the logical task can become an actual task. Otherwise, re-analysis must be made to reduce the amount of data and/or the number of instructions required in core for the task. Factors considered include the relationship (i.e., is the task linear, iterative, or a combination), the complexity, and the size of each function of the task.

If this re-analysis indicates that the task or data can be redesigned to fit into core, then the logical task can become an actual task. Otherwise, the logical task must be split into two or more logical tasks and the process of Task Design starts again.

Next, task designers assess all changes that have been made to the data files. This leads to integration of data and/or redefinition of tasks.

The completion of Preliminary Task Design is marked by the production of the gross task data requirements, both in terms of gross core configurations and data transfers.

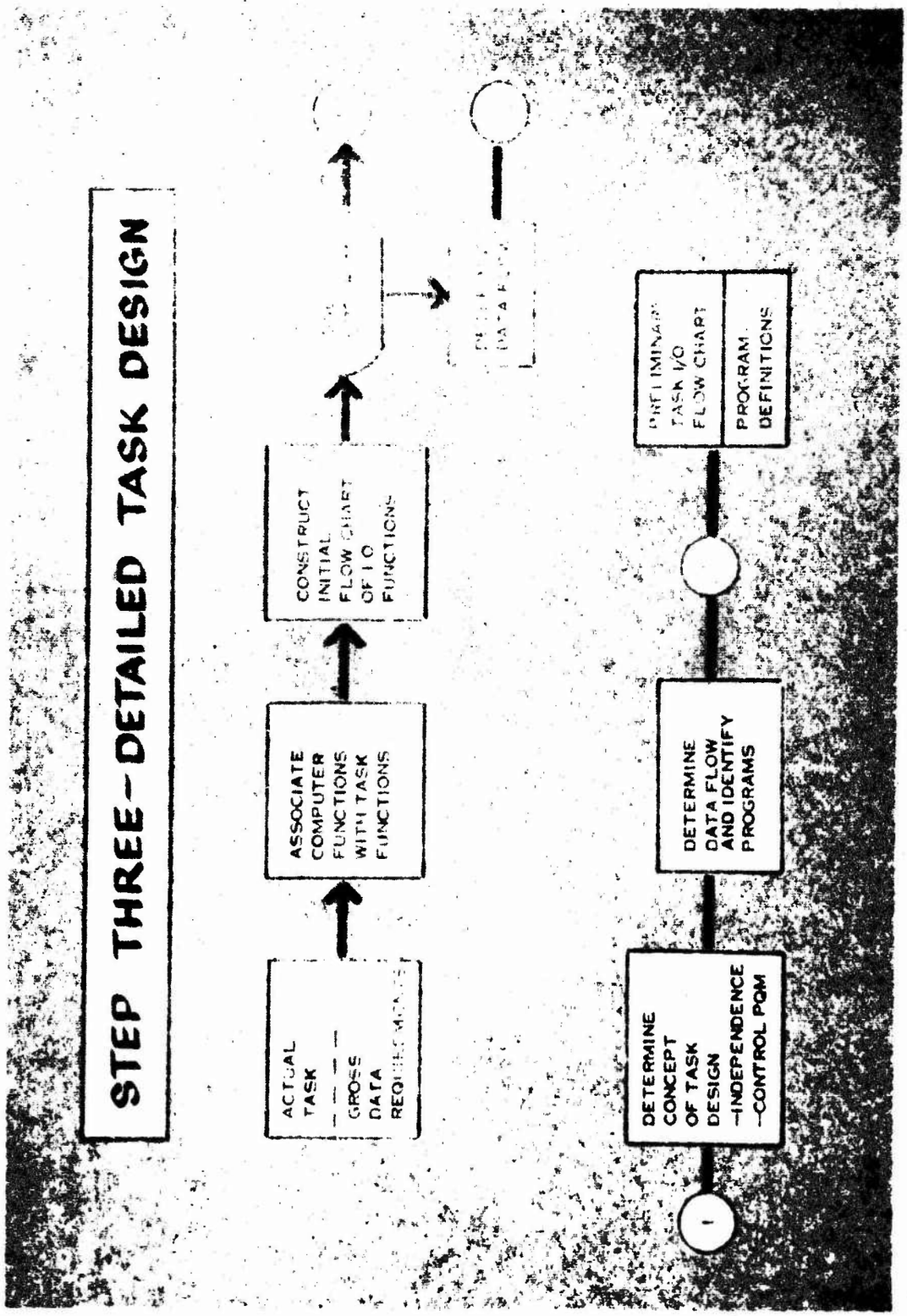


Figure 13

GRAPHIC NOT REPRODUCIBLE

Step Three--Detailed Task Design

Given actual tasks and gross data requirements, the functions of this step are to associate the computer functions necessary to accomplish the given task functions and to fraction the task into programs.

Computer functions such as reading, writing, sorting and searching are associated (wherever necessary) with the operational functions identified in the Translation Phase. This complete set of functions is illustrated in an initial task flow.

This initial task flow is analyzed to determine whether a single program would be sufficient to perform the processing required to satisfy all functions. (A single program would be sufficient if the amount of processing is small, or is not readily split into logical entities.) If a single program sufficies, design proceeds with a determination of the data flow and the preparation of a more specific form of the task flow (the preliminary task I/O flow chart.)

If a single program does not suffice, the initial task flow is then further analyzed to determine the concept of task design to be employed. There are two such concepts. One (the "control program" concept) is to have one program control the operation of all other programs. The other (the "independence" concept) is to have each program operate relatively independently of the others. Factors which argue for the adoption of the "control program" concept are a non-linear order of task function operation and a significant amount of common processing.

Once the concept has been determined, the design proceeds with a more detailed analysis of the data flow and a concurrent identification of programs. This process considers various constraints imposed by the physical configuration of the machine and by the System Control Program (the "Executive" in the 465L System.) The programs will consist of subsets of the functions performed by the task.

Task designers in each functional area identify common functions - and, therefore, programs - and insure that there is a consistent data base. Throughout the Design Phase, programmers whose sole responsibility is to insure a consistent and efficient data base are working with representatives of each functional area toward that end.

The completion of Detailed Task Design is signaled by the production of the preliminary task I/O flow charts and identification of each program.

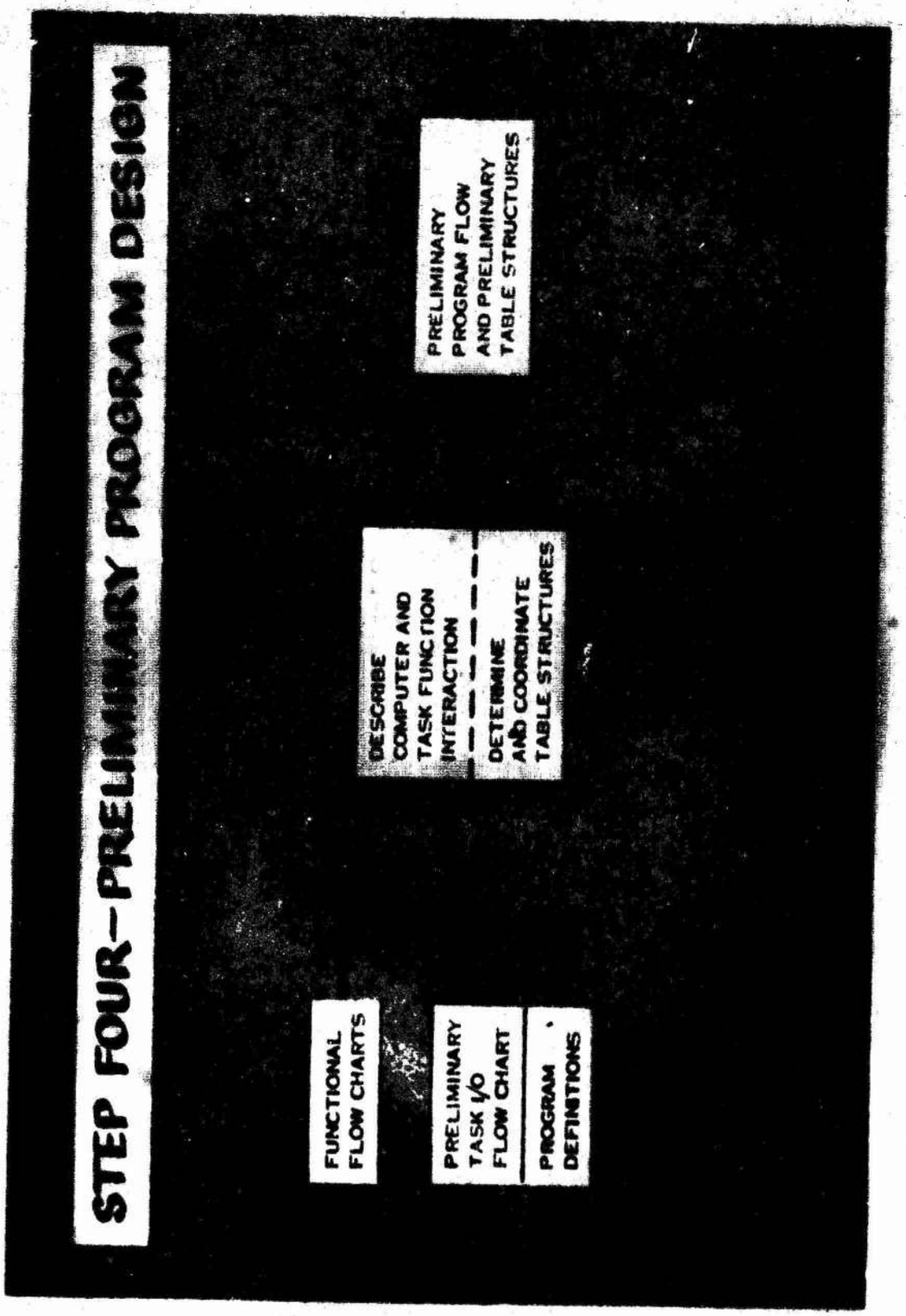

STEP FOUR—PRELIMINARY PROGRAM DESIGN
FUNCTIONAL FLOW CHARTS
PRELIMINARY TASK I/O FLOW CHART
PROGRAM DEFINITIONS
DESCRIBE COMPUTER AND TASK FUNCTION INTERACTION
DETERMINE AND COORDINATE TABLE STRUCTURES
PRELIMINARY PROGRAM FLOW AND PRELIMINARY TABLE STRUCTURES

Figure 14

Step Four--Preliminary Program Design

The functions of this step are to produce preliminary program design flows (from the functional flow charts and added computer functions) and to structure the tables containing the data to be processed. The more important constraints are imposed by types of table structures and tagging conventions which must be adhered to. Close coordination is mandatory since many programs process the same data.

Completion of this step is indicated by the production of preliminary program design flows and preliminary table structures.

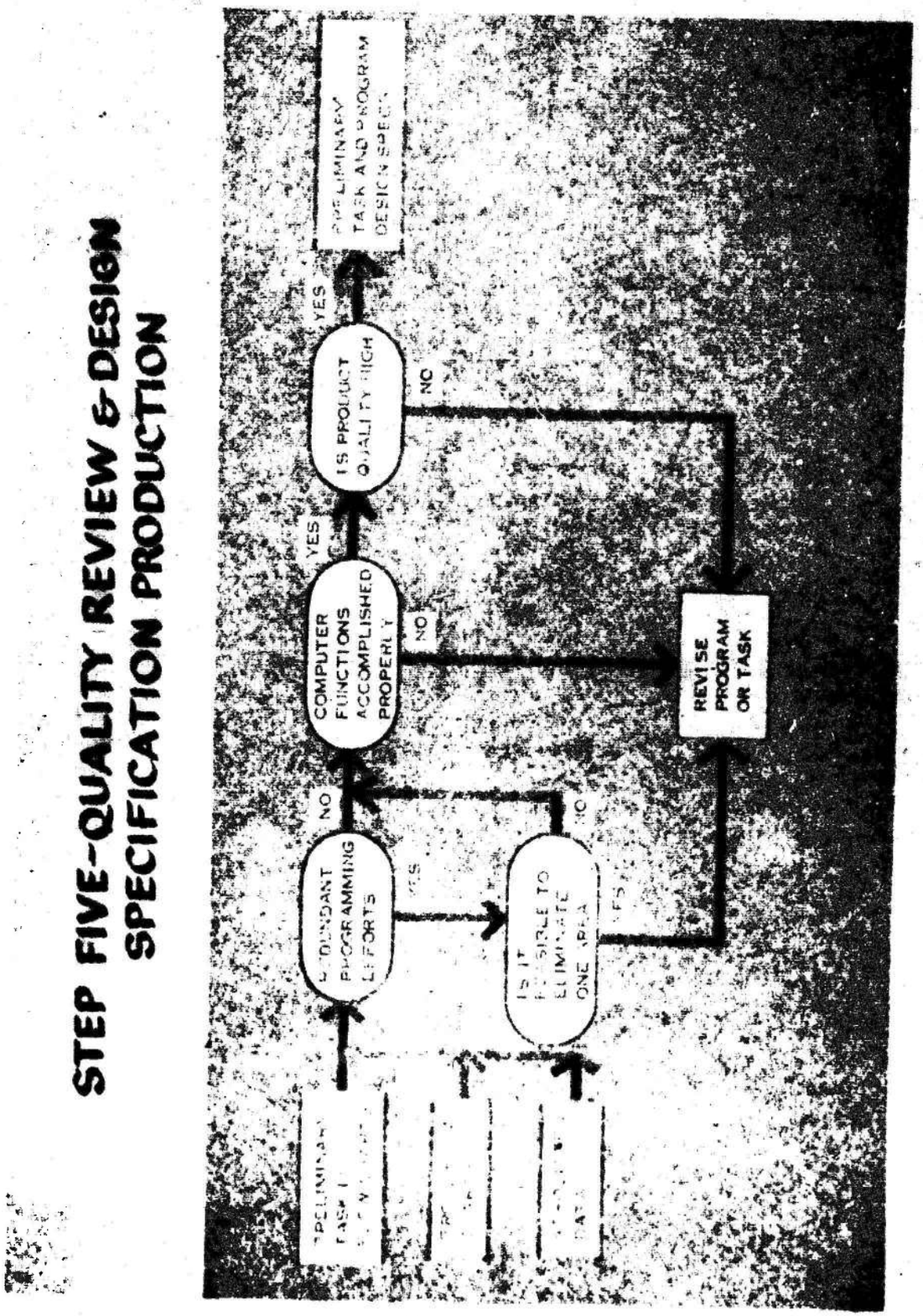


Figure 15

Step Five--Quality Review And Design Specification Production

The preliminary design of all programs and tables, and the detailed design of all tasks, are reviewed at a series of quality review meetings which are attended by representatives from all functional areas. The purpose of these meetings is three-fold:

1. To eliminate future redundant programming efforts;

2. To determine if computer functions are being accomplished at the most opportune points;

3. To insure that the quality of the product is high.

Upon completion of these meetings, the programs, tasks, and jobs are revised as necessary and are documented in accordance with detailed document format guidance. The preliminary Program Design Specifications include, for each program, a statement of the program's responsibility, a description of its environment, and a program design flow. The preliminary Task Design Specification includes a description of the task's responsibility, a description of its environment, its outputs, core configurations, auxiliary storage configurations, I/O flow and a detailed description of the I/O flow. The detailed job flows are prepared for later inclusion in the Final Program Subsystem Specification.

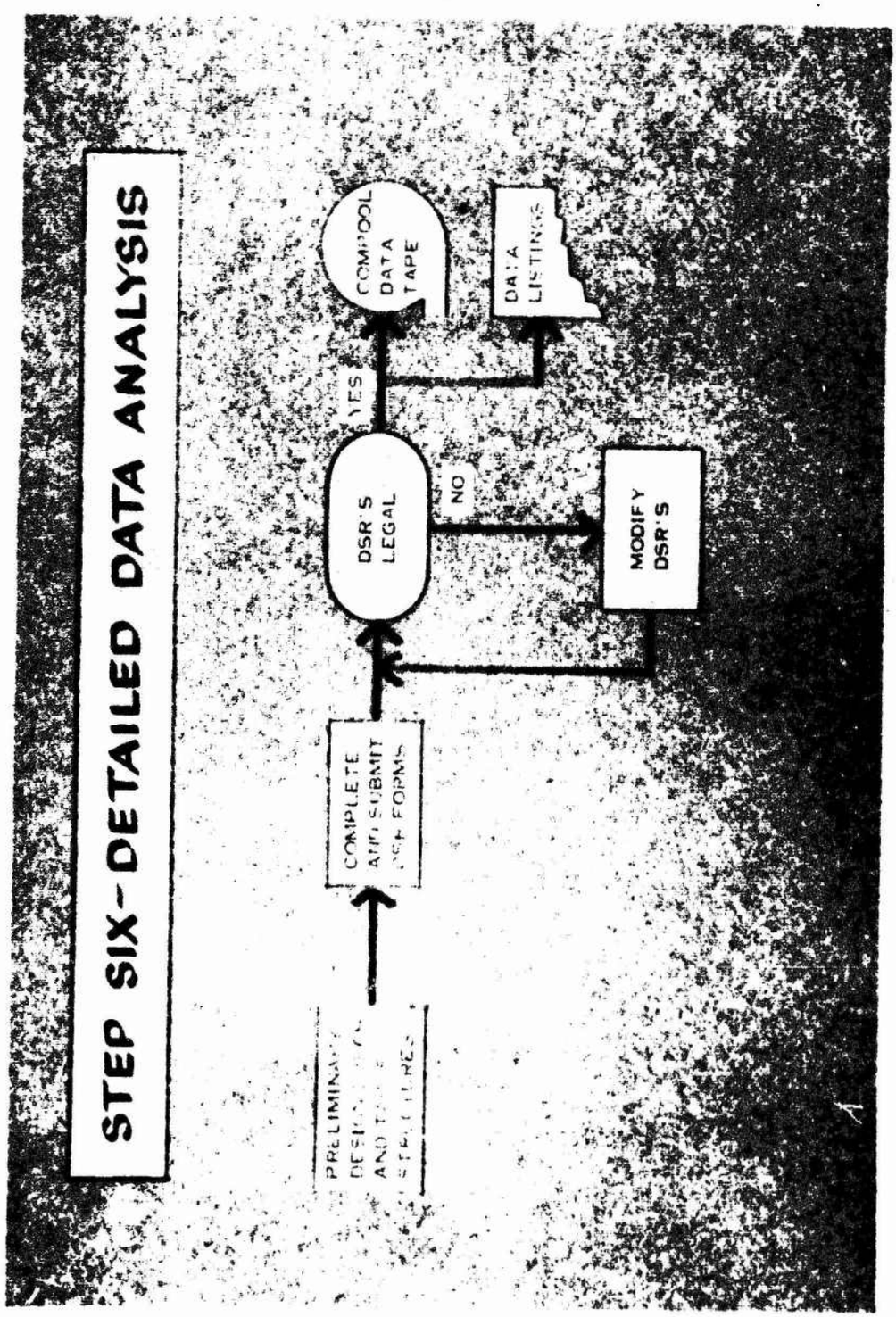

STEP SIX-DETAILED DATA ANALYSIS
COMPLETE
DSR'S LEGAL
YES
NO
COMPOOL DATA TAPE
DATA LISTINGS
MODIFY DSR'S

Figure 16

Step Six--Detailed Data Analysis

The functions of this step are to fix data definitions and to record the data in the data dictionary (known as the "Compool" in the 465L System.) Each item in each table is coordinated and fully defined. Data Specification Request (DSR) forms are filled out for each item, table and file. Data are legality checked and modifications made where necessary. The data are then placed onto the Compool tape and listings are produced.

From this point in the production process, the data base is fairly static, and changes are made on a more formal basis (i.e., by coordinating and submitting written change requests or DSR's.)

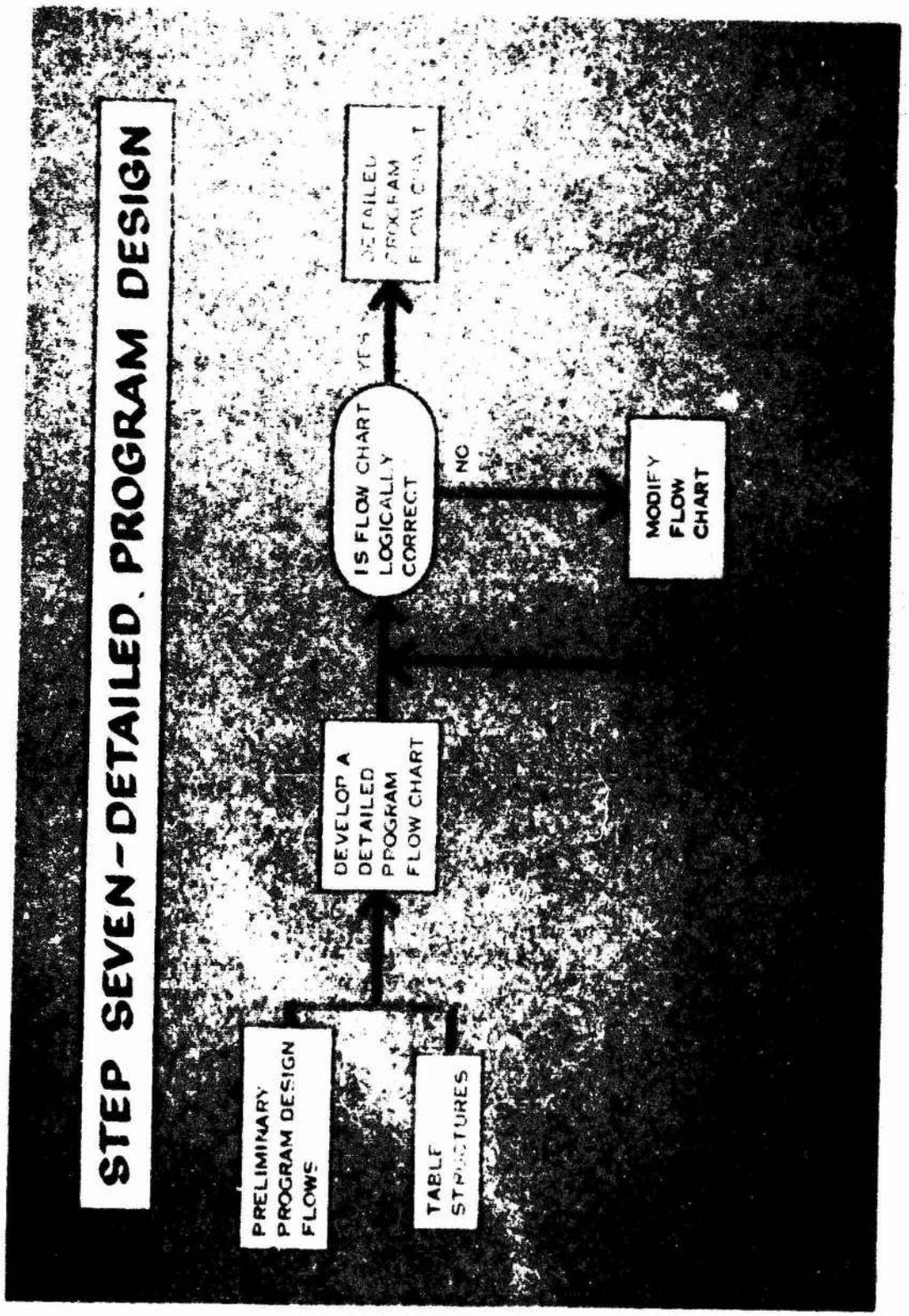

STEP SEVEN-DETAILED PROGRAM DESIGN
PRELIMINARY PROGRAM DESIGN FLOWS
TABLE STRUCTURES
DEVELOP A DETAILED PROGRAM FLOW CHART
IS FLOW CHART LOGICALLY CORRECT
NO
MODIFY FLOW CHART

Figure 17

Step Seven--Detailed Program Design

The last step in the Design Phase is the development of detailed program flows. This process involves a further definition of each box of the preliminary program design flows. Logical statements are produced which are at such a level that they can be translated one-for-one into JOVIAL statements (JOVIAL is the higher order programming language used at the System Development Corporation.) While developing his flows, the programmer uses the data specifications produced in Step Six above. When the flow is completed, the technical supervisor reviews it for accuracy and consistency, and changes are made as necessary.

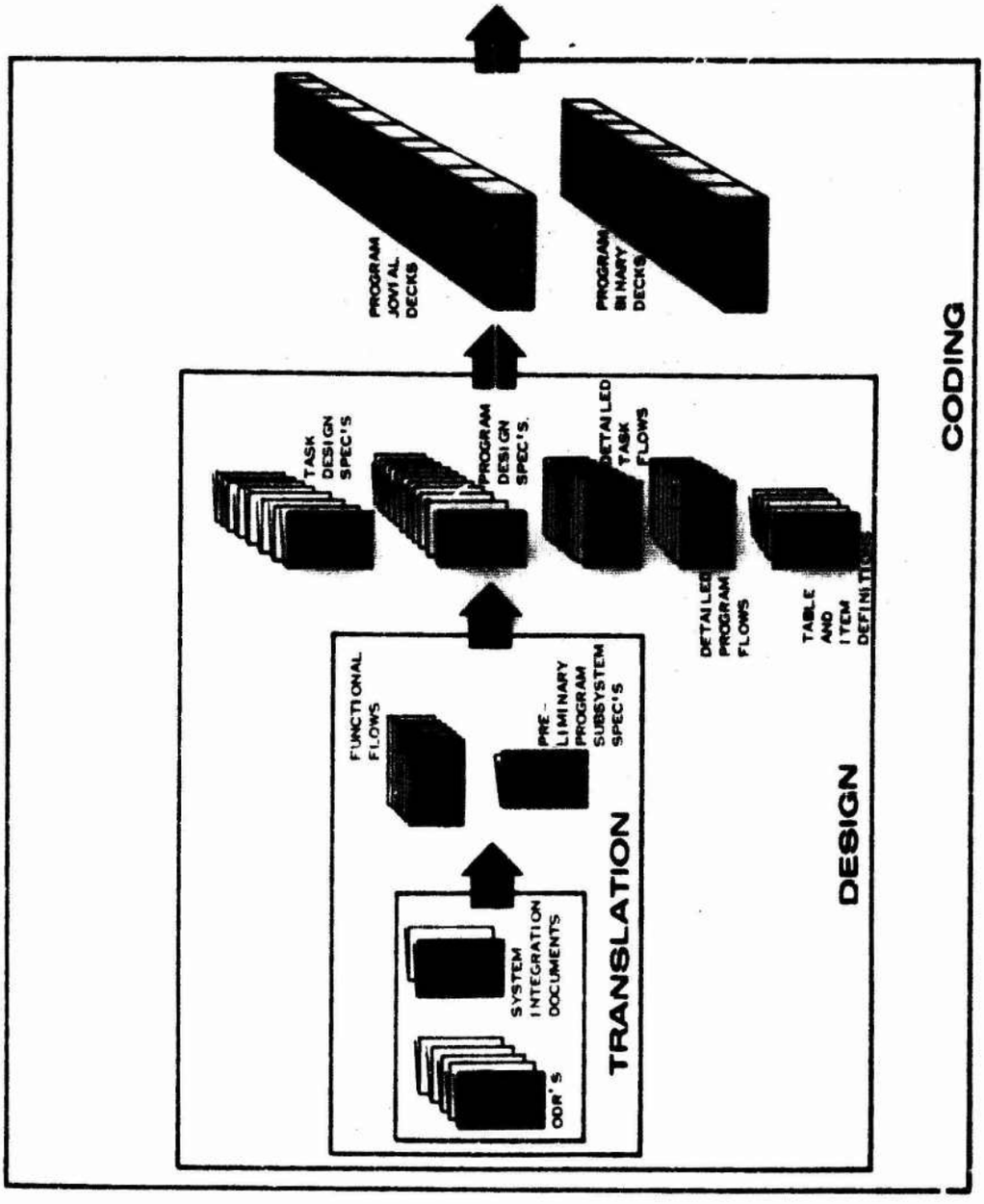

CODING
DESIGN
TRANSLATION
PROGRAM JOVIAL DECKS
TASK DESIGN SPEC'S
PROGRAM DESIGN SPEC'S.
DETAILED TASK FLOWS
FUNCTIONAL FLOWS
PRE-LIMINARY PROGRAM SUBSYSTEM SPEC'S
SYSTEM INTEGRATION DOCUMENTS
ODR'S

Figure 18

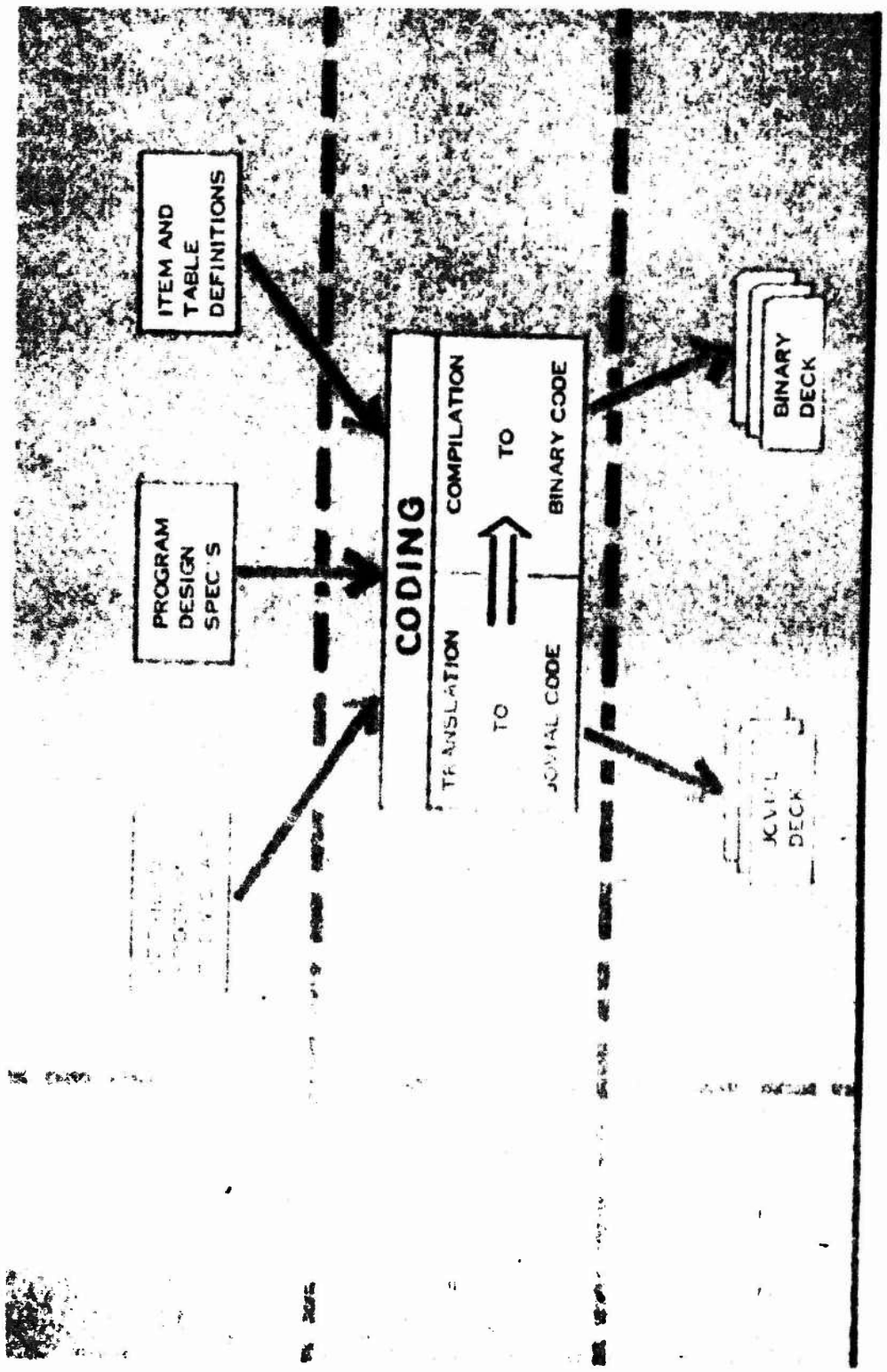
IV. CODING PHASE

Purpose Of The Coding Phase

The purpose of this phase is to translate programs heretofore defined in terms of design specifications and detailed program flows, into sets of higher order language statements, and to compile sets of machine language instructions from these higher order language statements.

Problems In Coding And Their Solution

Experience has shown that many problems arise because of the size and complexity of the program subsystem, and because of inexperienced programmers. Some of these problems are production of inefficient code (and consequent lengthening of the Verification Phase), and improper utilization of EAM and EDPM facilities. A way of minimizing these problems is to procedurize the Coding Phase.



METHODOLOGY OF CODING

Methodology Of Coding

In general, the inputs to this phase are the detailed program flow charts, the Program Design Specifications, and the complete data definitions.

Coding starts with the translation of the logical statements, contained on the detailed program flow diagrams, into equivalent JOVIAL higher order language statements. These higher order language statements are punched onto cards. The decks of cards are then submitted for compilation, and errors are corrected until an error-free binary deck is obtained.

Adherence to coding conventions and procedures, frequent review of the product by the technical supervisor, and the fact that much of the work usually done in the Coding Phase has already been done as the last part of the Design Phase, are sufficient to insure optimum progress.

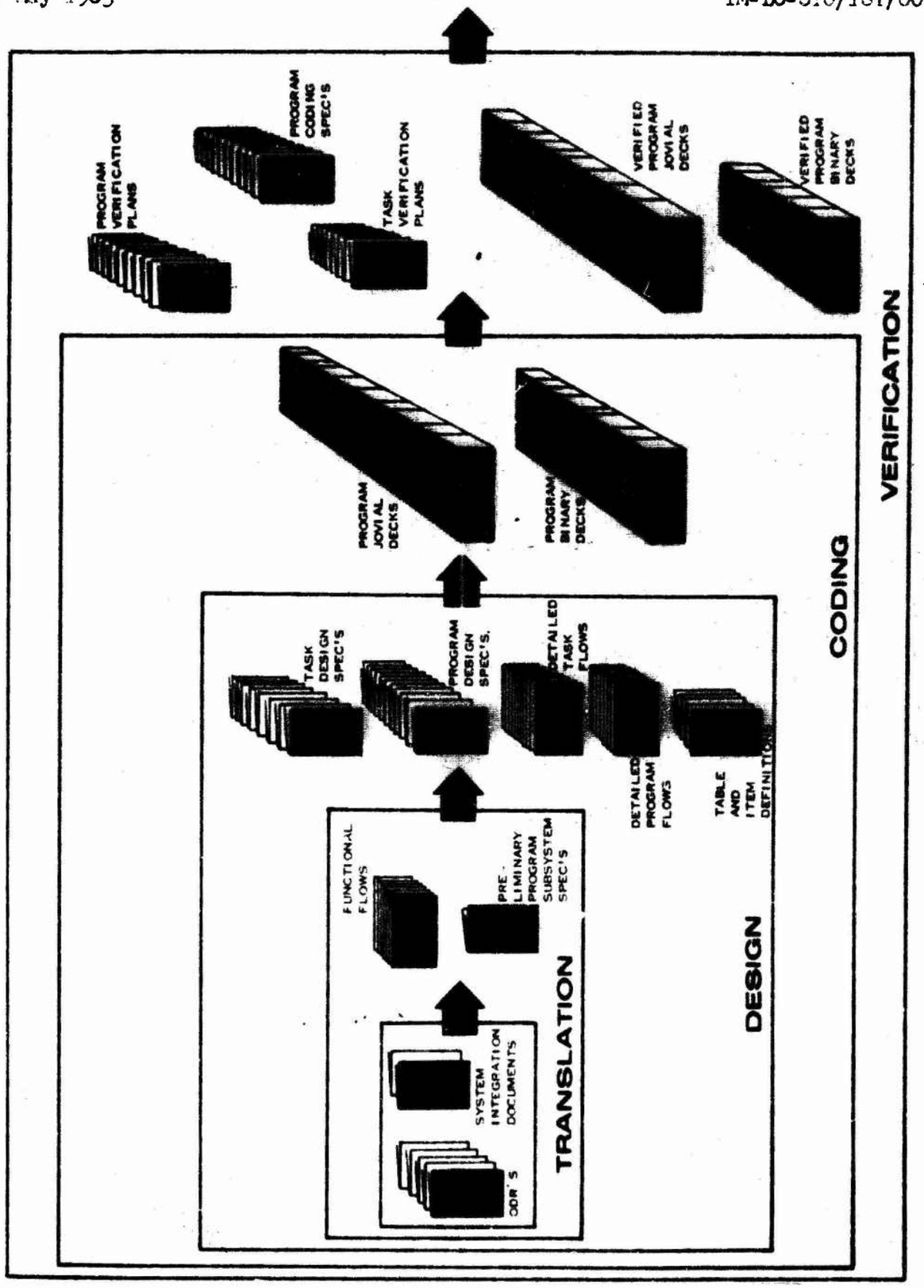

TRANSLATION
SYSTEM INTEGRATION DOCUMENTS
ODR'S
FUNCTIONAL FLOWS
PRELIMINARY PROGRAM SUBSYSTEM SPEC'S
DESIGN
TASK DESIGN SPEC'S
PROGRAM DESIGN SPEC'S.
DETAILED TASK FLOWS
DETAILED PROGRAM FLOWS
TABLE AND ITEM DEFINITION
CODING
PROGRAM JOVIAL DECKS
PROGRAM BINARY DECKS
VERIFICATION
PROGRAM VERIFICATION PLANS
PROGRAM CODING SPEC'S
TASK VERIFICATION PLANS
VERIFIED PROGRAM JOVIAL DECKS
VERIFIED PROGRAM BINARY DECKS

Figure 20

V. VERIFICATION PHASE

Purpose Of The Verification Phase

The purpose of this phase is to purge the program subsystem of errors. The goal of verification is to produce an error-free subsystem. It should be noted, however, that this is a goal which is never totally realized. In striving towards this goal, the producer of a large computer program subsystem corrects all the errors he detects as a result of running a set of preplanned test cases. The realities of life prevent him from verifying the literally millions of possible paths through the subsystem.

Problems In Verification And Their Solution

Historically, in this phase, production efforts bog down and schedules slip. Many reasons are presented to rationalize this problem. Examples are the complexity and size of the program subsystem and the inexperience of the programmers involved. But perhaps more basic causes of this problem are the absence of a defined verification methodology and the consequent inability of management to accurately assess progress and thereby to control production. If there is no organized approach, programmers tend to over-verify some areas and neglect others.

A solution is a system approach to verification, one in which levels of verification are introduced and for which, within each level, a well-defined procedure is established. The keys to establishing the procedures are the designation of a specific goal for each level of verification and the identification of interim products in the verification process. The interim products provide for managerial inspection and allow the programmer to direct his work toward the stated goal.

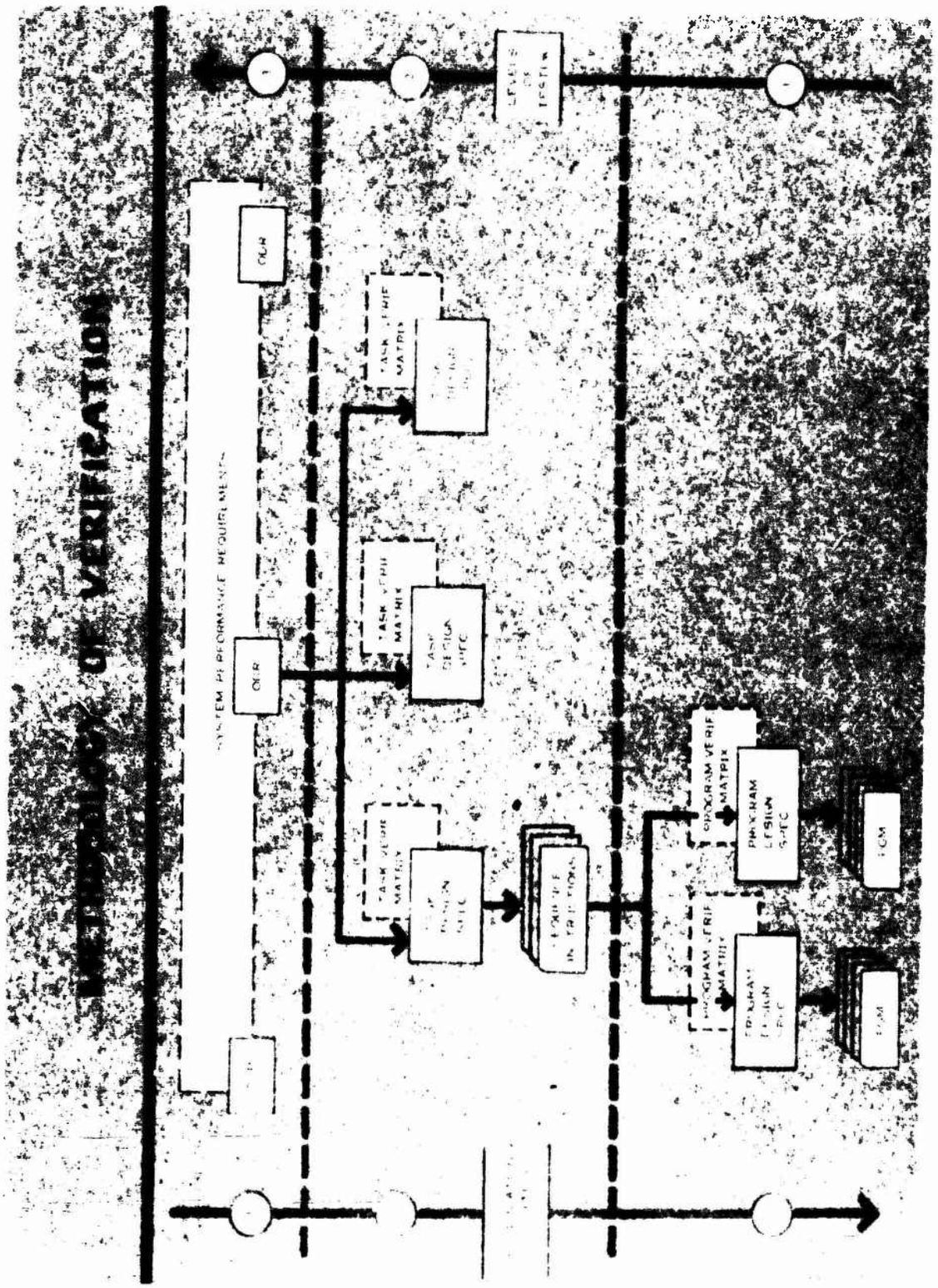

METHODOLOGY OF VERIFICATION
SYSTEM PERFORMANCE REQUIREMENTS
CDR
CDR
TASK VERIF MATRIX
TASK DESIGN SPEC
TASK VERIF MATRIX
TASK DESIGN SPEC
PROGRAM VERIF MATRIX
PROGRAM DESIGN SPEC
PROGRAM VERIF MATRIX
PROGRAM DESIGN SPEC
TCM
TCM

Figure 8

Methodology Of Verification

The system approach to verification relates directly to the manner in which the program subsystem has thus far been developed and documented. First, the system designers generated and published ODR's. Given these ODR's, the programmers designed tasks, then programs. They documented these in Task and Program Design Specifications. The "functions which are to be tested" are also documented at each level; i.e., system performance requirements, task verification matrices, and program verification matrices.

The approach, then, is to verify against each of the three levels of specifications. At the three levels, essentially the same set of instructions is being verified against three different sets of criteria.

Starting with the level of greatest detail, and utilizing the program verification matrices, programs are verified against Program Design Specifications. The manipulation of data in core is verified.

Utilizing the task verification matrices, tasks are then verified against Task Design Specifications. Core-to-I/O device and I/O device-to-core data transfers, and program intercommunication are verified.

Finally, utilizing the system performance requirements, the program subsystem is verified against the Operational Design Requirements. This level of testing is performed following the Internal Release Phase, and is therefore not covered in this paper.

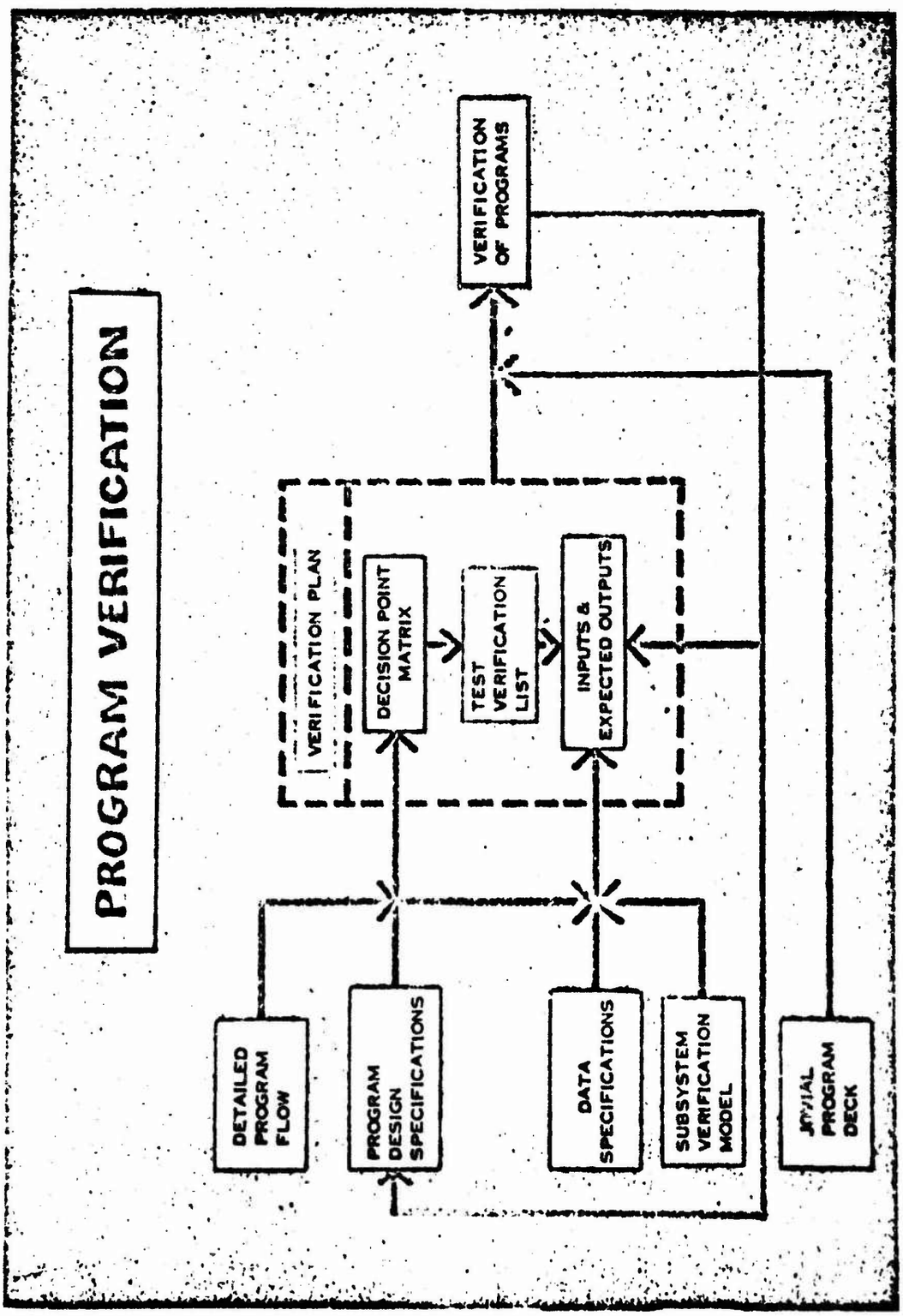

PROGRAM VERIFICATION
VERIFICATION OF PROGRAMS
VERIFICATION PLAN
DECISION POINT MATRIX
TEST VERIFICATION LIST
INPUTS & EXPECTED OUTPUTS
DETAILED PROGRAM FLOW
PROGRAM DESIGN SPECIFICATIONS
DATA SPECIFICATIONS
SUBSYSTEM VERIFICATION MODEL
JOVIAL PROGRAM DECK

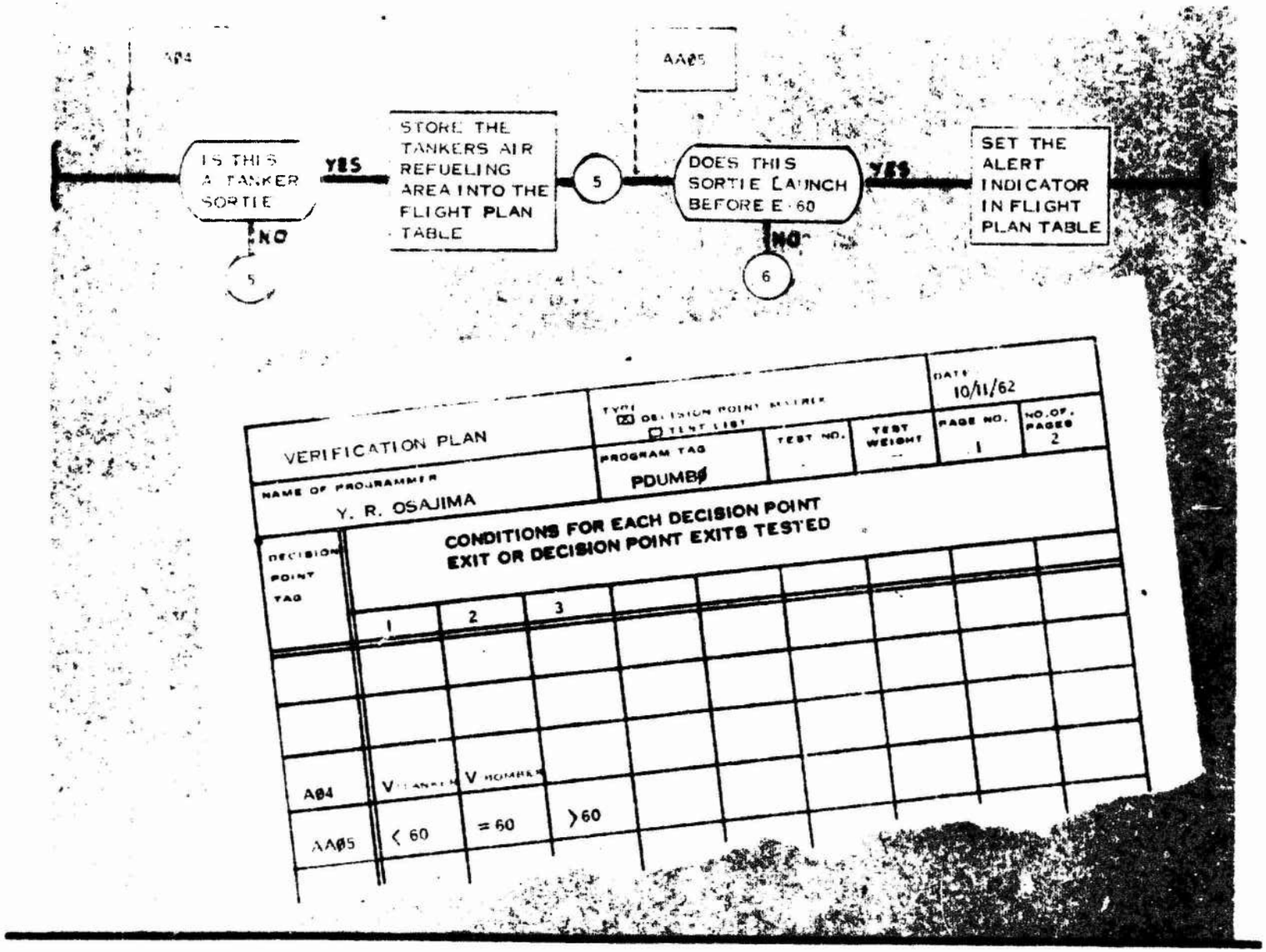
Figure 22

Program Verification

The specific goal of program verification is to verify each branch of each program decision point. Satisfaction of this goal insures the code's compatibility with the detailed program flow and verifies that the logic specified in the Design Phase is actually coded into the program. The inputs to program verification include the detailed program flow, Program Design Specifications, data specifications, subsystem verification model (discussed below), and, of course, the JOVIAL program deck. There are two basic steps in program verification, the preparation of a plan and the actual verification on the computer.

The first step of the procedure is the preparation of a verification plan. The verification plan consists of three elements:

1. a decision point matrix
2. test lists
3. inputs and expected outputs



THE DECISION POINT MATRIX

Figure 23

The decision point matrix is a device for presenting the program decision points in tabular form. Every decision point on the detailed program flow is labeled. If there is a corresponding symbolic region label in the code, then the same label will appear on the flow (e.g., AA05), otherwise a unique label will appear (e.g., A04.) The decision point labels are listed vertically on the matrix. Then, for each decision point, each branch is listed horizontally.

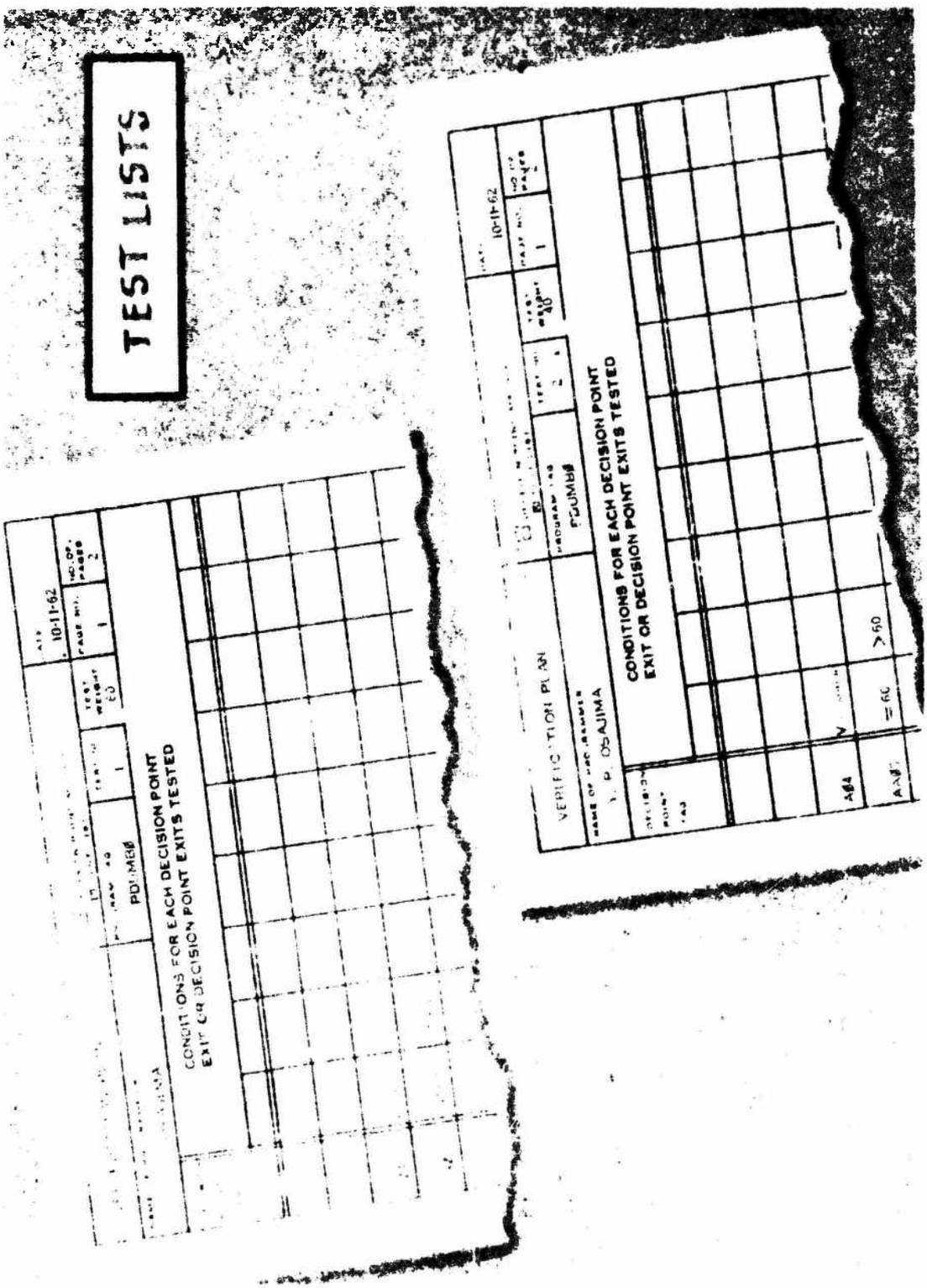

TEST LISTS
CONDITIONS FOR EACH DECISION POINT
EXIT OR DECISION POINT EXITS TESTED
VERIFICATION PLAN
CONDITIONS FOR EACH DECISION POINT
EXIT OR DECISION POINT EXITS TESTED
10-11-62
10-11-62

Figure 24

Next, the determination is made as to which branches of which decision points are to be activated in the first test. These decisions and these branches are indicated on a test list. The same form and method of presentation is used both for the decision point matrix and the test list. The header information on the form allows one to specify the "type" of use along with associated information (e.g., Test Number and Test Weight.) Next, the number of branches not yet activated is determined and additional tests (and test lists) are prepared. The fact that "paths" or combinations of branches have cumulative effects is recognized and as many paths as time permits are incorporated into the test lists.

In order to effectively prepare these test lists, a great deal of desk checking is performed on the program's logic. Errors found here can greatly minimize the time required for the actual verification on the computer.

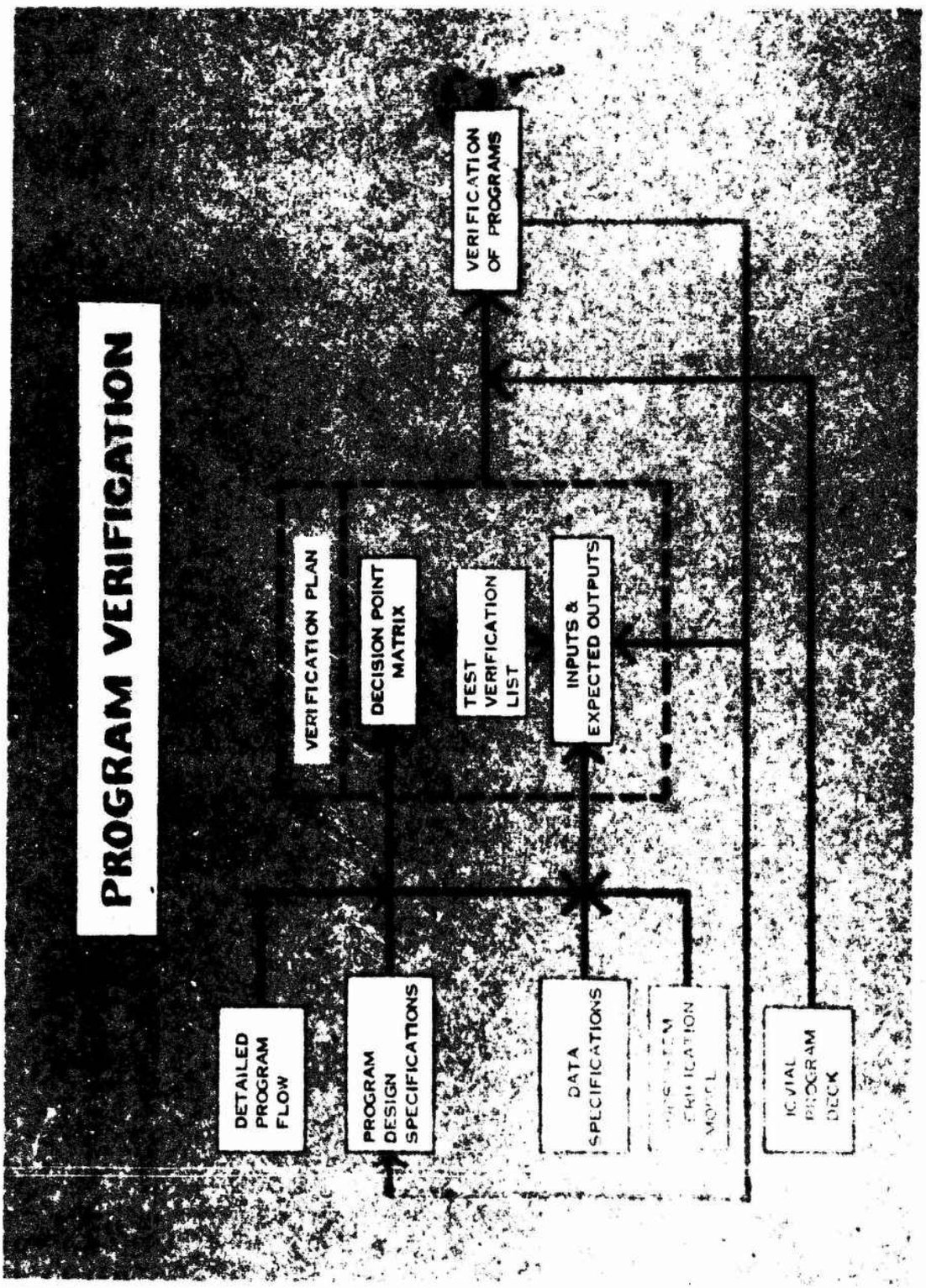

PROGRAM VERIFICATION
VERIFICATION OF PROGRAMS
VERIFICATION PLAN
DECISION POINT MATRIX
TEST VERIFICATION LIST
INPUTS & EXPECTED OUTPUTS
DETAILED PROGRAM FLOW
PROGRAM DESIGN SPECIFICATIONS
DATA SPECIFICATIONS
JOVIAL PROGRAM DECK

Figure 25

Upon completion of the test lists the inputs necessary to activate the branches specified in the test lists are generated and recorded. Wherever possible, these inputs are taken from the subsystem verification model. The subsystem verification model is a collection of representative data which describes the subsystem in miniature. In the Planning Subsystem it contains a sample attack force, a sample target system and the characteristics and capabilities of each. The usage of the verification model data insures a common basis for verification as will be explained in the discussion of Task Verification.

The expected outputs are then manually computed and recorded. The verification plan is reviewed for completeness and accuracy by the technical supervisor and revisions are made as needed.

Test weights are attached to each of the tests as a function of their size and complexity. The application of test weights facilitates a detailed schedule for verification. For example, if three tests were planned, the first weighted at 50, the second at 30, and the third at 20, and if the program is to be tested in ten weeks, then, in order for the schedule to be maintained, the first test should be completed after five weeks, the second after eight, the third after ten. This detailed schedule permits the manager to more closely assess program verification progress.

The second step in Program Verification is the actual verification of the program. Each test is run on the computer and expected outputs are compared with actual outputs. Variances are noted and then causes are searched out and corrected. The program is considered to be verified when all expected outputs and actual outputs agree.

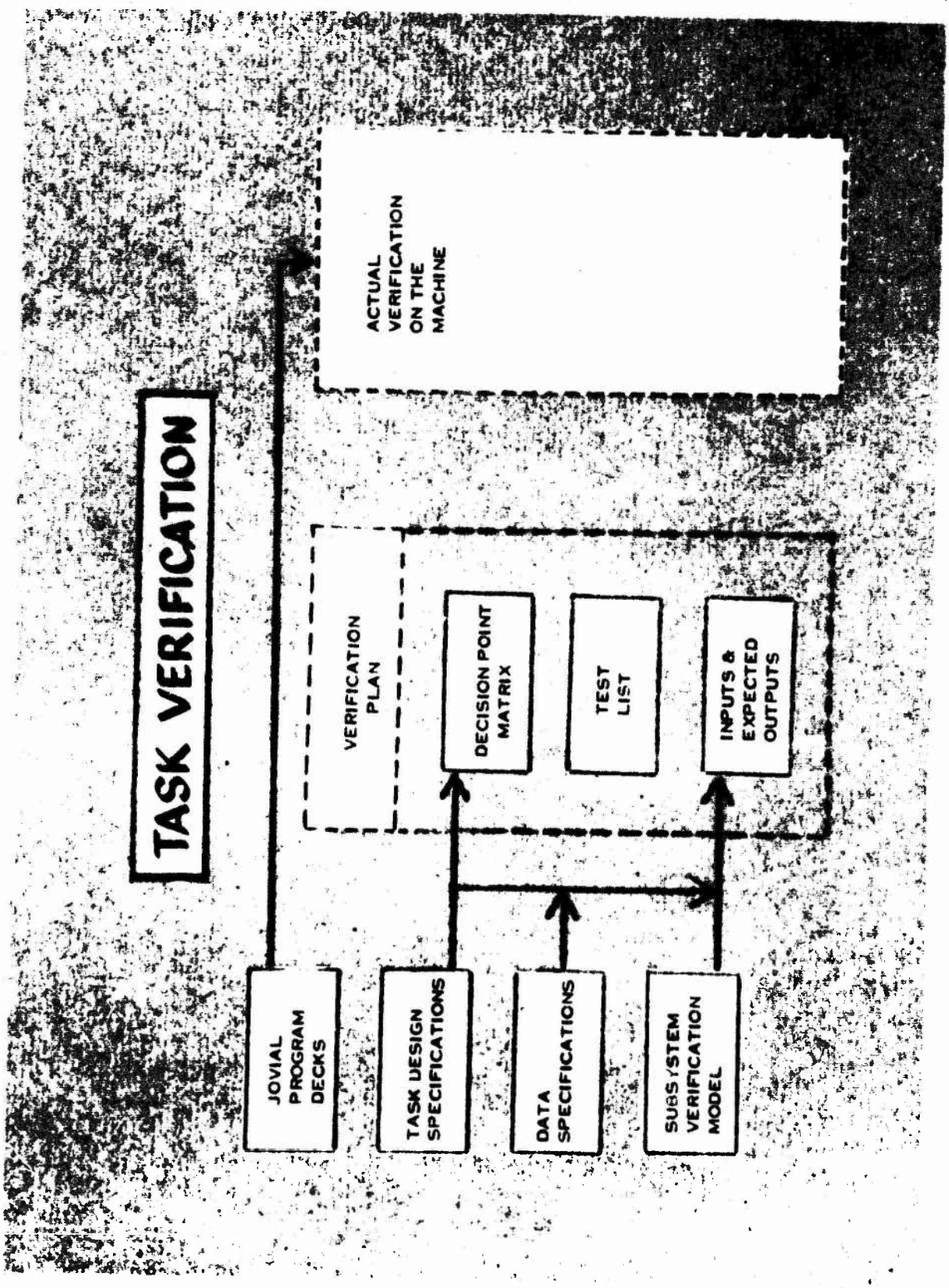

TASK VERIFICATION
ACTUAL VERIFICATION ON THE MACHINE
VERIFICATION PLAN
DECISION POINT MATRIX
TEST LIST
INPUTS & EXPECTED OUTPUTS
JOVIAL PROGRAM DECKS
TASK DESIGN SPECIFICATIONS
DATA SPECIFICATIONS
SUBSYSTEM VERIFICATION MODEL

Figure 26

Task Verification

The specific goal of Task Verification is to insure that each program interaction and each I/O operation functions properly. The inputs to Task Verification include the verified JOVIAL program decks, Task Design Specifications, data specifications, and the subsystem verification model.

In Task Verification, as in Program Verification, there are two basic steps, i.e., preparation of a plan and the actual verification on the machine.

The first step is the creation of a verification plan. As stated above, a specific goal of task verification is to insure that each I/O operation functions properly. This is accomplished by insuring that each sequence parameter (discussed below) is activated.

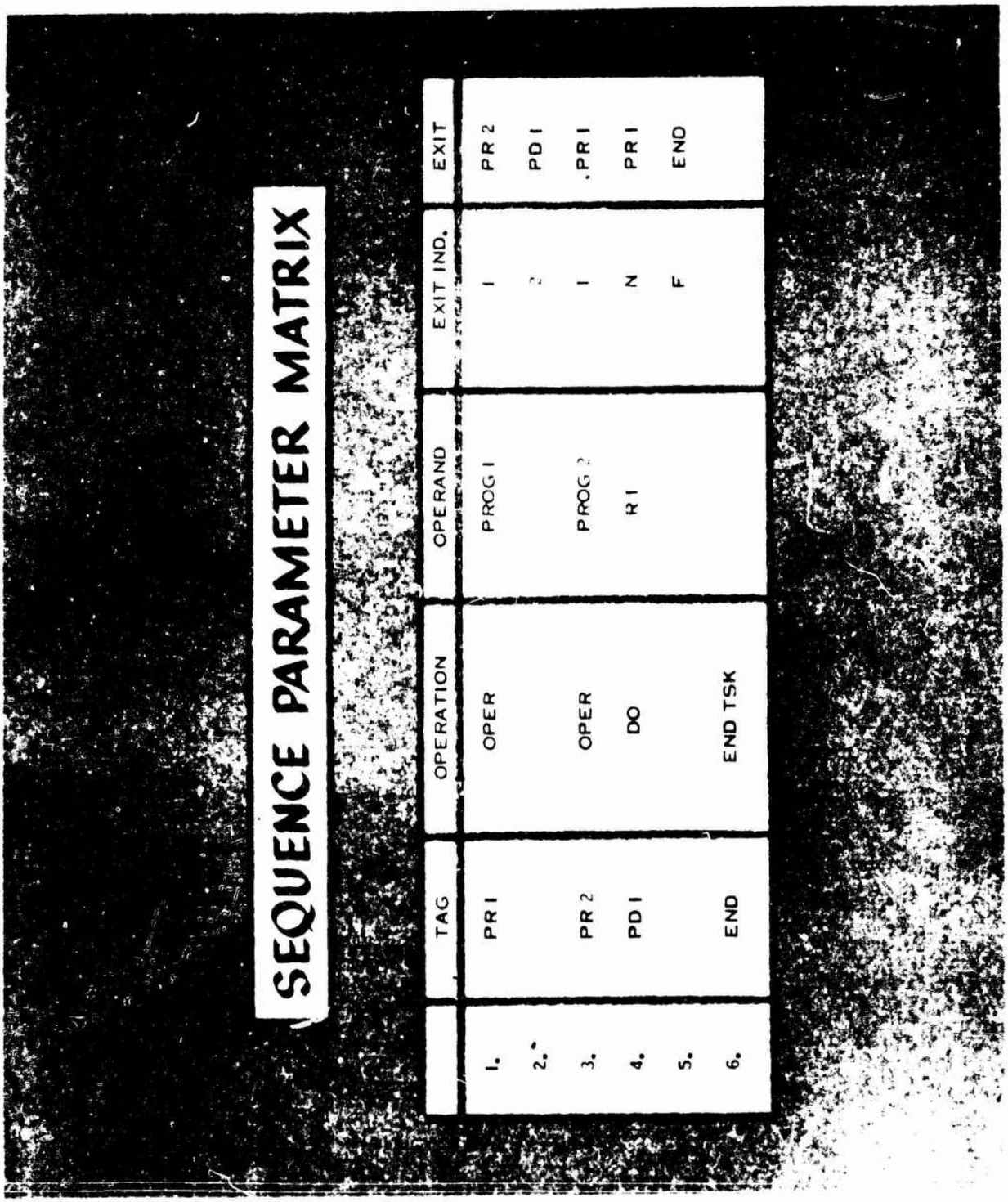

SEQUENCE PARAMETER MATRIX
TAG
OPERATION
OPERAND
EXIT IND.
EXIT
1.
PR 1
OPER
PROG 1
1
PR 2
2.
PD 1
3.
PR 2
OPER
PROG 2
1
PR 1
4.
PD 1
DO
R 1
N
PR 1
5.
F
END
6.
END
END TSK

Figure 27

Sequence parameters are higher order language statements which cause programs to be operated and I/O operations to be performed. They are the medium in which tasks are coded. These statements are prepared from a sequence parameter matrix which graphically portrays each operation and the order in which it is to be accomplished. This sequence parameter matrix is used as the task verification matrix.

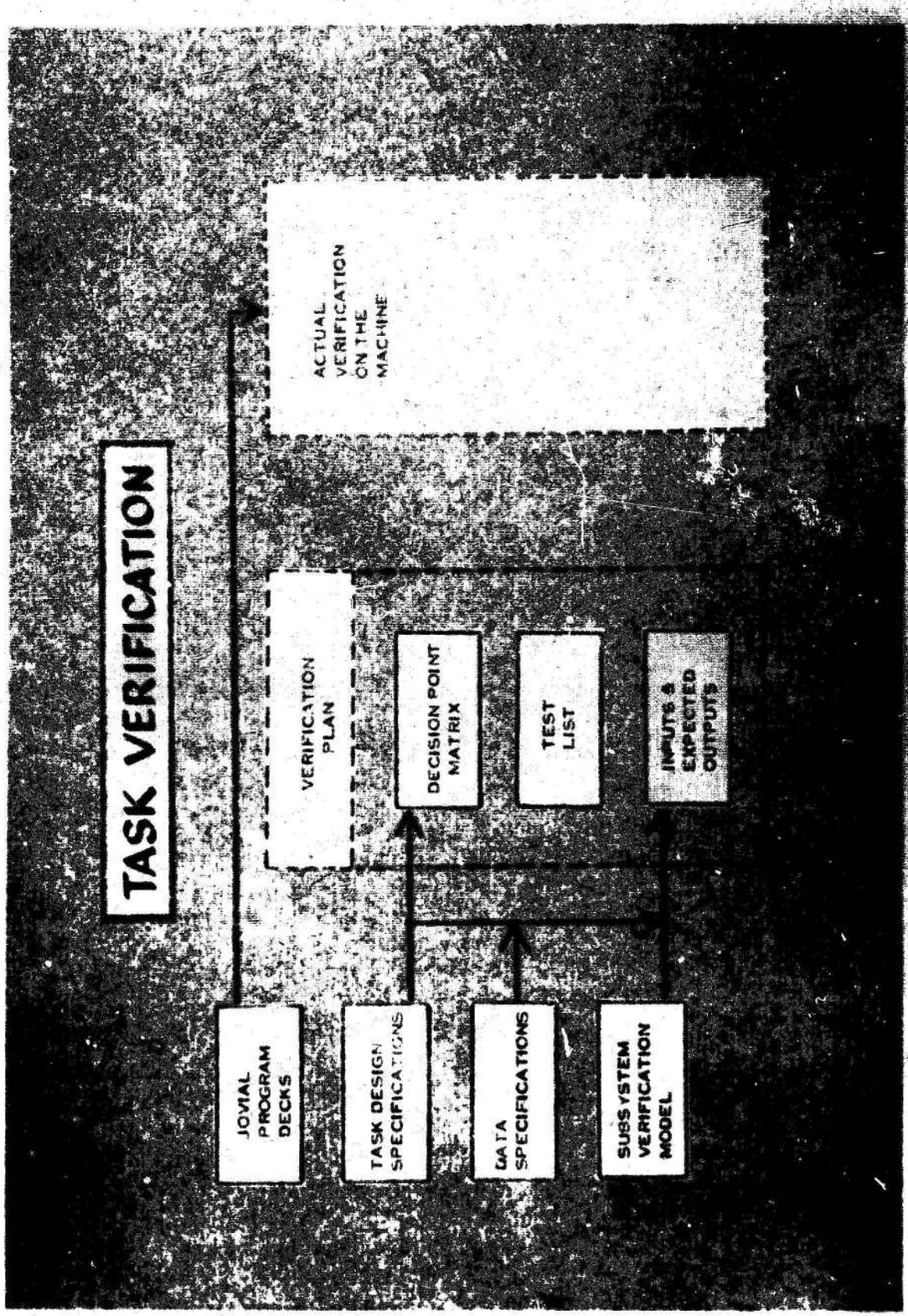

TASK VERIFICATION
JOVIAL PROGRAM DECKS
TASK DESIGN SPECIFICATIONS
DATA SPECIFICATIONS
SUBSYSTEM VERIFICATION MODEL
VERIFICATION PLAN
DECISION POINT MATRIX
TEST LIST
INPUTS & EXPECTED OUTPUTS
ACTUAL VERIFICATION ON THE MACHINE

Figure 28

Test lists indicating which sequence parameters are to be activated for each test are constructed from the task verification matrix.

In the Planning Subsystem a new concept of task verification evolved because many tasks contained a large number of programs and simultaneous verification of all programs at one time was found to be not feasible. This concept is called component verification, and is the verification of a portion of a task at a time. For example, assume that a task consists of programs A, B, C, D and E. One component, then, might consist of programs A and B, another of programs C and D, yet another of C, D and E, and finally, the largest component of A, B, C, D and E. When tasks are verified in this fashion, the task verification matrix developed for verifying components smaller than the total task is based on subsets of the sequence parameters which make up the total task.

Upon completion of the test lists, the inputs (needed to activate the specified sequence parameters and operate the tests) are devised and recorded. The subsystem verification model data previously employed in program verification are again used. Their usage in both program and task verification minimizes the need for manually calculating expected outputs during task verification, since the outputs calculated during program verification can be used.

Any expected outputs which have not been computed during program verification are now computed. All expected outputs are recorded.

The task verification plan is now complete. It is reviewed for accuracy and completeness by the technical supervisor and changes are made as needed. As in program verification, test weights are applied to enable the manager to more closely assess progress.

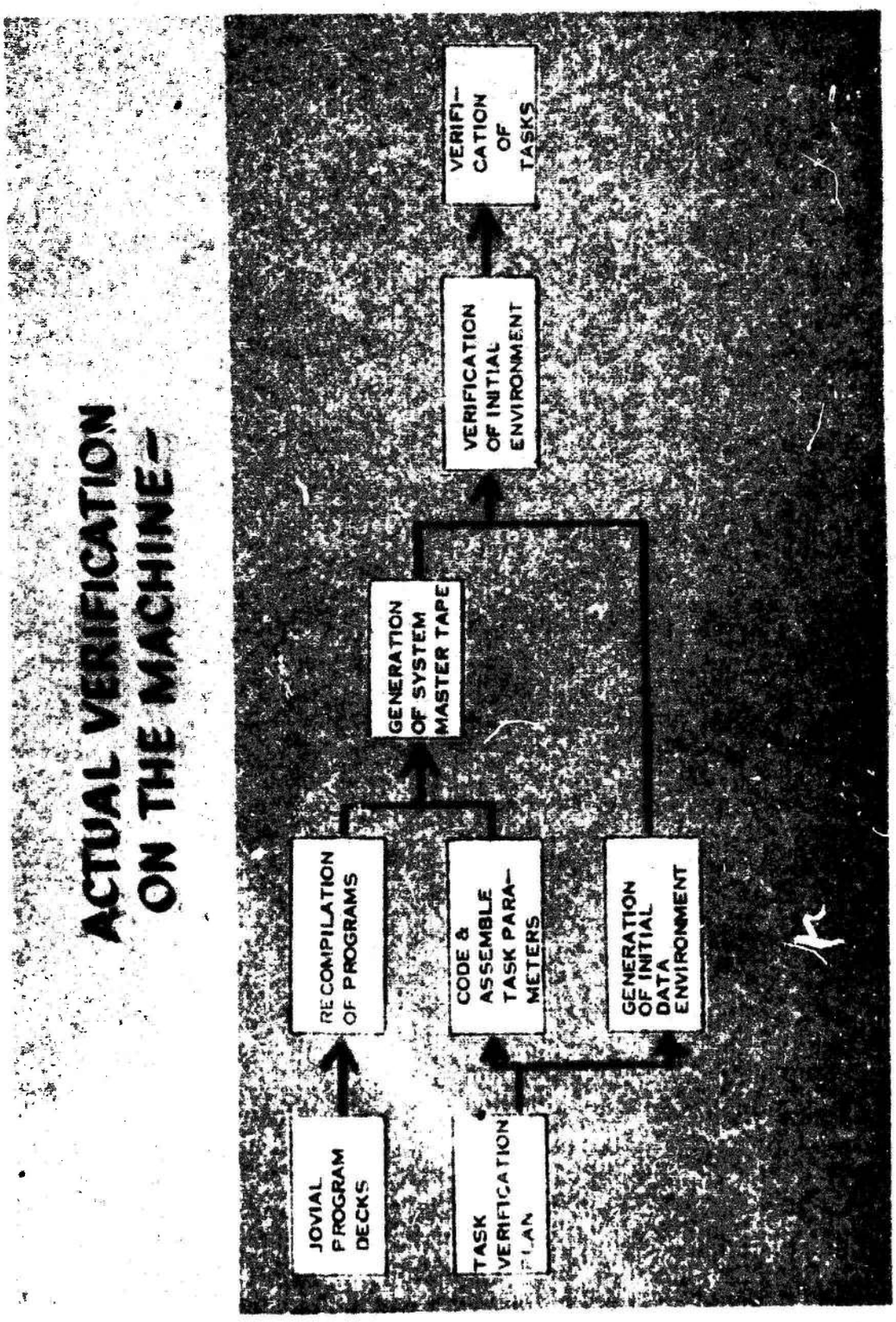

ACTUAL VERIFICATION
ON THE MACHINE
JOVIAL PROGRAM DECKS
TASK VERIFICATION PLAN
RECOMPILATION OF PROGRAMS
CODE & ASSEMBLE TASK PARA-METERS
GENERATION OF INITIAL DATA ENVIRONMENT
GENERATION OF SYSTEM MASTER TAPE
VERIFICATION OF INITIAL ENVIRONMENT
VERIFI-CATION OF TASKS

Figure 29

The second step in task verification is the actual verification on the computer. This key step in the production process requires many functions to be performed.

First - utilizing the JOVIAL program decks, every program is recompiled with the same version of the Compool (i.e., data dictionary.) This step is very important since the Compool changes fairly often and all programs must reflect the same data definitions. The results of the compilations are binary program decks.

Second - is the completion of the coding of task parameters. (There are two kinds of task parameters, the sequence parameters previously mentioned, and I/O parameters. I/O parameters completely define the I/O operations the task performs.) The task parameters are coded and symbolic decks and listings are produced. These are then submitted for assembly, which results in a task parameter binary deck and listing.

At this time the I/O Assignment cards are prepared. The System Control Program (the Executive), when initiating a task, first checks the mounted tapes against the assignments specified on these cards.

Third - is the generation or updating of the System Master Tape with the binary task parameter and program decks.

Fourth - is the generation of data environment. In devising the verification plan, the inputs were simply recorded on paper. The function of this step is to generate this same data on tape or disc.

Fifth - is the actual operation of the task for the purpose of determining whether the initial environment was correctly established by the Executive. The initial environment consists of all programs and data which are to be in core when the task begins to operate.

The last step is the complete operation of each test of the task on the machine. As in program verification, the expected outputs are compared with the actual computer outputs. Causes of variances are determined, changes are made and the task is rerun until the actual outputs match the expected outputs. When all actual and expected outputs agree, the task is considered to be verified.

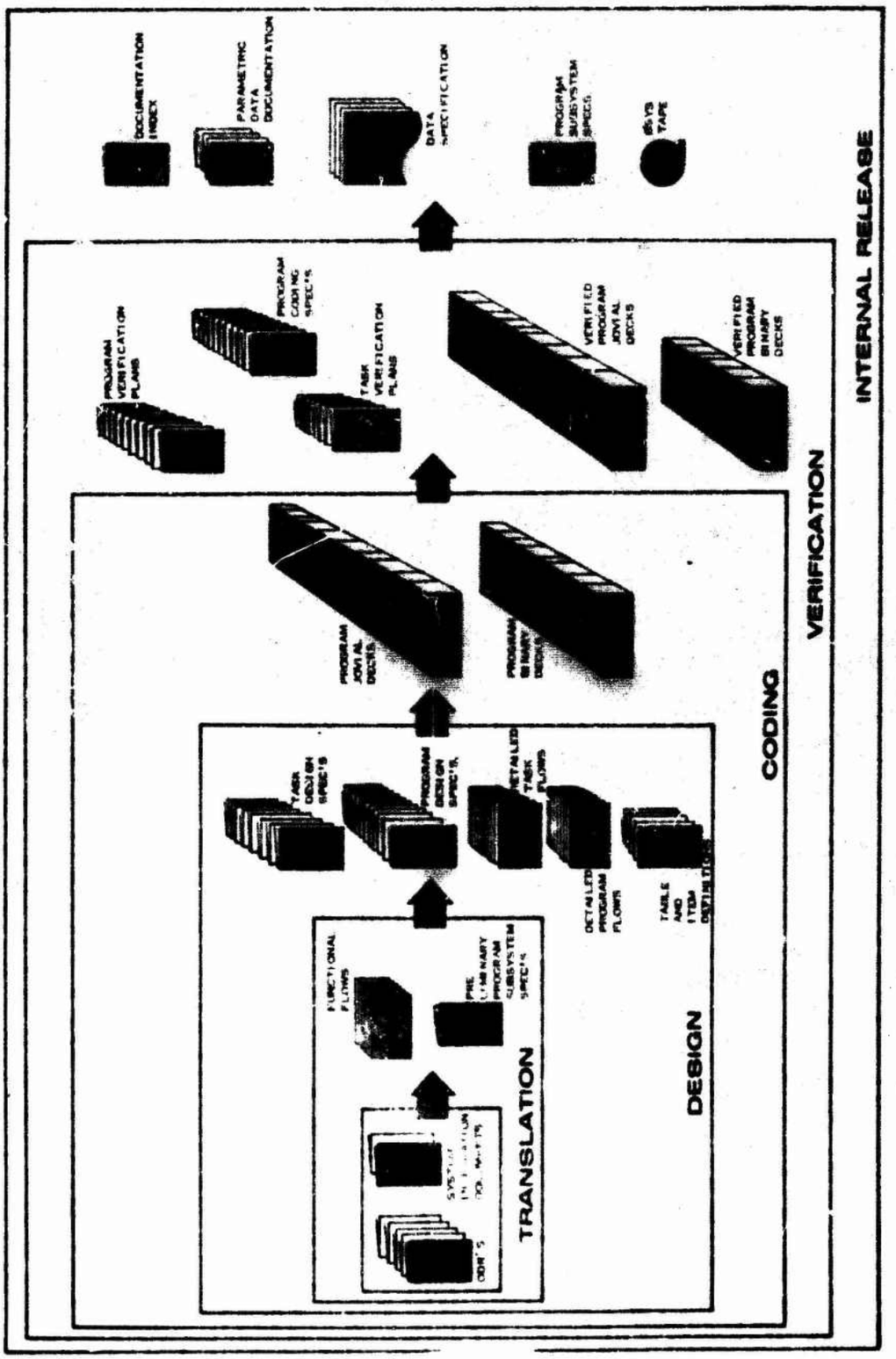

INTERNAL RELEASE
VERIFICATION
CODING
DESIGN
TRANSLATION
DOCUMENTATION INDEX
PARAMETRIC DATA DOCUMENTATION
DATA SPECIFICATION
PROGRAM SUBSYSTEM SPECS
SYS TAPE
PROGRAM VERIFICATION PLANS
PROGRAM CODING SPECS
TASK VERIFICATION PLANS
VERIFIED PROGRAM JOVIAL DECKS
VERIFIED PROGRAM BINARY DECKS
PROGRAM JOVIAL DECKS
PROGRAM BINARY DECKS
TASK DESIGN SPECS
PROGRAM DESIGN SPECS
DETAILED TASK FLOWS
DETAILED PROGRAM FLOWS
TABLE AND ITEM DEFINITION
FUNCTIONAL FLOWS
PRELIMINARY PROGRAM SUBSYSTEM SPECS

Figure 30

VI. INTERNAL RELEASE PHASE

Purpose Of The Internal Release Phase

This phase consists of the "packaging" of decks, listings, tapes, and documents that comprise the subsystem, and delivery of all these components to the group that will perform subsystem testing and installation on the user's computer(s).

This phase is of great importance. All of the previous phases may have been done extremely well, resulting in a high quality subsystem. But if little attention is given to the manner in which decks, etc. are assembled, sequenced, and turned over, this may create a poor first impression that can persist in causing the subsystem to be regarded as being of lesser quality than is really the case.

Equally important is the documentation. Documents perform several functions --

1) they give a general overview of the subsystem;
2) they present the specific methods (to operators and programmers) for actually running the system on the computer;
3) they present the specific methods by which the users will actually use the system;
4) they specify (via flow diagrams, coding specifications, etc.) how the system was produced and thus indicate how the system can be corrected, maintained, and modified for future needs. Without documents that are complete, accurate and clear, users would not know what to do with the decks, listings, and tapes.

Problems In Internal Release And Their Solution

Perhaps the main problem is that, all too often, the Internal Release Phase is not regarded in its true light . . . that is, the technical persons who produced the subsystem feel that they have done their job by producing a good program subsystem, and that it isn't too important to insure that the components are released in an orderly, organized manner.

Our solution has been to procedurize this phase, thus insuring that all necessary steps are followed without exception.

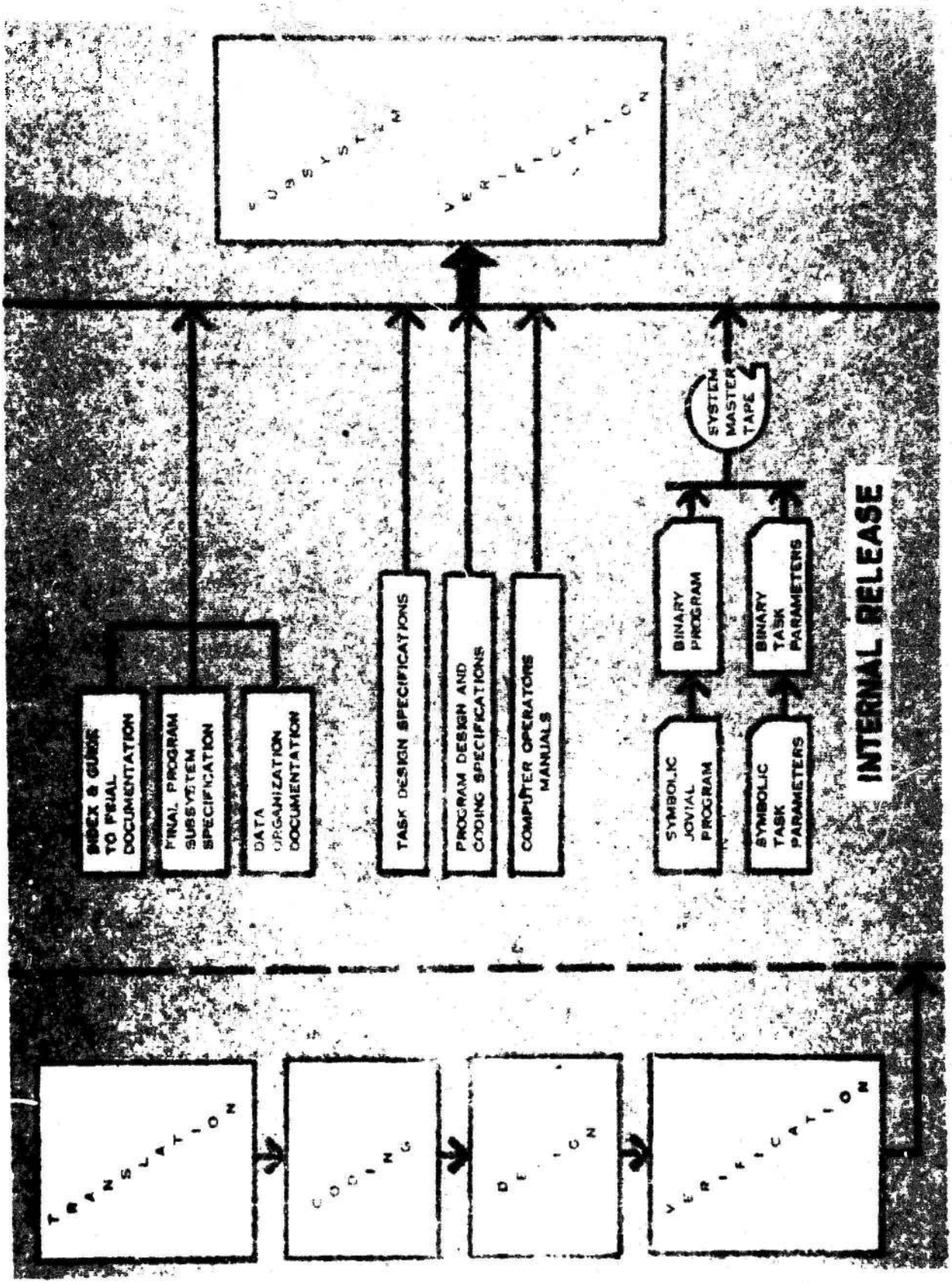


Figure 3

Methodology Of Internal Release

This phase consists of producing card decks and tapes, and writing the necessary documentation. Some of the documents will already have been written in previous phases of the production process, and would only need to be updated at the Release phase. Other documents, the contents of which depend on the results of the final phases of verification, are considered part of the release package; however they will not be completed until several weeks after the Internal Release Phase.

The Release Package consists of card decks, tapes, and documentation. All decks are accurately identified. They are assigned version numbers which will be updated each time a deck modification is made. The decks included in the Release Package are:

Symbolic JOVIAL Program Decks
Binary Program Decks
Symbolic Task Parameter Decks
Binary Task Parameter Decks

Only one tape is a part of the Release Package. It is the System Master Tape and contains the Executive programs, the necessary support programs and tasks, and all of the 465L Planning Subsystem programs and tasks.

Documentation is the last part of the Release Package. It consists of the more significant documents produced in the various phases. More precisely, it consists of:

An Over-all Index And Guide To Final Documentation
Final Program Subsystem Specification
Data Organization Documentation
Task Design Specifications
Program Design And Coding Specifications
Computer Operator Manuals

It should be noted, of course, that the group which has produced the subsystem does not divest itself of responsibility after Internal Release is completed. This group continues with on-going maintenance responsibilities for a specified period.

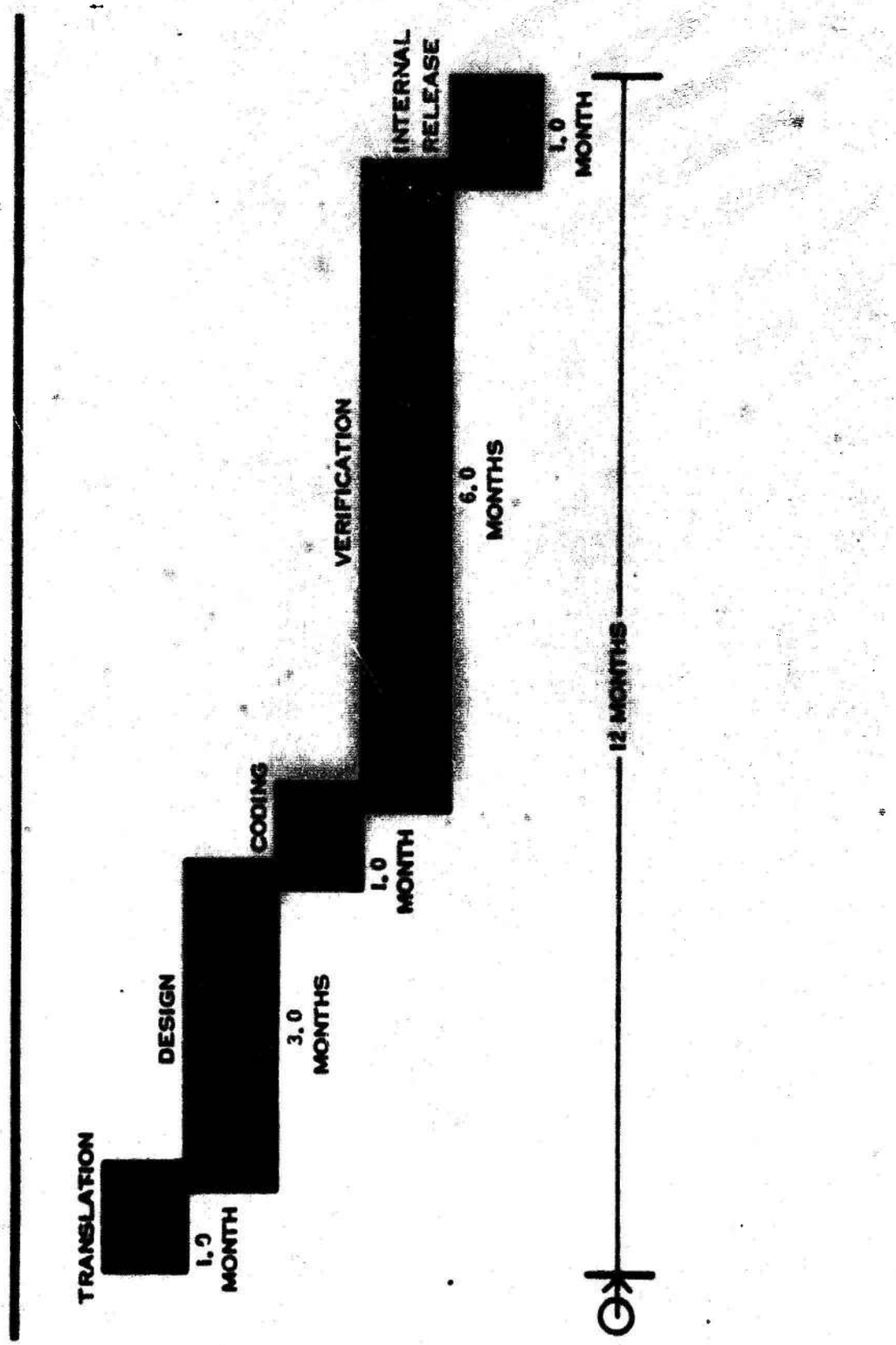

TIME PHASING
TRANSLATION
1.0 MONTH
DESIGN
3.0 MONTHS
CODING
1.0 MONTH
VERIFICATION
6.0 MONTHS
INTERNAL RELEASE
1.0 MONTH
12 MONTHS

Figure 32

(Page 70 Blank)

VII. TIME PHASING

In this paper, we have documented the actual management methods and controls used to produce the 465L Planning Subsystem. A question that will inevitably arise is - How much time should this production process take, and, given an estimate of the size of the program system to be produced, how many programmers are needed?

The authors of this paper have built up an extensive body of experience concerning these questions. We plan to document the results of this experience in a future paper, to be devoted to time phasing, scheduling, and budgeting. However, we would not want to conclude the present paper without touching on these topics.

Our experience indicates that the total production process for the initial development of a large-scale computer program system should take a minimum of twelve months. (By initial is meant the first time that the system is produced. If the same system is later updated, expanded, or otherwise revised, the time phasing would most likely be less than twelve months.) This assumes adequate manpower, computer time, and so forth. We believe that the relative weight to be placed on each phase is as follows (of course, there will necessarily be some overlapping of the phases):

Translation Phase	1 month
Design Phase	3 months
Coding Phase	1 month
Verification Phase	6 months
Internal Release Phase	1 month

Furthermore, our experience indicates that a useful working figure is to assume that production will be at the rate of 12 machine instructions per man per day (assuming average level of programming experience); or 240 machine instructions per month assuming 20 working days per month(this takes into account vacations, holidays, etc., during the year.) It should be made clear that this production rate covers the entire period from the start of Translation to the conclusion of Internal Release.

Of course, the figure of 12 machine instructions per man per day is one that will inevitably be increased as the production process becomes better defined, and as the programming state-of-the-art advances. Also, the minimum twelve month time span may be able to be reduced.

Assume that a manager is to produce a computer program system, which it is estimated will contain 72,000 instructions (this estimate must be the result of extensive data processing experience.) Using the rate of 12 machine instructions per man per day, we arrive at a needed manpower figure of 25 programmers for 12 months. Using the Time Phasing chart above, the manager can develop detailed work plans and thus keep close check on whether the schedule is being adhered to.

VIII. CONCLUSION

"Good order is the foundation of all good things."

- Edmund Burke

The raison d'etre for the existence of computer program systems is that they perform intricate calculations far more rapidly and accurately than human beings. A computer program system operates in an orderly fashion, at lightening speed.

But in order to produce a good program system, managers must themselves utilize a system of production, or what we have termed in this paper a "production process." We might say that managers need a "system for the system" which will help them to produce the best computer program system possible, at optimal time and manpower costs.

In this paper, we have delineated such a system, consisting of five primary phases: Translation, Design, Coding, Verification, and Release. Of course, we make no claims that these five phases constitute the "perfect" management system for producing a large-scale computer program system. No doubt, with the passage of time and the achievement of further experience, better "systems for the system" will be evolved. Indeed, the authors of this paper are seeking to refine and improve the system documented in this paper.

As we visualize it, the role of this paper is two-fold:

1. To emphasize that as computer program systems become larger and more complex, it is imperative that managers have a carefully conceived, workable plan for the production process.

2. To make available our experiences in managing the production of a 300,000 instruction computer program system.

It is our hope that this description of how the 465L Planning Subsystem is being produced will stimulate other managers to publish the systems they use for producing their large-scale program systems. With this cross-fertilization of ideas, techniques, and actual experiences, the state-of-the-art can be significantly advanced.

GLOSSARY

465L SYSTEM - The 465L System is the Strategic Air Command Control System. It is a large-scale, computer-based program system that is being designed and programmed by the System Development Corporation, in cooperation with other organizations, for use by the Strategic Air Command.

COMPOOL - Communications Tag Pool. This is a collection of information relating all item tags, table tags, constants, and parameters to absolute storage locations in core memory, or auxiliary storage. A compool may take the form of a magnetic tape, a deck of punched cards, or a printout.

EXECUTIVE - This is a set of support programs which was especially designed to control the operation of the Planning and Control programs which comprise the 465L System. Specifically, the Executive controls input/output operations, sequencing of programs, and so forth.

FUNCTIONAL AREA - Each major subdivision of the Planning Subsystem, as derived from the ODR's. The basic criterion employed in defining logical subdivisions is the functional interdependency of individual functions. (Refer to Translation Phase, Step 1.)

FUNCTIONAL FLOW CHART - This is a form of documentation which helps insure that the analysis (Translation Phase, Steps 1, 2 and 3) is performed correctly. It signifies completion of analysis, and is essentially a graphic presentation of the prose statement of requirements. Programmers are required to produce functional flow charts for each OSF. (Refer to Translation Phase, Step 3.)

JOVIAL - This is the higher-level programming language that has been developed at the System Development Corporation, and is being used in the 465L System.

LOGICAL TASK, LOGICAL JOB - Logical task is the same as the (actual) task defined below. It is the task in a preliminary stage of development. The primary distinction is that machine constraints have not yet been considered. Similarly, a logical job is an (actual) job. It is the job in a preliminary stage of development, for which machine constraints have not yet been considered.

ODR SUBFUNCTION (OSF) - An OSF is a further breakdown of a FUNCTIONAL AREA. The purpose of breaking down functional areas into OSF's is to distribute the work-load to programmers on an equitable basis. (Refer to Translation Phase, Step 2.)

OPERATIONAL DESIGN REQUIREMENTS (ODR's) - These documents constitute the information base for the production process. They are the primary inputs (along with the System Integration Document) to the Translation Phase. ODR's contain the following sections: general statement and description of the requirements; logical designs, including assumptions, in both prose and diagram form; specific requirements indicating the areas of human interaction with the machine; specific operational program requirements. (Refer to Introduction.)

PRELIMINARY PROGRAM SUBSYSTEM SPECIFICATION - This document incorporates the results of the Translation Phase. It identifies the logical tasks and jobs, the data sets, and the input and output requirements of the subsystem. It also contains prose and graphic descriptions of the manner in which the various logical tasks and jobs relate. Publication of this document signals the completion of the Translation Phase. (Refer to Translation Phase, Step 5.)

PROGRAM, TASK, JOB - The definition of a program is that which is standard throughout the programming profession. A task is a set of computer programs and associated data environment, designed to fulfill specific requirements stated in a part of, or one or more ODR's. Each set (i.e., each task) is discrete in that it has a unique identification, a definite beginning and end, and operates relatively independently of other sets. A job (for the Planning Subsystem) is defined as a set of tasks (this "set" may be comprised of one or more tasks) that will perform the functions called for by a "communication request" (a "communication request" is the means by which the 465L Planning Subsystem is utilized by SAC personnel. They input to the computer, via card inputs or a keyboard, requests for specific functions that are to be performed by the Planning Subsystem. These are called "communication requests", and it is the job that fulfills these requests.)

SYSTEM INTEGRATION DOCUMENT - This document is produced before the start of the production process as defined in this paper, and is thus one of the inputs to the Translation Phase. It specifies the formats of the user input messages and of output displays. (Refer to Introduction.)

TABLE, ITEM - A table is a definite allocation of core memory or auxiliary storage registers for the storage of specified information. An item consists of one or more bits in a table, set aside for the storage of specified information.

Unclassified

Security Classification

DOCUMENT CONTROL DATA - R&D

(Security classification of title, body of abstract and indexing annotation must be entered when the overall report is classified)

1. ORIGINATING ACTIVITY (Corporate author)

System Development Corporation
Santa Monica, California

2a. REPORT SECURITY CLASSIFICATION

Unclassified

2b. GROUP

3. REPORT TITLE

Management Report: Controlling Production of Complex Software

4. DESCRIPTIVE NOTES (Type of report and inclusive dates)

5. AUTHOR(S) (Last name, first name, initial)

Connelly, J. J., Osajima, Y. R.

6. REPORT DATE	7a. TOTAL NO. OF PAGES	7b. NO. OF REFS
17 May 1963	73	

8a. CONTRACT OR GRANT NO. AF 19(628)-1648
System 465L--SACCS, for Electronic
b. PROJECT NO. Systems Division, AFSC
(562.02)
c.
d.

9a. ORIGINATOR'S REPORT NUMBER(S)

TM-LO-810/101/00

9b. OTHER REPORT NO(S) (Any other numbers that may be assigned this report)

10. AVAILABILITY/LIMITATION NOTICES

Distribution of this document is unlimited

11. SUPPLEMENTARY NOTES

12. SPONSORING MILITARY ACTIVITY

13. ABSTRACT

Describes the development and production processes of the Planning Subsystem of the Strategic Air Command 465L Systems. Each of the Subsystems which compose the 465L System, alone, is in itself a large integrated program system. The"Planning Subsystem" is comprised of approximately 90 programs, 300,000 machine instructions and a data base of 6,000,000 words. This paper which was originally presented as a briefing is restricted to a definition of the basic functions of the production process. It includes 32 illustrations to clarify the text.

DD FORM 1 JAN 64 1473

Unclassified

Security Classification

14. KEY WORDS	LINK A		LINK B		LINK C	
	ROLE	WT	ROLE	WT	ROLE	WT
465L Software SACCS Planning Subsystems						

INSTRUCTIONS

1. ORIGINATING ACTIVITY: Enter the name and address of the contractor, subcontractor, grantee, Department of Defense activity or other organization *(corporate author)* issuing the report.

2a. REPORT SECURITY CLASSIFICATION: Enter the overall security classification of the report. Indicate whether "Restricted Data" is included. Marking is to be in accordance with appropriate security regulations.

2b. GROUP: Automatic downgrading is specified in DoD Directive 5200.10 and Armed Forces Industrial Manual. Enter the group number. Also, when applicable, show that optional markings have been used for Group 3 and Group 4 as authorized.

3. REPORT TITLE: Enter the complete report title in all capital letters. Titles in all cases should be unclassified. If a meaningful title cannot be selected without classification, show title classification in all capitals in parenthesis immediately following the title.

4. DESCRIPTIVE NOTES: If appropriate, enter the type of report, e.g., interim, progress, summary, annual, or final. Give the inclusive dates when a specific reporting period is covered.

5. AUTHOR(S): Enter the name(s) of author(s) as shown on or in the report. Enter last name, first name, middle initial. If military, show rank and branch of service. The name of the principal author is an absolute minimum requirement.

6. REPORT DATE: Enter the date of the report as day, month, year; or month, year. If more than one date appears on the report, use date of publication.

7a. TOTAL NUMBER OF PAGES: The total page count should follow normal pagination procedures, i.e., enter the number of pages containing information.

7b. NUMBER OF REFERENCES: Enter the total number of references cited in the report.

8a. CONTRACT OR GRANT NUMBER: If appropriate, enter the applicable number of the contract or grant under which the report was written.

8b, 8c, & 8d. PROJECT NUMBER: Enter the appropriate military department identification, such as project number, subproject number, system numbers, task number, etc.

9a. ORIGINATOR'S REPORT NUMBER(S): Enter the official report number by which the document will be identified and controlled by the originating activity. This number must be unique to this report.

9b. OTHER REPORT NUMBER(S): If the report has been assigned any other report numbers *(either by the originator or by the sponsor)*, also enter this number(s).

10. AVAILABILITY/LIMITATION NOTICES: Enter any limitations on further dissemination of the report, other than those imposed by security classification, using standard statements such as:

(1) "Qualified requesters may obtain copies of this report from DDC."

(2) "Foreign announcement and dissemination of this report by DDC is not authorized."

(3) "U. S. Government agencies may obtain copies of this report directly from DDC. Other qualified DDC users shall request through _____."

(4) "U. S. military agencies may obtain copies of this report directly from DDC. Other qualified users shall request through _____."

(5) "All distribution of this report is controlled. Qualified DDC users shall request through _____."

If the report has been furnished to the Office of Technical Services, Department of Commerce, for sale to the public, indicate this fact and enter the price, if known.

11. SUPPLEMENTARY NOTES: Use for additional explanatory notes.

12. SPONSORING MILITARY ACTIVITY: Enter the name of the departmental project office or laboratory sponsoring *(paying for)* the research and development. Include address.

13 ABSTRACT: Enter an abstract giving a brief and factual summary of the document indicative of the report, even though it may also appear elsewhere in the body of the technical report. If additional space is required, a continuation sheet shall be attached.

It is highly desirable that the abstract of classified reports be unclassified. Each paragraph of the abstract shall end with an indication of the military security classification of the information in the paragraph, represented as *(TS), (S), (C), or (U)*

There is no limitation on the length of the abstract. However, the suggested length is from 150 to 225 words.

14. KEY WORDS: Key words are technically meaningful terms or short phrases that characterize a report and may be used as index entries for cataloging the report. Key words must be selected so that no security classification is required. Identifiers, such as equipment model designation, trade name, military project code name, geographic location, may be used as key words but will be followed by an indication of technical context. The assignment of links, rules, and weights is optional.