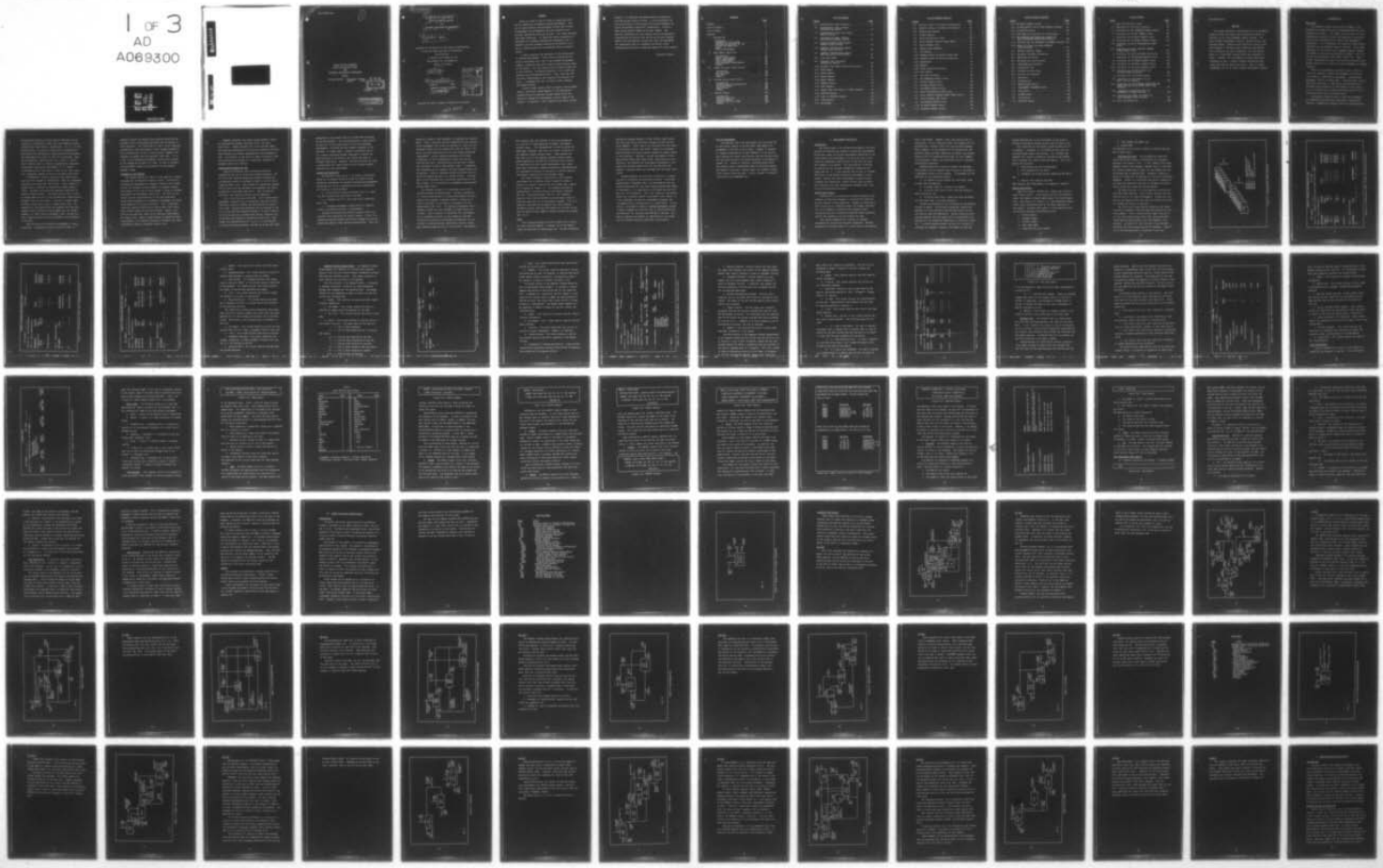
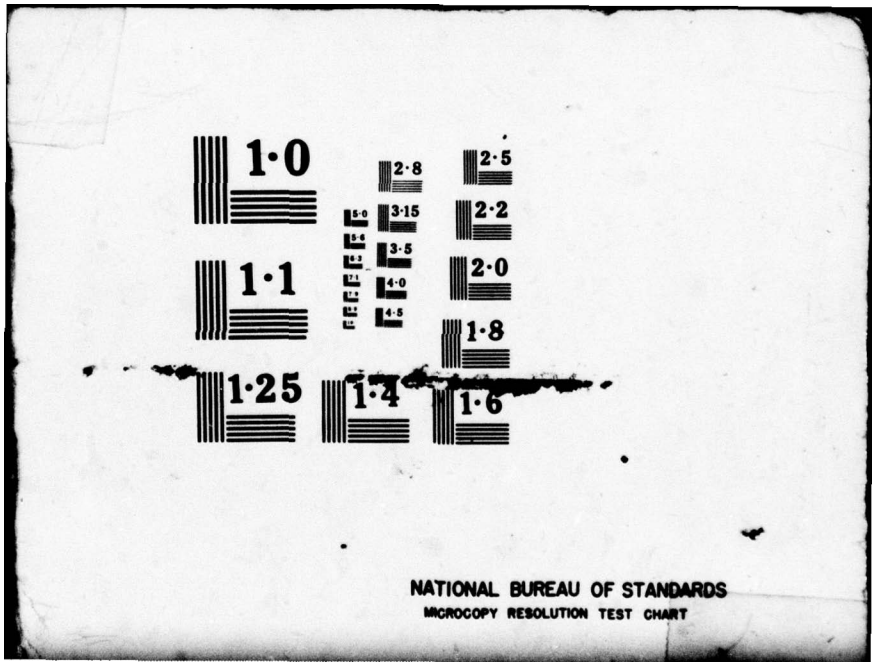
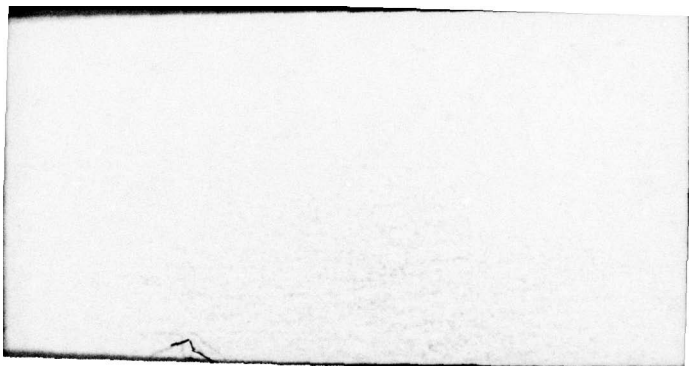
AD-A069 300 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN FOR AN AUTOMATED STATUS ACCOUNTING SYSTEM FOR SOFTWARE C--ETC(U)
MAR 79 A SCHUSTER

UNCLASSIFIED AFIT/GCS/EE/79-2 NL

1 OF 3
AD
A069300




1·0
1·1
1·25
1·4
1·6
1·8
2·0
2·2
2·5
2·8
3·15
3·5
4·0
4·5
5·0
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART



AFIT/GCS/EE/79-2

(1)

DESIGN FOR AN AUTOMATED STATUS ACCOUNTING SYSTEM FOR SOFTWARE CONFIGURATION MANAGEMENT

THESIS

AFIT/GCS/EE/79-2 Alexander Schuster
Captain USAF

DDC
RECEIVED
JUN 4 1979
A

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

79 05 30 231

(14) AFIT/GCS/EE/79-2

(6) DESIGN FOR AN AUTOMATED STATUS ACCOUNTING SYSTEM FOR SOFTWARE CONFIGURATION MANAGEMENT.

(9) Master's THESIS

Presented to the Faculty of the School of Engineering
of the Air Force Institute of Technology
Air University
in Partial Fulfillment of the
Requirements for the Degree of
Master of Science

(12) 243p.

by

(10) Alexander Schuster B.S.
Captain USAF
Graduate Computer Systems

(11) March 1979

Accession For
NTIS GRA&I
DDC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or special
A

Approved for public release; distribution unlimited

012 225 LB

Preface

There is a need in the Air Force to reduce the costs and the complexity of computer program development. This can be done by making the development efforts more visible to management and by demanding that more effective and reliable engineering practices be used. This study supported both these objectives. It identified status accounting requirements for software configuration management, and it adhered to current software engineering disciplines to obtain a software design for an automated status accounting system.

I cannot overstate the importance of the design phase of software development. Since errors in design are not usually discovered until late in the software development project, correcting them often requires reworking many parts of the system. This contributes both to project delays and to cost-overruns. The design presented in this report was obtained using methodologies which, I feel, make the software more understandable, thus reducing the potential for error. I recommend that they be used in other software development efforts.

I wish to thank Captain Frank E. Douglas, Mister Robert Kuhnert, and Mister Joseph Romanick of the Aeronautical Systems Division's Simulator System Program Office for helping me through the requirements analysis phase of my project. In addition, I feel indebted to my thesis advisor,

Captain J. B. Peterson, who enthusiastically guided and assisted me from start to finish. I am also grateful for the extra efforts of Debi Walters and Paulette Beaupre, my typist and graphics specialist, respectively. Finally, I want to give special thanks to my wife, Audrey. She deserves recognition not only because she was exceptionally understanding and patient with me during the past eighteen months, but also because, during that time, she shouldered the responsibilities of a housewife and mother, while simultaneously pursuing an advanced degree full-time, herself.

Alexander Schuster

Contents

		Page
Preface		ii
List of Figures		vi
List of Tables		x
Abstract		xi
I	Introduction	1
	Background	1
	Statement of the Problem	3
	Standardized Software for CSA	4
	Purpose and Objectives	5
	Scope	7
	Plan of Development	9
II	Requirements Definition	10
	Introduction	10
	General Requirements	10
	Report Requirements	12
	Input Requirements	29
	Data Management Requirements	41
	Summary	46
III	Formal Functional Specification	47
	Introduction	47
	Activity Model	49
	Data Model	69
	Summary	83
IV	Software Design Specification	84
	Introduction	84
	Software Design Considerations	84
	Structure Charts	87
	Observations	160
	Summary	163
V	Database Design	165
	Introduction	165
	Database Modelling	166
	Canonical Models	167
	SOFTSAS Canonical Schema	169
	Summary	172

List of Figures

Figure		Page
2-1	Configuration Index Structure	14
2-2	Configuration Index, Product Specification Section	15
2-3	Configuration Index, Test Plans/ Reports Section	16
2-4	Configuration Index, Version Description Documents Section	17
2-5	Computer Program Change Report, Change History Section	20
2-6	Computer Program Change Report, Change Status Section	22
2-7	Computer Program Change Report, Red Flag and Activity Section	25
2-8	Tree Code Format	26
2-9	Software Tree, Functional Description Section	28
2-10	Software Tree, Module Description Section . . .	30
2-11	OPEN Request	32
2-12	CREATE Request	34
2-13	DELETE Request	35
2-14	MODIFY Request	36
2-15	PRODUCE Request	36
2-16	PRINT Request	37
2-17	Sample Input and Output for PRINT Requests . .	38
2-18	DESCRIBE Request	39
2-19	Types of DESCRIBE Requests	40
2-20	CLOSE Request	41
2-21	END Request	41

List of Figures (Cont'd)

Figure		Page
3-1	Maintain Status of Software Configuration . . .	50
3-2	Maintain Status of Software Configuration . . .	52
3-3	Analyze User Request	55
3-4	Produce Reports	57
3-5	Build Configuration Index	59
3-6	Build Computer Program Change Report	61
3-7	Build Software Tree	63
3-8	Compose Query Response	65
3-9	Put Outputs	67
3-10	Software Status Accounting System Data	70
3-11	Software Status Accounting System Data	72
3-12	System Data	75
3-13	Requests	77
3-14	Status Accounting Database	79
3-15	User Outputs	81
4-1	Top Level Structure	88
4-2	GET CONVERTED REQUEST Branch	93
4-3	SATISFY REQUEST Branch	99
4-4	DO PRINT REQUEST Branch	105
4-5	DO PRODUCE REQUEST Branch	109
4-6	BUILD CONFIGURATION INDEX Branch	113
4-7	BUILD COMPUTER PROGRAM CHANGE REPORT Branch . .	116
4-8	BUILD SOFTWARE TREE Branch	121
4-9	DO DESCRIBE REQUEST Branch	126
4-10	DO CREATE REQUEST Branch	131
4-11	DO MODIFY REQUEST Branch	134

List of Figures (Cont'd)

Figure		Page
4-12	DO DELETE REQUEST Branch	138
4-13	DO OPEN REQUEST and DO CLOSE REQUEST Branches	141
4-14	PUT OUTPUTS Branch	144
4-15	READ FROM FILE and WRITE TO FILE Branches	147
4-16	GET APPROVED IDENTIFIERS and SEARCH FOR QUALIFIED RECORD IDENTIFIERS Branches	150
4-17	RETRIEVE DATA and REFERENCE DICTIONARY Branches	154
4-18	CHECK I/O STATUS and CHECK DATABASE STATUS Branches	157
5-1	SOFTSAS Canonical Schema	170
A-1	Box/Arrow Interface Conventions	186
A-2	Mechanism Call Arrow	187
A-3	OR Branch and Join Structure	188
A-4	ICOM Numbering Convention	189
B-1	Information Flow	192
B-2	Information Arrow Types	193
B-3	Decision and Iteration	194
C-1	ORG Record	196
C-2	SYSTEM Record	197
C-3	MANUFACTURERS Record	198
C-4	PROCUREMENT INSTRUMENT Record	199
C-5	CPCI Record	200
C-6	OP-MODE Record	201
C-7	CLASS Record	202
C-8	SUB-CLASS Record	203

List of Tables

Table		Page
I	Legal SOFTSAS Input Tokens	33
II	Interfaces for Top Level Structure	89
III	Interfaces for GET CONVERTED REQUEST Branch	94
IV	Interfaces for SATISFY REQUEST Branch	100
V	Interfaces for DO PRINT REQUEST Branch	106
VI	Interfaces for DO PRODUCE REQUEST Branch	110
VII	Interfaces for BUILD CONFIGURATION INDEX Branch	114
VIII	Interfaces for BUILD COMPUTER PROGRAM CHANGE REPORT Branch	117
IX	Interfaces for BUILD SOFTWARE TREE Branch	122
X	Interfaces for DO DESCRIBE REQUEST Branch	127
XI	Interfaces for DO CREATE REQUEST Branch	132
XII	Interfaces for DO MODIFY REQUEST Branch	135
XIII	Interfaces for DO DELETE REQUEST Branch	139
XIV	Interfaces for DO OPEN REQUEST and DO CLOSE REQUEST Branches	142
XV	Interfaces for PUT OUTPUTS Branch	145
XVI	Interfaces for READ FROM FILE and WRITE TO FILE Branches	148
XVII	Interfaces for GET APPROVED IDENTIFIERS and SEARCH FOR QUALIFIED RECORD IDENTIFIERS Branches	151
XVIII	Interfaces for RETRIEVE DATA and REFERENCE DICTIONARY Branches	155
XIX	Interfaces for CHECK I/O STATUS and CHECK DATABASE STATUS Branches	158
C-I	Data Item Definitions	223

Abstract

This report documents a design effort for an automated system to record and report the configuration status of software for Air Force embedded computer systems. The work included requirements analysis, software design, and database design. Because of the flexibility given to program managers in tailoring their reporting requirements and in selecting the data "integator", only the requirements of a single AFSC program office were presented in detail. However, the requirements of other offices were considered as well. Current software engineering techniques were used to derive the design. They are highly recommended for use on other software development projects.

I Introduction

Background

The Simulator System Program Office (SIMSPO) of the Aeronautical Systems Division at Wright-Patterson AFB, Ohio, sponsored this thesis. The SIMSPO is one of many Air Force organizations tasked with acquiring resources in accordance with the Air Force 800 series, "Acquisition Management", regulations. It is responsible for acquiring flight trainer simulators for 12 different types of aircraft. Since these simulators will be controlled by embedded computer systems, the SIMSPO's activities are governed by Volumes I and II of the Air Force Regulation (AFR) 800-14 (Refs 3,4). These volumes consolidate and explain policies and procedures that pertain specifically to the acquisition management of computer resources.

Acquisition management encompasses several highly related, yet separately identified, management disciplines, one of which is configuration management. Configuration management consists of three tasks: identification, control, and status accounting of items being acquired (Refs 2,3). Although status accounting is the subject of this thesis, it is so closely tied to identification and control that all three tasks are briefly introduced below.

Configuration identification is the set of documents and engineering drawings which describes a system being acquired. Hardware and software which will be delivered to

the procuring activity to serve end-use functions are designated as configuration items (CIs) and computer program configuration items (CPCIs), respectively. As a CI or CPCI design evolves, its technical description (documentation) constitutes its configuration identification and represents the requirements for the CI or CPCI to be developed. When the configuration identification for an item is "sufficiently" defined (that is, when the documentation represents a meaningful contractual requirement), it is "frozen". The documentation at that point in time is the item's baseline, subsequent changes to which are formally controlled.

"Configuration control consists of the formal procedures by which changes to configuration items are documented, processed, and authorized" (Ref 34:53). These changes are accomplished in response to engineering change proposals (ECPs): documents which describe proposed alterations to baselined specifications. Associated with an ECP is a set of change pages for each of the configuration identification documents that the ECP affects. Each set of change pages includes a cover sheet called a change notice (CN). CNs enumerate all the pages within the document that have been changed since a revision to the document was last issued. Together, an item's baseline documents, ECPs, and CNs provide a complete history of the development and maintenance of the item.

The third task of configuration management is status accounting. Configuration status accounting (CSA) is a

system of reports and records that provides visibility and traceability of configuration baselines and the changes to those baselines. CSA "identifies an item's initial approved configuration, then continuously tracks changes proposed to that configuration, as well as the priority, schedule, and progress of changes that are approved" (Ref 5:30). It is a tool to aid configuration managers perform tasks related to the changes being made to an item. As such, a program's CSA requirements should be tailored to fit the program manager's needs.

Statement of the Problem

The impetus behind this thesis is the need for a system to manage the configuration status of CPCIs. Historically, configuration management for computer resources has focused only on the hardware-related elements of computer systems, even though 4-5% of the Air Force budget is spent on software development and maintenance (Ref 28:12). But CPCIs do not exhibit certain characteristics of CIs upon which many established procedures of configuration management are based. Three examples of this are as follows. First, system malfunctions caused by CPCIs are not due to wear; thus corrections require change to the item's design. Second, since CPCIs do not wear out, there are no logistics requirements for spare parts. Lastly, responsiveness to changing system requirements is an inherent advantage of software; yet, many configuration control procedures inhibit this.

Computer programs, like other system elements, should be configuration managed throughout their life-cycle (Ref 4:6,1). Indeed, Appendix VIII of MIL-STD-483 (USAF) has contained maintenance and accounting procedures for CPCI specifications and support documentation since December of 1970. Particularly in the realm of status accounting, software should not be treated as a "tag-along" component of hardware (Refs 6,12,17,32,41).

Standardized Software for CSA

Should an automated status accounting system be standardized for use by acquisition program managers? The Air Force uses standard mechanized systems to manage the configuration status of equipment already in the Air Force inventory (Refs 1,37). Such systems are appropriate because the Air Force owns the equipment being managed, and status accounting is performed organically. Thus, collecting, analyzing, and reporting the data can be regulated.

This is not the case within AFSC. Until configuration management responsibility is transferred from AFSC to AFLC, contractors normally perform the CSA functions on acquisition items. Although the contractor can be directed to use an Air Force provided computer program or to use an organic Air Force capability to produce the reports, these may not be the most cost-effective alternatives (Ref 5:33). Even though non-standardization will result in a proliferation of status accounting software, savings can be realized where

manufacturers use software that has already been developed for earlier projects. But the manufacturer's system should be used only if the output data will satisfy Government requirements, including the data element standards of MIL-STD-482A. At other times, a standard, modifiable design for a status accounting system, developed in advance and made available to contractors, may reduce the costs of software acquisition. Moreover, the specifications for such a system would be a useful guide to help program managers identify their CSA requirements.

Purpose and Objectives

The purpose of this thesis is to evolve a preliminary design for a software system to record and report status accounting information for software configuration management. The design will be for a database processing system which will consist of three elements:

a. An applications program (SOFTSAS) that will supervise the recording and reporting of the data;

b. A database that will store the status accounting data; and,

c. A database management system that will organize, control, and protect the data in the database.

Database processing was selected instead of more conventional file processing for several reasons. First, in a database processing system, application programs are not concerned with physical data storage considerations. Thus, the

system as a whole is more adaptable to changing user requirements. This is important because responsibility for configuration management (and status accounting) of a CPCI will be assigned to more than one organization over the CPCI's lifecycle, and each organization has its own status accounting requirements. Second, database processing facilitates integrating all of an organization's data into one centralized data bank. Centralization can make more information available to each of the system's users. In addition, data redundancy can be reduced, and data standardization can be achieved. Finally, database processing permits users to access and employ the data in a variety of ways. The database is not structured for a particular application of the data; rather, it is structured to accommodate unanticipated queries (Ref 16:3-5).

The software system will be designed using current software engineering methodologies. Software engineering is a disciplined approach to software development and maintenance that attempts to manage software's complexity in order to increase its reliability and enhance its cost-effectiveness (Ref 7,42). The approach presupposes that a software project will pass through six stages during its lifetime. Once a problem is identified, the requirements for an acceptable solution to the problem are defined, alternative solutions are developed and compared, and one of them is selected (system requirements analysis stage). When software comprises part of the solution, requirements

that identify what the software is to do are documented in a precise and unambiguous way (software requirements analysis stage). This documentation is used to design the algorithms, database, test data, and overall structure of the computer program (design stage). These designs are then used to write the computer programs (coding stage). Coded programs are verified for correctness and validated for compliance with the stated requirements (test stage). Those that successfully complete testing are implemented and are subsequently modified as necessary to satisfy changing requirements (operations and maintenance stage).

This thesis provides a description of the software requirements for a software status accounting system, a design of an overall structure for the system, and a design for a database to support it. The software engineering methodologies "Structured Analysis and Design Technique" (Ref 35), "Structured Design" (Ref 11), and "Database Canonical Form" (Ref 20:248-290) are used because they enhance the modifiability of the computer program. This is an important consideration throughout this thesis because AFR 800-14 directs that CSA requirements be tailored to the specific needs of each computer resource acquisition program (Ref 4:6,3).

Scope

Since program managers are given flexibility in tailoring their CSA requirements, a standard set of CSA reports cannot be specified for wide-spread use. The CSA information

required by program managers at AFSC differs significantly from that required by functional managers at AFLC and the using commands. (AFSC reports are concerned with the status of contractual actions; the others' reports are concerned with the status of actions against systems and items already in the inventory.) Moreover, the CSA requirements of program offices within AFSC often differ significantly as well. Therefore, the software CSA requirements documented in this report are tailored specifically for the Simulator SPO. However, a database model was developed that has wider application.

Program managers are also given flexibility in selecting who is to perform the accounting function. Since AFSC pamphlet 800-7 recommends that it be performed by the contractor developing the item (even though program funds might be increased), it is impossible to anticipate how the software will be implemented. Thus, only a general, preliminary design for a software system is provided in this thesis; it is as independent of machine, programming language, and telecommunication considerations as is practicable. In particular, no attempt to design a database management system is made, although the functions that it should perform are described and its interface with SOFTSAS is defined. Only when an external environment is identified for the system can the software design be specified in more detail.

Plan of Development

The Simulator SPO's CSA requirements and the design for an application program to satisfy these requirements are documented in the rest of this report. Chapter II presents an informal specification of what SOFTSAS is to do, while a more rigorous and formal specification that was derived using the Structured Analysis and Design Technique is presented in Chapter III. Chapter IV contains a software design which was achieved using a methodology known as Structured Design, and Chapter V describes a logical model for software configuration status accounting data. Finally, Chapter VI presents conclusions and recommendations.

II Requirements Definition

Introduction

The second stage in the software development life-cycle consists of defining the software's processing requirements. During this stage, the system requirements identified in the first stage of the development life-cycle are used to formulate an unambiguous specification of the functions that the software is to perform. The specification is derived by focusing on the interface between the system and the people who use it. In the sections that follow, an informal definition of the SOFTSAS processing requirements is presented. Sources for these requirements include military standards and regulations, reports on work done by others, and interviews with AFSC configuration managers (Ref 3,4,5, 6,12,17,23,24,25,26,32,33,34,41).

General Requirements

SOFTSAS will be an application program designed to aid software configuration managers in tracking CPCI baselines and the changes to those baselines. Capable of executing in both a batch and an interactive mode, its primary functions will be to maintain a repository for software status accounting data and to produce pre-defined status accouting reports and responses to ad hoc queries for data.

User inputs (requests) will consist of commands accompanied by required and optional parameters. SOFTSAS, executing in the batch mode, will usually obtain the requests

from a card reader. However, other input devices may be substituted. In the interactive mode, the requests will be obtained from a terminal after the user has been prompted with a message. No new request will be fetched until the previous request has been completely processed, and program execution will be terminated when either an 'End' command is encountered or when a non-recoverable error is detected during batch processing.

SOFTSAS output will consist of reports and messages. Three reports can be produced, each of which is outlined in a subsequent section of this chapter. The messages will be generated for the following reasons:

a. To respond to user requests for individual items of data from the database;

b. To prompt users at a terminal for inputs;

c. To notify users that an error occurred during request processing; and,

d. To notify users when a request has been validated and then again when it has been processed.

Reports and messages will be considered as two separate classes of output. During batch processing, reports and messages will be accumulated and then delivered to an on-line printer upon job termination. During interactive processing however, only the reports will be accumulated; the messages will be dispatched immediately to the user's terminal. In either processing mode, the user will be able to configure the program's external environment so that the

outputs destined for an on-line printer can be directed instead to peripheral storage devices such as tape or disk.

The repository for the status accounting data will be a database. A database is a collection of data that can be shared among several applications. As a minimum, the database will contain information required by the SOFTSAS user to produce the status accounting reports. This information will describe:

1. CPCIs being acquired or maintained;
2. ECPs prepared for the CPCIs;
3. Documents and change notices supporting the CPCIs; and,
4. Modules that comprise the CPCIs.

More specific data requirements are itemized in Chapter V.

Report Requirements

Upon user request, SOFTSAS will produce one or more of three formal reports. These reports are the Configuration Index, the Computer Program Change Report, and the Software Tree. Each report will contain, in a pre-determined format, timely status accounting data for the CPCIs specified in the request. Preceding each report, a title page will bear the following document identification data:

a. Issuing agency;
b. Document number;
c. Contract number;
d. CDRL item number
e. Issue date and issue number;

f. CPCI numbers and names; and,

g. System name.

The remainder of this section contains a detailed description of each report.

Configuration Index. The Configuration Index will report the current status of maintainable and traceable documents that support a CPCI. It will identify the latest complete issue of the documents (whether basic or revision), the interim changes made to them, and the changes that are anticipated due to approved ECPs. Documents issued as a series of volumes will be reported as a series of individual documents. Figure 2-1 illustrates the overall structure of the Index and identifies the types of documents for which status accounting records must be maintained.

The Index consists of a title page and a division for each CPCI being reported. Each division contains one section for each of six types of documents. Figures 2-2, 2-3, and 2-4 are examples which show that the sections have similar formats and consist of two parts.

The first part of each section identifies the latest revisions of documents that have been delivered to the procuring agency. Also, in every section except section VI, it lists the change notices (with their associated ECPs) issued against each document since the document's last revision. In section VI, the first part lists the ECPs delivered with the Version Description Documents. Specifically, the following data is contained in part one:

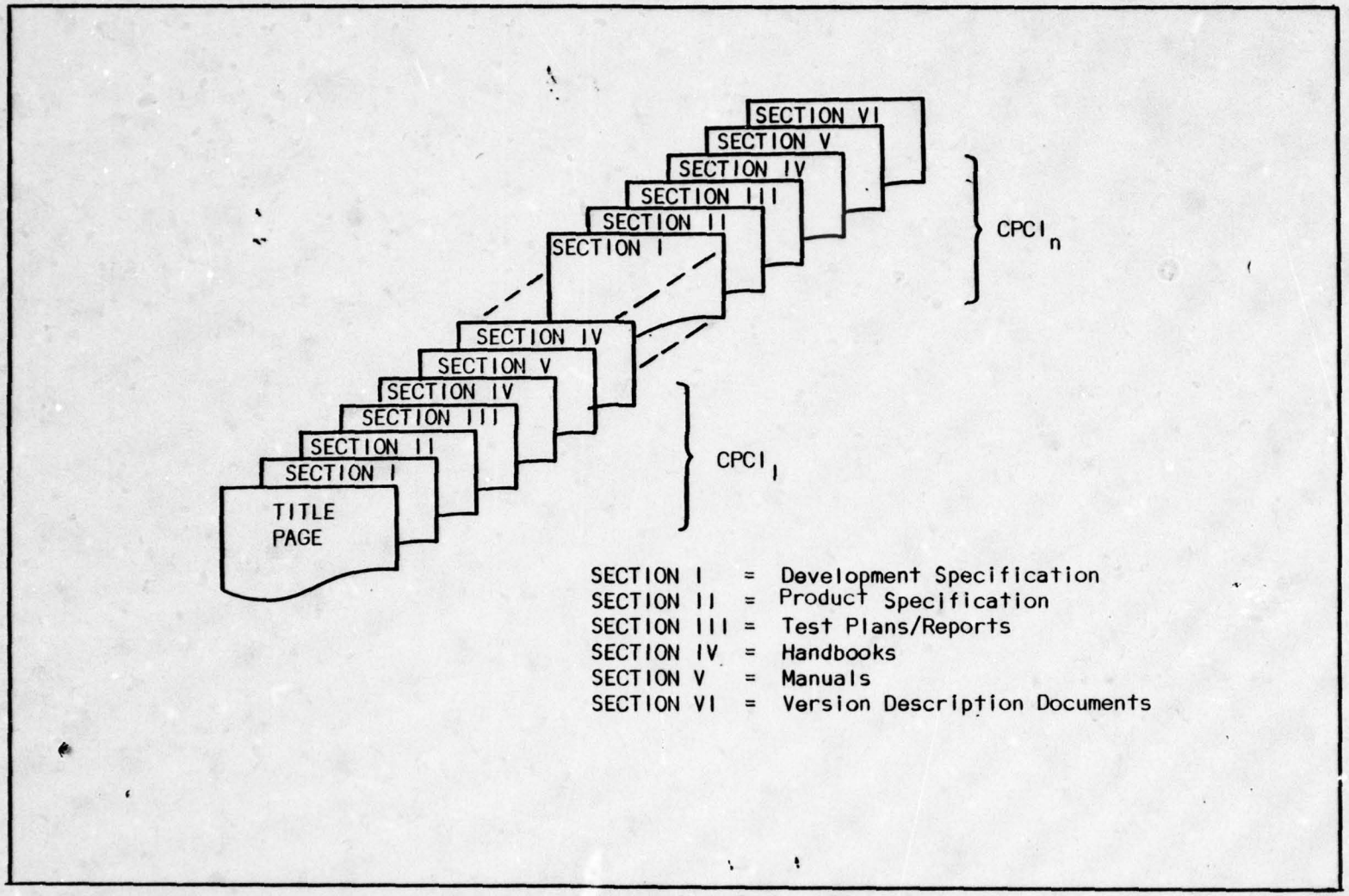


Figure 2-1 Configuration Index Structure

COMPUTER PROGRAM CONFIGURATION INDEX

CPCI Number _____ CPCI Name _____

Section II - Computer Program Product Specification, Number _____

Part 1. Basic Documentation

Doc/Change	Rev/ECP	Title	Date Dist.
Volume 1	BASIC	----------	XX-XX-XX
SCN 1-1	ECP 1	----------	XX-XX-XX
SCN 2-1	ECP 2	----------	XX-XX-XX
:	:	:	:
SCN 4-1	ECP 7	----------	XX-XX-XX
Volume 2	A	----------	XX-XX-XX
SCN 1-2	ECP 1	----------	XX-XX-XX
SCN 2-2	ECP 2	----------	XX-XX-XX
:	:	:	:
Appendix I	BASIC	----------	XX-XX-XX
:	:	:	:

Part 2. Approved Changes

ECP Number	Title	Doc Affected	Approved
90	----------	1, 5, 7, I, IV	XX-XX-XX
:	:	:	:

Figure 2.2 Configuration Index, Product Specification Section

COMPUTER PROGRAM CONFIGURATION INDEX

CPCI Number _____ CPCI Name _____

Section III - Test Plans/Reports

Part 1. Basic Documentation

DOC/Change	Rev/ECP	Title	Date Dist.
TR-XXXX-XXX-XX	BASIC	----------	XX-XX-XX
CN 01	ECP XX	----------	XX-XX-XX
CN 02	ECP XX	----------	XX-XX-XX
⋮	⋮	⋮	⋮
TR-XXXX-XXX-XX	A	----------	XX-XX-XX
⋮	⋮	⋮	⋮

Part 2. Approved Changes

ECP Number	Title	Doc Affected	Approved
XX	----------	TR-XXXX-XXX-XX	XX-XX-XX
⋮	⋮	⋮	⋮
XX	----------	TR-XXXX-XXX-XX	XX-XX-XX

Figure 2-3 Configuration Index, Test Plans/Reports Section

COMPUTER PROGRAM CONFIGURATION INDEX

CPCI Number _____ CPCI Name _____

Section VI - Version Description Documents

Part 1. Basic Documentation

Doc Number	Rev/ECP	Title	Date Dist.
AA-XXXX-01	VDD-1	Version Desc Doc	XX-XX-XX
AA-XXXX-02	VDD-2	Version Desc Doc	XX-XX-XX
	ECP 43	----------	XX-XX-XX
	⋮	⋮	⋮
AA-XXXX-03	VDD-3	Version Desc Doc	XX-XX-XX
⋮	⋮	⋮	⋮

Part 2. Approved Changes

ECP Number	Title	Doc Affected	Approved
97-5	----------	VDD-5	XX-XX-XX
⋮	⋮	⋮	⋮

Figure 2-4 Configuration Index, Version Description Documents Section

a. Header - This consists of section and CPCI identification data.

b. Document/Change - This column contains an entry to identify each document or change notice by number.

c. Revision/ECP - For document entries, this column contains the work "BASIC" or the current revision identifier of the document. For change notices, the column contains the number of the ECP that generated the change.

d. Title - This column contains the title of either the document or the ECP, as appropriate.

e. Date Distributed - This column contains the date that a document revision or a change notice was delivered to the procuring activity.

The second part of each section identifies those documents which will require changes due to ECPs that have been approved but for which change notices have not as yet been issued. Specifically, the following data is contained in part two:

a. ECP Number - This column contains an entry for each approved ECP that will impact a document listed in part one.

b. Title - This column contains the title of the ECP.

c. Documents Affected - This column identifies the volumes, appendices, or other documents listed in part one that will be affected by the ECP.

d. Approval Date - This column contains the date that the ECP was approved.

Computer Program Change Report. The Computer Program Change Report will identify all the ECPs ever prepared against a CPCI and will provide status information on those ECPs that are currently active. This report consists of three sections for each CPCI to be reported.

The first section is the Change History. It contains an ordered, historical listing of all the engineering changes ever proposed for the specified CPCI. The Change History will be formatted as shown in Figure 2-5 and will contain the following data:

a. Header - This consists of section and CPCI identification information.

b. ECP Number - This column contains an entry to identify by number every ECP prepared for the CPCI.

c. ECP Title - This column contains the titles of the ECPs.

d. Status - This column contains a code for the current status of an ECP. The codes that will be used are:

i. P - ECP is being prepared;

ii. S - ECP has been submitted and is awaiting CCB action;

iii. A - ECP has been approved by the CCB;

iv. D - ECP has been disapproved by the CCB;

v. X - ECP has been deferred by the CCB;

vi. C - ECP is being tested by the contractor;

vii. T - ECP is being organically tested;

viii. I - ECP has been implemented.

COMPUTER PROGRAM CHANGE REPORT

CPCI Number _____ CPCI Name _____

Section I - Change History

ECP Number	ECP Title	Status	Date	Remarks
XX	---------	------	XX-XX-XX	-------
⋮	⋮	⋮	⋮	⋮

Figure 2-5 Computer Program Change Report, Change History Section

e. Date - This column contains the date that the ECP entered its current status.

f. Remarks - This column contains additional information which can be used, for example, to identify ECPs which either impact another contractor's configuration item or are caused as a result of another contractor's ECP.

The second section of the Computer Program Change Report is the Change Status summary. It contains a concise summary description of each active ECP for the specified CPCI. A summary for a particular ECP will appear in every issue of this section after a number has been assigned to the ECP and until one issue after either the ECP is disapproved or is implemented. The Change Status Summary will be formatted as shown in Figure 2-6 and will contain the following data:

a. Header - This consists of section and CPCI identification information.

b. ECP Number, Title - These entries identify the ECP being reported.

c. Priority - This entry identifies the priority of the ECP as either "Emergency", "Urgent", or "Routine".

d. Current Status - This entry contains a code for the current status of the ECP as reported in the Change History.

e. Statements of Problem and Solution - These entries contain brief narratives extracted from the ECP to describe the problem and its proposed solution.

COMPUTER PROGRAM CHANGE REPORT

CPCI Number _____ CPCI Name _____

Section II - Change Status

1. Identification:

ECP Number _____ Title _____ Priority _____ Current Status _____

2. Statement of Problem:

3. Statement of Solution:

4. Modules Affected: 5. Documents Affected: 6. Related Configuration Changes:

Tree Code	Name	DOC Number	Type	CN	Date Due	Date Issued	CI/CPCI Number	ECP Number

7. Change Progress:

Milestone	Date Sched.	Date Actual	Comment
Prepared			
Tested*			*Organic Changes
Submitted			
CCB Action			**Contractor Changes
Tested**			
TCTO RQST'D			
TCTO DIST.			

8. Remarks:

Figure 2-6 Computer Program Change Report, Change Status Section

f. Modules Affected - Entries contain the tree codes and names (see Software Tree below) of the computer program modules that require changing in order to implement the ECP.

g. Documents Affected - Entries identify by type, number, and title the documents that require changing in order to implement the ECP. In addition, the numbers and delivery schedules of the CNs that will accompany the ECP will be reported when known.

h. Related Configuration Changes - Entries cross-reference the CIs and other CPCIs that are affected by this change. The number of the ECP written against each related item is also provided.

i. Change Progress - Entries report the originally scheduled date and the actually accomplished date for each ECP development milestone. All milestones must be scheduled when an ECP enters the system; unknown dates may be entered as a series of 9's. Comments on an ECP's progress, such as rescheduled milestones, can also be reported.

j. Remarks - This entry may be used to include other pertinent information in the report.

The third section of the Computer Program Change Report is an exception report called the Red Flag and Activity List. It contains a list of ECPs, documents, change notices, and program modules for which a noteworthy event has occurred or for which a suspense date is in jeopardy. This List, sorted in order of descending priority of the items, is produced to aid the configuration manager locate those items that

most likely will require his attention. The List will be formatted as shown in Figure 2-7 and will contain the following data:

a. Header - This contains section- and CPCI-identification information.

b. Priority - This column contains the priority of the item being reported.

c. Item Type - Entries in this column identify the type of item being reported as "ECP", "Document", "Change Notice", or "Module".

d. ID Code - This column contains the identification of the item. Identification code elements for each type of item are specified in Chapter V.

e. Title - This column contains the title of the item being reported.

f. Reason Code - Entries in this column indicate why the item is being reported. The following codes will be used:

i. D - Item is delinquent. An item is reported delinquent when a suspense date is missed, that is, when an action has not been accomplished yet the date of the report is later than the date scheduled for the action.

ii. A - Item requires action. An item is reported as requiring action when it will become delinquent within 30 days of the date of the report.

iii. N - Item is not scheduled. An item is reported not scheduled when no suspense date has been recorded for

COMPUTER PROGRAM CHANGE REPORT

CPCI Number _____ CPCI Name _____

Section III - Red Flag and Activity, Changes Since _____

Priorty	Item Type	Number	Title	Reason	Remarks
Emergency	ECP	XXXX	----------	D	----------
	ECP	XXXX	----------	C	----------
	Test Plan	XXXX	----------	C	----------
		⋮	⋮	⋮	⋮
Urgent	ECP	XXXX	----------	A	----------
	VDD	XXXX	----------	N	----------
		⋮	⋮	⋮	⋮
Routine	Handbook	XXXX	----------	A	----------
		⋮	⋮	⋮	⋮

Figure 2-7 Computer Program Change Report, Red Flag and Activity Section

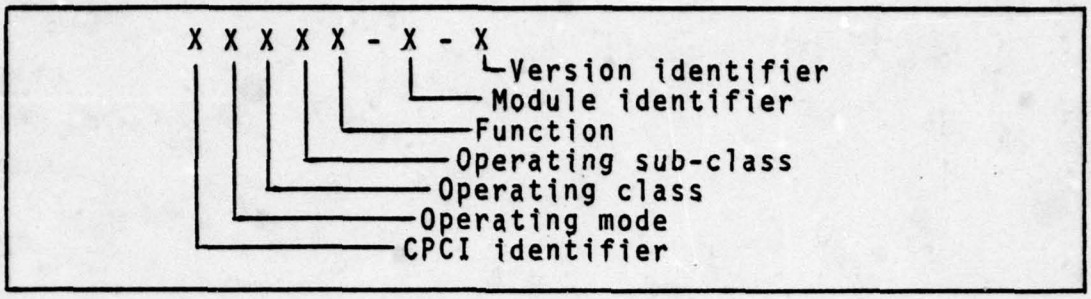


Figure 2-8 Tree Code Format

it in the database or when the date has been represented as a series of 9's.

iv. C - Item has been changed. Items are reported changed when either an ECP changes status or priority, a new ECP is submitted, a new document or change notice is issued, or changes are made to the lists of documents or modules affected by an ECP.

g. Remarks - This column will normally contain a narrative to explain action being take on red flag items.

Software Tree. The Software Tree is a report which will describe the program modules that comprise specified CPCIs. It will identify the status of each of the modules and will provide a hierarchically-indentured representation of a general tree structure used to model the functions that the modules perform. A Software Tree will be produced for each CPCI specified in the user's request.

Program modules are identified by unique, three-part "tree codes" assigned to them by the user (Figure 2-8). The first part of the code describes the function that the

module performs. Each of the five character positions represents a corresponding level in the tree, and the contents of the concatenated positions specify a unique function that SOFTSAS modules will perform. The second part of the code distinguishes between the members of a group of modules that, together, perform the specified function. The third part of the code specifies the current revision of the module being described.

The Software Tree consists of two sections. The first section lists the elements (nodes) that make up the first five levels of the tree structure (see Figure 2-9). These nodes are established by the user according to the following guidelines:

a. Each node at the first level identifies a different CPCI;

b. Each node at the second level identifies an operating mode as either "real-time" or "non-real-time";

c. Each node at the third level specifies an operating class as either operational, support, or test (Ref 4:10,1);

d. Each node at the fourth level specifies an operating sub-class as deemed appropriate by the contractor (DOC STDS 48-49)

3. Each node at the fifth level specifies a function as deemed appropriate by the contractor.

The second section specifies the tree at the sixth and seventh levels. It contains on subsection for each "func-

SOFTWARE TREE

CPCI Number _____ CPCI Name _____ Rev _____ Date _____

Section I - Functional Description

	Level	Tree Code
"Operating Mode"	2	x1000
"Classification"	3	x1100
"Sub-classification"	4	x1110
"Function"	5	x1111
⋮		
"Function"	5	x111n
"Sub-classification	4	x1120
⋮		
"Classification"	3	x1200
⋮		
"Operating Mode"	2	x2000
⋮		

Figure 2-9 Software Tree, Functional Description Section

tion" in order to describe specific characteristics of the modules supporting that function. As illustrated in Figure 2-10, each subsection contains the following data:

a. Header - This consists of function identification information;

b. Module Code - This column contains the tree codes of the modules which, together, perform the designated function;

c. Revision Letter and Date - These columns contain the current revision identifier for each module and the date that the revision was delivered to the procuring activity;

d. Source and Object Media Numbers - These columns contain the identifiers of the devices on which the module is stored in its source and object forms, respectively.

e. Source and Object File Addresses - These columns contain the addresses of the source and object files on the storage media;

f. Related Documents - This column contains the identifiers of documents that specifically address the operation, test, or maintenance of a listed module;

g. Description - This column contains the names of the listed modules.

Input Requirements

Inputs to SOFTSAS will consist of user requests to operate on the database in some way. Since discussions

SOFTWARE TREE

CPCI Number _____ CPCI Name _____

Section II - Module Description, Tree Code XXXXX

"Operating Mode"
"Classification"
"Subclassification"
"Function"

Module Number	Version Ltr	Version Date	Module Descrip	Source Media	Source Addr	Object Media	Object Addr	Related Docs

Figure 2-10 Software Tree, Module Description Section

about the database appear in this and in subsequent sections of this paper, some of the terms used to describe the structure of the database will be described here. Then, a description of legal SOFTSAS requests will be presented.

Basic Terms. The following terms will be used when describing the logical structure of the database, that is, the structure as viewed by the applications programmer:

1. Field - A field is the smallest unit of named data.

2. Record - A record is a named collection of ordered fields.

3. Database File - A database file is a logical collection of all occurrences (instances) of a given type of database record.

4. Database - A database is a collection of multiple record types (database files).

5. Realm - A realm is a defined subset of database records.

6. Primary Key - A primary key is one or more fields that can be used to differentiate between each of the records in a database file.

7. Secondary Key - A secondary key is one or more fields that can be used to identify a group of records in a database; however, it cannot distinguish between the records in a group.

User Requests. User requests consist of a sequence of words and symbols that resemble an English sentence written

```
OPEN realm-name SETTING USER = user-identifier
and MODE = [READ, EXCLUSIVE_WRITE, PROTECTED_WRITE]
```

Figure 2-11 OPEN Request

in the imperative mood. Table I lists the types of words and symbols that can be used. Each request begins with a command word. The command word is followed by any required and optional parameters, according to syntactic rules established for that command. The rules for each type of request are presented below in a metalanguage which utilizes the following conventions:

a. Words appearing in upper-case letters are a required part of a request or clause;

b. Words appearing in lower-case letters are abstractions for data to be supplied by the user;

c. Brackets enclose a group of words or clauses from which one must be singled out for use in the request;

d. Braces enclose a word or clause that may be repeated in the request;

e. Parentheses enclose a word or clause that may be included in the request at the user's options.

The remainder of this section describes the legal SOFTSAS requests.

1. Open. The OPEN request (Fig 2-11) instructs SOFTSAS to prepare the named database realm for processing. This request must be issued before any occurrence of a record in the realm can be accessed. The OPEN request must

Table I

Legal SOFTSAS Input Tokens

Token	Type*	Token	Type*
AND	L	OR	L
ASCENDING	X	PRINT	C
CLOSE	C	PRODUCE	C
COMPLETELY	X	PROTECTED_WRITE	X
CREATE	C	READ	X
DELETE	C	Realm-name	V
DESCRIBE	C	RECORD	X
END	C	Record-name	V
EQ	R	Report-name	V
EXCLUSIVE_WRITE	X	SETTING	V
Field-name	V	SORTED_ON	K
GE	R	USER	X
GT	R	User-identifier	V
LE	R	WHERE	K
LIMITED	X	= (EQ)	S
LS	R	¬ (NOT)	S
MODE	X	< (LT)	S
MODIFY	C	> (GT)	S
NQ	R	_	S
OPEN	C	; (end of request)	S
Operand	V		

*C=Command L=Logical operator K=Clause identifier
R=Relational operator V=Variable name X=Other keywords

```
CREATE  record-name SETTING field-name = operand
 ({AND field-name = operand})
```

Figure 2-12 CREATE Request

include a SETTING clause which is used to specify the identity of the user and the mode in which he wishes to access the realm.

Specifying an access mode aids SOFTSAS in protecting the integrity of the database. In order to process an update request such as CREATE, DELETE, or MODIFY, the user must specify either the EXCLUSIVE-WRITE or the PROTECTEX-WRITE mode. A realm that is being accessed in the EXCLUSIVE-WRITE mode by a program cannot be accessed by a concurrently operating program. A realm that is being accessed in the PROTECTED-WRITE mode by a program can only be opened in the READ mode for another program.

Upon encountering an OPEN request, SOFTSAS will verify that the named realm exists and that the user is authorized to access it. SOFTSAS will then attempt to prepare the realm in the requested mode for the user. If an attempt fails, another attempt will be made after a short delay.

2. Create. The CREATE request (Fig 2-12) instructs SOFTSAS to add a new occurrence of a record to the database. The request's parameters must specify the name of the record to be created and the value to be used as the primary key for the occurrence. The parameters may also specify other data to be stored in the record as well.

```
DELETE  record-name
 (WHERE field-name [EQ, GE, GT, LE, LT, NQ] operand
 ({[AND, OR] field-name [EQ, GE, GT, LE, LT, NQ]
                        operand})
```

Figure 2-13 DELETE Request

SOFTSAS will use the record's name to obtain its description from the database. It will then reserve and format storage space as required, sotring all data provided by the user in the request and recording all values for key fields (both primary and secondary) in the appropriate database indexes.

3. Delete. The DELETE request (Fig 2-13) instructs SOFTSAS to remove occurrences of a named record from the database. Unless a WHERE clause is provided as a parameter, SOFTSAS will delete all occurrences of the named record and all pointer to them. A WHERE clause contains search criteria (values for key fields) used by SOFTSAS to limit the effect of a database operation to certain occurrences of a record. When a WHERE clause is present, SOFTSAS will delete only those occurrences of the record whose key fields contain values that satisfy the search criteria.

Upon encountering a DELETE request, SOFTSAS will first locate and then erase all data associated with qualified record occurrences.

4. Modify. The MODIFY request (Fig 2-14) instructs SOFTSAS to alter the contents of occurrences of a named re-

```
MODIFY  record-name
 SETTING field-name = operand ({AND field-name=operand})
 (WHERE field-name [EQ, GE, GT, LE, LT, NQ] operand
  ({[AND,OR] field-name [EQ, GE, GT, LE, LT, NQ]
                          operand})]
```

Figure 2-14 MODIFY Request

cord, the request must alos include a SETTING clause. The SETTING clause will contain the names of the fields to be modified and the new data that they are to contain. Any fields of a record may be changed except for primary key fields. The user also has the option of specifying a WHERE clause. This clause has the same effect here as it does for the DELETE request.

Upon encountering a MODIFY request, SOFTSAS first locates the occurrences to be modified and then, one occurrence at a time, updates the fields specified in the request.

5. Produce. The PRODUCE request (Fig 2-15) instructs SOFTSAS to build and print one or more of the status accounting reports described earlier in this chapter. The

```
PRODUCE report-name ({AND report-name})
 (WHERE field-name [EQ, GE, GT, LE, LT, NQ] operand
 ({[AND,OR] field-name [EQ, GE, GT, LE, LT, NQ]
                          operand})]
```

Figure 2-15 PRODUCE Request

```
PRINT ([field-name ({AND field-name}), RECORD])
 (SORTED_ON [ASCENDING, DESCENDING] field-name
 ({AND [ASCENDING, DESCENDING] field-name})
 (WHERE RECORD = record-name ({AND field-name=operand})
```

Figure 2-16 PRINT Request

request will specify which reports are to be produced and may include a WHERE clause to restrict the reporting to certain CPCIs. Otherwise, the reports will contain staus accounting information for each of the CPCIs in the database.

6. Print. The PRINT request (Fig 2-16) instructs SOFTSAS to retrieve specific items of status accounting data. The request not only names the data to be retrieved but also dictates the format in which it is to be printed.

Three elements of the request identify the data to be retrieved. First, the user will provide, in a WHERE clause, the name of the record to be accessed. In addition, the user can specify search criteria in the WHERE cluase that restrict request processing to certain occurrences of the named record. Finally, the user can ask for all the data stored in qualified occurances of the record or, by specifying field names, can ask for only a subset of that data.

Two elements of the request dictate the format for the output data. All the data retrieved from a single occurrence of a record will constitute one line of output. In each line, the data will be presented in the order that their

```
PRINT ECP_ID AND ECP_MILESTONE_NAME AND ECP_MILESTONE_
SCHED_DATE SORTED_ON ASCENDING ECP_MILESTONE_SCHED_DATE AND
ASCENDING ECP_ID WHERE RECORD = ECP_MILESTONES AND
CPCI_ID =XXX;
```

ECP-#	MILESTONE	DUE-DATE
00001	Preparation	1 Jan 78
00002	Preparation	5 Jan 78
00002	Submittal to CCB	20 Jan 78
00001	Submittal to CCB	28 Jan 78
00003	Preparation	28 Jan 78
00002	CCB action	10 Feb 78

```
PRINT ECP_ID AND ECP_MILESTONE_SCHED_DATE SORTED_ON
ASCENDING ECP_ID WHERE RECORD = ECP AND CPCI_ID =XXX;
```

ECP-#	DUE DATE	MILESTONE
00001	1 Jan 78	Preparation
00001	28 Jan 78	Submittal to CCB
00002	5 Jan 78	Preparation
00002	20 Jan 78	Submittal to CCB
00002	10 Feb 78	CCB Action
00003	28 Jan 78	Preparation

Figure 2-17 Sample Input and Output for PRINT Requests

```
DESCRIBE ([[COMPLETELY, LIMITED, record-name
          ({AND record-name}) (COMPLETELY),
          field-name ({AND field-name})]]
```

Figure 2-18 DESCRIBE Request

corresponding fields were named in the request. If fields were not named in the request, the data will be presented in the order that the fields were defined when the structure of the database was created. Besides controlling the order in which the data will be presented in a line, the user may specify that the lines of output be sorted on the contents of one or more fields in the record. However, if no SORTED_ ON clause is provided, the lines will be written out in the order the record occurrences were accessed.

The examples in Figure 2-17 illustrate how the format of the PRINT request affects the format of its output.

7. Describe. The DESCRIBE request (Fig 2-18) instructs SOFTSAS to provide information to the user about the logical structure of the database. The request can be submitted in one of six forms. Figure 2-19 contains a list of the forms and the outputs they produce.

The majority of responses to DESCRIBE requests will consist of record descriptions or field descriptions or both. The description of a record consists of:

a. The name of the record;

b. The names of the fields that comprise it;

c. The number of times the record occurs in the realm;

Form	Output
DESCRIBE record(s) COMPLETELY	Descriptions of named records and their fields
DESCRIBE COMPLETELY	Descriptions of each type of record in the realm and descriptions of each record's fields
DESCRIBE (no parameters)	Descriptions of each type of record in the realm
DESCRIBE LIMITED	Names of records in the realm
DESCRIBE field(s)	Descriptions of named fields
DESCRIBE record(s)	Descriptions of named records

Figure 2-19 Types of DESCRIBE Requests

```
CLOSE  realm-name
```

Figure 2-20 CLOSE Request

d. The number of bytes of storage space that the record occupies; and,

e. The identity of the record's primary and secondary key fields.

The description of a field consists of:

a. The name of the field;

b. The length of the field in bytes;

c. The type of data the field contains; and,

d. The relative position the field occupies within the record.

8. Close. The CLOSE request (Fig 2-20) instructs SOFTSAS to revoke the user's access to the named realm. The request, in effect, cancels a previous OPEN request.

9. End. The END request (Fig 2-21) instructs SOFTSAS to terminate job execution. This request must be included in the user's input stream, and any requests submitted after the END request will not be processed.

Data Management Requirements

In a database processing environment, a database manage-

```
END
```

Figure 2-21 END Request

ment system (DBMS) interfaces between the database and the application programs to disencumber the programs of data storage details and to protect the data from misuse and abuse. User application programs interact with the DBMS only to update and retrieve data. Other data management functions such as data base construction and security are not entrusted to the user. Rather, they are performed by a single manager, called the Database Administrator (DBA), who uses special utility programs furnished with the DBMS. In either case, the data itself is referenced in logical, or symbolic, terms.

Although it is beyond the scope of this thesis to select or design a DBMS to support SOFTSAS, the requirements that SOFTSAS will levy on its DBMS will be presented here.

Data Manipulation. SOFTSAS will generate database manipulation requests expressed in terms of logical data relationships. The DBMS must transform these requests into basic input/output operations on the database file, dispatch them to the host operating system, and then report back to SOFTSAS the status of the request's outcome. The following are descriptions of the data manipulation functions to be performed by the DBMS:

a. Find - The purpose of this function is to locate, and to return unique identifiers for, occurrences of records that satisfy a specific search criterion. SOFTSAS will provide:

i. The name of the record to be located;

ii. A relational expression consisting of the name of a key field, a value for the key field, and a relational operator code; and,

iii. The address and size of a buffer to contain the identifiers.

b. Open - The purpose of this function is to prepare a database realm for processing given the name of the user, the name of the realm he wishes to access, and the type of access protection he needs.

c. Close - The purpose of this function is to terminate a user's access to a realm.

d. Get-dict - The purpose of this function is to retrieve specific information about the logical structure of the database. SOFTSAS will specify exactly what information is needed and the address and size of a buffer in which to store it.

e. Get-data - The purpose of this function is to retrieve status accounting data from the database given:

i. A record occurrence identifier that was previously "found";

ii. The names of the fields in the record to be queried; and,

iii. The address and size of a buffer to hold the retrieved data.

f. Update - The purpose of this function is to change data which had been stored in the database. SOFTSAS will supply a record occurrence identifier that was previously

"found", the names of the fields to be changed, and the address of a buffer that contains the new data.

g. Add-new - The purpose of this function is to store a new occurrence of a record in the database and to create all the appropriate linkages and addressing facilities. SOFTSAS will supply the name of the record to be added, the names of fields in the record for which the user has provided data, and the address of a buffer containing the status accounting data. Unique data values must be provided for the record's primary key field(s).

h. Delete - The purpose of this function is to remove an occurrence of a record from the database and to erase all linkages to it. The identifier of the record occurrence will be provided by SOFTSAS.

Data Description. To reference the status accounting data, SOFTSAS will use a logical, or symbolic, representation of the database. In this way, the program will not be affected by changes to the database's physical structure.

The logical model of the database will be constructed (by the Database Administrator) by using a data description language (DDL). A DDL provides the means to assign names and types to fields and to define the way that fields are grouped into records. In addition, a DDL is used to designate primary and secondary keys, to establish inter-record relationships, and to impose privacy controls. The vocabulary for the DDL will not be known until a DBMS has been

selected to support SOFTSAS. This is because most database management systems provide their own DDL to represent the frequently unique constructs used by the DBMS in organizing its database.

A utility program will need to be acquired with the SOFTSAS DBMS to build a dictionary of data definitions from a set of DDL specifications. A similar program, capable of modifying an existing dictionary, must also be acquired. The dictionary will form part of the database, and its information will be directly available to the DBMS and indirectly available to SOFTSAS (via the Get-dict database function).

Data Security. SOFTSAS and the DBMS will protect the status accounting data to the extent that software is able to do so. The threats of most concern are unauthorized access to the data and loss of the data. Unauthorized access to the data refers to illegal data acquisition or modification, including modification to the structure of the database. Data can be lost because of natural hazards (fire, flood, or earthquake), human hazards (physical damage due to dropping a disk pack), and operating hazards (program error or bad input).

To protect against illegal data acquisition and modification, SOFTSAS will validate all user's requests against a list containing the names of legal users and the types of requests each is authorized to submit. In addition, some

users who may be authorized to submit a particular SOFTSAS request may not be authorized access to all the data in the database. Therefore, the DBMS will check each database request against privacy controls imposed on the data when the database was defined.

To protect against loss of data, a utility program will need to be acquired with the DBMS to provide a database backup and recovery capability. All database modifications will be recorded in a journal, and a copy of the database will periodically be created and saved. When data is lost, a copy of the database that was created before the problem occurred will replace the damaged database. Then, selected modifications (called after-images) can be reapplied from the journal to complete the recovery process. The DBA should be responsible for the saving, restoring, and recovering of the status accounting data.

Summary

This chapter has presented an informal definition of the SOFTSAS processing requirements. Briefly stated, SOFTSAS must maintain status accounting data and provide formal reports and responses to ad hoc queries.

These requirements are the product of the second stage of the software development life-cycle and form the basis for a formal functional specification using SADT models in Chapter III.

III Formal Functional Specification

Introduction

The formal functional specification of the SOFTSAS system is presented as two models depicting "what" the system is to do. The models were developed using a disciplined approach to functional analysis introduced by Softech, Inc., as part of their Structured Analysis and Design Technique (SADT) (Ref 35).

The purpose of the models is to provide an unambiguous specification of the system. The system is represented by an organized sequence of well-defined, interrelated diagrams. SADT uses the principle of top-down systems design to decompose a very general view of the system into successive levels of smaller, component views called modules. As the modules at each level are decomposed, more details about the system are exposed. This process continues until a level is reached at which the modules are small enough such that they can be easily understood and their interfaces can be completely delineated.

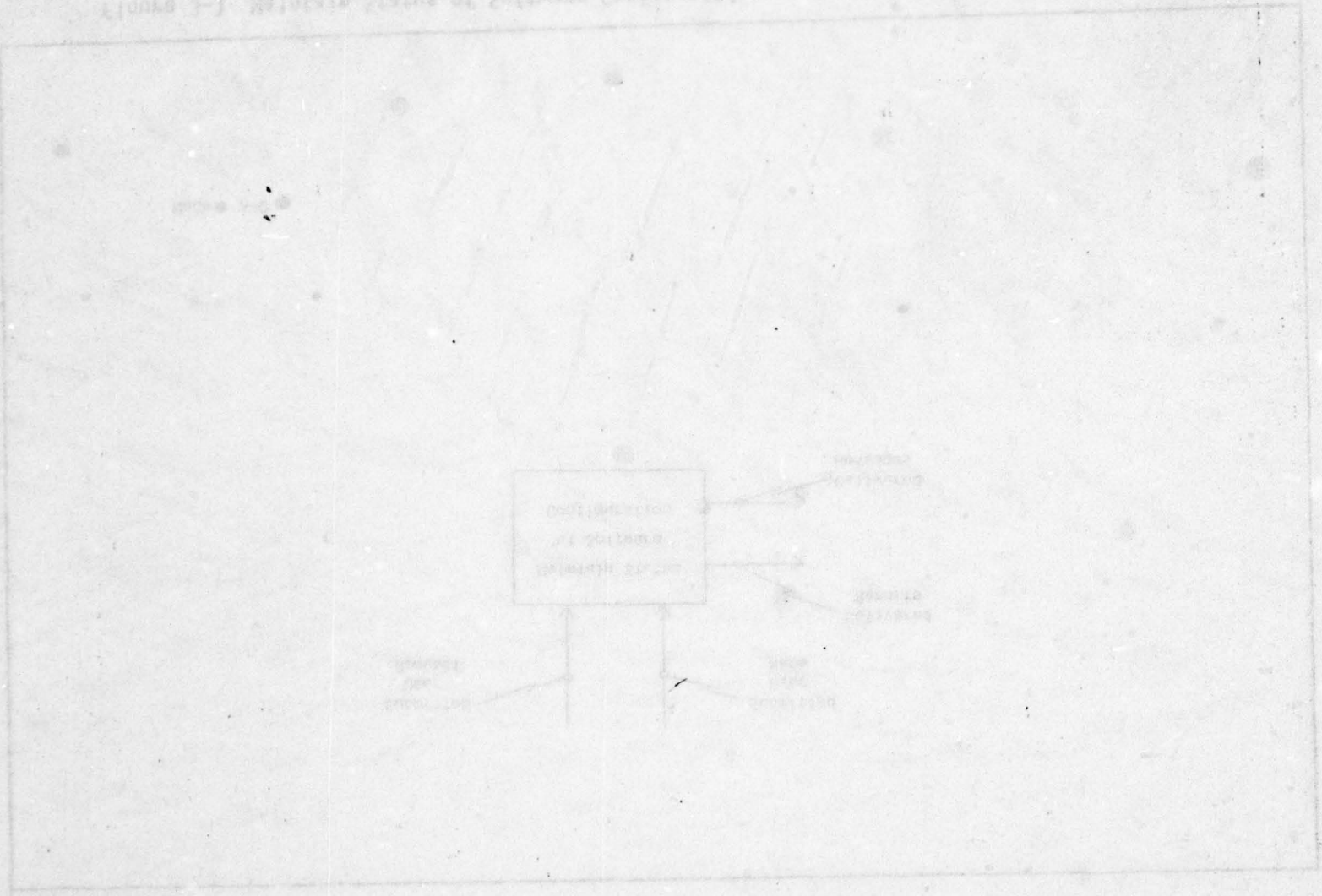
Since systems can be thought of as a collection of things (data) and the processes that act on them (activities), SADT requires that a system be modelled twice. In one model, "actigrams" emphasize the activities with data items interfacting between them. In the other model, "datagrams" emphasize the data with activities interrelating them. Together, these models provide a graphic representa-

tion that clearly reveals the relationships between all the elements and functions of the system.

The rest of this chapter contains the SOFTSAS activity and data models with supporting text for each. Preceeding each model is a "node index" which gives an overview of the decomposition structure of the system. Instructions for reading the SADT "diagramming language" are contained in Appendix A and are further described in Refs. 35 and 36.

Activity Model

Node	Title
A-0	Maintain Status of Software Configuration
A0	Maintain Status of Software Configuration
A1	Analyze User Request
A11	Check User Request
A12	Check User Authorization
A13	Report Status of Outcome
A14	Determine Output Data Formats
A15	Generate Database Requests
A16	Generate Messages
A2	Process Database Requests
A3	Produce Reports
A31	Dispatch Report Formats
A32	Build Configuration Index
A321	Produce Section Header
A322	Describe Current Document Configuration
A323	Describe Scheduled Changes
A324	Build Title Page
A33	Build Computer Program Change Report
A331	Produce Change History Section
A332	Produce Change Status Section
A333	Produce Red Flag and Activity Section
A34	Build Software Tree
A341	Describe Software Functions
A342	Produce Module Descriptions
A4	Compose Query Response
A41	Sort Data Items
A42	Compose PRINT Response
A43	Compose DESCRIBE Response
A5	Put Outputs
A51	Determine Message Destination
A52	Deliver Reports to Printer
A53	Deliver Messages to Printer
A54	Deliver Messages to Terminal



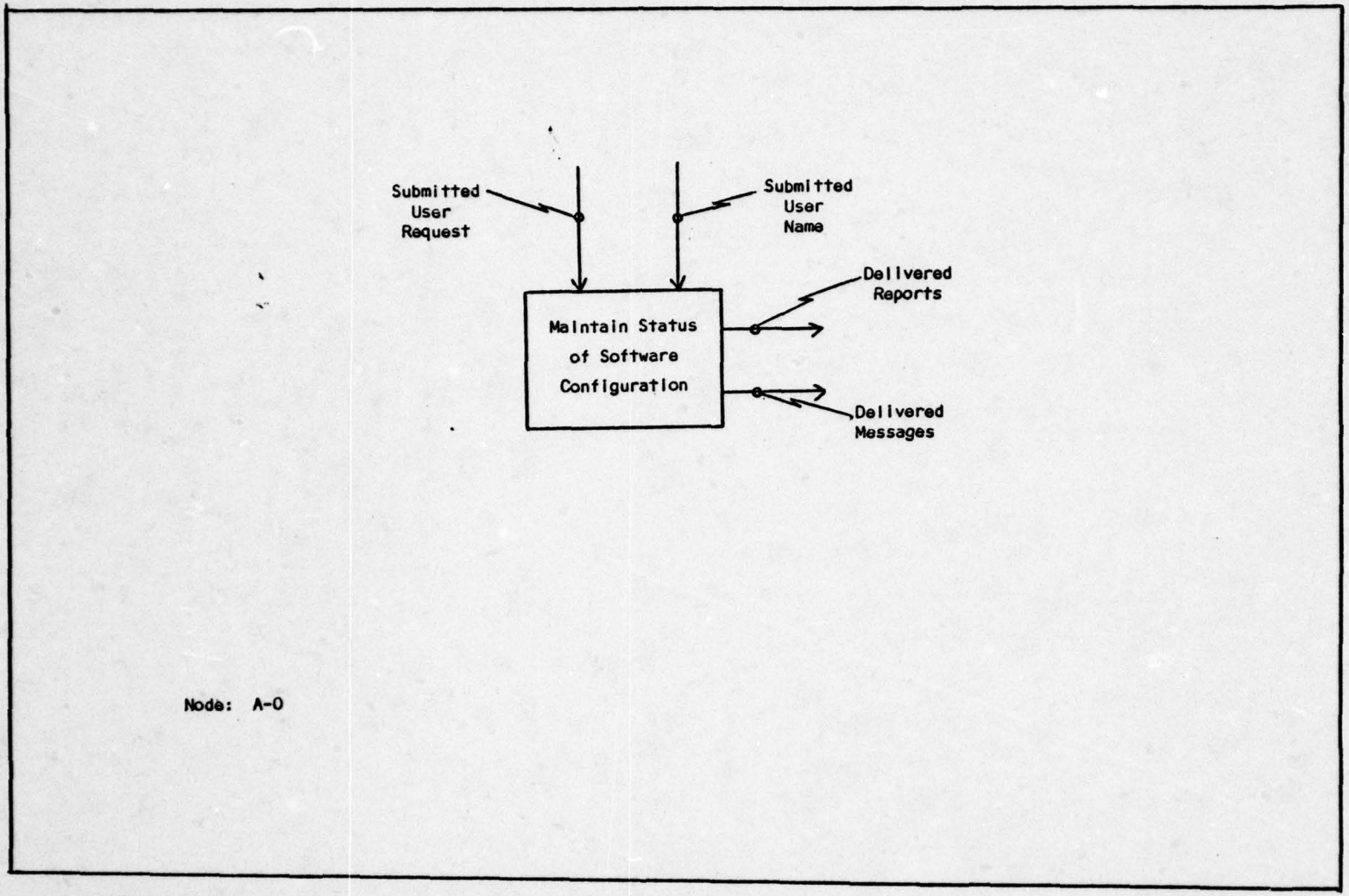


Figure 3-1 Maintain Status of Software Configuration

Viewpoint and Purpose

These models were developed as a tool for systems analysts who are tasked with designing a "software status accounting" applications program to be called SOFTSAS. SOFTSAS will be used, by offices responsible for tracking CPCI baselines, to maintain data which identifies the status of a software system's configuration. The system should insure that only authorized users are allowed access to the system's data and that retrieved data can be presented to the user in a meaningful format.

A-0 Text

Users (C2) interface with SOFTSAS via submitted requests (C1) which control the operation of the system. These requests instruct SOFTSAS to record new status accounting data and to relate this data back to the user in the form of formal reports (01) or of formatted responses to ad hoc queries for specific information (02).

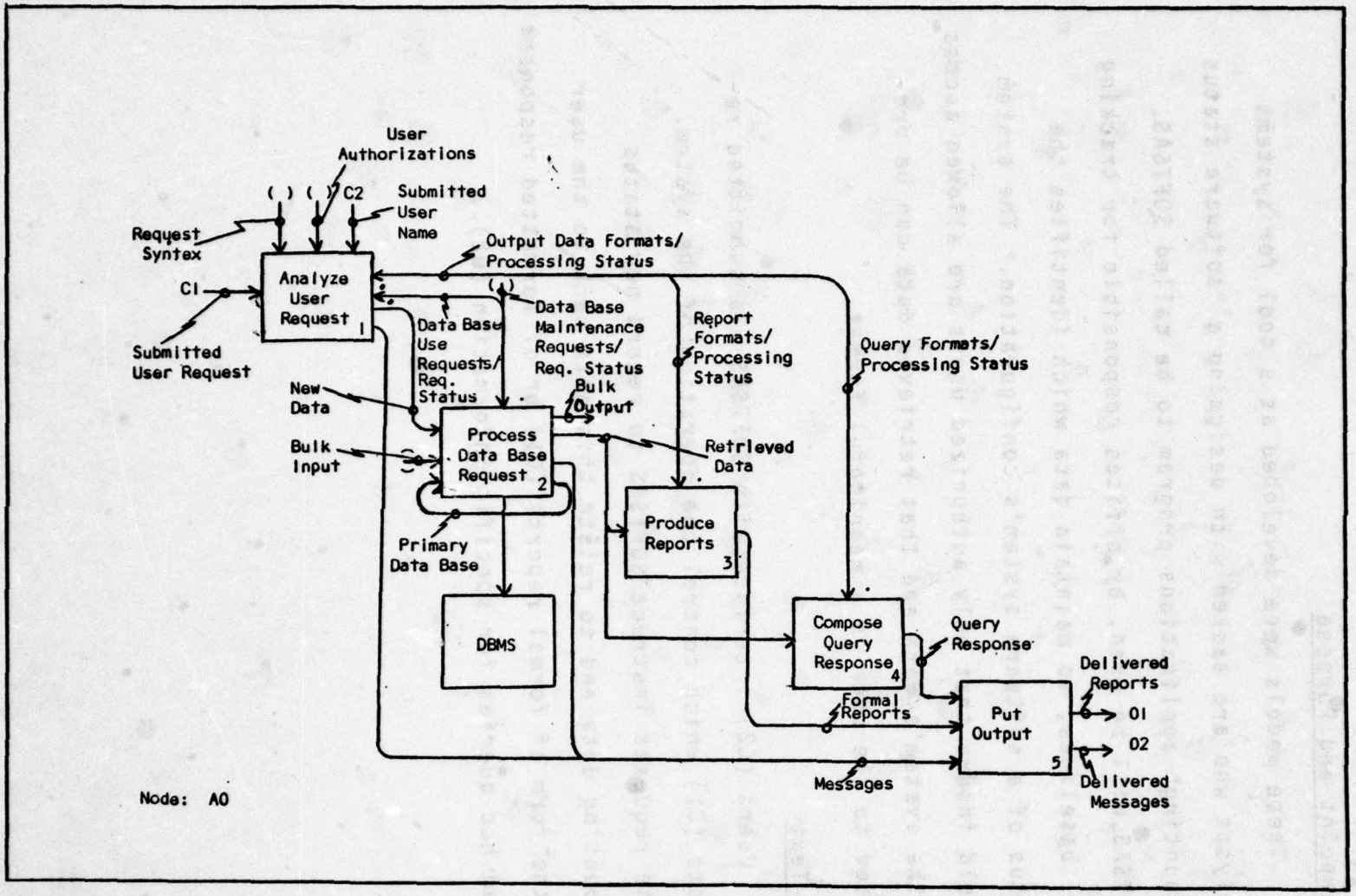


Figure 3-2 Maintain Status of Software Configuration

A0 Text

Submitted user requests (1I1) are evaluated by the "Handle User Request" activity (1). If the user (1C2) submits a request that does not meet the validation criteria (1C1, 1C2), the request is rejected and the user is notified (104). Otherwise, (1) directs that the status accounting data (2I1, 202) be stored or retrieved as per request (1Q2). In addition, for data retrieval requests, (1) determines the output formats (101) to be used by (3) and (4).

"Process Database Request" (2) is a sub-system (Database Management System) which accesses and maintains the primary database (2I3, 204) inresponse to database requests (2C1). New user data (2I1) are obtained from (1) and retrieved data (202) are dispatched to the output formatting activities (3, 4). This activity (2) also makes back-up copies (201) of the database and database modifications journal for future use as (2I1) should recovery from inadvertant data loss be required. An indicator (2C1) is returned to SOFTSAS to signal if the database request was successfully processed. Activity (2) is not modelled further in this report since its specifications are beyond the scope of this thesis. However, the requirements that SOFTSAS levies on (2) are discussed in Chapter II.

"Produce Report" (3) uses the retrieved status accounting data (3I1) and specified formatting requirements

(3C1) to build formal status accounting reports (3O1). "Compose Query Response" (4) uses the retrieved data (4I1) and specified formatting requirements (4C1) to build responses (4O1) to ad hoc user requests for data.

All information (5I1, 5I2, 5I3) prepared for the user by SOFTSAS are outputted by (5) in the form of formal reports (5O1) and user messages (5O2).

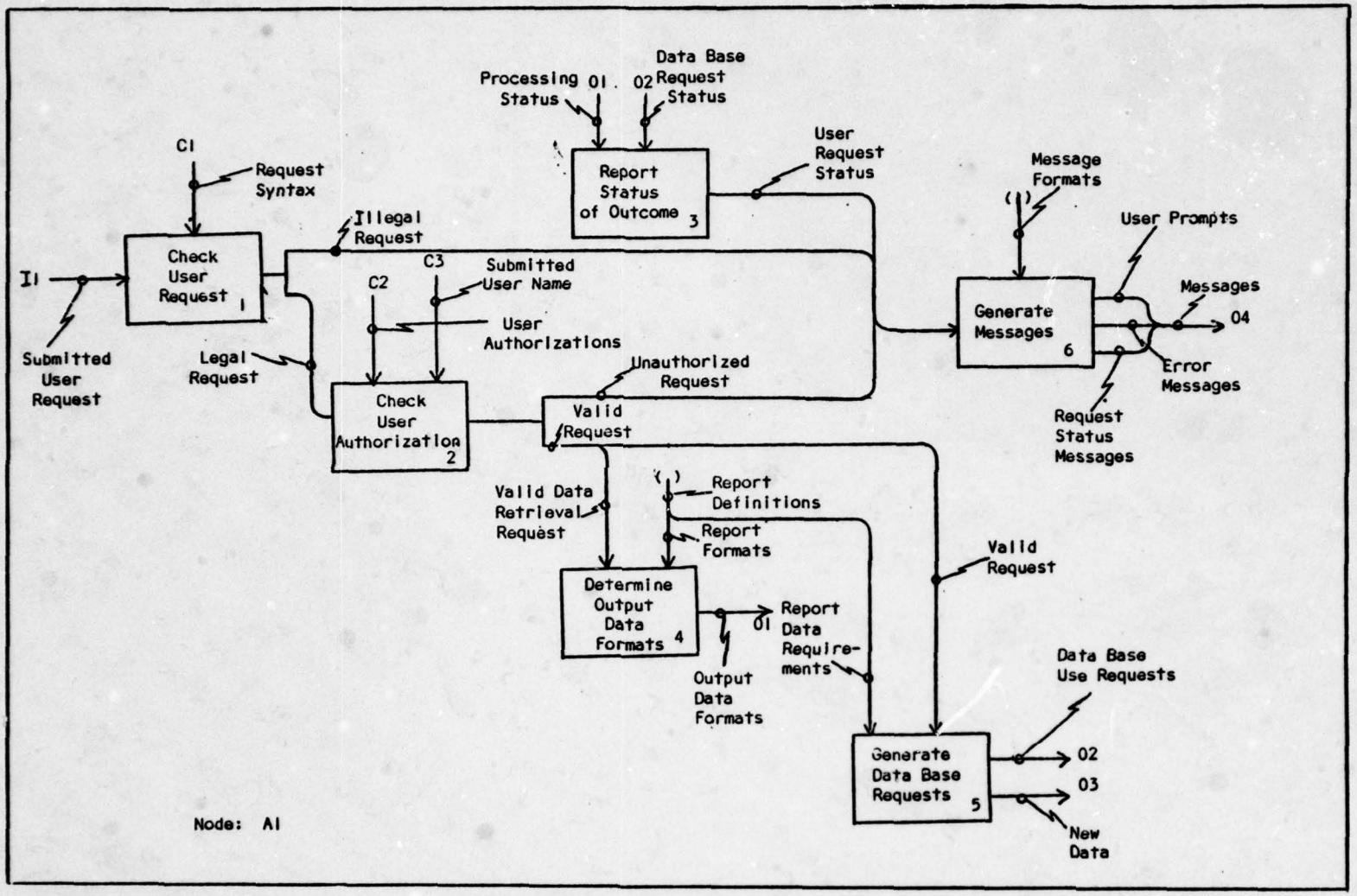


Figure 3-3 Analyze User Request

A1 Text

A submitted request (1I1) is checked by (1) for legal request syntax (1C1). If it is a legal SOFTAS request (2I1), it is checked against established user authorizations (2C1) by (2) to insure that the user (2C2) is authorized to make the request. Invalid requests (101, 201) are rejected, and the user is notified by (6).

For SOFTSAS data retrieval requests (4C1), output data formats (401) are prepared by (4) and dispatched to the activity responsible either for building formal reports or for constructing query responses. Report formats are predetermined (4C2), whereas query response formats are specified in the data retrieval request (4C1) itself.

"Generate Database Requests" (5) prepares database commands and parameters (501, 502). It uses the infomation provided by the valid SOFTSAS requests (502) or, in the case of a PRODUCE request, it uses the predetermined data requirements of the reports to be built (5C1).

Status indicators (3C1, 3C2) are used by (3) to report back to the user the final outcome of each valid request (301). This data (6I3), together with user prompts (6I1) and error notifications (602), is embodied in messages (04) prepared by (6) according to predetermined formats (6C1).

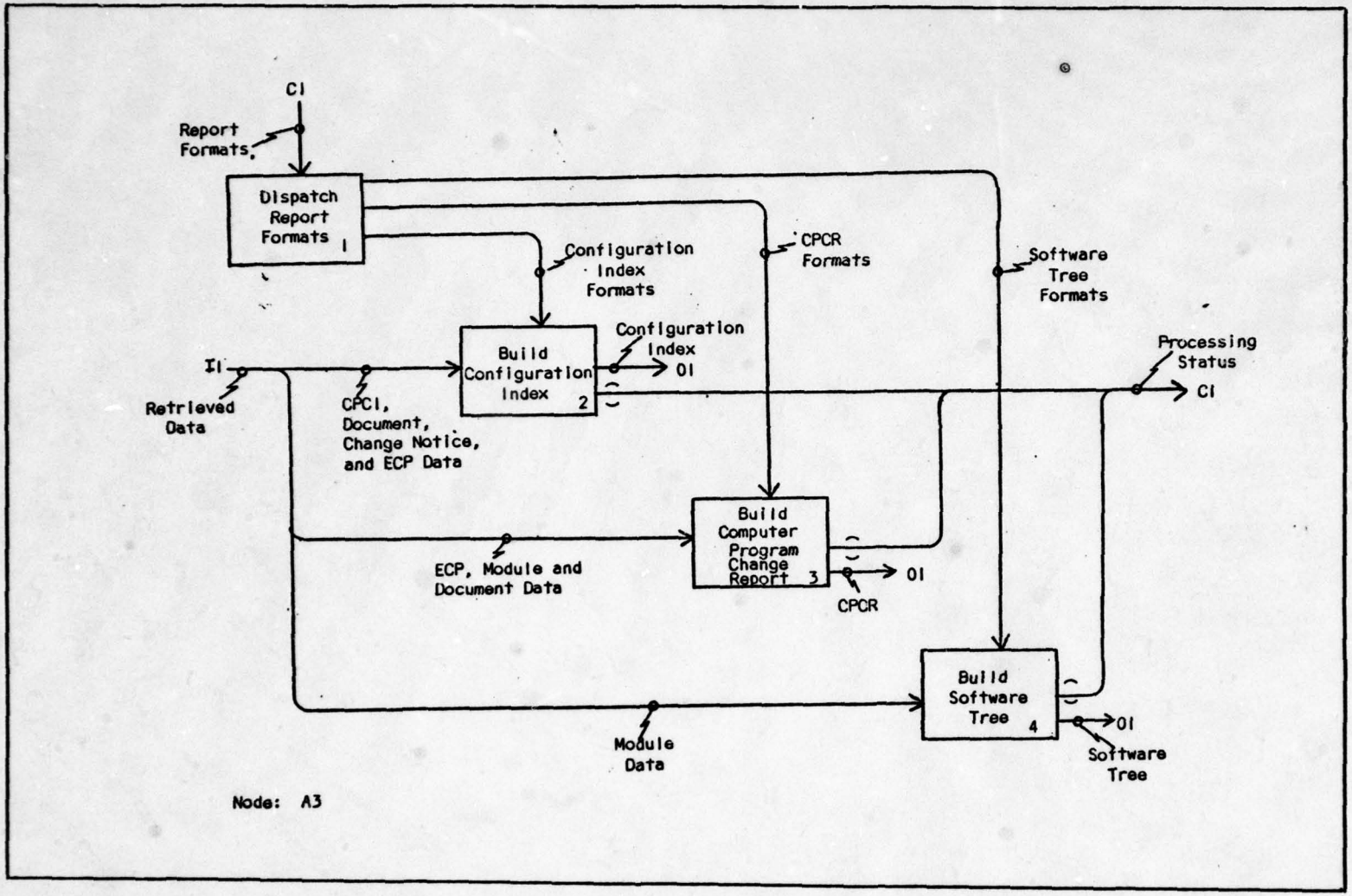


Figure 3-4 Produce Reports

A3 Text

Report formats (1C1) are dispatched by (1) to the appropriate report-generating activity (2, 3, 4). There, the formats (2C1, 3C1, 4C1) control how the retrieved status accounting data (2I1, 3I1, 4I1) is edited for output (201, 302, 402). The outcome status of the report building activity is delivered at (202, 301, 401).

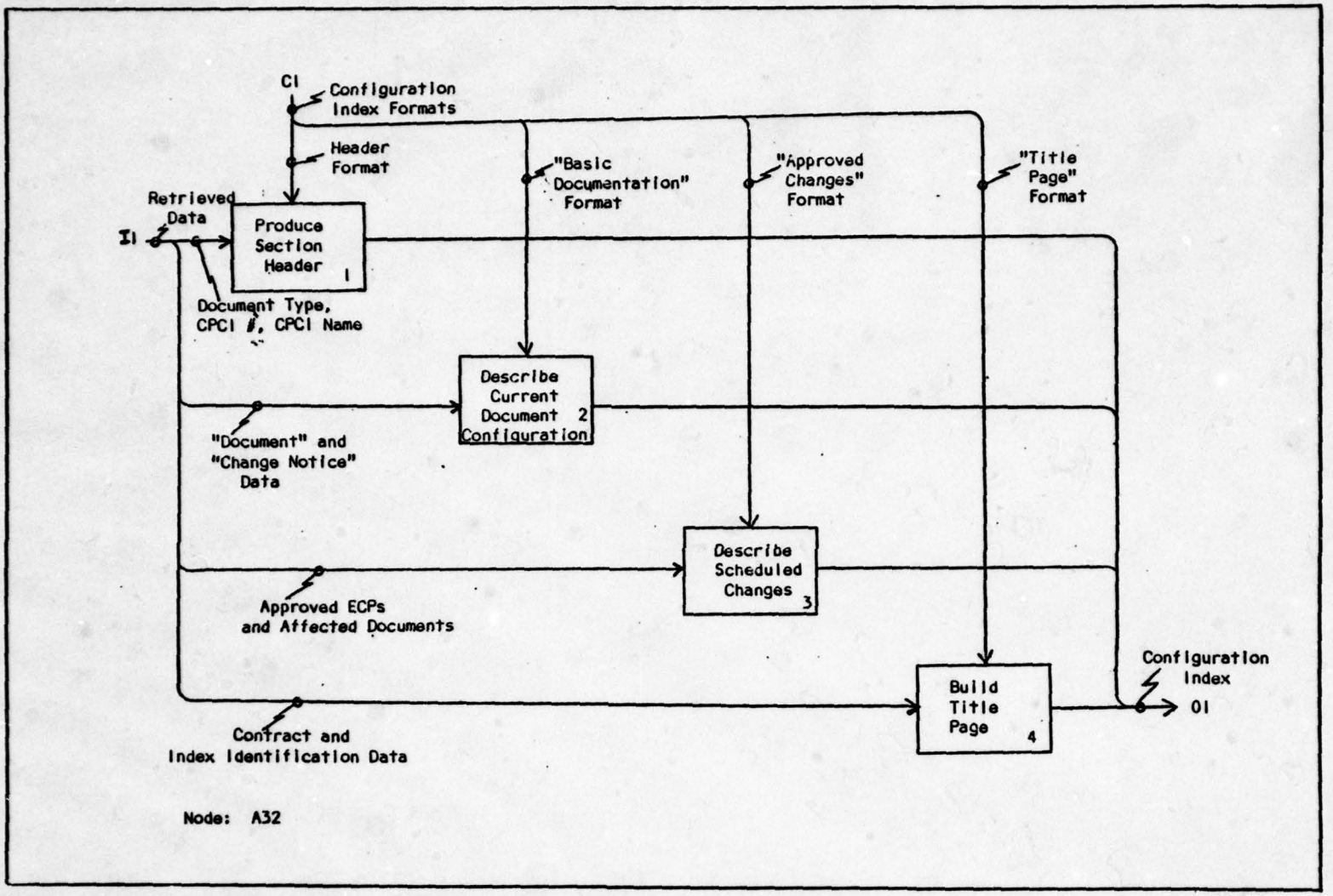


Figure 3-5 Build Configuration Index

A32 Text

The Configuration Index (C1) is built according to predetermined formats (C1). It consists of a title page (401) and a division for each CPCI to be reported. Each division consists of six sections. Retrieved data pertaining to a single CPCI (I1) will comprise each division of the Index.

Activity 'Build Title Page' (4) will be performed once for each issue of the Index. The other activities (2, 3, 4) will combine to produce a single section and will be performed six times for each CPCI being reported.

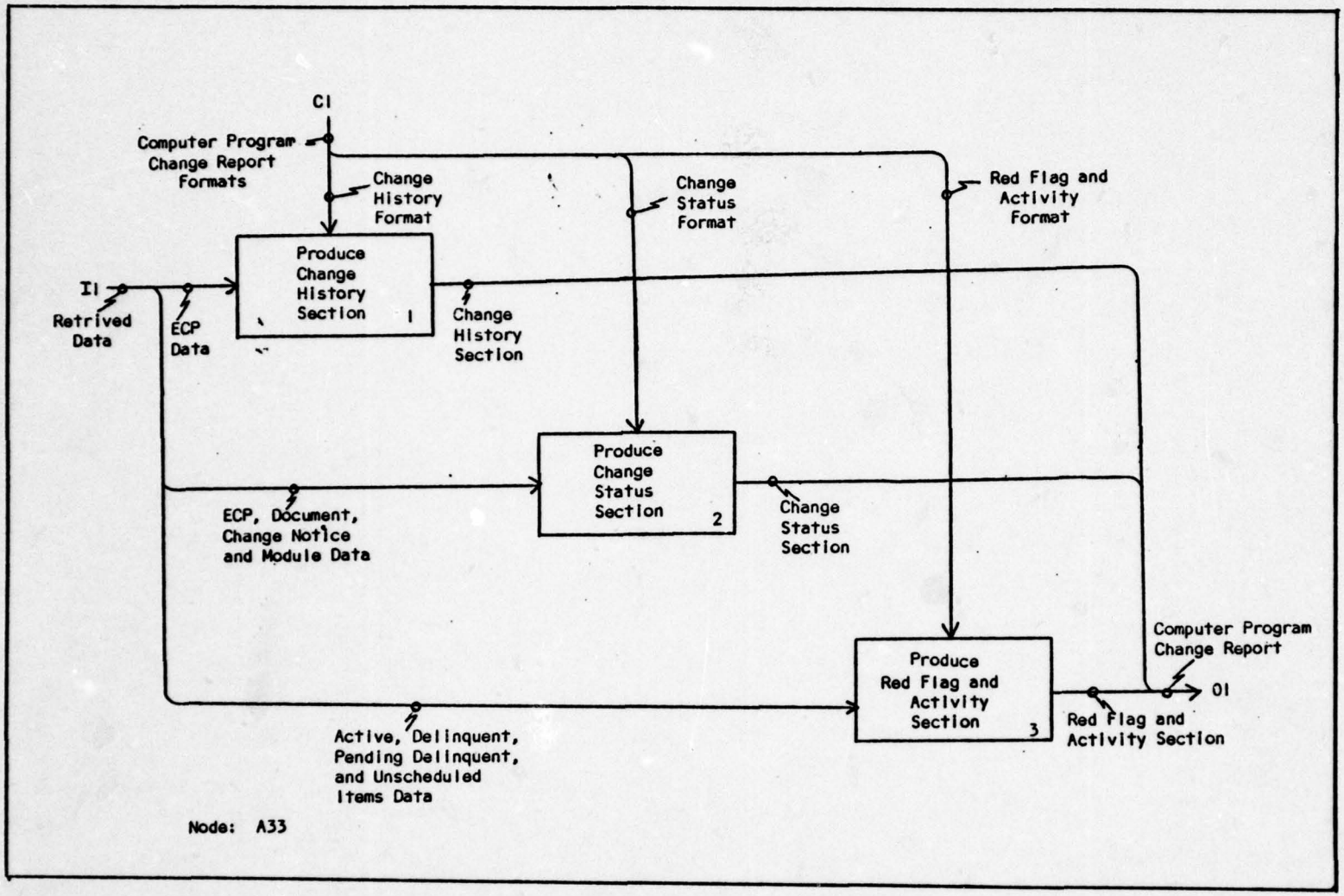


Figure 3-6 Build Computer Program Change Report

A33 Text

The Computer Program Change Report (01) identifies the status of engineering changes proposed for CPCI. For each CPCI being reported, it consists of a Change History section (101), a Change Status section (201), and a Red Flag and Activity section (301).

Activity (1) produces the Change History section (101) which contains a list of all the change (1I1) ever proposed against a specified CPCI (1C1).

Activity (2) produces the Change Status section (201) which contains information about each active change proposal (2I1) for a specified CPCI (2C1).

Activity (3) produces the Red Flag and Activity section (301) which identifies ECPs, documents, and change notices (3I1) that have missed a suspense date, that are within 30 days of missing a suspense date, or that have not recorded a suspense date for a milestone. In addition, the section identifies:

1. ECPs that have changed status or priority;

2. Documents or change notices issued since the last report was produced; and,

3. Changes to lists of documents and modules which are affected by an ECP.

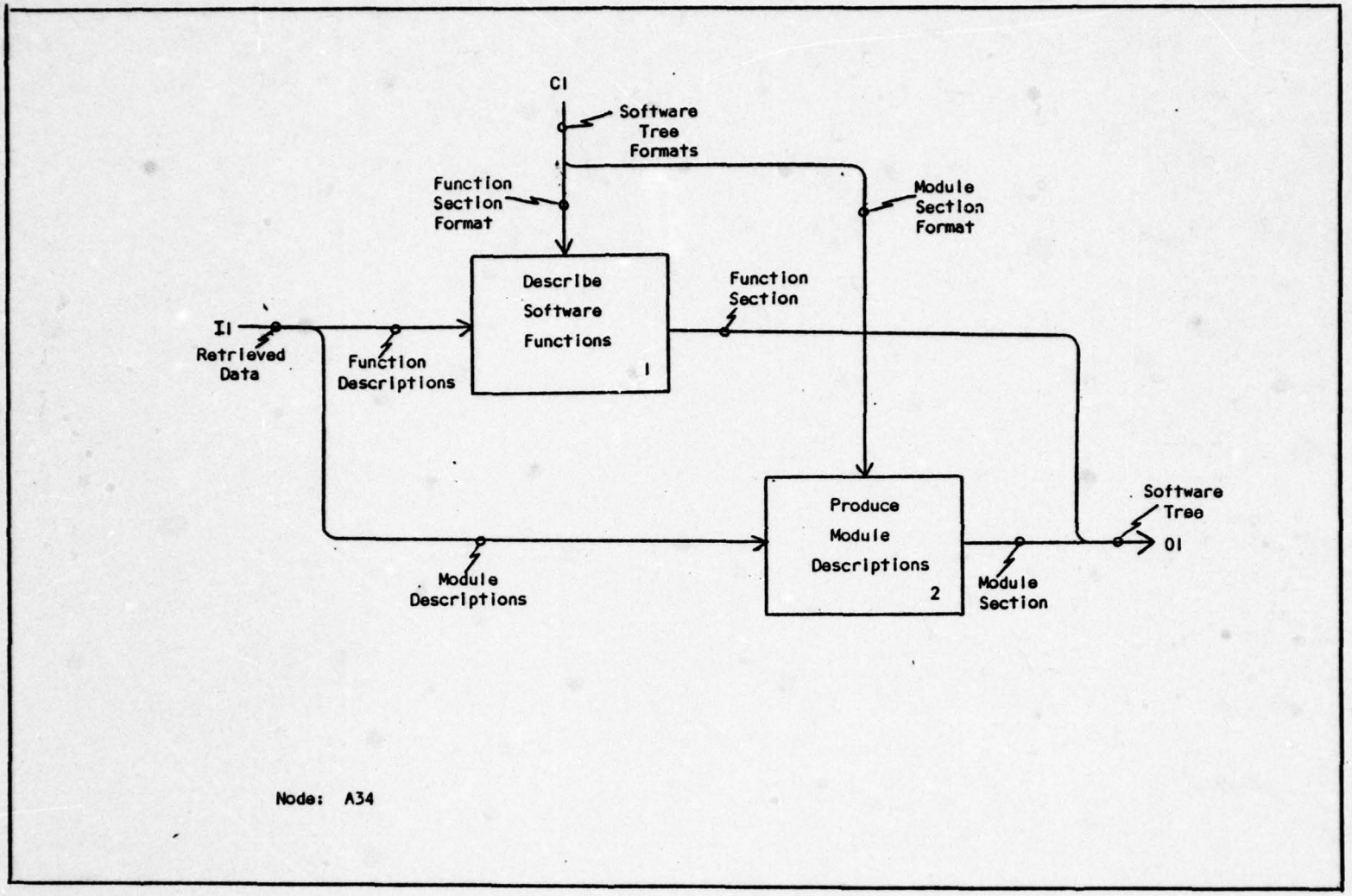


Figure 3-7 Build Software Tree

A34 Text

The Software Tree (01) is a two-section report that describes the configuration and status (I1) of the modules that comprise a specified CPCI. The function section (101) is produced by (1) and contains a hierarchically-indentured description of the functions (1I1) performed by the modules. The module section (201) is produced by (2) and contains descriptions of the modules (2I1) that perform each of the described functions. Construction of the Software Tree (01) is controlled by predetermined formats (C1), and identifiers of the CPCIs to be reported comprise a part of the formats.

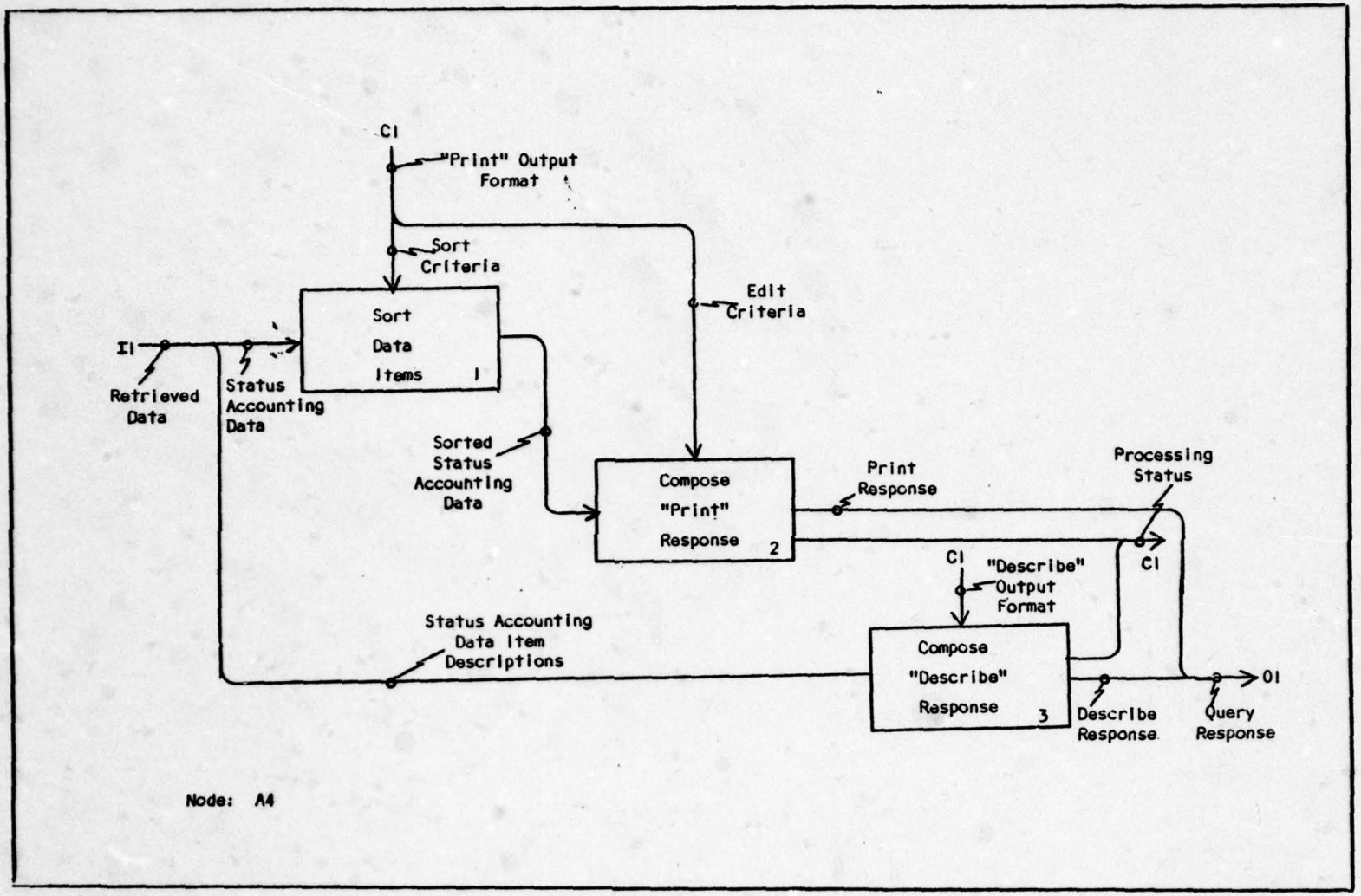


Figure 3-8 Compose Query Response

A4 Text

Query responses (01) result from either a user PRINT (1C1) or DESCRIBE (2C2) request. PRINT responses (201) consist of status accounting data (1I1) that have been sorted by (1) based on certain field values (1C) and then edited by (2) based on formatting guidelines (2C1) specified in the user's request. DESCRIBE responses (301) consist of descriptions (3I1) of status accounting data items; the descriptions are formatted by (3) according to predetermined instructions (3C1). The outcome status of query processing is provided at (202, 302).

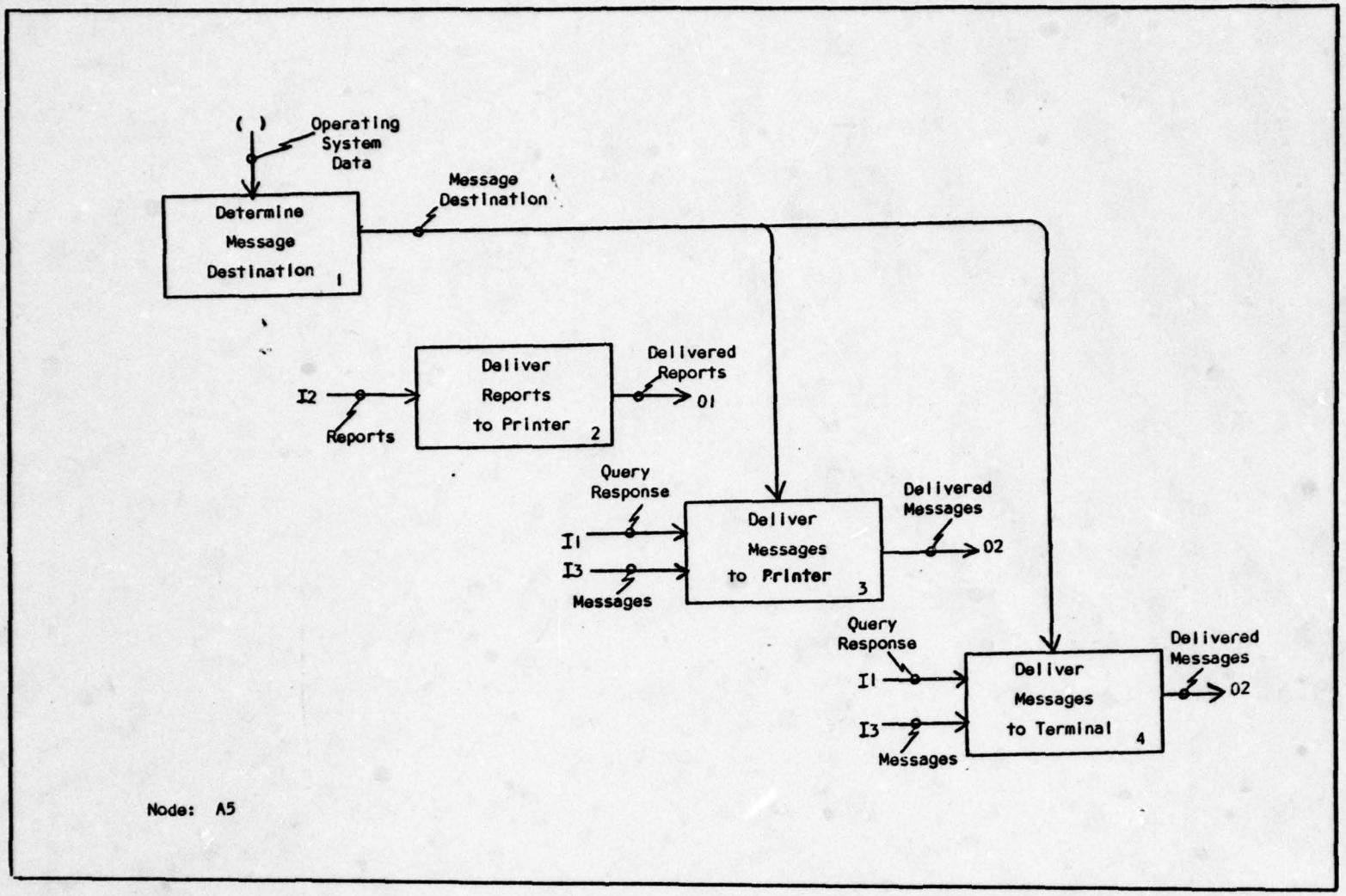


Figure 3-9 Put Outputs

A5 Text

SOFTSAS outputs consist of reports (2O1) and messages (3O1, 4O1). All reports (2I1) are delivered to the on-line printer by (2). The detination (1O1) of messages (3I1, 3I2, 4I1, 4I2) is determined by (1) based upon the source of user inputs identified at (1C1). Messages for users submitting inputs from a terminal will be delivered by (4) to the user's terminal. Messages for users submitting inputs from a card reader or other such device will be delivered by (3) to the on-line printer.

Data Model

Node	Title
D-0	Software Status Accounting System Data
D0	Software Status Accounting System Data
D1	System Data
D11	User Authorizations
D12	Request Syntax
D13	Report Definitions
D2	Requests
D21	User Request
D22	Invalid Request
D23	Valid Request
D24	Database Use Requests
D3	Status Accounting Database
D31	Database Dictionary
D32	Database Directory
D33	Status Accounting Data
D34	Processing Status
D35	After-images
D4	User Outputs
D41	Retrieved Data
D42	Reports
D43	Messages

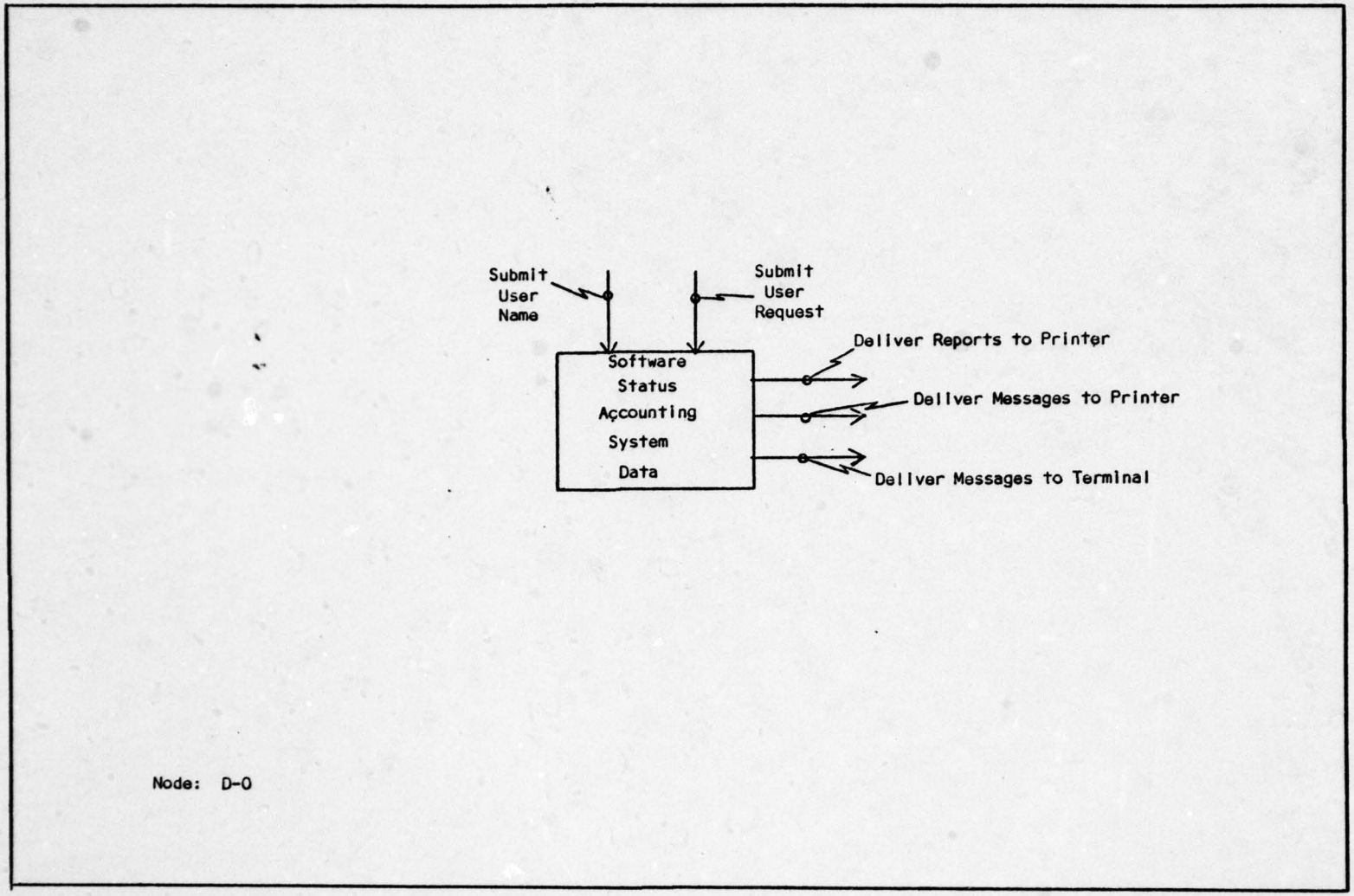


Figure 3-10 Software Status Accounting System Data

D-0 Text

"Submit User Request" (C2) controls the modification and use of SOFTSAS data. Since restrictions can be placed on the types of requests users are authorized to submit, "Submit User Name" (C1) identifies each user to the system.

Two types of output will be delivered to the user: formal reports and messages. All formal reports are written to the on-line printer (01). However, messages are only delivered to the printer (02) when SOFTSAS is executing in the batch mode. When executing interactively, that is, when a user is submitting inputs from a terminal, SOFTSAS will deliver all messages to the user's terminal (03).

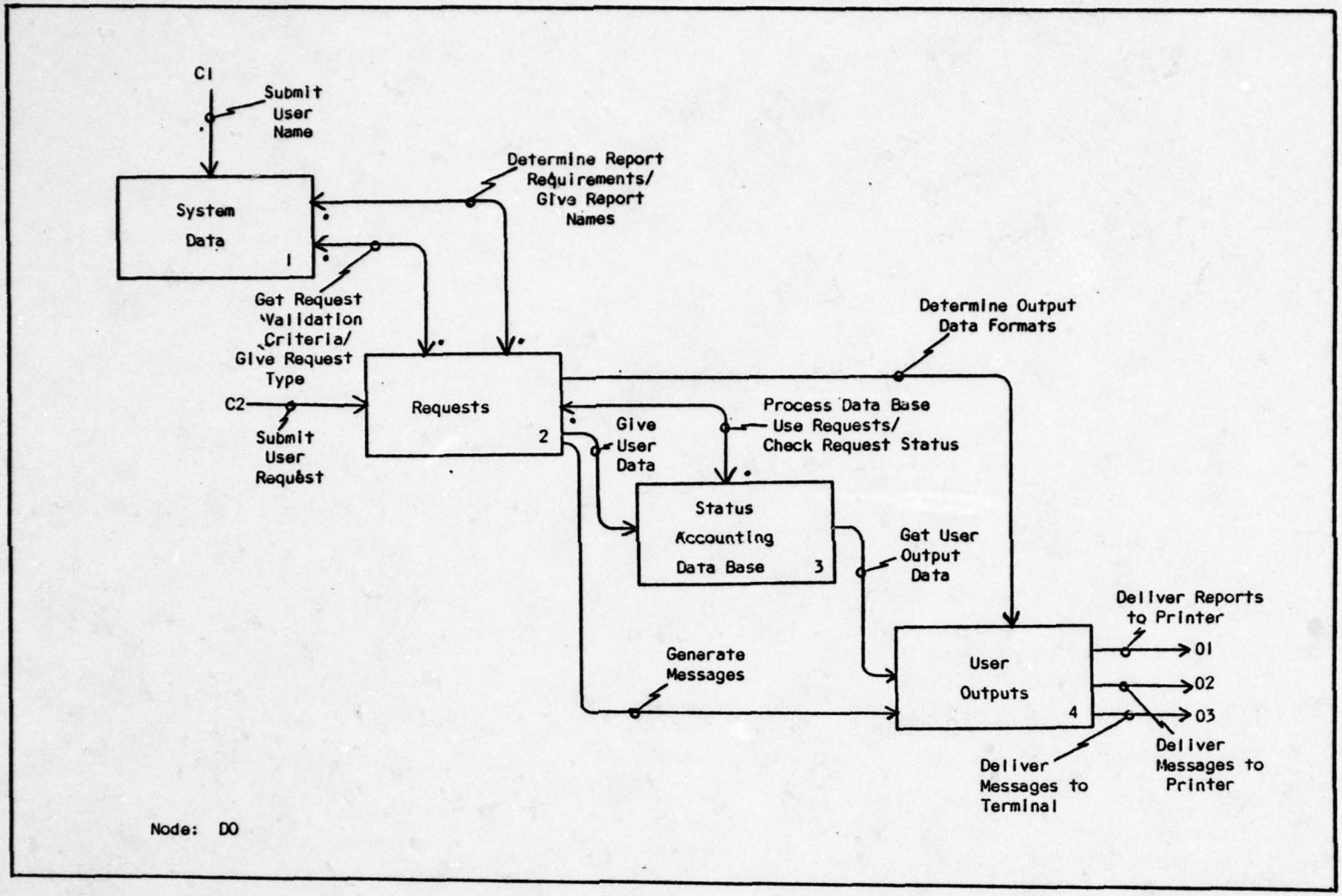


Figure 3-11 Software Status Accounting System Data

DQ Text

"System Data" (1) is information that is "hard-coded" into the SOFTSAS software. It provides formatting and data retrieval requirements (1Q1) for formal reports as needed and supplies the appropriate SOFTSAS request validation criteria (1Q2) for the user identified by (1C1).

"Requests" (2) consists of user requests and database requests. Activity (2I1) supplies user requests which are echoed back to the user by (2Q4), and are checked against validation criteria obtained by (2C1). Activity (2Q2) processes database requests and returns the status of the outcome. For data update requests, (2Q3) provides the new data to be stored in the database. For data retrieval requests and depending upon the type of request, (2Q1) uses the information supplied in the request or supplied by (2C2) to determine output formatting requirements. Any messages for the user caused by request processing are generated by (2Q4).

The "Status Accounting Database" (3) consists of a directory of data descriptions, a dictionary of data storage locations, and the status accounting data itself. The processing of database requests (3C1) controls whether data is to be stored (3I1) or retrieved (3Q1).

"User Outputs" (4) consist of reports and messages. Data obtained by (4I1) are formatted for output as determined by (4C1), while messages generated by (4I2) are out-

putted without change. All reports are delivered to the on-line printer (401). Messages can be delivered to the user's terminal (403) or the on-line printer (402).

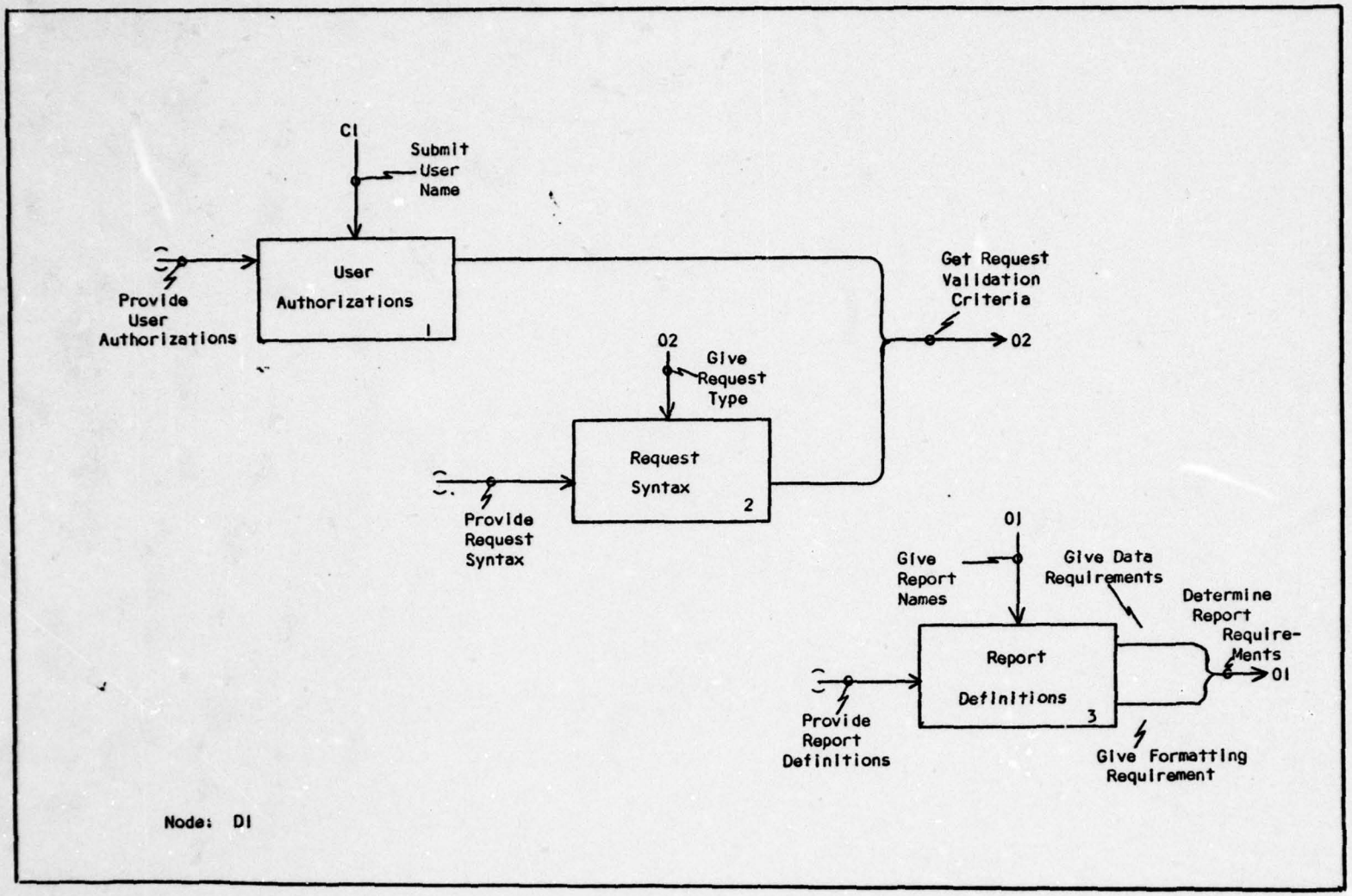


Figure 3-12 System Data

D1 Text

"User Authorizations" (1) is a list of the types of request that each user is authorized to submit (401). "Request Syntax" (2) are the legal syntax for each type of SOFTSAS request (201). Together, (101) and (201) deliver validation criteria for checking the input of the user identified by (1C1).

"Report Definitions" (3) consist of data and formatting requirements for SOFTSAS formal reports. Activity (3C1) names those requirements to be delivered by (301) for processing of PRODUCE requests.

The above data (1, 2, 3) are a predefined part of SOFTSAS.

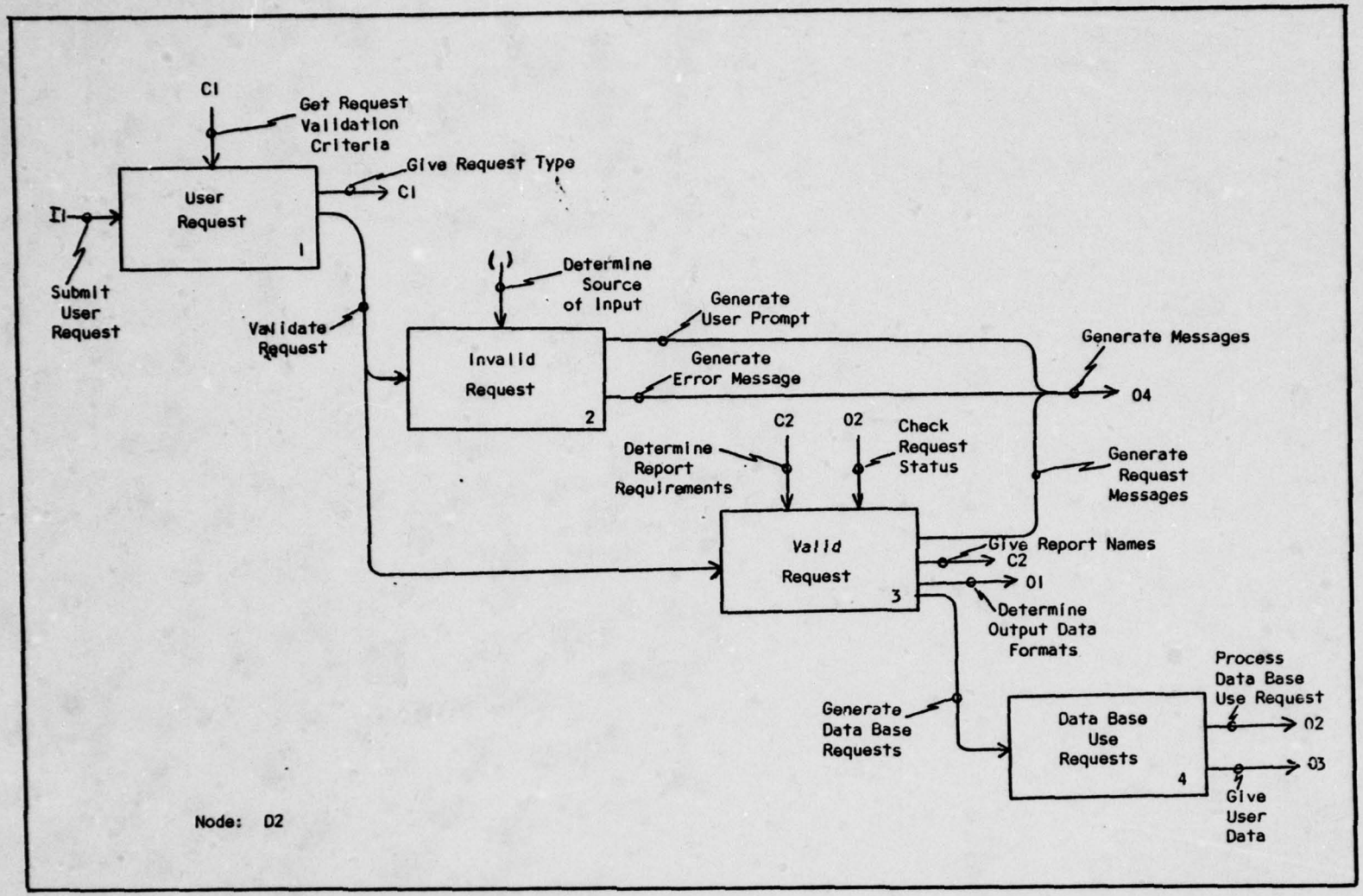


Figure 3-13 Requests

D2 Text

A "User Request" (1) is submitted (1I1) and then validated (1O1) against criteria obtained by (1C1). The validation process (2I1, 3I1) will produce either an invalid request (2) or a valid one (3). If a request is deemed invalid because it is incomplete and if the source of the input is determined by (2C1) to be terminal, a user prompt is generated (2O1) for any missing parameters. Otherwise, the request is rejected and the user is notified by (2O2).

For valid SOFTSAS requests (OPEN, CLOSE, CREATE, DELETE, MODIFY, PRINT, PRODUCE, AND DESCRIBE), appropriate database requests are generated (3O3) based upon the requirements specified in the request or, in the special case of the PRODUCE request, based upon requirements obtained by (3C1). Similarly, output data formats are determined (3O3) for data retrieval requests from requirements specified in the PRINT or DESCRIBE requests or, in the case of the PRODUCE request, from (3O1). Activity (3C2) controls the preparation of user messages (3O1) when a request has been handled.

"Database Use Requests" (4) are generated (4I1) from valid SOFTSAS requests and are accomplished by (4O1). In addition, new data is supplied to the database by (4O2).

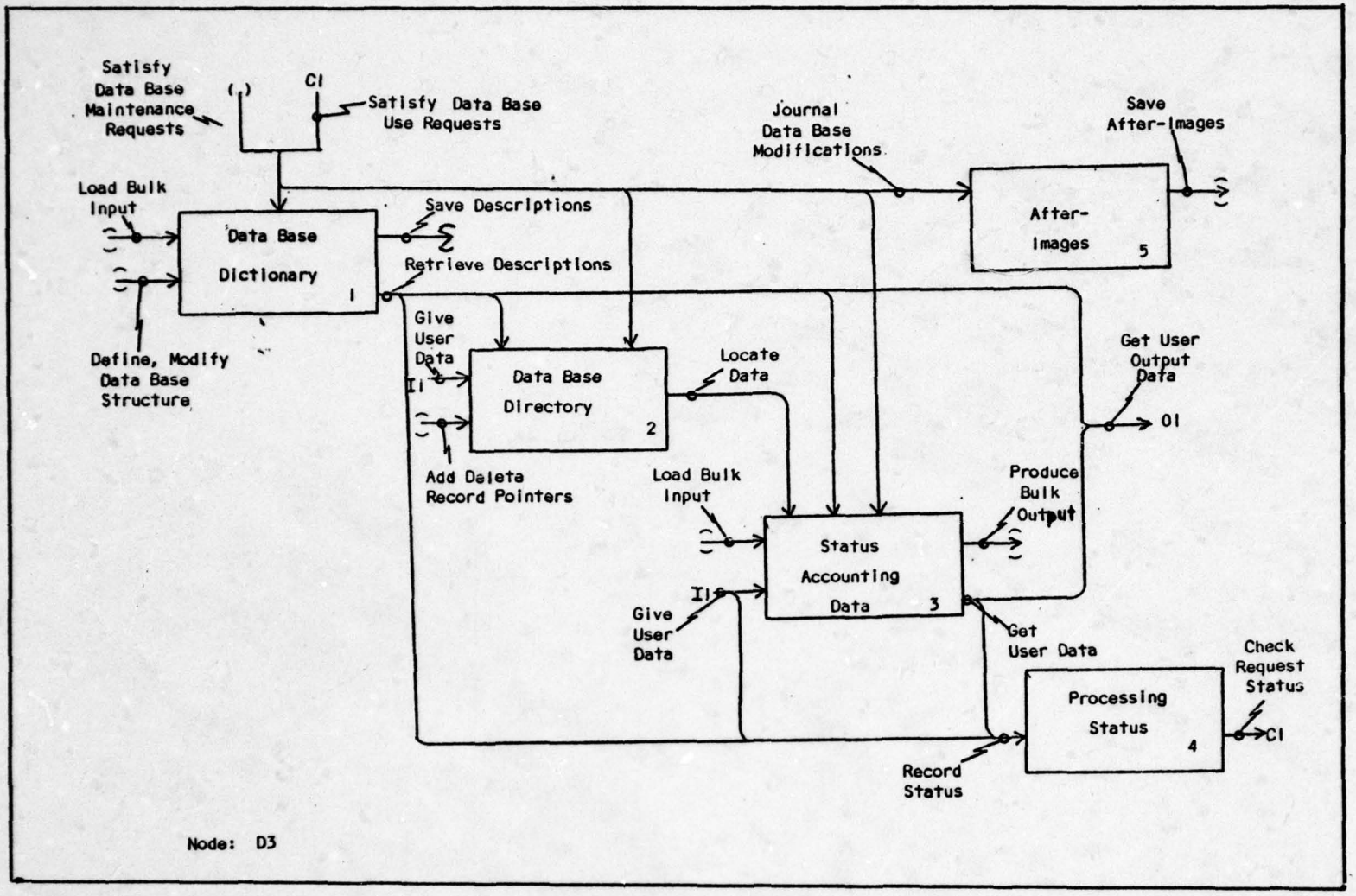


Figure 3-14 Status Accounting Database

D3 Text

All activity on the database (1, 2, 3) result from either database maintenance or database use requests. Database maintenance requests are processed for programs run by the Database Administrator. These requests direct that the dictionary (1) be created or modified (1I2), that an old database replace the current one (1I1, 3I1), or that a copy of the database be saved (1O1, 3O1). Database use requests are processed for user application programs. These requests control adding, modifying, deleting, and retrieving user data (3I2, 3O1) and retrieving data descriptions (1O2).

The "Database Director" (2) is modified by (2I2) when processing ADD-NEW and DELETE requests (2C2) and their values are obtained by (2I1). The directory is used by (2O1) to locate data when processing a FIND request (2C2).

Data storage locations and data formats obtained by (3C1) and (3C2), respectively, control (3I2) and (3O2) when processing ADD-NEW, DELETE, UPDATE, and GET-DATA requests (3C3).

"Processing Status" (4) is an indicator of the outcome status of a request. The status is recorded (4I1) for-checking (4O1) after completion of each request.

"After-images" (5) are modifications to the database that are recorded (5I1) and saved (5O1) to aid in database recovery (1I2, 3I2) after a failure.

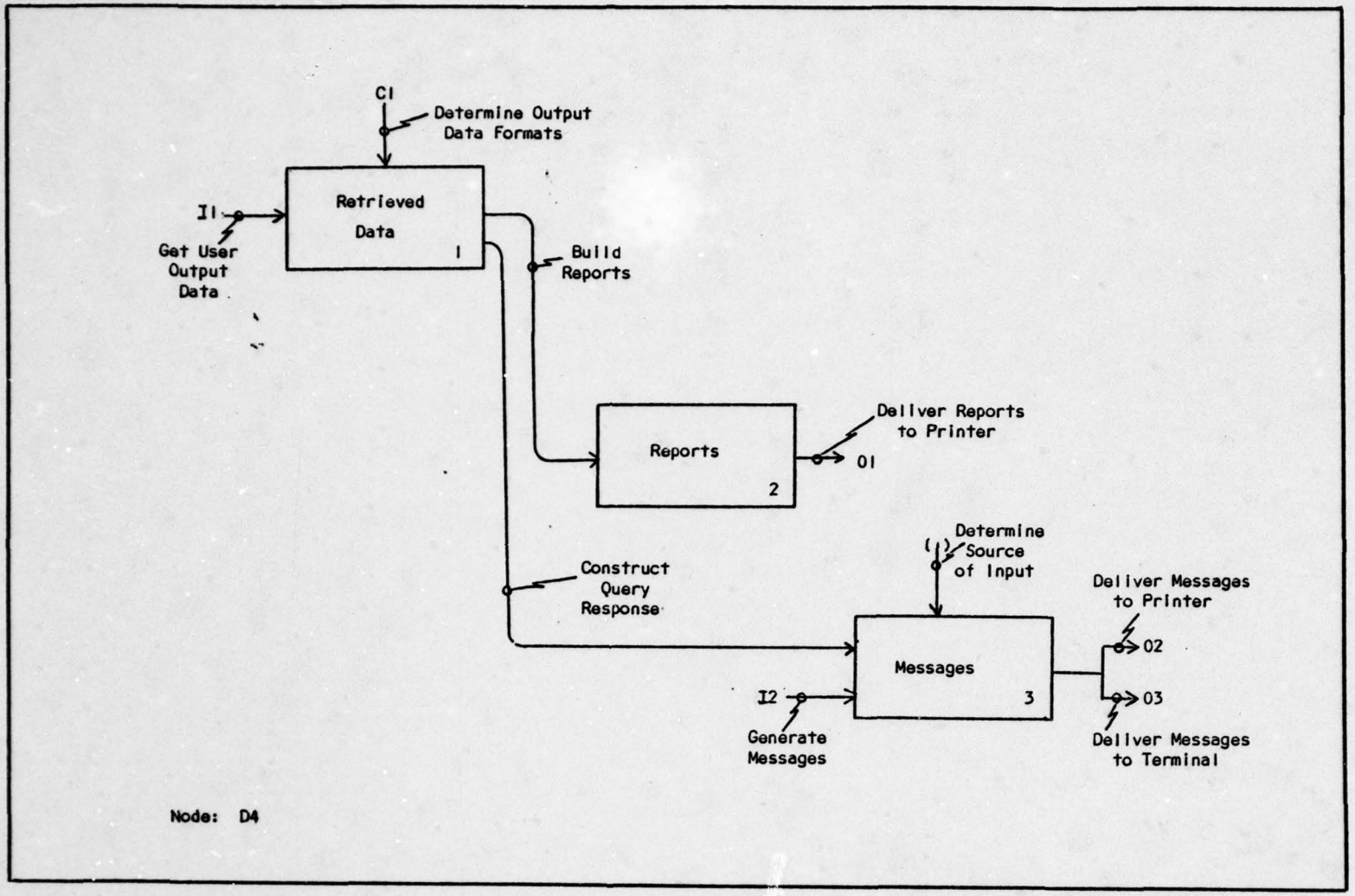


Figure 3-15 User Outputs

D4 Text

"Retrieved Data" (1) is obtained (1I1) from the database and is used (1O1, 1O2) to build reports and responses to ad hoc queries for data according to formatting requirements specified by (1C1). "Reports" (2) are constructed (2I1) and output to the on-line printer (2O1). "Messages" (3) consist of query responses, user prompts, error notifications, and copies of the input requests. Once constructed (3I1, 3I2), these messages are output (3O1) to the user. They are written to the user's terminal (O2) if (3C1) determines the inputs were received from a terminal. Otherwise, they are written to the on-line printer (O3).

Summary

This chapter presented the formal functional specification for SOFTSAS in the form of SADT activity and data models. These models were used to design a structure for the software which will satisfy the status accounting requirements for software configuration management. The structure that was designed is presented in Chapter IV.

IV Software Design Specification

Introduction

The third stage in the software development life-cycle consists of deriving an internal structural design for the software to satisfy the requirements specified during stage two. This design provides the missing link between the external specifications of the system and the internal algorithms which comprise individual program modules. It identifies the major components of the system, the functions that they perform, and the interfaces between them. The sections in this chapter present an overview of some fundamental software design considerations, a preliminary design specification for SOFTSAS, and a brief summary of observations that were made while developing the design.

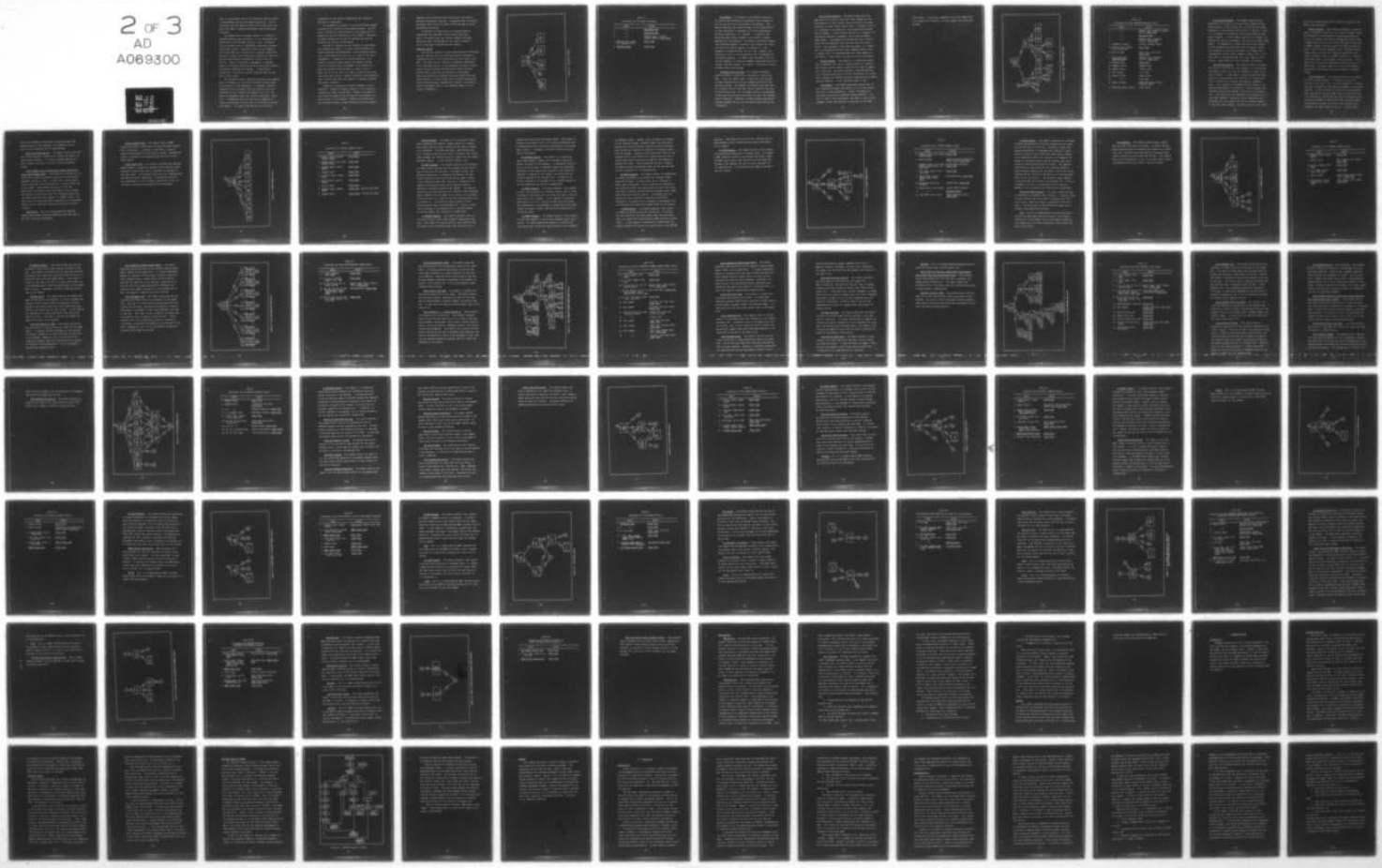
Software Design Considerations

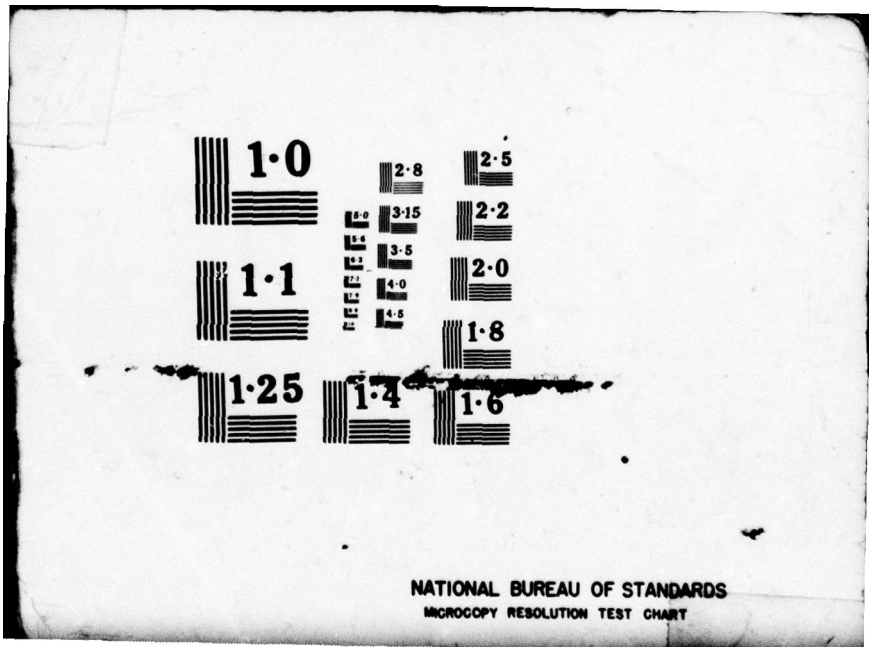
Design plays an important role in the development of software. Thayer et al (Ref 38:4,160-163) determined that, in one software project, 28.7% of the errors that were introduced as a result of coding changes and detected during subsystem/system testing could have been prevented had some sort of design standard been used. In other studies, Boehm (Ref 9) discovered that 64% of detected errors were due to design, and, in the NASA Apollo project, Hamilton and Zeldin (Ref 15) found that 73% of all errors were design errors. Moreover, Lehman (Ref 18) surveyed major aerospace firms that were managing 57 software projects and found

AD-A069 300 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN FOR AN AUTOMATED STATUS ACCOUNTING SYSTEM FOR SOFTWARE C--ETC(U)
MAR 79 A SCHUSTER

UNCLASSIFIED AFIT/GCS/EE/79-2 NL

2 OF 3
AD
A069300




1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
2·5
2·2
2·0
1·8
1·4
1·6
NATIONAL BUREAU OF STANDARDS
MICROCOPY RESOLUTION TEST CHART

that, on the average, 44% of all production time was spent on requirements analysis and design activities. This is consistent with the report by Weiss (Ref 39:26) which suggests that 40% of software development time be allocated to design.

The primary goal of software design is to develop a program structure that satisfies a set of requirements and constraints at the minimum life-cycle cost. Life-cycle costs incude the costs of developing, operating, and maintaining the system. Constantine (Ref 11:9-12) suggests that these costs are a function of a program's quality as measured in terms of its efficiency, reliability, maintainability, modifiability, generality, flexibility, and utility. Several "structural" approaches, or methodologies, for software design exist which show how to develop and portray "good quality" designs. In particular, Constantine's "Structured Design" technique (Ref 11) was used in this thesis.

The Structured Design technique recognizes that development of software is a non-trivial problem and that humans do not deal well with complexity. It requires iterative decompositions of software tasks into smaller sub-tasks (sometimes called "stepwise refinement") until the sub-tasks are manageably small and solvable separately. Moreover, it recommends that each sub-task be composed of a single, well-defined function that is performed by one program "module". As long as the modules are relatively

independent of one another, decomposing the system will decrease its complexity.

The decomposition process of Structured Design imposes a hierarchical structure on the program modules. The program's structure is characterized by the contents of its modules and by their dependency on one another. Measures of these characteristics such as cohesion and coupling determine the quality of the design.

Coupling is a measure of the strength of interconnection between two modules. Strongly coupled modules are highly connected (interdependent), while uncoupled modules have no interconnections and are, in a sense of the word, independent. An objective of Structured Design is to achieve a structure whose modules are loosely coupled. This will minimize what a programmer of one module will need to know about another module in order to use it. Myers (Ref 27:33-53) ranks six types of coupling from best (data-coupling) to worst (content-coupling), and Constantine (Ref 11:76-90) describes four aspects of computer programs that affect coupling.

Cohesion is a measure of modular strength, or functionality. A module is highly cohesive if its processing elements are highly interrelated. However, the relationship should be based on the structure of the problem, not the structure of the program. Software structures that group highly-related, problem-oriented processing elements

together tend to be more effectively modular and tend to minimize intermodular coupling. Constantine (Ref 11:96-126) describes seven levels of cohesion differentiated by seven associative principles.

In addition to these measures of program quality, Constantine also describes several design heuristics which, if properly used, may improve systems structures. These heuristics include module size, span of control, fan-in, and scope of effect/scope of control.

Structure Charts

This section contains a design specification for SOFTSAS that was derived using the Structured Design strategies of "transform analysis" and "transaction analysis" (Ref 11:171-222). The specification consists of graphical representations (structure charts) of the program's structural organization, tables of module interface definitions, and supporting text to describe the function of each module. These were derived by conceptualizing the formal functional specifications in terms of data flows to which Constantine's design guidelines were systematically applied. An explanation of the symbols used in the structure charts is contained in Appendix B.

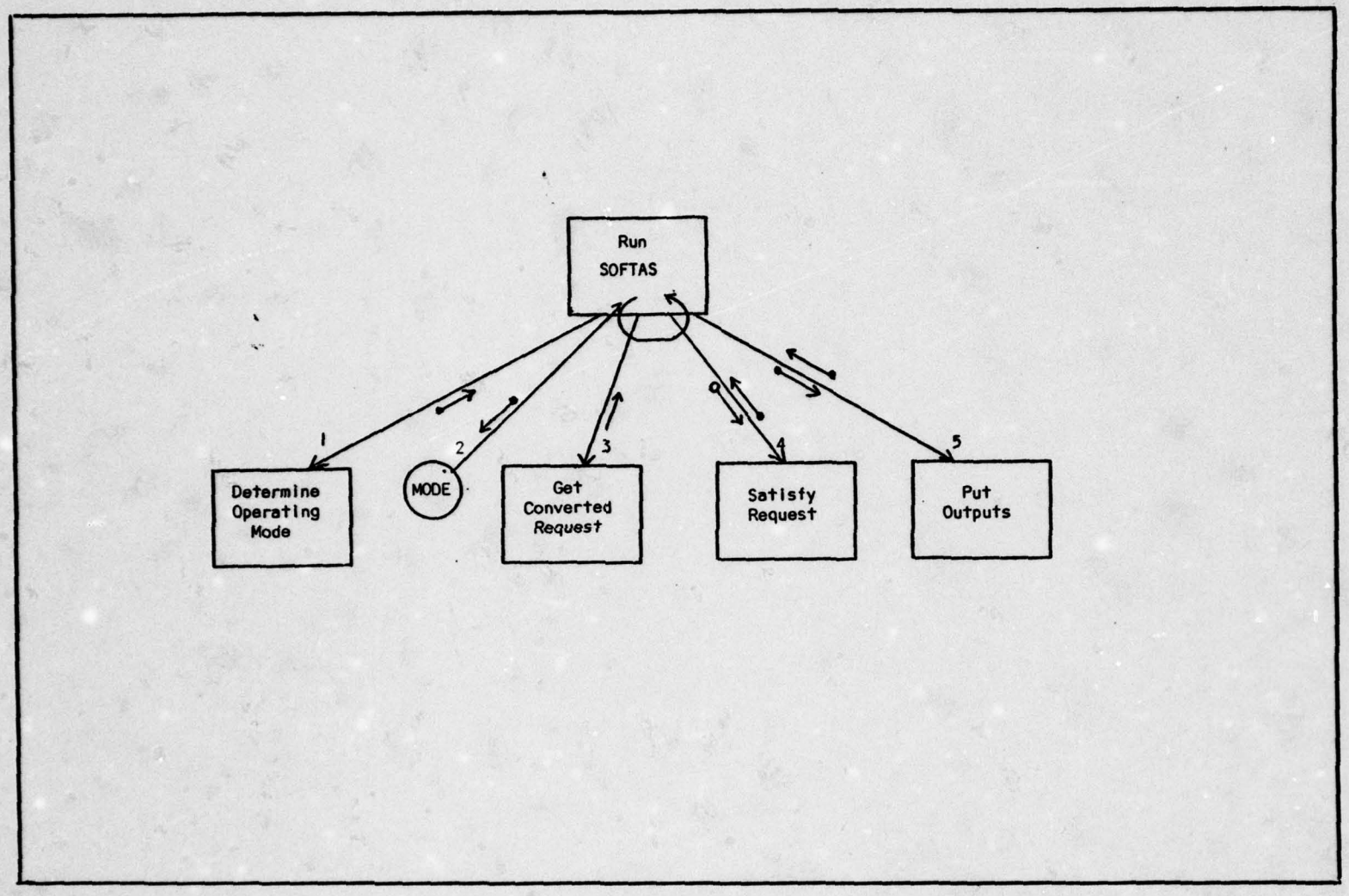


Figure 4-1 Top Level Structure

Table II

Interfaces for Top Level Structure

	INPUT	OUTPUT
1	-----	Operating mode
2	-----	Operating mode
3	-----	Request type, internal request tables, error flag
4	Request type, internal request tables	Error flag
5	Operating mode	Error flag

Run SOFTSAS. This module is the program supervisor. It controls the reading and processing of user requests as well as the writing of user reports and messages. The module determines the operating mode of the program which, in turn, determines if messages are to be accumulated or delivered immediately to a terminal. In addition, the operating mode dictates whether or not certain "prompt" messages will be generated. The module alternately invokes 'Get Converted Request' to obtain a user request (in internal form) and 'Satisfy Request' to process it. This sequence is iterated either until an 'End' request is encountered or until a non-recoverable error is detected during batch processing. In either case, the module calls on 'Satisfy Request' to close any database realms that may still be open, and then invokes 'Put Outputs' to deliver all accumulated reports and messages.

Determine Operating Mode. This module determines whether SOFTSAS is executing in the batch or interactive mode. There are several ways in which this can be accomplished, depending upon the user support provided by the operating system. For example, the module could query the file control block of the input file to ascertain the type of device being used. Or, an unused switch in the program status word could be set by the user to signal the intended mode of operation. Moreover, a simple question-and-answer sequence between the user and the module could provide the information.

Get Converted Request. This module supervises the acquisition of a single, valid user input request and the transformation of that request from its English-like representation to a more usable tabular form. A request is valid if it conforms to the syntactic rules established for that type of request. A valid request consists of a command, its required and optional parameters, and an end-of-request (EOR) symbol. The command must precede the rest of the request, and the parameters may span more than one logical record. Each parameter clause that can appear in a request has a corresponding internal table (command, search, sort, or update) that is constructed when the clause appears in the request. These tables are used to process the request.

Satisfy Request. This module is a transaction center which dispatches the internal request tables to a subordinate request handler module, based upon the type of request to be processed. The module performs a coordinating function, relaying processing and status information (via "error flag") between the 'SOFTSAS Executive' module and the request processor modules. It also monitors realms in use.

Put Outputs. This module supervises the delivery of any accumulated messages and reports to an on-line printer just prior to job termination. Output line images are collected in two files by SOFTSAS request handlers. One file is used to hold the reports; the other is used to hold messages, except when SOFTSAS is executing in the inter-

active mode. In this case, messages are written immediately to the user at his terminal, and only reports are put to the printer.

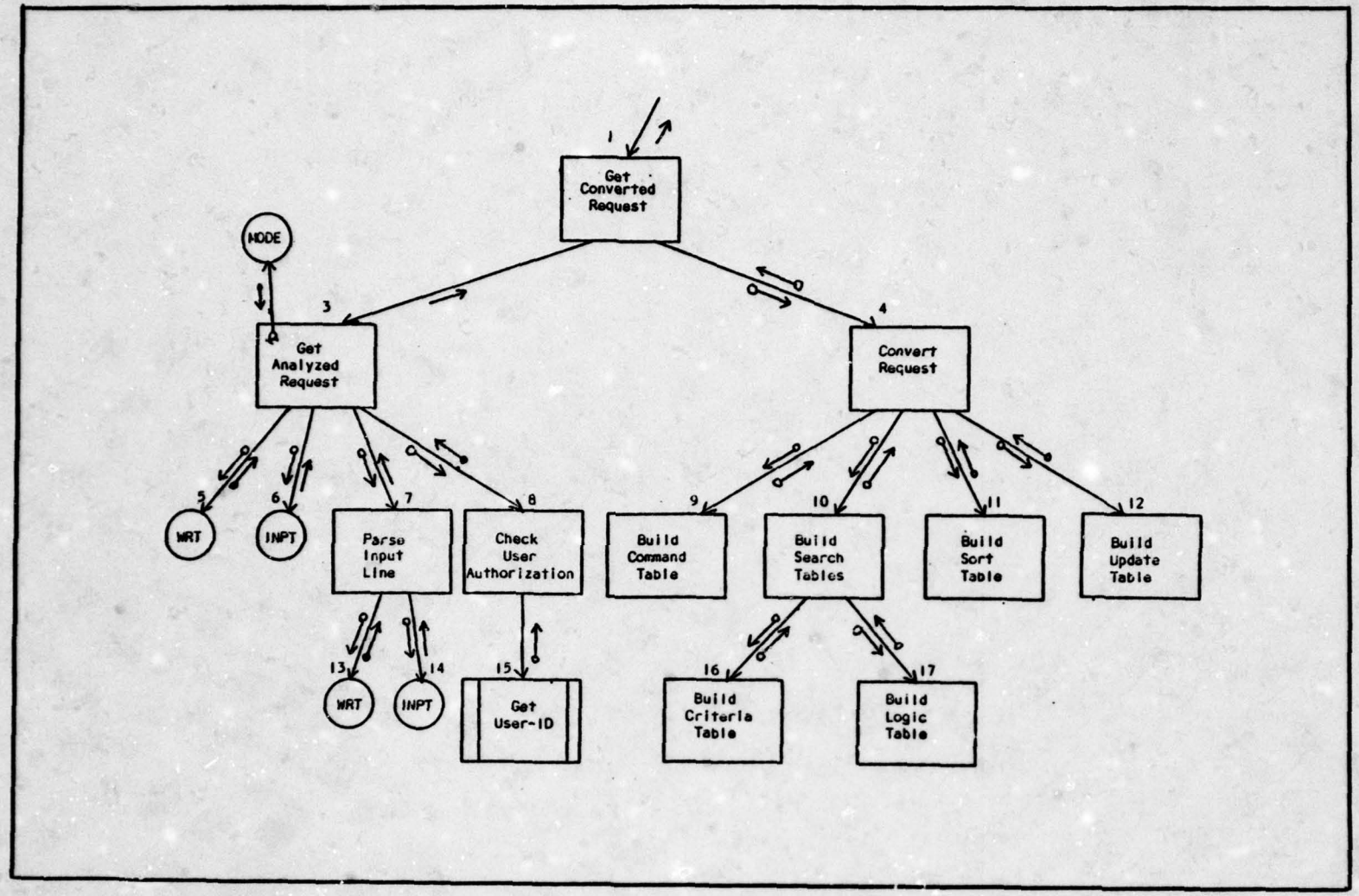


Figure 4-2 GET CONVERTED REQUEST Branch

Table III

Interfaces for GET CONVERTED REQUEST Branch

	INPUT	OUTPUT
1	-----	Request type, internal request tables, error flag
2	-----	Operating mode
3	-----	Request type, parameter clauses, error flag
4	Parameter clauses	Internal request tables
5, 13	File name, output line, line size	Error flag
6, 14	File name	Input line, line size, error flag
7	User input line, operating mode	Request type, parameter clauses, error flag
8	Request type	Error flag
9	Command clause	Command table
10	Search clause	Search tables
11	Sort clause	Sort table
12	Update Clause	Update table
15	-----	User name
16	Search clause	Modified search clause, criteria table
17	Modified search clause	Logic table

Get Converted Request. This module supervises the acquisition of a single, valid user input request and the transformation of that request from its English-like representation to a more usable tabular form. A request is valid if it conforms to the syntactic rules established for that type of request. A valid request consists of a command, its required and optional parameters, and an end-of-request (EOR) symbol. The command must precede the rest of the request, and the parameters may span more than one logical record. Each parameter clause that can appear in a request has a corresponding internal table (command, search, sort, or update) that is constructed when the clause appears in the request. These tables are used to process the request.

Get Analyzed Request. This module controls the input and syntactic analysis of a user request. It first writes a message to advise the user that SOFTSAS is ready to process a request. It then reads a logical record from input, echoes it to the message file, and calls on 'Parse Input Line' to identify and validate the request's component parts. Valid requests are accepted and are passed to the superordinate module for processing. Invalid requests are rejected, and an error message is written to the message file. User errors that are detected by this module are nonfatal when SOFTSAS is executing in the interactive mode; the module will just prompt the user for another request. In the batch mode, however, the module sets an error indica-

tor for its superordinate which ultimately results in premature job termination.

Convert Request. Upon accepting parameter clauses as input, this module "zeroes out" all internal request tables and dispatches to the appropriate subordinate modules to convert the clauses into their corresponding tabular forms. A command table will be used to hold the report names from a PRODUCE request, the realm names from OPEN and CLOSE requests, and all of the parameters from a DESCRIBE request. Search tables will hold any relational expressions and logical operators found in a WHERE clause and the names of the record (type) to be accessed, when applicable. A sort table will hold the field names and qualifiers found in a SORT clause. An update table will hold the equivalence expressions (field names and values) found in a SETTING clause.

Parse Input Line. This module contains the validation criteria for each request. It accepts and lexically analyzes a user input line. For valid requests, it returns the type of request and the request parameters to the superordinate. If the line does not contain an end-of-request mark, the module obtains another logical record from input. If an EOR mark is encountered when required parameters are missing and SOFTSAS is executing in the interactive mode, the module will prompt the user for the required input parameter and then attempt to read it. If the attempt

fails or if SOFTSAS is executing in the batch mode, the module writes an error message to the message file and sets an error indicator for its superordinate.

Check User Authorization. This module insures that the user of the program is authorized to submit the request currently being analyzed. It invokes a system-supplied routine to obtain the name of the user from the operating system.

Build Command Table, Build Search Tables, Build Sort Table, Build Update Table. These modules are called to construct their respective tables, given their corresponding parameter clauses as input. In particular, 'Build Search Tables' supervises the construction of a criteria table and a logic table. These tables are used to "find" the identifiers of database record occurrences to be accessed. Each entry in the criteria table will completely describe a relational expression that appears in a WHERE clause. The logic table will describe logical operations to be performed on the lists of identifiers obtained using the relational criteria.

Get User-ID. This is a system-supplied interface module which obtains from the operating system the name of the user executing the program.

Build Criteria Table. This module scans a WHERE clause and creates a table entry for each relational expression it encounters. A pointer to that entry is used to replace the expression in the original clause. The modified clause is returned to the superordinate with the criteria table entries.

Build Logic Table. This module transforms the modified WHERE clause to its postfix (reverse Polish notation) representation, stacking the logical operators and operands in the logic table. The entries of the table will unambiguously describe the logical operations - intersection and union - to be performed on lists of record identifiers, as well as the order in which the operations will be performed.

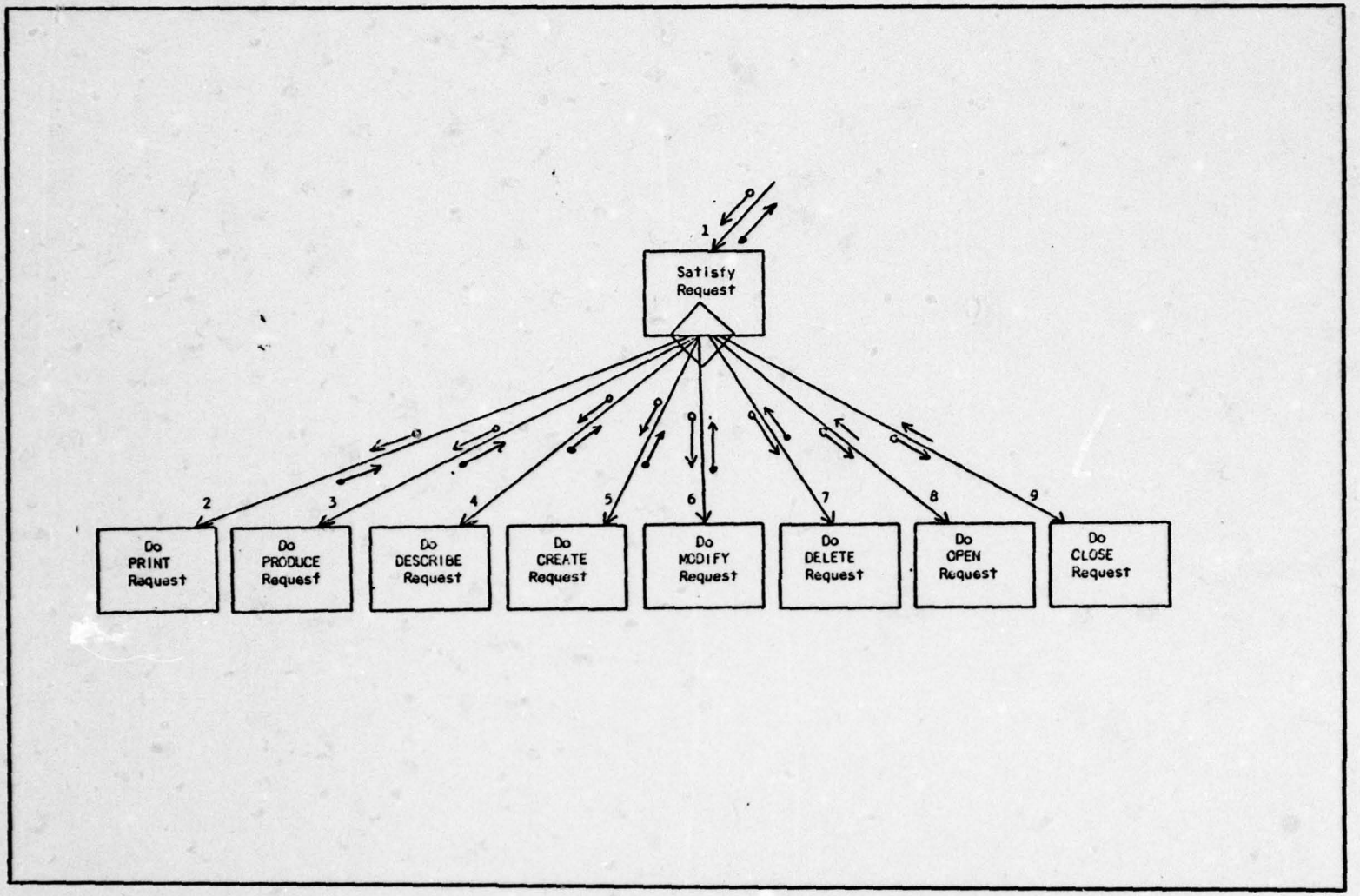


Figure 4-3 SATISFY REQUEST Branch

Table IV

Interfaces for SATISFY REQUEST Branch

	INPUT	OUTPUT
1	Request type, internal request tables	Error flag
2	Command table, sort table search tables	Error flag
3	Command table, search tables	Error flag
4	Command table	Error flag
5	Criteria table, update table	Error flag
6	Update table, search tables	Error flag
7	Search tables	Error flag
8	Command table, update table	Error flag, opened realm name
9	Command table	Error flag, closed realm name

Satisfy Request. This module is a transaction center which dispatches the internal request tables to a subordinate request handler module, based upon the type of request to be processed. The module performs a coordinating function, relaying processing and status information (via "error flag") between the 'SOFTSAS Executive' module and the request processor modules. It also monitors realms in use.

Do PRINT Request. This module supervises the retrieval of status accounting data from the database and the formatting of this data for printing. The module invokes 'Get Approved Identifiers' to determine the identity of those record occurrences that satisfy the search table criteria. In the interactive mode only, the user is informed of the number of occurrences found, and his approval must be obtained to continue processing the request. The approved identifiers are then passed to the "Construct PRINT Response' module together with the names of the fields to be accessed and the criteria to be used to sort the retrieved data prior to printing it. The resultant query response is then obtained and written to the message file. Finally, a message is written to the message file to indicate whether or not the request was successfully accomplished.

Do PRODUCE Request. This module supervises the construction of one or more formal reqports requested by the user. Each report can contain data for more than one CPCI. The module first identifies those CPCIs which are to be

reported and then builds the named reports. Each report is written to the report file as it is being constructed, and a nonrecoverable error terminates processing of the request when it occurs. A message is written indicating the outcome of the request.

Do DESCRIBE Request. This module is a transaction center which handles requests for information about the logical structure of the database. It determines which of the six possible types of DESCRIBE request was submitted, dispatches to the appropriate transaction level module to construct a response to the request, and writes the constructed response to the message file. In addition, it writes a message to the message file to indicate whether or not the request was successfully accomplished.

Do CREATE Request. This module handles a user request to add a new occurrence to a database record, and it writes a message to the message file to inform the user of the outcome status of his request. If the request is processed successfully, and if it reflects either a new ECP having been submitted or a new document or change notice have been issued, the module invokes 'Add Active-item Occurrence' to record the event.

Do MODIFY Request. This module handles a user request to alter the contents of one or more database record occurrences. It invokes subordinate modules to identify the occurrences that satisfy the search criteria and to perform

the database update. However, prior to making any changes in the interactive mode, the module warns the user with a message which states how many qualified record occurrences were located and requests his approval to continue processing the request. Changing certain monitored fields in the database (such as ECP priority or status) requires that an active-item record occurrence be added to the database to register the event. After processing the user's request, the module reports the outcome status to the user and sets an error indicator for its superordinate.

Do DELETE Request. This module obtains the identifiers of record occurrences to be deleted and invokes 'Remove Record Occurrences' to delete them, and all pointers to them, from the database. Prior to making the deletions in the interactive mode, the module warns the user with a message which indicates how many record occurrences were found that satisfy the search criteria. The message also requests the user's approval to perform the deletions. Afterwards, the module writes a message to the message file to inform the user about the outcome status of his request.

Do OPEN Request. This module handles a user request to prepare a database realm for processing. It extracts the user identifier and realm access mode from the update table and invokes the system-supplied DBMS interface routine 'Open'. If the request is successfully accomplished, the module transmits the name of the opened realm to the SOFTSAS

executive. The module will set an error indicator for its superordinate and will notify the user about the outcome of his request.

Do CLOSE Request. This module handles a user request to discontinue processing of a database realm. It invokes a DBMS interface routine to de-allocate the realm, checks the DBMS status return code, notifies the user about the outcome of the request, and sets an error indicator for its superordinate. It also returns the name of the realm that was closed.

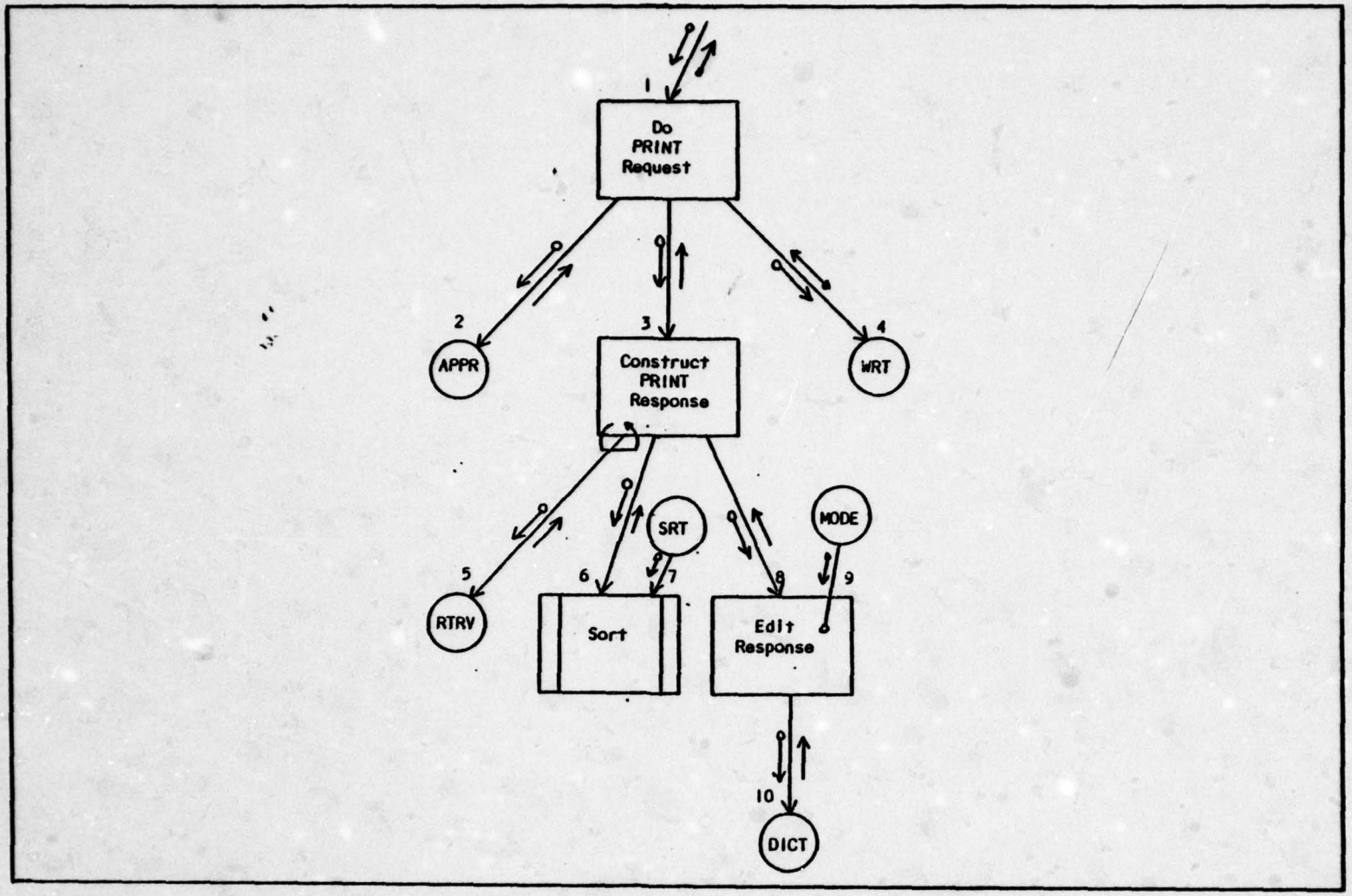


Figure 4-4 DO PRINT REQUEST Branch

Table V

Interfaces for DO PRINT REQUEST Branch

	INPUT	OUTPUT
1	Command table, sort table, search tables	Error flag
2	Search tables	Approved record identifiers, record name, error flag
3	Command table, sort table, record identifiiers, record name	Edited PRINT response, error flag
4	File name, output line, line size	Error flag
5	Record name, record identifiers, field names	Retrieved data, error flag
6,7	Retrieved data, sort criteria	Sorted data, error flag
8	Sorted data, field names	Edited PRINT response
9	-----	Operating mode
10	Item name, action code	Data item description, error flag

Do PRINT Request. This module supervises the retrieval of status accounting data from the database and the formatting of this data for printing. The module invokes 'Get Approved Identifiers' to determine the identity of those record occurrences that satisfy the search table criteria. In the interactive mode only, the user is informed of the number of occurrences found, and his approval must be obtained to continue processing the request. The approved identifiers are then passed to the "Construct PRINT Response' module together with the names of the fields to be accessed and the criteria to be used to sort the retrieved data prior to printing it. The resultant query response is then obtained and written to the message file. Finally, a message is written to the message file to indicate whether or not the request was successfully accomplished.

Construct PRINT Response. This module supervises the construction of a database query response. It obtains items of data from each qualified record occurrence and, if no error occurs, orders this data as specified in the sort table. The sorted data is then edited for output and is passed to the superordinate module.

Sort. This is a system-supplied routine which sorts tabled data based upon specific criteria. The criteria consists of a prioritized list of field names on which to sort (keys) and a sort mode (ascending or descending) for each of the fields.

Edit Response. This module accepts sorted, tabled data and edits each entry for output. It obtains the program's operating mode to determine the length for an output line, and it obtains field descriptions to determine the format in which to present each item of data. The items of data will be presented in a line in the same order as their corresponding fields were named in the user request.

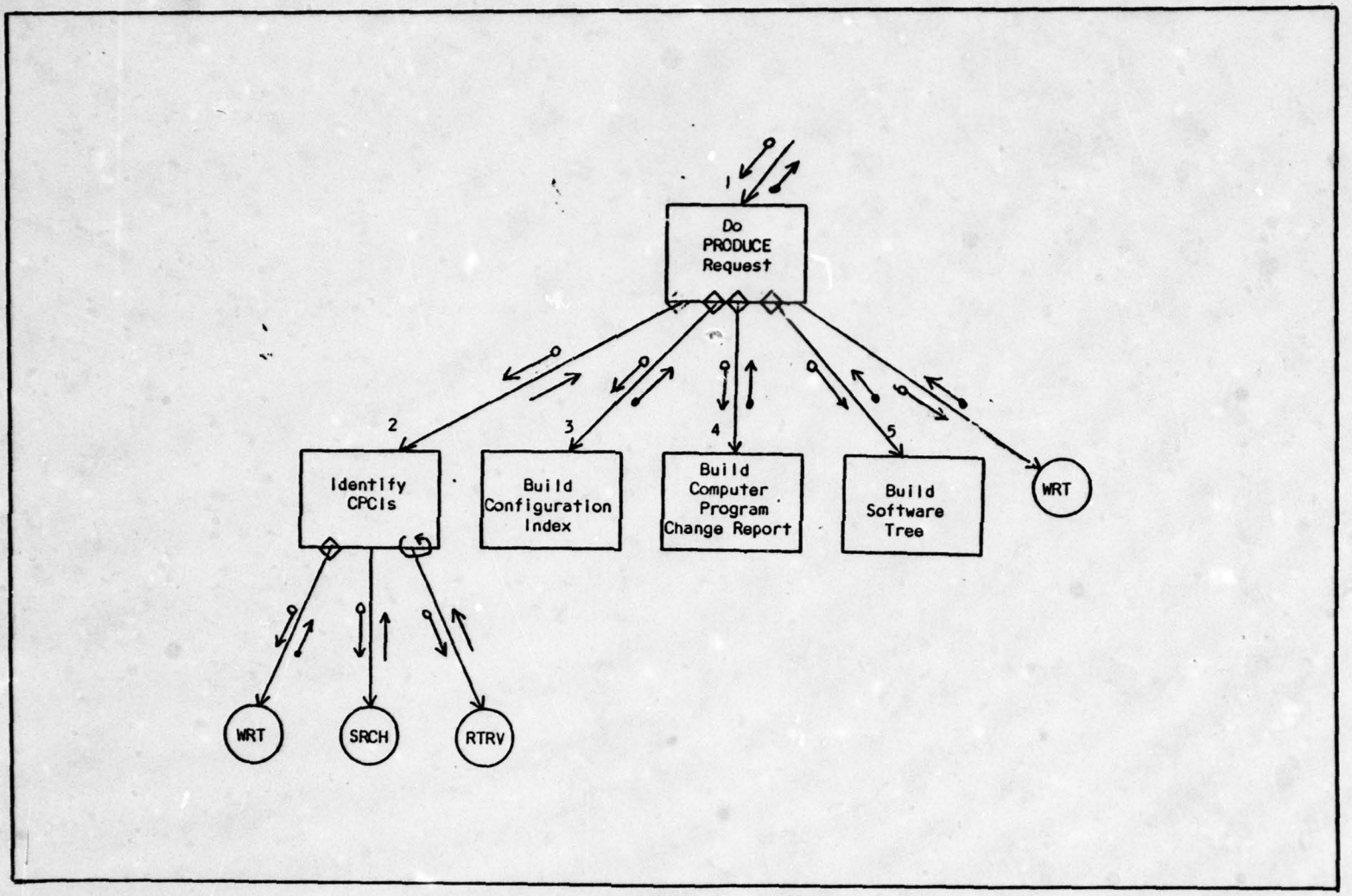


Figure 4-5 DO PRODUCE REQUEST Branch

Table VI

Interfaces for DO PRODUCE REQUEST Branch

	INPUT	OUTPUT
1	Command table, search tables	Error flag
2	Search tables	CPCI numbers, CPCI names, error flag
3,4,5	CPCI numbers, CPCI names	Error flag
6,7	File name, output line, line size	Error flag
8	Search tables	Record name, record identifiers, error flag
9	Record name, record identifiers, field names	CPCI numbers, CPCI names, error flag

Do PRODUCE Request. This module supervises the construction of one or more formal reports requested by the user. Each report can contain data for more than one CPCI. The module first identifies those CPCIs which are to be reported and then builds the named reports. Each report is written to the report file as it is being constructed, and a nonrecoverable error terminates processing of the request when it occurs. A message is written indicating the outcome of the request.

Identify CPCIs. This module obtains the numbers and names of the CPCIs to be reported. It first validates the contents of the criteria table, and generates an error message if the table contains any field names other than those of the primary key for CPCI records. It then obtains the identifiers of record occurrences that satisfy the search criteria, using them to retrieve the numbers and names of the CPCIs to be reported.

Build Configuration Index. This module supervises the construction of the Configuration Index for the named CPCIs. It invokes subordinate modules to construct one title page followed by six report sections for each CPCI to be reported. Detection of an error by any of the subordinate modules terminates processing of the request and results in this module and, in turn, this module's superordinate being notified.

Build Computer Program Change Report. This module supervises the construction of the Computer Program Change Report (CPCR) for the named CPCIs. It invokes subordinate modules to construct one title page followed by three sections (change history, change status, and red flag and activity) for each CPCI to be reported. Detection of an error by any of the subordinate modules terminates processing of the request and results in this module and, in turn, this module's superordinate being notified.

Build Software Tree. This module supervises the construction of a family tree for the modules of each named CPCI. The tree is a hierarchical data structure which describes the functional organization of the CPCI's modules. Each module is assigned a unique seven-character tree code by the user. This code is used to identify the module and the functions that it supports within the CPCI. 'Build Software Tree' calls subordinate modules to construct a title page, a hierarchical breakdown of the function of the CPCI's components, and a list of the modules (grouped by function) which comprise the CPCI.

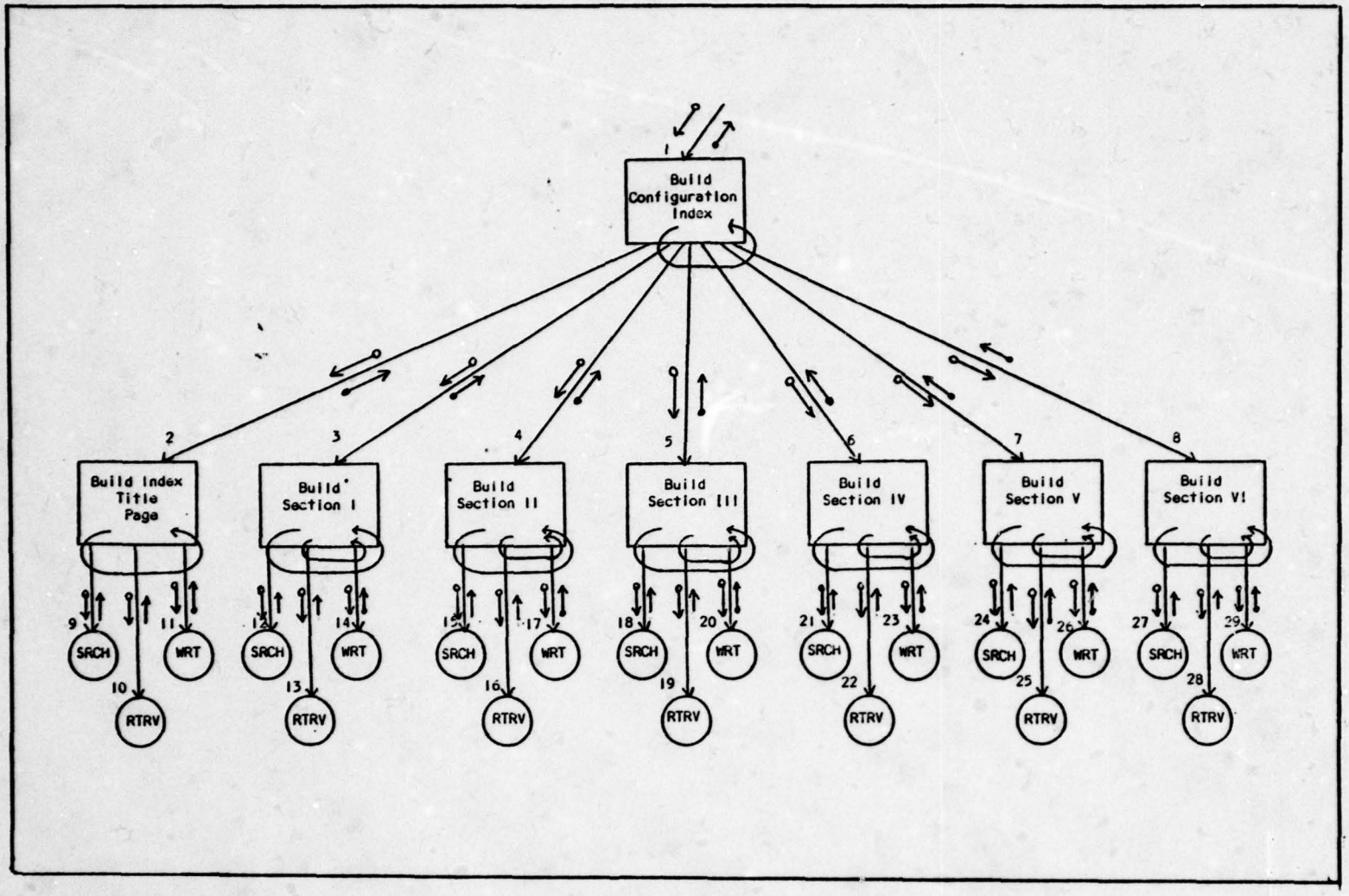


Figure 4-6 BUILD CONFIGURATION INDEX Branch

Table VII

Interfaces for BUILD CONFIGURATION INDEX Branch

INPUT	OUTPUT
1, 2 CPCI numbers, CPCI names	Error flag
3, 4, 5, 6, 7, 8 CPCI number, CPCI name	Error flag
9, 12, 15, 18, 21, 24, 27 Search tables	Record name, record identifiers, error flag
10, 13, 16, 19, 22, 25, 28 Record name, record identifiers, field names	Retrieved data, error flag
11, 14, 17, 20, 23, 26, 29 File name, output line, line size	Error flag

Build Configuration Index. This module supervises the construction of the Configuration Index for the named CPCIs. It invokes subordinate modules to construct one title page followed by six report sections for each CPCI to be reported. Detection of an error by any of the subordinate modules terminates processing of the request and results in this module and, in turn, this module's superordinate being notified.

Build Index Title Page. This module is invoked once during the construction of the Index. The module locates and retrieves document identification data for the Index such as issuing agency, document number, and contract number. It then edits this data and writes output lines to the report file.

Build Section I,..., Build Section VI. These modules are invoked once for each CPCI. They produce formatted status accounting data about each of six types of maintainable documents (development specification, produce specification, test plans/reports, handbooks, manuals, and version description documents). Each section will consist of information identjfying the type of document being reported, the current configuration of each of the documents of that type, and any approved engineering changes that will affect the documents in the future.

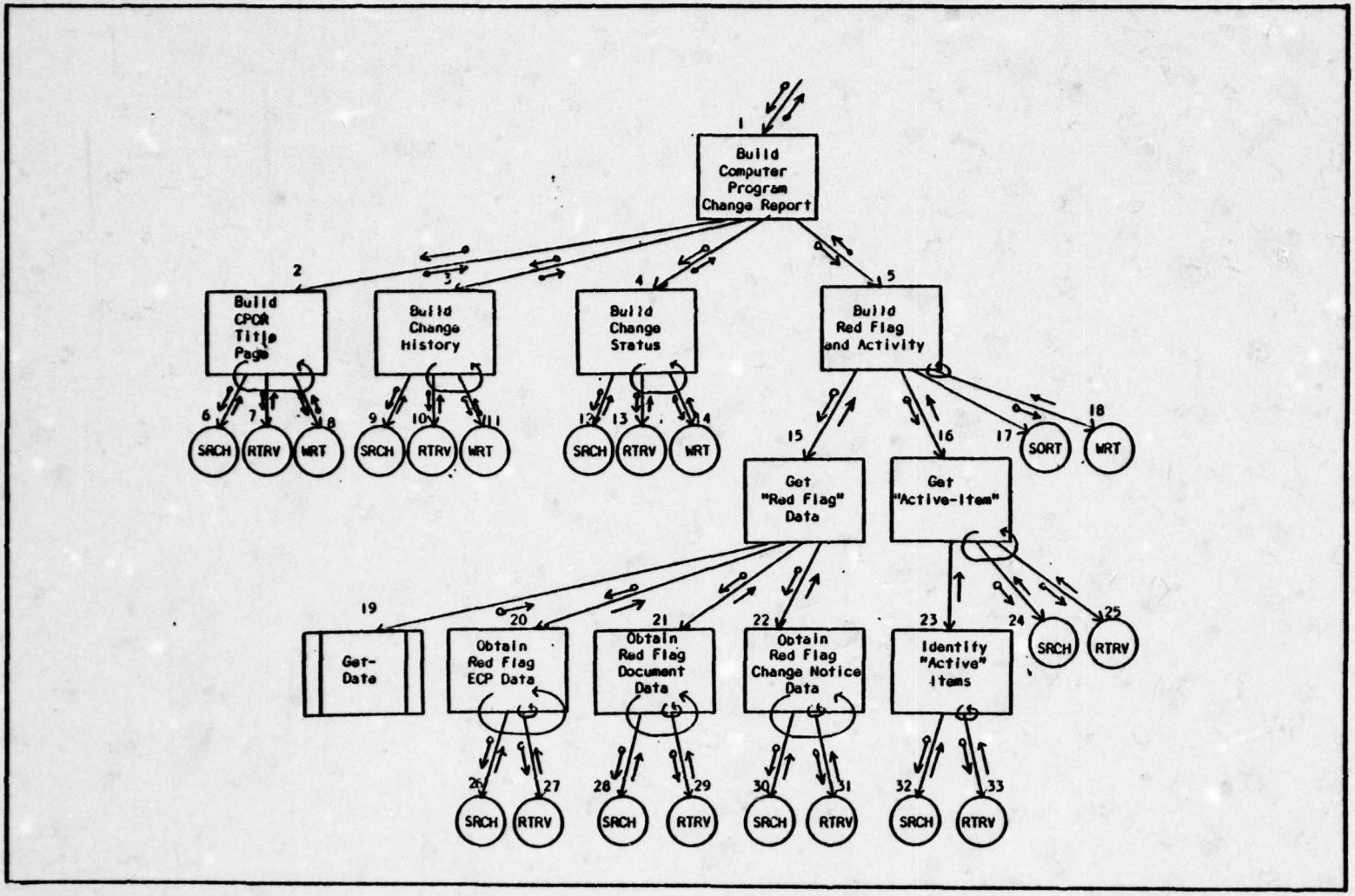


Figure 4-7 BUILD COMPUTER PROGRAM CHANGE REPORT Branch

Table VIII

Interfaces for BUILD COMPUTER PROGRAM CHANGE REPORT Branch

INPUT	OUTPUT
1, 2 CPCI numbers, CPCI names	Error flag
3, 4, 5 CPCI number, CPCI name	Error flag
6,9, 12, 24, 26, 28, 30, 32 Search tables	Record name, record identifiers, error flag
7, 10, 13, 25, 27, 29, 31, 33 Record name, record identifiers, field names	Retrieved data, error flag
8, 11, 14 File name, output line, line size	Error flag
15 CPCI number	Unsorted "red flag" data, error flag
16 CPCI number	Unsorted "activity" data, error flag
17 Unsorted "red flag" and "activity" data	Sorted "red flag" and "activity" data
19 -----	Current date
20 CPCI number	"Red flag" ECP data, error flag
21 CPCI number	"Red flag" document data, error flag
22 CPCI number	"Red flag" change notice data, error flag
23 -----	"Active" item identifiers, error flag

Build Computer Program Change Report. This module supervises the construction of the Computer Program Change Report (CPCR) for the named CPCIs. It invokes subordinate modules to construct one title page followed by three sections (change history, change status, and red flag and activity) for each CPCI to be reported. Detection of an error by any of the subordinate modules terminates processing of the request and results in this module and, in turn, this module's superordinate being notified.

Build CPCR Title Page. This module is invoked once during the construction of the CPCR. It locates and retrieves document identification data for the CPCR such as issuing agency, document number, and contract number. It then edits this data and writes output lines to the report file.

Build Change History. This module finds all the ECP records in the database that have affected, or will affect, the named CPCI. Information such as the number and title of the ECPs, their current status and date of status, and the contract numbers under which they were approved is obtained and is written to the report file.

Build Change Status. This module finds all the currently active ECPs in the database that affect the named CPCI. Active ECPs are those which have been approved but have not been completed. Extensive information concerning the identification and the development schedule of each

qualified engineering change, together with lists of modules, documents, hardware, and other CPCIs affected by the change, are retrieved from the database and written to the report file.

Build Red Flag and Activity. This module constructs the third section of the CPCR for each named CPCI. A table of data for "red flag" and "active" items is formed, sorted by priority, edited for output, and written to the report file. A red flag item is an ECP, document, or change notice that has missed a suspense date, will miss a suspense date within 30 days, or has not had a suspense date scheduled for a milestone. An active item is one that has undergone certain specific changes within the current reporting period.

Get Red Flag Data. This module supervises the identification of red flag items and the retrieval of red flag data for reporting. It invokes subordinates to compare the current date against scheduled (or non-scheduled) ECP, document, and change notice milestone dates. The number, title, priority and remarks of each red flag item is obtained and passed to the module's superordinate.

Get "Active-item" Data. This module invokes a subordinate to find data items that have been "active" during the reporting period. It then retrieves the number, title, priority and remarks of each found item and passes this information to its superordinate for reporting.

Get-Date. This is a system-supplied module which provides the current date from the system clock.

Obtain Red Flag ECP Data, Obtain Red Flag Document Data, Obtain Red Flag Change Notice Data. These modules search the database for "red flag" ECPs, documents, and change notices respectively. The names, numbers and priorities of all "red flag" items are retrieved and passed to these modules' superordinate.

Identify "Active" Items. This module obtains the contents of all the "active-item" record occurrences in the database. The data in these occurrences uniquely identify other record occurrences which have been designated "active" (see 'Add Active-Item Occurrence').

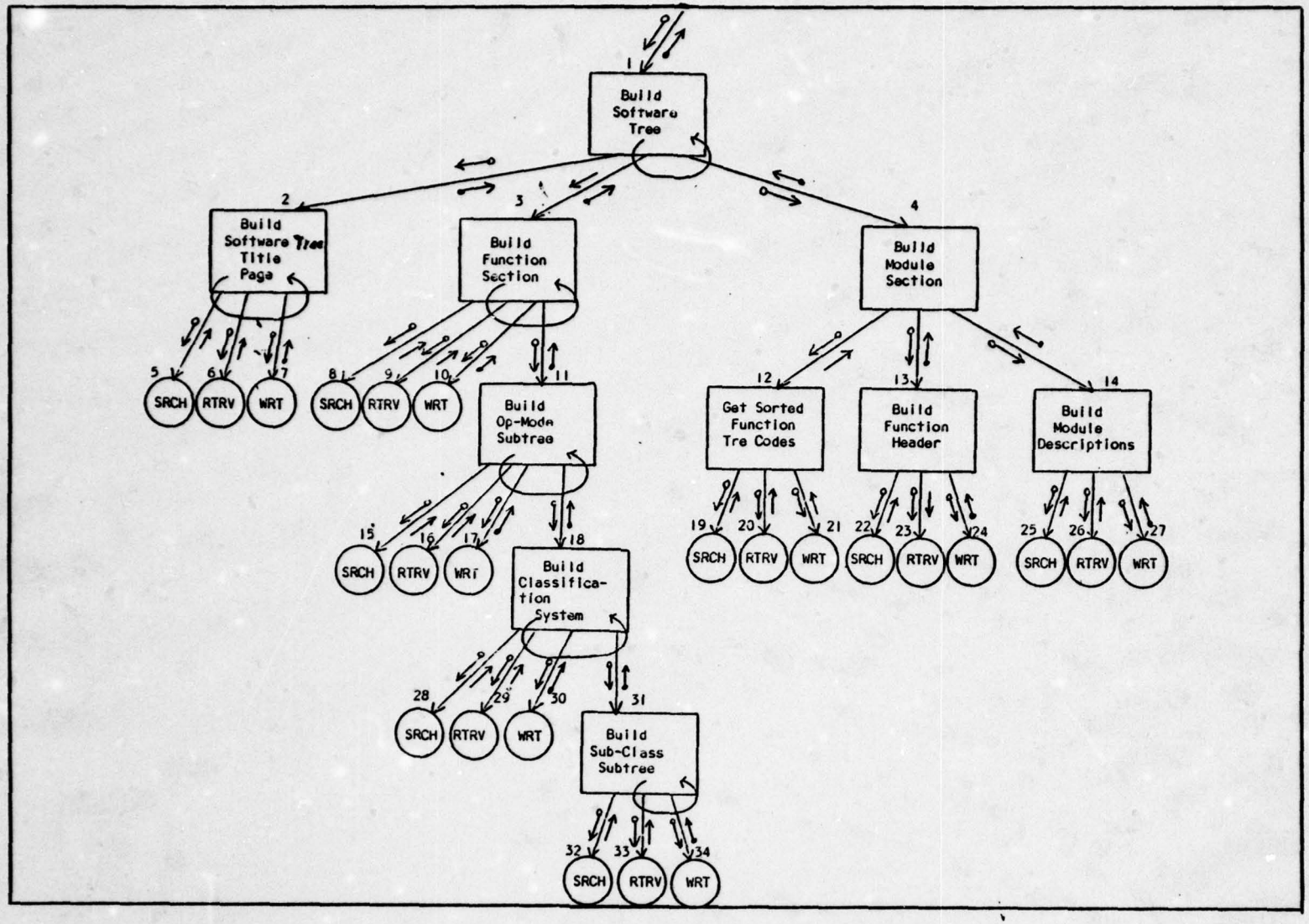


Figure 4-8 BUILD SOFTWARE TREE Branch

Table IX

Interfaces for BUILD SOFTWARE TREE Branch

INPUT	OUTPUT
1, 2 CPCI numbers, CPCI names	Error flag
3, 4 CPCI number, CPCI names	Error flag
5, 8, 15, 19, 22, 25, 28, 32 Search tables	Record name, record identifiers, error flag
6. 9, 16, 20, 23, 26, 29, 33 Record name, record identifier, field names	Tree code numbers and descriptions, error flag
7, 10, 17, 24, 27, 30, 34 File name, output line, line size	Error flag
11 Operating mode tree code	Error flag
12 CPCI number	Sorted "function" tree codes, error flag
13 Function tree code	Error flag
14 Module tree code	Error flag
18 Classification tree code	Error flag
21 Unsorted function tree codes	Sorted function tree codes, error flag
31 Sub-classification tree code	Error flag

Build Software Tree. This module supervises the construction of a family tree for the modules of each named CPCI. The tree is a hierarchical data structue which describes the functional organization of the CPCI's modules. Each module is assigned a unique seyen-character tree code by the user. This code is used to identify the module and the functions that it supports within the CPCI. 'Build Software Tree' calls subordinate modules to construct a title page, a hierarchical breakdown of the functions of the CPCI's components, and a list of the modules (grouped by function) which comprise the CPCI.

Build Software Tree Title Page. This module is invoked once during the construction of the Software Tree. It locates and retrieves document identification data for the Software Tree such as issuing agency, document number, and contract number. It then edits this data and writes output lines to the report file.

Build Function Section. This module outputs a hierarchical description of the functional organization of a CPCI's modules. Each CPCI's family tree consists of four levels of functions (operating mode, classification, subclassification, and function) that a module can support. This module obtains and outputs a description of each of the operating modes defined for the named CPCI. Additionally, for each operating mode, it invokes subordinate modules to output descriptions of component functions performed in support of that operating mode.

Build Module Section. Upon activation, 'Build Module Section' produces descriptions of the modules which comprise the named CPCI. Modules that, together, perform a single function are presented as a group. For each function, a header (describing the function's organizational superstructure) and a list of module descriptions is produced. A module's description consists of its name, number, version, delivery date, source and object file addresses, and list of supporting documents.

Build Op-mode Subtree, Build Classification Subtree, Build Sub-classification Subtree. These modules obtain and report hierarchical descriptions of classifications, sub-classifications, and functions that support an operating mode. A set of one or more functions support a given sub-classification; a set of one or more sub-classifications support a given classification; and, a set of one or more classifications support a given operating mode.

Get Sorted Function Tree Codes. This module obtains function-level tree codes for a CPCI and insures that they are sorted in ascending order.

Build Function Header. This module uses a "function" tree code which was passed to it as a parameter to obtain the description of the function and the descriptions of the operating mode, classification, and sub-classification that the function supports. This information is written to the

report file just ahead of the descriptions of the modules that perform the specified function.

Build Module Descriptions. This module obtains and writes to the report file a description of each of the modules that, together, perform a single function.

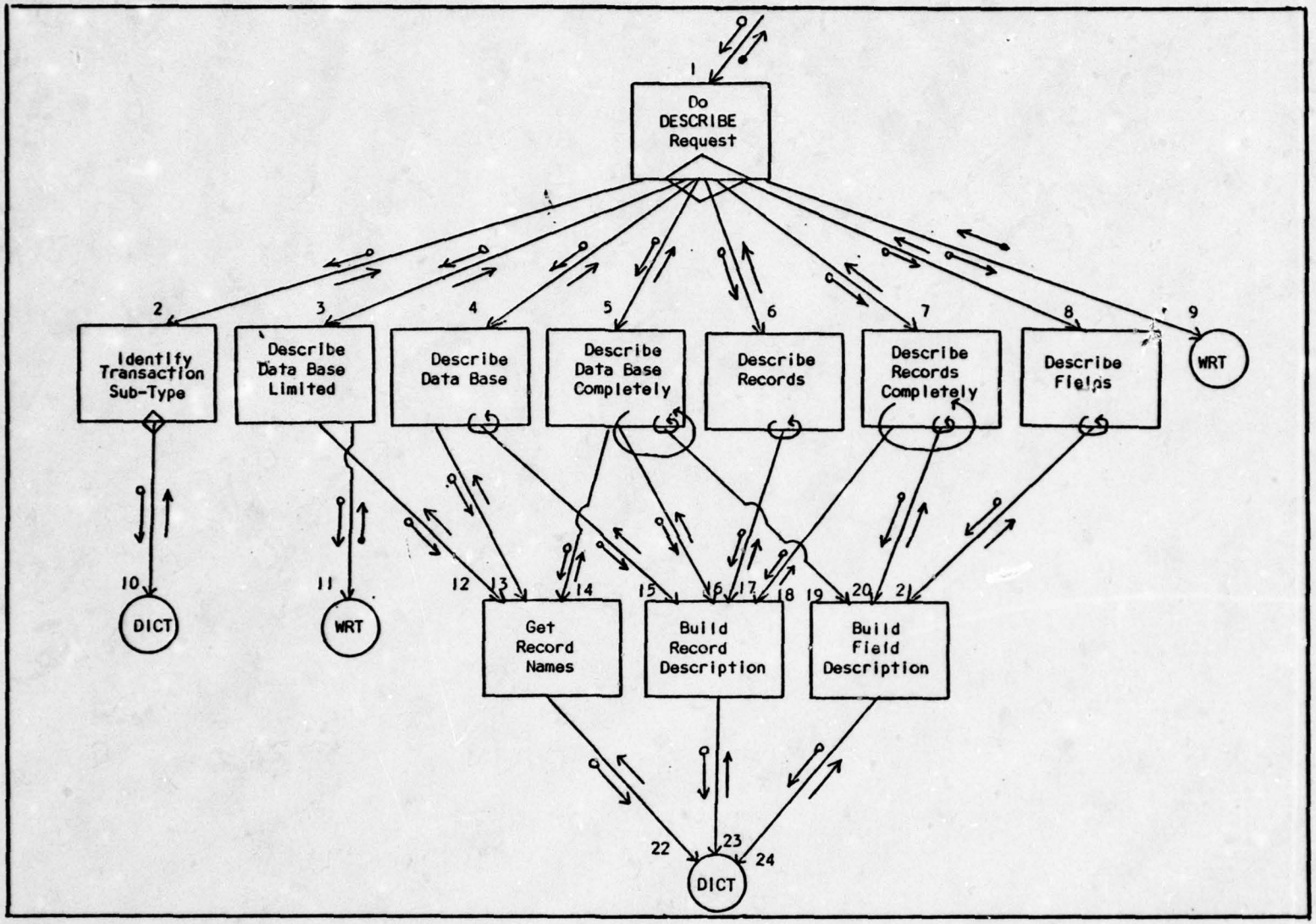


Figure 4-9 DO DESCRIBE REQUEST Branch

Table X

Interfaces for DO DESCRIBE REQUEST Branch

	INPUT	OUTPUT
1	Command table	Error flag
2	Command table	Transaction sub-type, error flag
3, 4, 5	-----	DESCRIBE response, error flag
6, 7, 8	Command table	DESCRIBE response, error flag
9, 11	File name, output line, line size	Error flag
10, 22, 23, 24	Item name, action code	Data item description, error flag
12, 13, 14	-----	Record names, error flag
15, 16, 17, 18	Record name	Record description, error flag
19, 20, 21	Field name	Field description, error flag

Do DESCRIBE Request. This module is a transaction center which handles requests for information about the logical structure of the database. It determines which of the six possible types of DESCRIBE request was submitted, dispatches to the appropriate transaction level module to construct a response to the request, and writes the constructed response to the message file. In addition, it writes a message to the message file to indicate whether or not the request was successfully accomplished.

Identify Transaction Sub-Type. This module examines the command table's entries to determine the type of DESCRIBE request that was submitted by the user. This may require references to the database dictionary to determine if variable names are of records or of fields. In addition, the command table entries are checked for consistency.

Describe Database Limited. This module obtains the names of all of the record types defined for the opened database realm. These names are edited into a response format which is written to the message file.

Describe Database. This module obtains the names of all the record types defined for the opened database realm and then obtains edited descriptions of these records for delivery to the user.

Describe Database Completely. This module obtains the names of all the record types defined for the opened data-

base realm, obtains an edited description of each of the records, and then obtains an edited description of each of the fields that comprise each record.

Describe Records. This module obtains an edited description of each record explicitly named in the command table. An error indicator is set if any of the named records cannot be found in the database dictionary.

Describe Records Completely. This module obtains edited descriptions of each record explicitly named in the command table and of each field within each record. An error indicator is set if any of the named records cannot be found in the database dictionary.

Describe Fields. This module obtains an edited description of each field explictly named in the command table. An error indicator is set if any of the named records cannot be found in the database dictionary.

Get Record Names. This module queries the database dictionary for the names of all the types of records defined in the database. It notifies its superordinate when an error is detected.

Build Record Description. This module obtains and edits a description of a specified data base record. A record's description will identify its: name, component filed names, storage space requirements, key fields, and number of occurrences in the realm. The module notifies its superordinate when any processing error occurs.

Build Field Description. This module obtains and edits a description of a specified database field. A field's description identifies the field's name, length in bytes, data type, and relative position within the logical record in which it's defined. The module notifies its superordinate when any processing error occurs.

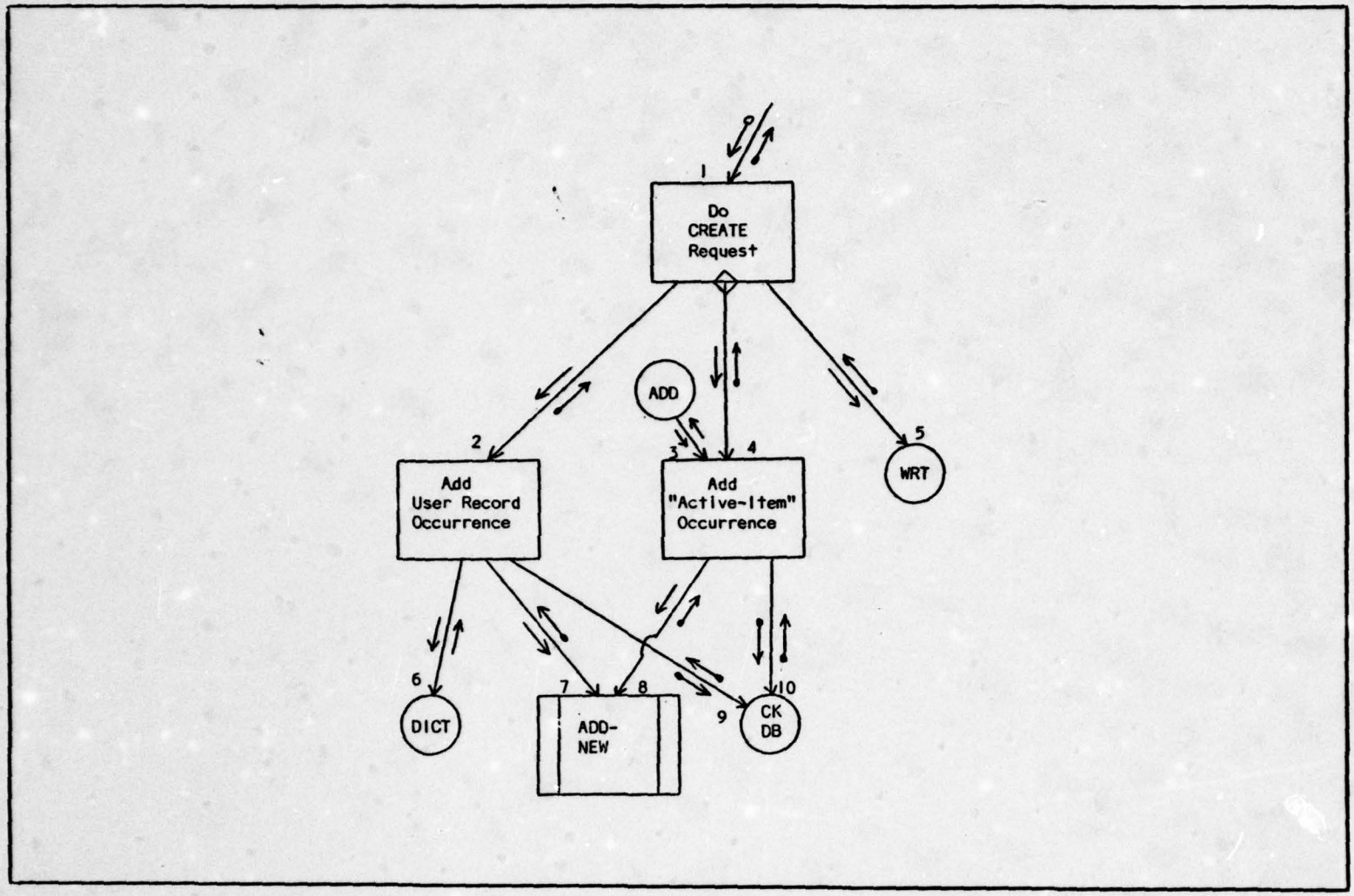


Figure 4-10 DO CREATE REQUEST Branch

Table XI

Interfaces for DO CREATE REQUEST Branch

	INPUT	OUTPUT
1	Criteria table, update table	Error flag
2	Criteria table, update table	Error flag
3, 4	"Active" item identifier	Error flag
5	File name, output line, line size	Error flag
6	Item name, action code	Data item description, error flag
7, 8	Record name, field names, buffer address	DBMS status code
9, 10	DBMS status code	Error flag

Do CREATE Request. This module handles a user request to add a new occurrence to a database record, and it writes a message to the message file to inform the user of the outcome status of his request. If the request is processed successfully, and if it reflects either a new ECP having been submitted or a new document or change notice have been issued, the module invokes 'Add Active-item Occurrence' to record the event.

Add User Record Occurrence. This module adds an occurrence of an active-item record to the database. SOFTSAS must report when additions to certain, predetermined records in the database have been made. An "active-item" record will contain information to identify occurrences of those monitored records that have been changed.

Add Active-item Occurrence. This module adds an occurrence of an active-item record to the database. SOFTSAS must report when additions to certain predetermined records in the database have been made. An active-item record will contain information to identify occurrences of monitored records that have been changed.

Add-New. This is a system-supplied DBMS interface module which calls on the DBMS to add a new occurrence of the specified record to the database.

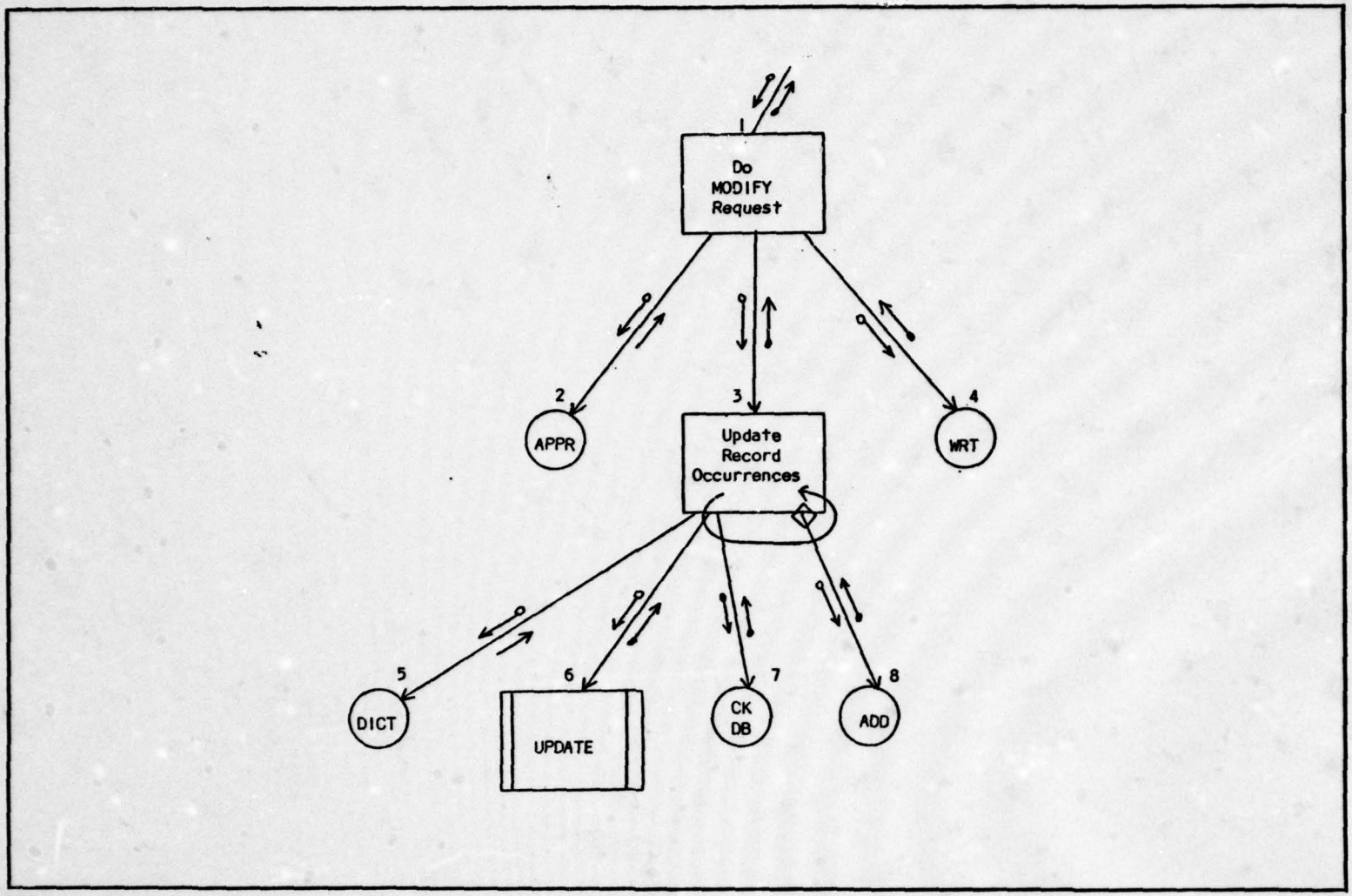


FIGURE 4-11 DO MODIFY REQUEST Branch

Table XII

Interfaces for DO MODIFY REQUEST Branch

	INPUT	OUTPUT
1	Search tables, update table	Error flag
2	Search tables	Approved record identifiers, record name, error flag
3	Update table, record name, approved record identifiers	Error flag
4	File name, output line line size	Error flag
5	Item name, action code	Data item description, error flag
6	Record name, record identifier, field names, buffer address	DBMS status return code
7	DBMS status return code	Error flag
8	"Active" item identifier	Error flag

Do MODIFY Request. This module handles a user request to alter the contents of one or more database record occurrences. It invokes subordinate modules to identify the occurrences that satisfy the search criteria and to perform the database update. However, prior to making any changes in the interactive mode, the module warns the user with a message which states how many qualified record occurrences were located and requests his approval to continue processing the request. Changing certain monitored fields in the database (such as ECP priority or status) requires that an active-item record occurrence be added to the database to register the event. After processing the user's request, the module reports the outcome status of the request to the user and sets an error indicator for its superordinate.

Update Record Occurrences. This module invokes the DBMS 'update' interface routine to store buffered data in the database. In order to prepare the buffer region to hold the new status accounting data, the module obtains descriptions from the database dictionary of those fields to be changed. If the DBMS status return code indicates that the request was successfully accomplished and if the change affected a monitored field, an acitve-item record occurrence is added to the database. Any errors encountered during request processing are reported to the module's superordinate.

Update. This is a system-supplied DBMS interface module which calls on the DBMS to transfer status accounting data from a buffer region to fields within a particular record occurrence in the database.

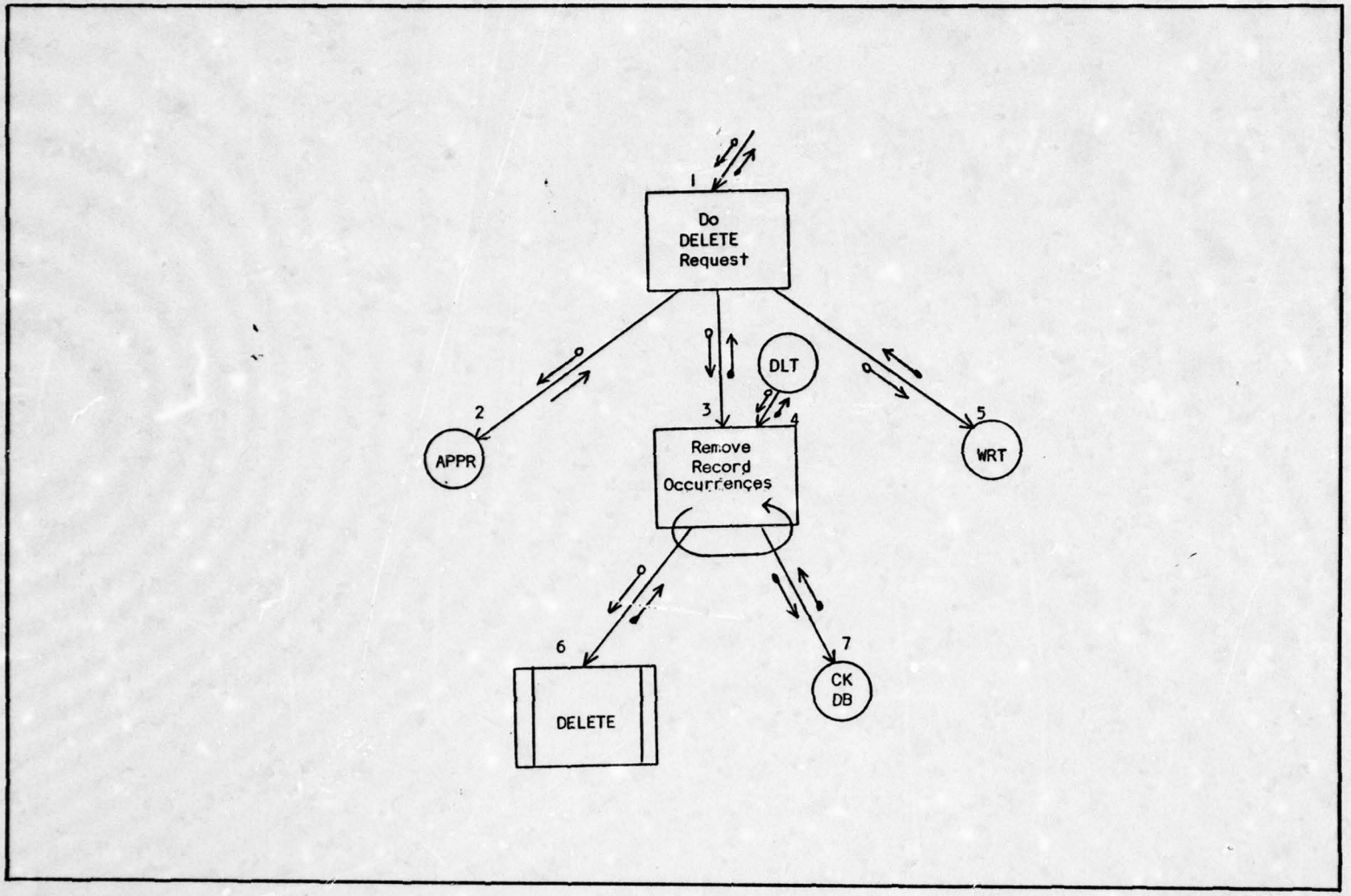


Figure 4-12 DO DELETE REQUEST Branch

Table XIII

Interface for DO DELETE REQUEST Branch

	INPUT	OUTPUT
1	Search tables	Error flag
2	Search tables	Approved record identifiers, record name, error flag
3, 4	Record name, record identifiers	Error flag
5	File name, output line, line size	Error flag
6	Record name, record identifier	DBMS status code
7	DBMS status code	Error flag

Do DELETE Request. This module obtains the identifiers of record occurrences to be deleted and invokes 'Remove Record Occurrences' to delete them (and all pointers to them) from the database. Prior to making the deletions in the interactive mode, the module warns the user with a message which indicates how many record occurrences were found that satisfy the search criteria. The message also requests the user's approval to perform the deletions. Afterwards, the module writes a message to the message file to inform the user about the outcome status of his request.

Remove Record Occurrences. When activated, this module accepts as input the identifiers of those record occurrences to be deleted. For each occurrence, it then invokes a DBMS interface routine to satisfy the user's request. In addition, this module checks the DBMS status return code upon completion of the request and sets an error indicator for its superordinate.

Delete. This is a system-supplied DBMS interface module which calls on the DBMS to remove an occurrence of a record from the database.

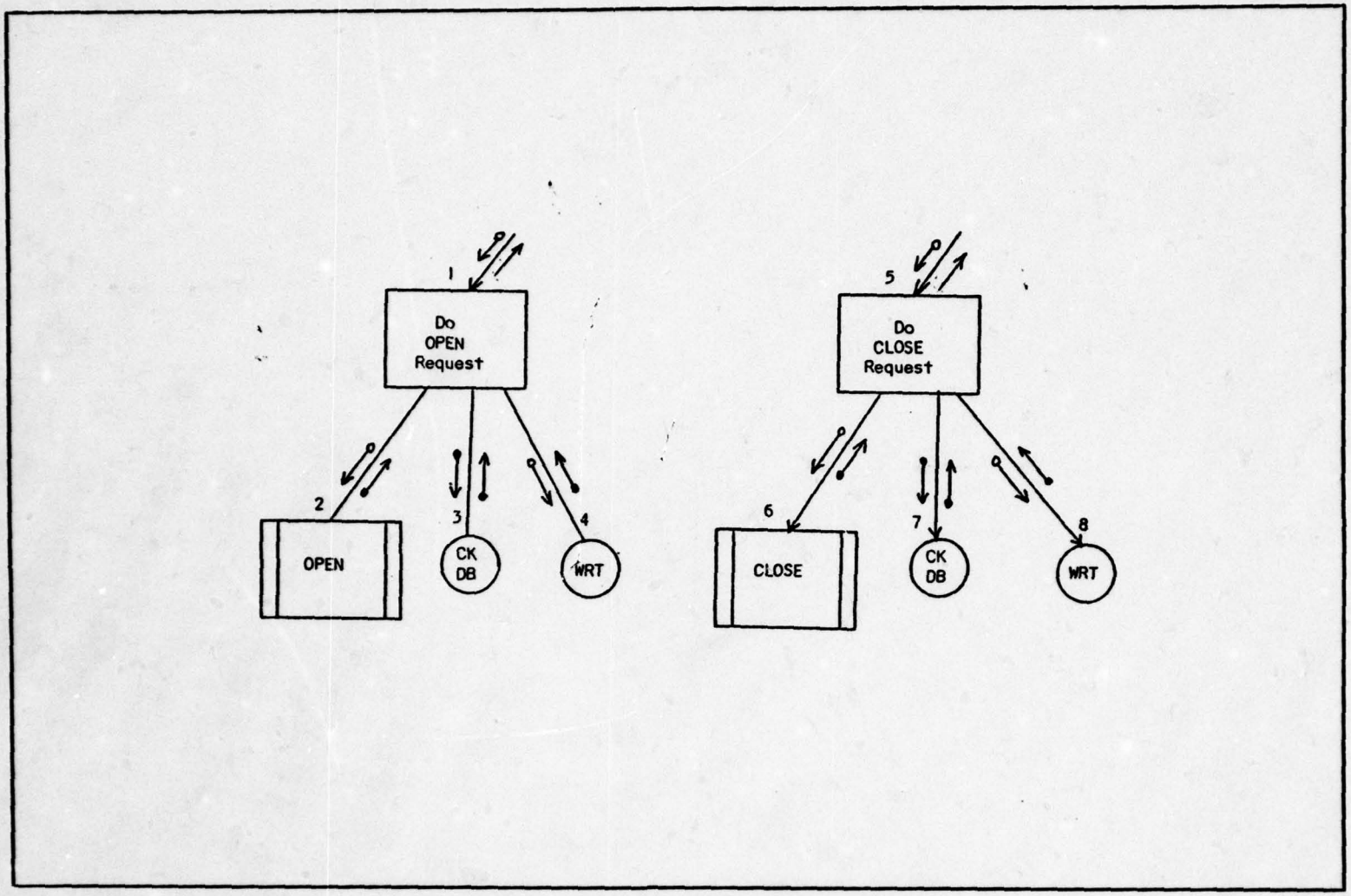


Figure 4-13 DO OPEN REQUEST and DO CLOSE REQUEST Branches

Table XIV

Interfaces for DO OPEN REQUEST and DO CLOSE REQUEST Branches

	INPUT	OUTPUT
1	Command table, update table	Error flag, opened realm name
2	User identifier, access mode, realm name	DBMS status code
3	DBMS status code	Error flag
4	File name, output line line size	Error flag
5	Command table	Error flag
6	Realm name	DBMS status code
7	DBMS status code	Error flag
8	File name, output line, line size	Error flag

Do OPEN Request. This module handles a user request to prepare a database realm for processing. It extracts the user identifier and realm access mode from the update table and invokes the system-supplied DBMS interface routine 'Open'. If the request is successfully accomplished, the module transmits the name of the opened realm to the SOFTSAS executive. The module will set an error indicator for its superordinate and will notify the user about the outcome of his request.

Open. This is a system-supplied DBMS interface routine which calls on the DBMS to validate a user's request for access to a realm and to prepare that realm for processing.

Do CLOSE Request. This module handles a user request to discontinue processing of a database realm. It invokes a DBMS interface routine to de-allocate the realm, checks the DBMS status return code, notifies the user about the outcome of the request, and sets an error indicator for its superordinate.

Close. This is a system-supplied DBMS interface module which calls on the DBMS to terminate processing of a realm and to de-allocate it from the program.

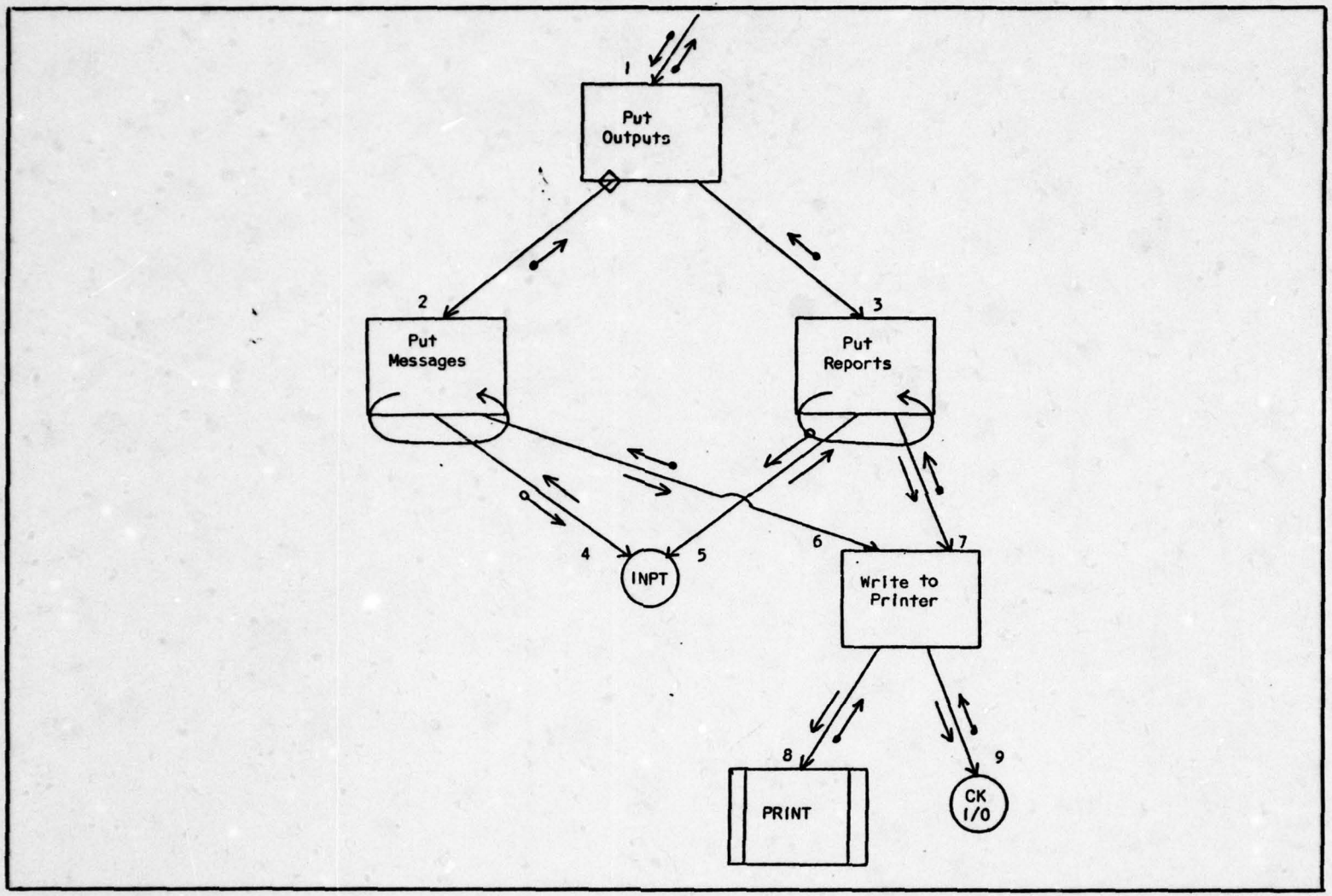


Figure 4-14 PUT OUTPUTS Branch

Table XV

Interfaces for PUT OUTPUTS Branch

	INPUT	OUTPUT
1	Operating mode	Error flag
2, 3	-----	Error flag
4, 5	File name	Input line, line size, error flag
6, 7	File name, action code, buffer address, buffer size	Error flag
8	Logical output device, output line, line size	I/O status return code
9	I/O status return code	Error flag

Put Outputs. This module supervises the delivery of any accumulated messages and reports to an on-line printer just prior to job termination. Output line images are collected in two files by SOFTSAS request handlers. One file is used to hold the reports; the other is used to hold messages, except when SOFTSAS is executing in the interactive mode. In this case, messages are written immediately to the user at his terminal, and only reports are put to the printer.

Put Messages, Put Reports. These modules read output line images from the message and report files, respectively, and dispatch them to the 'Write To Printer' module. The modules set error indicators for their superordinate.

Write To Printer. This module calls the system-supplied I/O routine 'Print' to write a logical record to an output buffer of an on-line printer. The module then checks the I/O status return code and sets an error indicator for the modules that invoke it.

Print. This is a system-supplied I/O routine that accepts an output line in print-image format and writes it to the system output buffer.

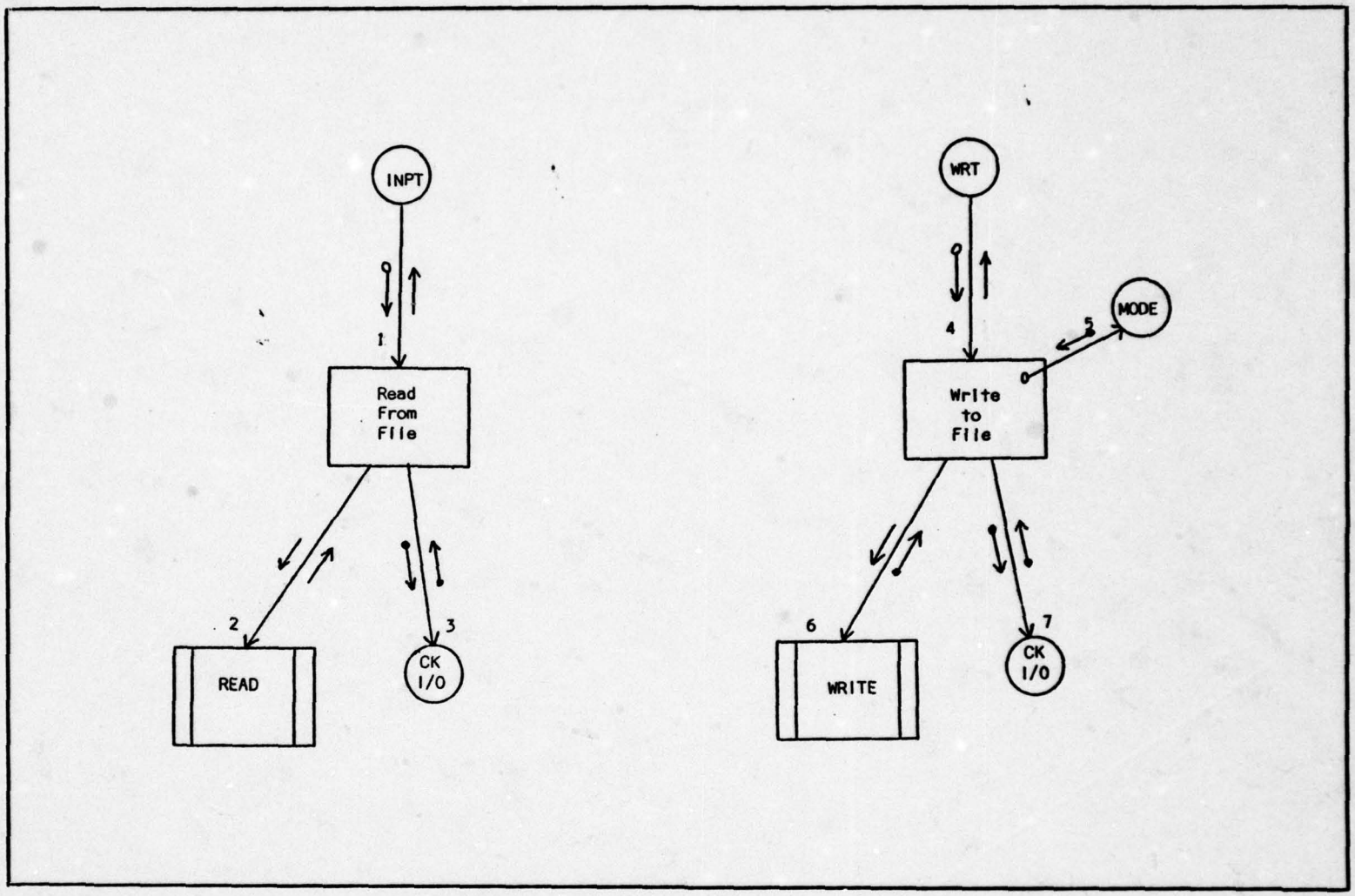


FIGURE 4-15 READ FROM FILE and WRITE TO FILE Branches

Table XVI

Interfaces for READ FROM FILE and WRITE TO FILE Branches

	INPUT	OUTPUT
1	File name	Input line, line size, error flag
2	File name, action code, buffer address, buffer size	Input line, line size, I/O status code
3,7	I/O status code	Error flag
4	File name, output line, line size	Error flag
5	-----	Operating mode
6	File name, action code, buffer address, line size	I/O status code

Read From File. This module calls a system-supplied I/O routine to read a logical record from a file whose logical name is provided as an input parameter. The module then checks the I/O status return code and sets an error indicator for its superordinates.

Write To File. This module calls a system-supplied I/O routine to write a logical record to a file whose logical name is provided as an input parameter. All SOFTSAS files will be located on mass storage devices with one exception. When SOFTSAS is executing in the interactive mode, the message file will be tied to the user's terminal. After the I/O request is satisfied, the module will check the I/O status return code and will set an error indicator for its superordinates.

Read. This is a system-supplied I/O routine that reads a logical record from a specified input device and stores it in a program work area. The module reports the number of bytes of storage occupied by the record.

Write. This is a system-supplied I/O interface module which writes logical records of a specified size to a specified output device.

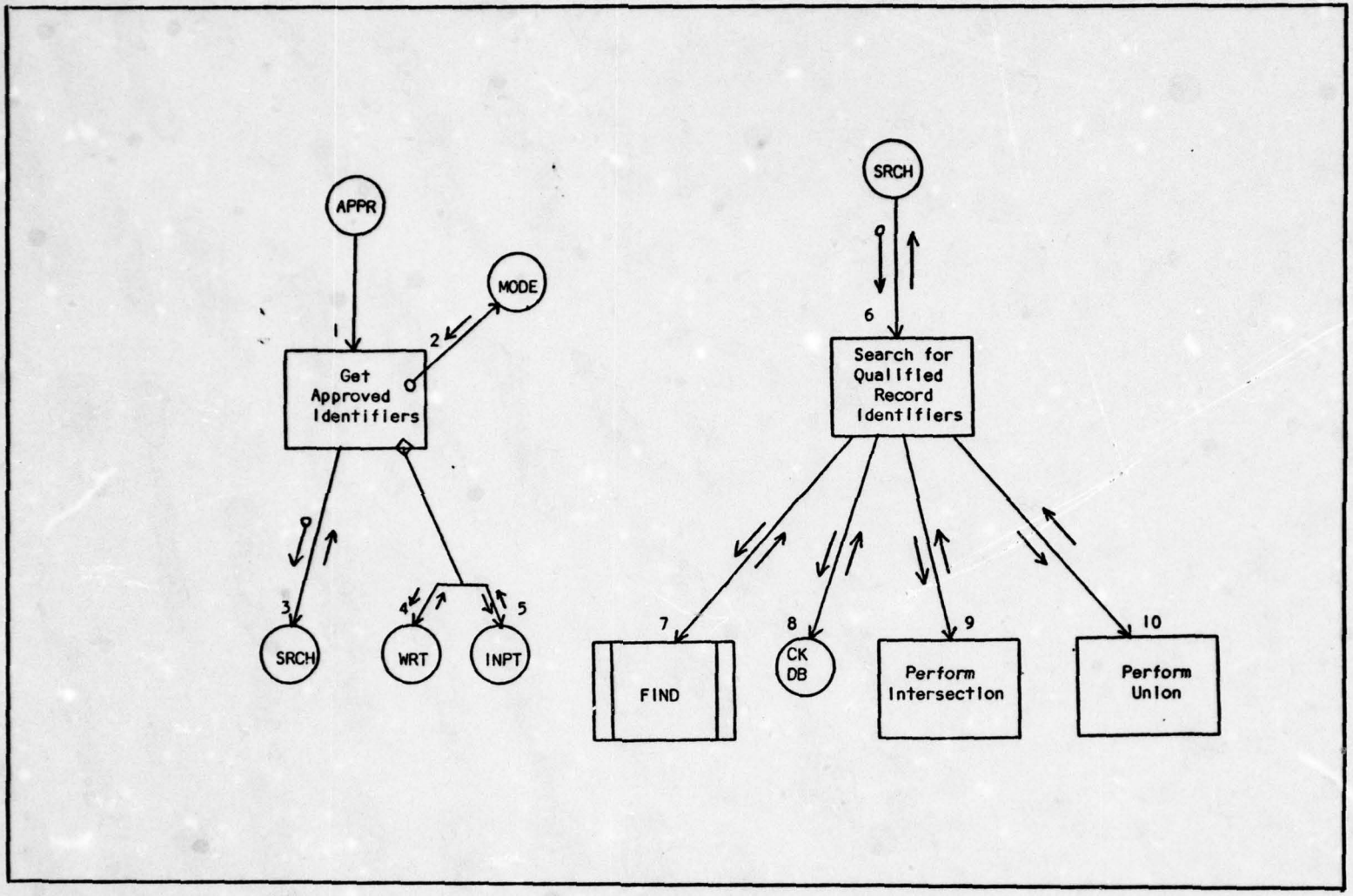


FIGURE 4-16 GET APPROVED IDENTIFIERS and SEARCH FOR QUALIFIED RECORD IDENTIFIERS Branches

Table XVII

Interfaces for GET APPROVED IDENTIFIERS and SEARCH FOR QUALIFIED RECORD IDENTIFIERS Branches

	INPUT	OUTPUT
1	Search tables	Approved record identifiers, record name, error flag
2	-----	Operating mode
3, 6	Search tables	Record name, record identifiers, error flag
4	File name, output line, line size	Error flag
5	File name	Input line, line size, error flag
7	Record name, key field name, key value, relational operator, buffer addr, buffer size	Record identifiers, DBMS status return code
8	DBMS status return code	Error flag
9, 10	Identifiers list A, identifiers list B	Resultant identifiers list

Get Approved Identifiers. This module invokes a subordinate to identify the occurrences of database records that satisfy search criteria contained in the search tables. When SOFTSAS is operating in the interactive mode, this module will write a message to the user stating how many qualified record occurrences were located and requesting that the user enter "Yes" or "No" to continue processing of the request. The module will then obtain the user's response from input. If the response is negative, the module will notify its superordinate that no approved record occurrences were located.

Search for Qualified Record Identifiers. This module invokes a system-supplied DBMS interface routine to obtain the identifiers of record occurrences that satisfy specific search criteria. The search criteria consist of: a record's name; one or more relational expressions, each of which describes a range of values within which a qualified occurrence's key must lie; and, a logical relationship between each of the relational expressions. The relational expressions are contained in the criteria table. The order in which the criteria table entries are processed is dictated by entries in the logic table. Logic table entries comprise a postfix representation of the user's WHERE clause, describing the order in which logical operations (intersection, union) are to be performed on the lists of identifiers. The module checks the database status return code

after each call to the DBMS and sets an error indicator for its superordinates.

Find. This is a DMBS interface module which calls on the DBMS to locate record occurrences that satisfy a specific search criterion. A unique record identifier is obtained for each qualified occurrence.

Perform Intersection, Perform Union. Each of these modules performs a logical operation on two sets of record occurrence identifiers.

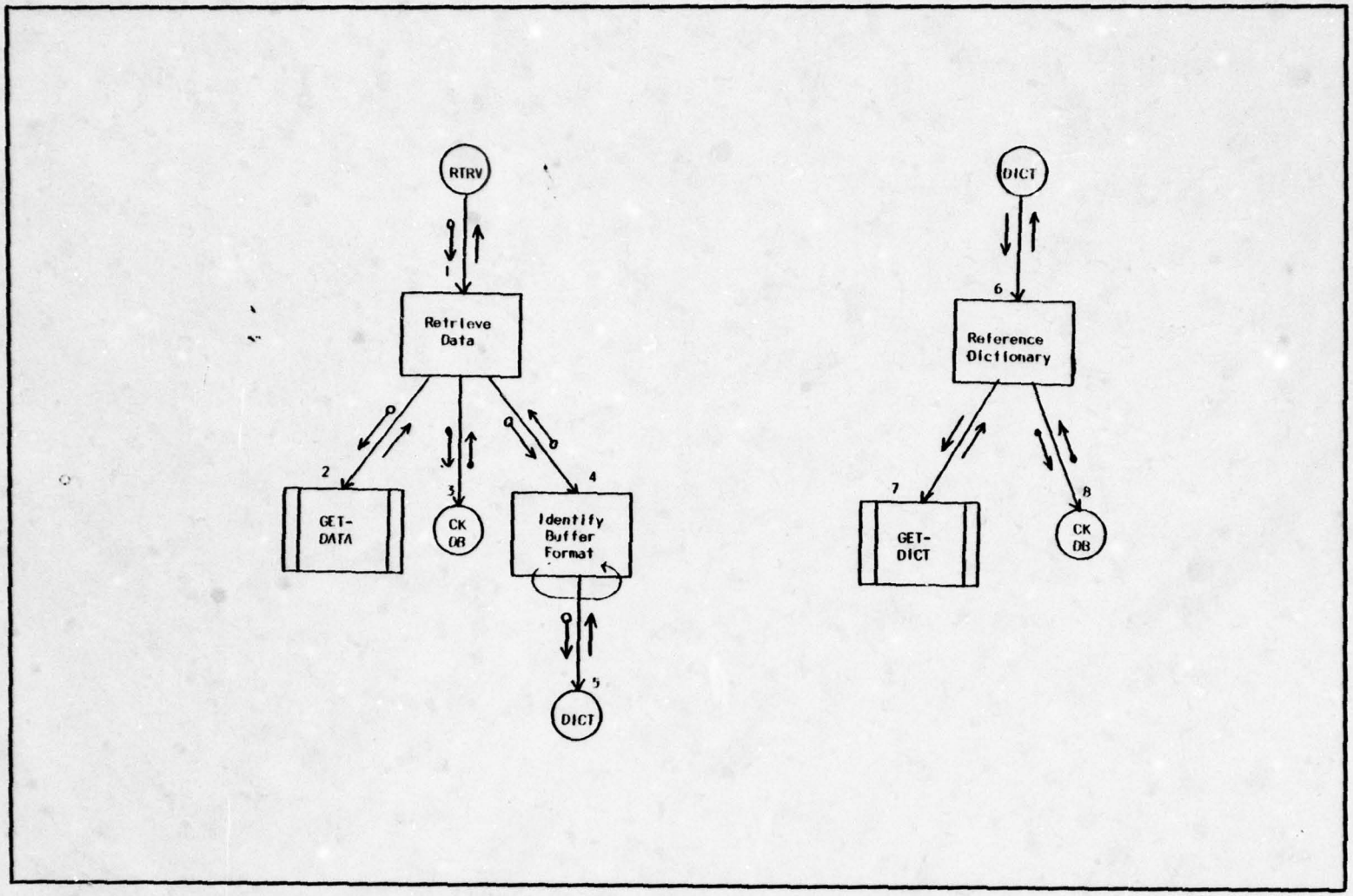


Figure 4-17 RETRIEVE DATA and REFERENCE DICTIONARY Branches

Table XVIII

Interfaces for RETRIEVE DATA and
REFERENCE DICTIONARY Branches

	INPUT	OUTPUT
1	Record name, record identifiers, field names	Retrieved data, error flag
2	Record name, record identifier, field names, buffer addr, buffer size	Retrieved data, DBMS status code
3	DBMS status code	Error flag
4	Field names	Relative buffer addresses
5, 6	Item names, action code	Data item description, error flag
7	Action code, item name, buffer address	Data item description, DBMS status code
8	DBMS status code	Error flag

Retrieve Data. This module invokes a system-supplied DBMS interface routine to copy specific items of data from a database record occurrence into a buffer. It also calls a subordinate to identify the data type and relative buffer location for each item that was obtained. If no fields are named, data from each of the fields in the record are obtained. The module checks the DBMS status return code, setting an error indicator for its superordinate.

Reference Dictionary. This module invokes a system-supplied DBMS interface routine to retrieve a description of a particular part of the logical structure of the database. It then checks the DBMS status return code and sets an error indicator for its superordinates.

Get-Data. This is a DBMS interface module which calls on the DBMS to retrieve data from specific fields in a single record occurrence.

Identify Buffer Format. This module determines the format in which the retrieved data items were stored by the DBMS in a buffer. As required, it obtains record and field descriptions from the database dictionary.

Get-Dict. This is a DBMS interface module which calls on the DBMS to obtain database descriptive information from the database dictionary. A specified "action code" instructs the module to retrieve either record names, record descriptions, or field descriptions.

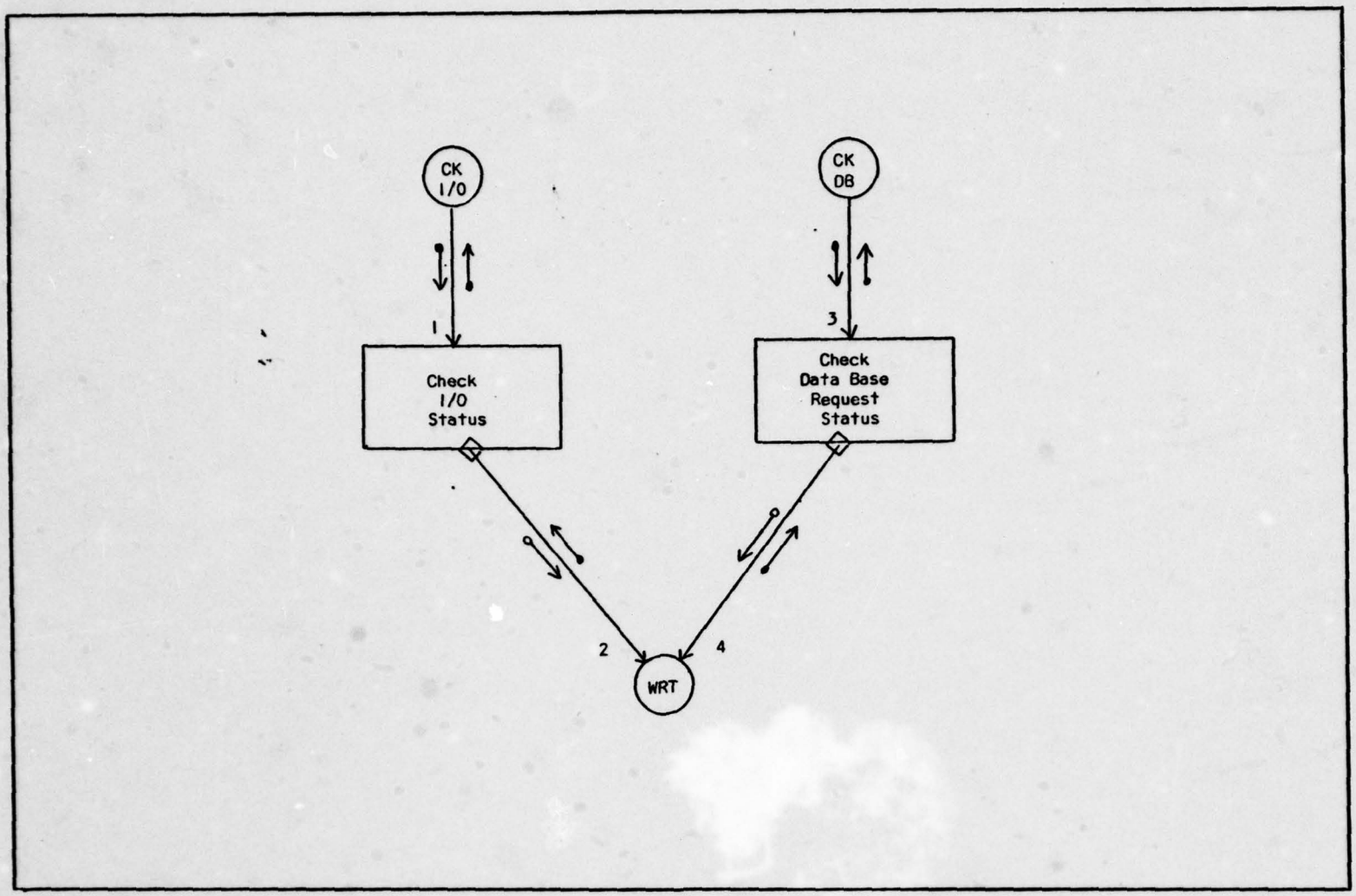


Figure 4-18 CHECK I/O STATUS and CHECK DATABASE STATUS Branches

Table XIX

Interfaces for CHECK I/O STATUS and
CHECK DATABASE STATUS Branches

	INPUT	OUTPUT
1	I/O status return code	Error flag
2, 4	File name, output line, line size	Error flag
3	DBMS status return code	Error flag

Check I/O Status, Check Database Status. These modules inspect database and I/O status return codes, respectively. When either of these modules detects that an error has occurred, it generates an error message, writes it to the message file, and sets an error indicator for its superordinates.

Observations

Abstraction. The structure charts presented in this chapter constitute a preliminary design for SOFTSAS which will require refining before it can be used. It is an abstract solution to the general status accounting problem. The descriptions of the modules and the data communicated between them were specified such that flexibility in the choice of a more specific implementation was provided. For example, a user's input request is converted to an internal form and is stored in various "internal tables". But a format was not provided for the tables, and the manner in which they are to be passed as arguments (i.e. by name or by value) was not identified.

Modifiability. The Structured Design approach to hierarchically functional decomposition enhances the modifiability of the system. As reflected by the structure charts, a user's input request is read and completely analyzed by the 'Get Analyzed Request' module before being converted to an internal form. Since knowledge of the formats of submitted requests is localized to one sub-tree of the program's hierarchy, these formats can be changed without impacting other parts of the program. In addition, the design readily allows for more serious changes as well. Implementation of a DBMS which communicates entire records of data instead of individual fields would require changes to the modules that prepare and interpret the database buffer as well as those that interface with the DBMS. Also,

such a change may require that module 'Update Record Occurrences' first retrieve data that is to remain unaltered in order to completely specify the record's new contents. Although these changes may be substantial, the affects are still localized to a set of well-defined modules whose identity can readily be dtermined from the charts.

Error Handling. Although decomposing a program into highly independent modules makes it less complex and easier to develop, test, and modify, there is no way to ensure that a program will always execute correctly (Ref 22). Therefore one of the primary goals of software design is to build a program that is reliable. For software, reliability is defined as the probability that a software error does not cause deviation from required output by more than specified tolerances in a specified environment over a specified time interval (Ref 31, 38:5,2). For a program to be reliable, its errors must be reported when they occur, and their adverse effects must be minimized. This requires that:

a. Processing errors be detected at the earliest possible time;

b. Errors be prevented from propagating and damaging other parts of the system; and,

c. An attempt be made to recover any losses or damages (Ref 11:337-338, 40:429).

The third requirement implies that a program abort should

not occur; this would only preclude the possibility of instantaneous recovery attempts and loss minimization.

"In most cases, the module detecting an error should not be the module which corrects it" [Ref 29:359]. Often, errors are dtected by modules at low levels of a program's structural hierarchy. These modules are usually designed to perform a function that can be used in many applications. It is not desirable that these modules handle "exception processing" because it tends to limit their re-usability. In addition, what appears to be an error condition at a low level may constitute an acceptable condition at a higher level. (An example of this might be a "no data qualify" condition for a data retrievel request.) But neither is it desirable to complicate high-level modules by having them react to errors that can be handled at lower levels. A rule of thumb is that errors should be handled by the lowest level module having sufficient knowledge to direct a recovery. When recovery is not possible, the problem should be passed to higher-level modules to be resolved.

The SOFTSAS modules which will most often detect processing errors are those which interface with the I/O facility or with the DBMS and those which see parts of submitted user requests. When a processing error is detected, SOFTSAS will notify the user, reporting:

a. The type of error that occurred;

b. A description of the data that was in error;

c. An indication of the outcome of any recovery actions that the program attempted; and,

d. A suggestion for avoiding a reoccurrence of the error.

When executing in batch mode, a non-recoverable error will cause the SOFTSAS executive module to terminate execution of the program. However, when executing in the interactive mode, only execution of the request will be terminated. In either case, the "error-detecting" modules need some medium for communicating the presence of the error to the program executive. Many of the module interfaces defined in the preceeding section contain an output parameter, "error flag", to inform a module's superordinate when it perceives that a detected error is non-recoverable. If the superordinate cannot effect a recovery of its own, it will pass the error notification along to its superordinate. Since there is no specification for a host machine or a DBMS as yet, the specific error types and related recovery procedures have not been defined in this thesis.

Summary

This chapter presented the design specification for SOFTSAS which was developed using the Structured Design strategies of transform-analysis and transaction-analysis. The design was derived by conceptualizing the formal functional specifications as data flows and then following the design guidelines set forth by Constantine. A modification/

evalution process was repeated several times until the final set of structure charts was obtained.

V Database Design

Introduction

Database processing consists of three components: the user's applications programs, the database management system (DBMS), and the database itself. Chapter II discussed the functions that the DBMS must perform in order to support SOFTSAS, and Chapter IV suggested a SOFTSAS/DBMS interface. Although the design of a DBMS is beyond the scope of this thesis, this chapter presents a model which describes a logical structure for the database on which the SOFTSAS DBMS will operate.

Database Modelling

Succinctly stated, a database is a collection of interrelated data items stored to serve multiple applications. The description of a database takes two forms. A physical description, of interest to systems programmers and designers, specifies the manner in which the data items are physically stored and referenced on the hardware. A logical description, used by applications programmers and users, specifies a format for each of the data items and identifies the relationships among the data items as perceived by the user.

Application programs use logical data descriptions to communicate with the DBMS. The DBMS converts these into physical descriptions which it uses when invoking the basic input/output routines of the host operating system. This two-tiered system for referencing the data protects application programs from physical data storage considerations, thus enhancing their modifiability.

The logical structure of a database is modelled at two levels of abstraction called the schema and the sub-schema, respectively (Ref 10). A schema is an overall description of the true logical structure of all the data contained in the database; it is defined and controlled by the Database Administrator. But since a database serves multiple applications, individual applications seldom utilize all the data it contains. In addition, two applications may each view differently the data that they share. Therefore,

sub-schemas are used to describe subsets of the database for individual applications. Despite their differences, both schemas and sub-schemas are "maps" which describe the structure of data items and the associations among them. These maps can be drawn in many ways.

Canonical Models

Martin (Ref 20:248-290) has proposed a methodology for obtaining a "canonical" schema from a set of user views of the data. He defines a canonical schema as, "A model of data which represents the inherent structure of that data and hence is independent of individual applications of the data and also of the software or hardware mechanisms which are employed in representing and using the data" (Ref 20: 248). His approach is as follows.

Each end-user's view of the data is analyzed to obtain a bubble chart consisting of the data item types (bubbles) and the associations among them (directed links) that are of interest to the user. Between any two data items, A and B, there are two possible types of associations. First, there can be a one-to-one mapping (1-association) from data item A to data item B such that each value of A always uniquely identifies one value of B. This type of association would be reflected on the bubble chart by a single-headed arrow connecting A to B. Second, there can be a one-to-many mapping (M-association) such that each value of A identifies zero, one, or more values of B. This type of association

would be reflected by a double-headed arrow on the bubble chart connecting A to B. In some cases, arrows will be used that point both from A to B and from B to A.

Once the bubble charts have been developed for each user, they are combined according to a specific procedure (Ref 20:274-276). The procedure essentially entails merging user views, one at a time, such that redundancies in data and relationships are eliminated while user-defined logical structures are preserved. The result is a canonical model for the database which can be implemented using one of several database management systems, including CODASYL DBTG-based software, DL/1-based software, and relational database software.

A canonical model is interpreted as follows. The primary keys are those bubbles which have one or more single-headed arrows leaving them. (A primary key that consists of more than one data item is called a concatenated key, and its items are treated as a unit.) The secondary keys are those bubbles which have one or more double-headed arrows leaving them. A record consists of a primary key and data items (called attributes) that satisfy the following two criteria: a. The attribute must be fully functionally dependent on the entire primary key; b. The attribute cannot be functionally dependent on any other attribute in the record, unless that other attribute could itself be used as the record's primary key.

SOFTSAS Canonical Schema

The canonical schema presented in this thesis models software configuration status accounting (CSA) data of concern to AFSC program managers. The schema was derived by analyzing "user views" of the data. However, a user was not considered to be a software configuration manager; rather, a user was considered to be a section of a CSA report to be produced for that manager. For example, to satisfy the CSA requirements of the SIMSPO, a total of 8 sections for 3 reports need to be constructed on a periodic basis (see Chapter III); thus, there are at least 8 separate user views of the data for the SIMSPO. Once a preliminary schema was derived, additional data that would be useful to SIMSPO software configuration managers, both now and possibly at some later time, were also included. The schema that evolved models the software CSA data for only one AFSC program office. However, since a canonical schema portrays the inherent relationships among data, and since much of the software CSA data to be reported to AFSC program offices is the same (the formats of the reports are what differ most), it is felt that the CSA reporting requirements of other SOFTSAS users within AFSC could be satisfied without radical changes to the model.

Figure 5-1 and Figures C-1 through C-27 in Appendix C portray the canonical schema for the SOFTSAS database. Figure 5-1 identifies the logical database records (bubbles)

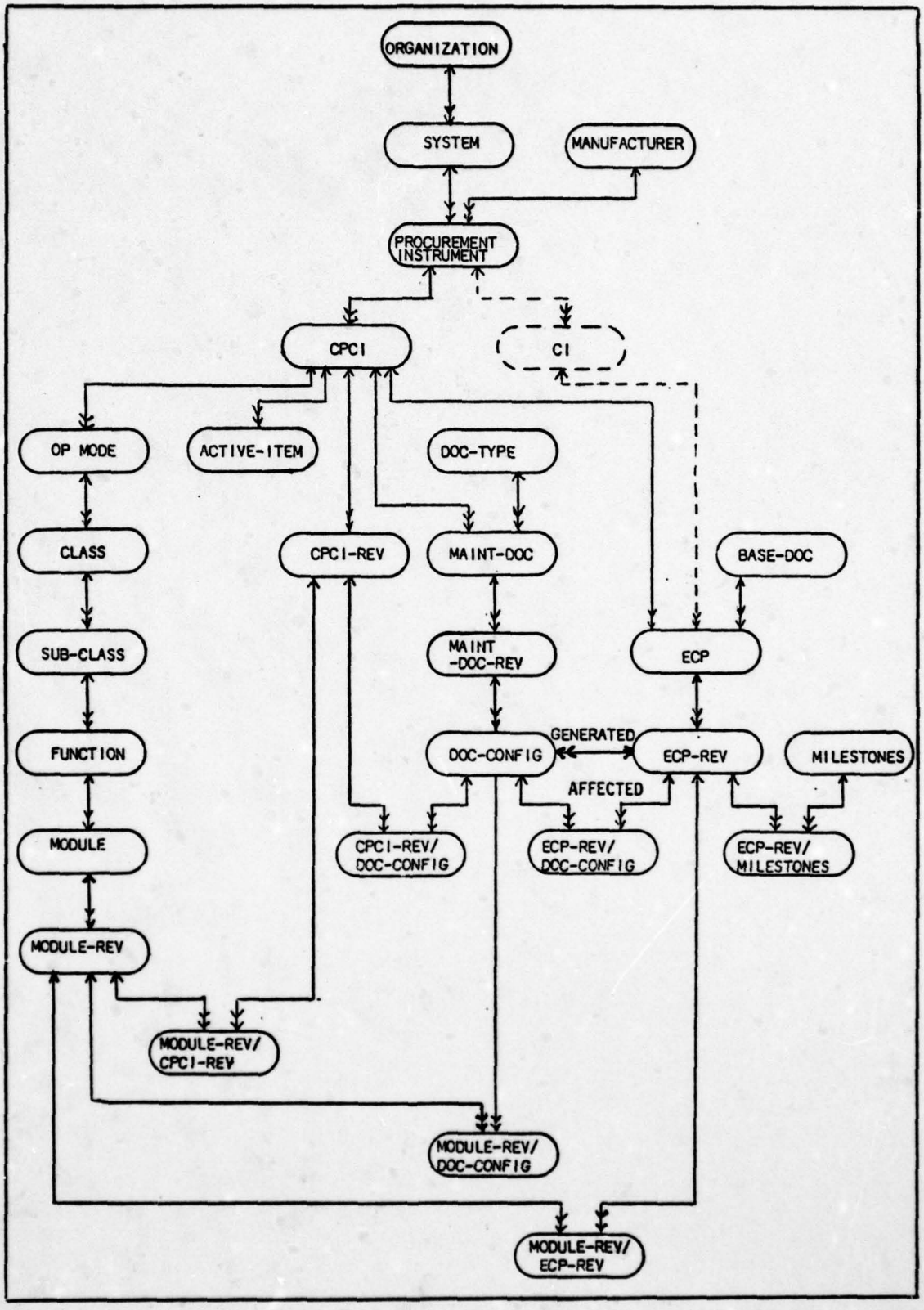


Figure 5-1 SOFTSAS Canonical Schema

and the relationships among them (arrows). The figures in the appendix describe the contents of these records.

Each of Figures C-1 through C-27 describes one complete database record. A record is depicted as a set of conjoined boxes, each of which identifies a data item belonging to the record. The contents of the first box in each figure are underlined for emphasis because they identify the record's primary key. Data items that form a concatenated key are co-located in the first box and are separated by a plus (+) sign. The other boxes identify the record's attributes. Although the single-headed arrows directed from the record's primary key to each of its attributes were deleted, a double-headed arrow was retained to identify where an attribute is to be used as a secondary key.

Table C-I defines each of the data items named in the schema. Format descriptions for most of the items can be found in MIL-STD-482A.

Summary

This chapter presented a canonical schema to describe the structure of the status accounting database. The schema was derived from the user's views of the data. Unfortunately, most database management systems today cannot use canonical schemas. However, it is a relatively straightforward process to convert this schema into any of several logical structures that can be implemented with present-day database management systems. Despite the overhead for conversion, the canonical schema is useful because it provides a "minimal structure" for the database and identifies the best way to group and interconnect the data: according to its inherent properties.

VI Conclusion

Observations

Status accounting is an essential element of configuration management which can contribute significantly to the success of an acquisition program. Historically, accounting for CPCIs has been neglected in favor of the more visible, and usually more expensive, CIs. But with the rise in software costs and complexity, configuration management of CPCIs is essential.

The CSA report requirements presented in Chapter II were tailored to the needs of the Simulator SPO and may not be appropriate for other procurement agencies. This is because AFSC configuration management policies and procedures give program offices flexibility in accounting for the status of their configuration items. Since program offices are directed to obtain only that information needed to effectively and economically do that, the potential exists for much variation in their status accounting requirements.

General designs for a modifiable computer program plus a comprehensive database were presented in Chapters IV and V. They were derived without any specific hardware, DBMS, support software, or programming language constraints. These constraints could not be specified because each program office determines the agency to perform the status accounting function (itself or the contractor) based on cost-effectiveness considerations. To have imposed a standard

set of constraints would have been counterproductive, particularly since many contractors already have hardware and support software available for use. The designs can be more stringently specified only after an external environment has been defined for a particular application of SOFTSAS.

One desirable requirement not levied on SOFTSAS by the SPO, yet included in the requirements definition, was for an interactive processing capability. Interactive processing allows a user to communicate with the software as it is executing. This can be used, for example, to verify which records in the database will be affected by a specific update request just prior to submitting the request. In addition, it lets users spontaneously correct certain types of errors detected by the software without having to terminate execution of the program. However, interactive processing is most suited to those users with small, inconstant input streams such as those employed when updating the database. Since the Simulator SPO program managers anticipate contracting for status accounting services, they did not levy the interactive processing requirement on SOFTSAS. Nevertheless, the capability was considered desirable enough to conceivably be required by other SOFTSAS users.

A conclusion was reached that the data management function could best be performed in a database processing environment. A DBMS would "hide" information from SOFTSAS about the physical storage of the data and would provide a range of options to protect the data from misuse and abuse. This

simplifies the SOFTSAS program requirements, thus enhancing its maintainability and reducing SOFTSAS life-cycle costs, particularly when a contractor's DBMS is used. In addition, increases in program efficiency can be realized because a DBMS provides the means by which:

a. Data redundancy in the system can be reduced;

b. The same data can be perceived and used in different ways by different users;

c. The data can be obtained using different access paths; and,

d. Data standardization can be achieved.

The logical structure for the database was defined in terms of a canonical schema. A canonical schema depicts the "real-world" relationships between the items of data: without DBMS or physical storage constraints. It is constructed from a collection of user views together with assumptions about future uses of the data. Although a canonical schema cannot be implemented with all DBMS software packages, it can be used to derive a database schema for most hierarchical, CODASYL DBTG-based, and relational systems. This conversion step would be relatively straightforward to an analyst who is familiar with the data definition language of the target DBMS.

The database model presented in this report was called a schema. Indeed, it is a schema for software CSA applications within AFSC. However, the model is really a sub-schema which should be merged with other sub-schemas (such as those

for hardware CSA, manpower accounting, and budgeting) to form a more comprehensive model of an organization's data. In this way, an organization's data can be processed as an integrated whole.

Recommendations

Once an external environment is specified for SOFTSAS, the design presented earlier can be refined. But since many designs can be derived for a particular system, it is important that alternative designs be developed and the best one selected for full-scale development.

Several software design methodologies are currently in vogue (Ref 21). For example, the Structured Design technique derives a design from an analysis of the flow of data associated with the problem; other techniques rely on aspects such as the flow of control or the structure of the data associated with the problem. These methodologies are not firm rules, but rather are guidelines which identify trade-offs so that intelligent choices can be made. Thus, obtaining several competing designs for a computer program, even when using the same technique, will not be technically difficult; but as yet, comparing them will.

Research in the areas of software engineering and software quality assurance has identified some attributes of software quality and some measurable program characteristics that have been demonstrated to affect these attributes (Ref 8,13,14,29:81-85,38:4,68-71]. Some of the characteristics reflect the logic complexity and the documentation of a

module; these pertain more to the implementation of a module than to the design of the program. However, other characteristics reflect the way in which processes are allocated to the program's modules and the way in which these processes are connected; these pertain directly to the design of the program.

In order to select the best of several competing designs, the quality attributes to be built into the software must first be identified. Second, any conflicts between the attributes (Ref 8:594,19:46-47) must be resolved, and the levels of quality to be achieved for each attribute must be explicitly stated in the software's specifications. Finally, tools with which to measure a design's compliance with its specifications and to measure its cost-effectiveness against that of other designs must be developed. Myers (Ref 24:137-149) for one has proposed a model to quantify the independence of modules in a program in order to evaluate the program's complexity and, hence, its maintainability. However, much work remains to be done in the area of quantifying software metrics.

Before much effort is expended in preparing more detailed designs for SOFTSAS, consideration should be given to whether a software status accounting system, separate from a system for hardware, is truly required. It is felt that, by modifying the preliminary SOFTSAS design (primarily the report generator modules), a single status accounting system could serve both purposes. In addition, by expanding

the database to include a structure for hardware data (such as spares status and equipment effectivity), only a single database need be maintained. These actions would not only reduce software costs, but would also allow for combined, yet distinct, status reporting of CIs and CPCIs being acquired for a system.

In addition, some enhancements to the basic SOFTSAS requirements should be considered by the user. For example, a new input keyword could be used together with a clause identifier to signal SOFTSAS to re-use a parameter specified in the previous request. Also, SOFTSAS could be designed to perform some arithmetic or statistical functions on the data upon request. One use of these functions, for example, might be to compute the average number of ECPs per quarter that were incorporated behind schedule into the CPCIs of a system over the course of the previous year.

Beyond SOFTSAS design, it is recommended that a rigorous specification for SOFTSAS coding practices be developed. For example, restrictions should be placed on the use of:

a. Poorly commented code;

b. Assembly language when a high-level language can be used;

c. Language constructs which allow a program to modify itself; and,

d. Aberrant statements or algorithms to gain marginal efficiencies in speed or memory.

Moreover, use of programming techniques such as structured programming and self-check coding should be encouraged. When enforced, these practices enhance the maintainability and usability of the program because the resulting code is made more understandable.

It is also recommended that a centralized CSA data bank be used when more than one contractor is involved in the development of a system. Separate contractors often work on the same system when functional requirements are allocated among several CIs and CPCIs. In such cases, consideration should be given to designating one contractor to engineer the development of the entire system (Ref 5:32-33). He could then be assigned the responsibility of integrating all the status accounting information and producing the required reports. This would not only reduce CSA costs, but it would also make CSA more responsive to the information needs of the system's configuration managers.

As the number of SOFTSAS users increases, status accounting data elements and data relationships will undoubtedly be identified which are not represented by the database model. If the new data elements are not documented in MIL-STD-482A, they should be forwarded to the custodian of the standard for approval and incorporation (Ref 24:1). In addition, an effort should be made to maintain the canonical schema presented in Chapter V, thus enhancing its usefulness as a guide for structuring (or restructuring)

status accounting databases. In fact, it is felt that such a model could itself be a worthwhile addition to MIL-STD-482A, and consideration should be given to incorporating it in the standard. This would also, in effect, assign maintenance responsibility for the model.

Finally, the measure of SOFTSAS' effectiveness will be whether or not it provides the information that a software configuration manager needs in a timely and accurate manner. Therefore, specifications for the use of SOFTSAS must necessarily include such details as:

a. When should SOFTSAS reporting begin;

b. When should an item be added to the database;

c. How frequently should reports be provided, and to whom;

d. How long should an item continue to be reported;

e. Under what circumstances should an item be deleted from the database; and,

f. How quickly must unanticipated requests for reports be satisfied?

Although these details deal more with human interaction with the system than with computer support for the system, variations in these "design" requirements will affect what information the manager receives and thereby affect his ability to manage his acquisition program.

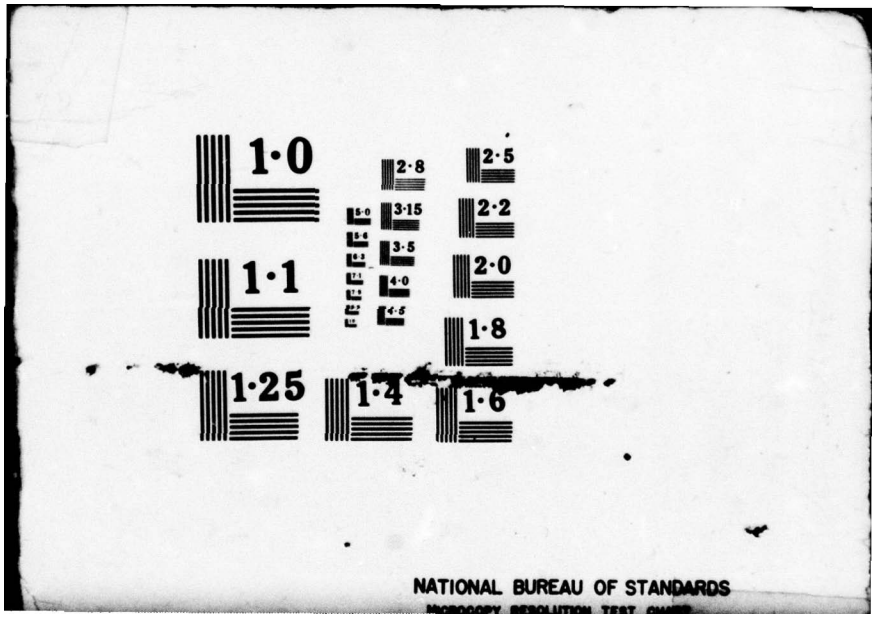
AD-A069 300 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 9/2
DESIGN FOR AN AUTOMATED STATUS ACCOUNTING SYSTEM FOR SOFTWARE C--ETC(U)
MAR 79 A SCHUSTER

UNCLASSIFIED AFIT/GCS/EE/79-2 NL

3 OF 3
AD
A069300



END
DATE
FILMED
7-79
DDC


1·0
1·1
1·25
2·8
3·15
3·5
4·0
4·5
1·4
2·5
2·2
2·0
1·8
1·6
NATIONAL BUREAU OF STANDARDS

Bibliography

1. AFM 66-271. Mechanized Time Compliance Technical Order Reporting System. Washington: Department of the Air Force, April 1976.

2. AFR 65-3. Configuration Management. Washington: Department of the Air Force, July 1974.

3. AFR 800-14, Volume I. Management of Computer Resources in Systems. Washington: Department of the Air Force, September 1975.

4. AFR 800-14, Volume II. Acquisition and Support Procedures for Computer Resources in Systems. Washington: Department of the Air Force, September 1975.

5. AFSCP 800-7. Configuration Management. Washington: Department of the Air Force, December 1977.

6. Allred, 2Lt Doyle E. Configuration Management Officer (Status Accounting), ASD/A-10 SPO. (personel interview). Wright-Patterson AFB, OH, August 1978.

7. Boehm, B. W. "Software Engineering," IEEE Transactions on Computers, C-25(12): 1226-1241 (December 1976).

8. Boehm, B. W., et al. "Quantitative Evaluation of Software Quality," Proceedings, Second International Conference on Software Engineering. IEEE Cat. #76CH1125-4C. 592-605. San Francisco: IEEE, October 1976.

9. Boehm, B. W., et al. "Some Experiences With Automated Aids to the Design of Large-Scale Reliable Software," Proceedings, 1975 International Conference on Reliable Software. IEEE Cat. #75CH0940-7CSR. 105-113. NY: ACM, April 1975.

10. CODASYL. CODASYL Data Base Task Group Report. New York: ACM, 1971.

11. Constantine, Larry L. and Edward Yourdan. Structured Design (Second Edition). New York: YOURDAN Press, 1978.

12. Douglas, Capt Frank E. Software Configuration Manager, ASD/Simulator SPO. (personal interviews). Wright-Patterson AFB, OH, 1978-1979.

13. Gilb, T. Software Metrics. Cambridge, Mass.: Winthrop Publishers, Inc., 1976.

14. Goodenough, J. B. and Douglas T. Ross. The Effect of Software Structure on Reliability, Modifiability, Reusability, Efficiency: A Preliminary Analysis. Report R-2099. Philadelphia: Frankford Arsenal, December 1973. (AD-780841).

15. Hamilton, M. and S. Zeldin. "Higher-Order Software - A Methodology for Defining Software," IEEE Transactions on Software Engineering, 2(1):9-32 (March 1976).

16. Kroenke, David. Database Processing: Fundamentals, Modeling, Applications. Chicago: Science Research Associates, Inc., 1977.

17. Kuhnert, Robert. Configuration Management Specialist, ASD/Simulator SPO. (personal interviews). Wright-Patterson AFB, OH, 1978-1979.

18. Lehman, John H. "How Software Projects are Really Managed," Datamation: 119-129 (January 1979).

19. Lipow, M. Airborne Systems Software Acquisition Engineering Guidebook for Quality Assurance. ASD-TR-78-8. Wright-Patterson AFB, OH: Aeronautical Systems Division, November 1977. (AD-A059068).

20. Martin, James. Computer Data Base Organization (Second Edition). Englewood Cliffs: Prentice-Hall, Inc., 1977.

21. McGowan, Clement L. and John R. Kelly. "A Review of Some Design Methodologies," Info Tech State of Art Conference on Structured Design. TP053. Amsterdam: Softech, Inc., October 1976.

22. Mills, H. D. "How to Write Correct Programs and Know It," Proceedings, 1975 International Conference on Reliable Software. IEEE Cat. #75CH0940-7CSR: 363-370. Los Angeles: ACM, April 1975.

23. MIL-STD-480. Configuration Control - Engineering Changes, Deviations, and Waivers. Washington: Department of Defense, October 1968.

24. MIL-STD-482A. Configuration Status Accounting Data Elements and Related Features. Washington: Department of Defense, April 1974.

25. MIL-STD-483(USAF). Configuration Management Practices for Systems, Equipment, Munitions, and Computer Programs. Washington: Department of the Air Force, December 1970.

26. MIL-STD-490. Specification Practices. Washington: Department of Defense, October 1968.

27. Myers, Glenford J. Reliable Software Through Composite Design. New York: Petrocelli/Charter, 1975.

28. Myers, Ware. "The Need for Software Engineering," Computer. 11(2):12-26 (February 1978).

29. Parnas, D. L. "The Influence of Software Structure on Reliability," Proceedings, 1975 International Conference on Reliable Software. IEEE Cat. #75CH0940-7CSR: 358-362. Los Angeles, April 1975.

30. Reeve, Capt William H. and Capt Jerry L. Stinson. Software Design for a Visually-Coupled Airborne Systems Simulator (VCASS). AFIT/GCS/EE/78-6. (Master's thesis). Wright-Patterson AFB, OH: (1978).

31. Regulinski, Thaddeus L. Professor, School of Engineering, Air Force Institute of Technology, Wright-Patterson AFB, OH, 1978 (Lecture notes, class EE 7.53).

32. Romanick, Joseph. Systems Management Engineer, ASD/Simulator SPO. (personal interviews). Wright-Patterson AFB, OH, 1978-1979.

33. Schoeffel, W. L. An Air Force Guide to Software Documentation Requirements. ESD-TR-76-159. Hanscom AFB, MA: Electronic Systems Division, June 1976. (AD-027051).

34. Searle, Lloyd V. An Air Foce Guide to Computer Program Configuration Management. ESD-TR-77-254. Hanscom AFB, MA: Electronic Systems Division, August 1977.

35. Softech, Inc. An Introduction to SADT. 9022-78R. Waltham, MA: Softech, Inc., November 1976.

36. -------. Structured Analysis Reader Guide. 9022-73.2. Waltham, MA: Softech, Inc., May 1975.

37. Technical Order TO-00-20-4. Configuration Management Systems. Washington: Secretary of the Air Force, April 1976.

38. Thayer, T. A., et al. Software Reliability Study. RADC-TR-76-238. Griffiss AFB, NY: Rome Air Development Center, August 1976.

39. Weiss, David M. The Mudd Report: A Case Study of Navy Software Development Practices. NRL Report 7909. Washington: Naval Research Laboratory, May 1975.

40. Yau, S. S., et al. "An Approach to Error-Resistant Software Design," Proceedings, Second International Conference on Software Engineering. IEEE Cat. #76CH1125-4C. 429-436. San Francisco: IEEE, October 1976.

41. Zamadics, 2Lt Robert S. Retrofit Management Officer, ASD/F-15 SPO. (personal interview). Wright-Patterson AFB, OH, 1978.

42. Zelkowitz, Marvin V. "Perspectives on Software Engineering," Computing Surveys, 10(2) (June 1978).

Appendix A

STRUCTURED ANALYSIS DIAGRAMS

(excerpted with permission from Ref 30)

Introduction

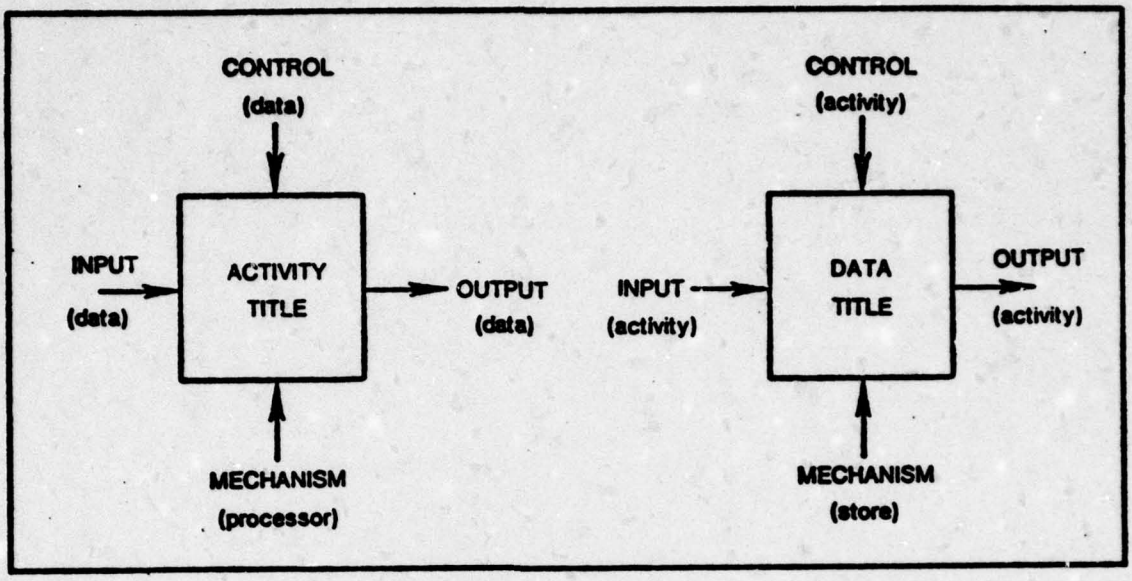
This appendix contains a brief explanation of the functional analysis phase of structured analysis to aid the reader in understanding the development of the design structure. A more detailed discussion can be found in Ref 35.

Structured Analysis

Structured analysis is a comprehensive methodology for performing functional analysis and design. In the functional analysis phase, the emphasis is on analyzing and documenting "what" the system is supposed to do. Two sets of diagrams result: one describes the system in terms of activities, the other describes the system in terms of data. These diagrams are created by decomposing the system into smaller and smaller pieces. The two sets of diagrams are cross-checked and sequenced to provide a model of the system functions.

Diagram Syntax

Structured analysis diagrams consist of labeled boxes and arrows for expressing the system activity and data models. Figure A-1 illustrates the basic syntax of the models. Inside a box is the name of the activity model or data model. For an activity model the name expresses the action taking place. For a data model the name is a noun or noun phrase expressing the data item.



A-1 Box/Arrow Interface Conventions

The boxes of the parent diagram are decomposed into more detailed diagrams called children. Each diagram is numbered in a Dewey-decimal manner (Ref 35:2-3), which represents the parent-child relationship. Diagrams 311, 312, 313, and 314 would be children of diagram 31. Each diagram is referenced as a node.

The boxes of a diagram are connected by arrows which represent the interface between the boxes. The sides of the box defines the kind of arrow(s) which may enter or leave that side of the box.

Four types of arrows represent the kinds of interface. As in reference 35:3-6, these are:

A. Activity Diagrams:

1. Input: data transformed by the activity into the output.

2. Output: Data created by the activity.

3. Control: Data used to control the process of converting the input into output.

4. Mechanism: The processor which performs the activity (person computer, program, etc.)

B. Data Diagrams:

1. Input: Activity which creates the data.

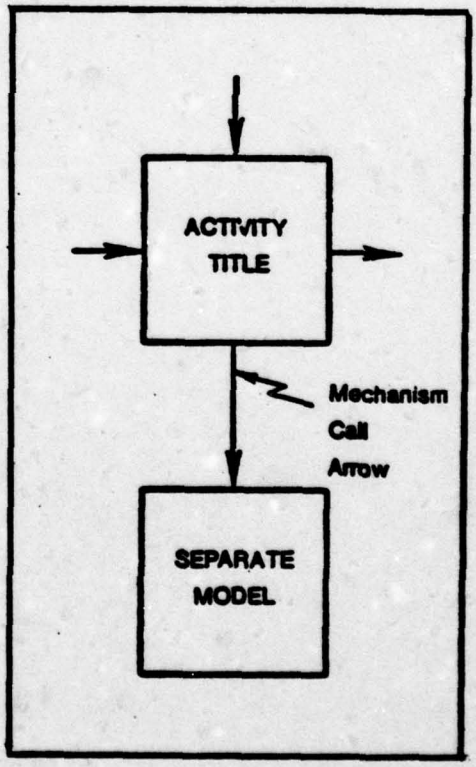
2. Output: Activity which uses the data.

3. Control: Activity which controls the creation or use of the data.

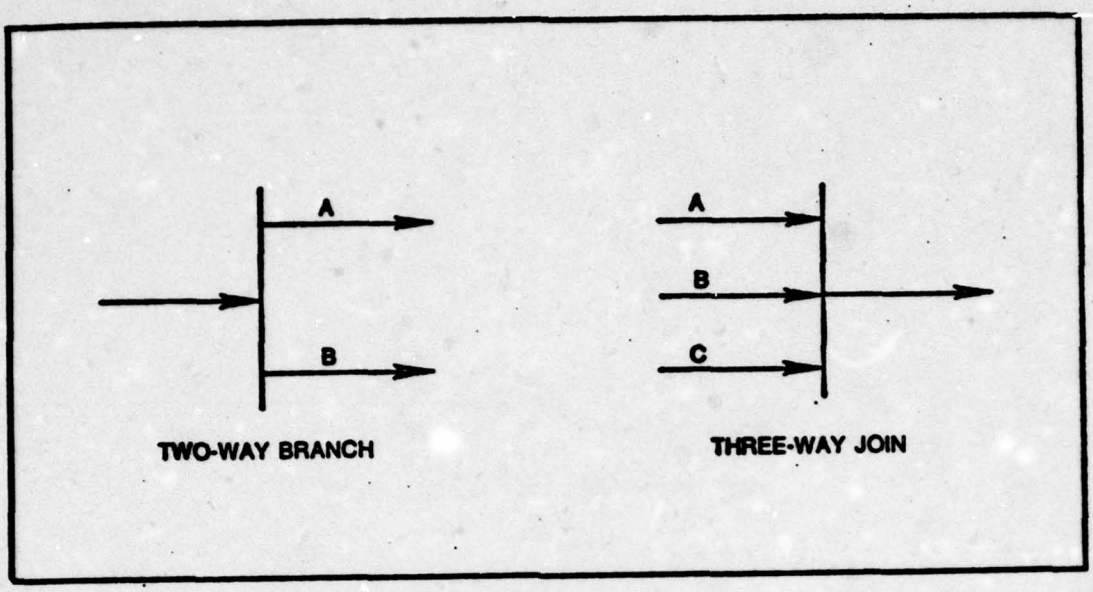
4. Mechanism: The storage device used to hold the data (buffer, computer memory, etc.)

It should be noted that the "mechanism" arrow represents the tool necessary to "realize the box" (Ref 35:3-4); since it is usually evident from the title of the box the "mechanism" arrow is not always shown.

The "mechanism call" arrow points downward. This indicates a separate system which completely performs the function of the box. In such cases there will be no child diagram of the box and its



A-2 Mechanism Call Arrow



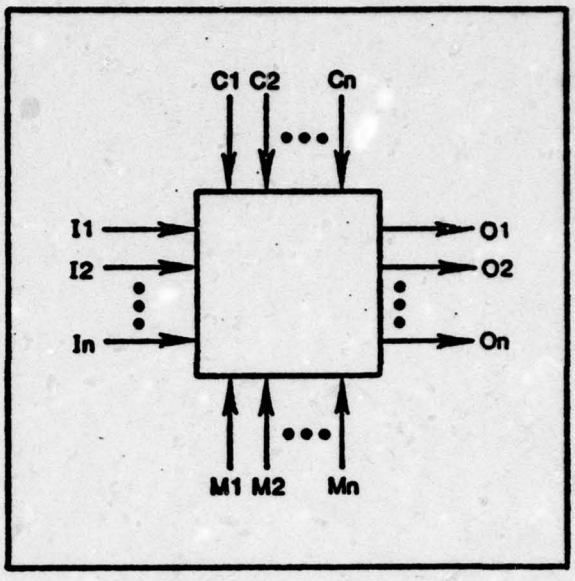
A-3 OR Branch and Join Structure

detail would be found in a separate model of the mechanism. This "mechanism call" is illustrated in Figure A-2.

The "multiple branch" (EXCLUSIVE OR) is used to indicate multiple, but not simultaneous, outputs. The "multiple join" indicates multiple, but not simultaneous, inputs. Both conventions are illustrated in Figure A-3.

An "ICOM Code" is used to connect arrows across the parent/child boundaries. The name ICOM is derived from the arrow names: input, control, output, and mechanism. Each boundary crossing arrow (ones which do not have both ends connected to a box) is labeled with its parent-context ICOM code, in addition to its normal label. This aids the reader in locating the matching parent arrow. The "ICOM Code" is written near the unconnected end of the arrow and consists of the letter I, C, O, or M followed by a number. This

number gives the relative position that the arrow enters or leaves the side of the parent box. Numbering is done from left to right and top-to-bottom as illustrated in Figure A-4. For example, "C2" on an arrow in a child diagram indicates the arrow is the second control arrow entering the parent box.



A-4 ICOM Numbering Convention

In the text associated with each diagram, the arrows are identified with an "ICOM code" consisting of a letter (*I, O, C or M*), *a prefix number, and a suffix number.* The prefix number refers to the box within the diagram and the suffix number refers to the top-down or left-right order of the arrow on the box. For example, "2C3" means the third control going into box 2 on the diagram.

Reading Sequence (Ref 35:4-2)

The following sequence is suggested for reading each diagram in a top-down order.

1. Scan only the boxes of the diagram to gain a first impression of its decomposition.

2. Using the parent diagram, rethink the message of the parent, observing the arrows feeding to and from the current diagram.

3. Referring back to the current diagram, see how and where each arrow from the parent context attaches to the factors in the current diagram: using ICOM codes.

4. Consider the internal arrows of the current diagram to see how it works in detail. Consider the boxes from top to bottom and from left to right. Examine the arrows by going clockwise around each box.

5. Finally, read the text of the current diagram to confirm or to alter the interrelation gained from consideration of the diagrams themselves.

Appendix B

STRUCTURED DESIGN CHARTS

(excerpted with permission from Ref 30)

Introduction

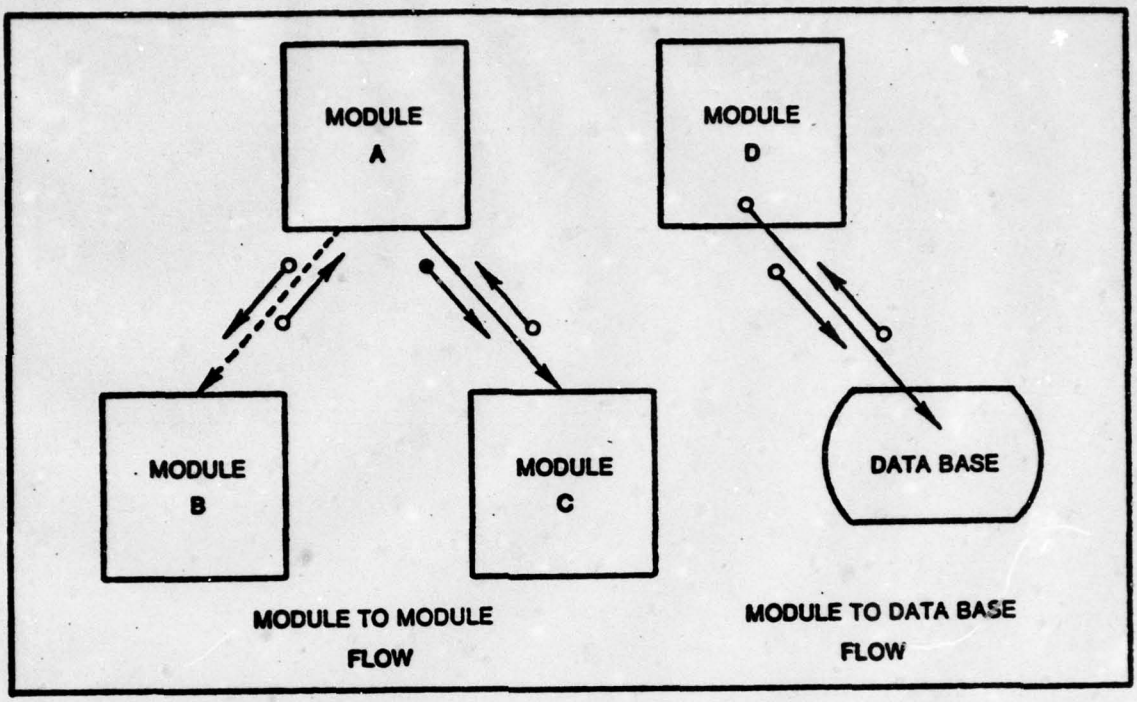
This appendix contains a brief explanation of structured design in order that the reader may understand the development and the reading of the structured charts. A more detailed discussion can be found in Ref 11.

Structured Design

Structured design is a set of general program design considerations and techniques for making coding, debugging, and modification easier, faster, and less expensive by reducing complexity. It simplifies the software by dividing it into modules in such a way that the modules can be implemented and modified with minimal effect on other parts of the system.

Structured design starts by stating a problem as a data flow graph, or bubble chart. A first cut obtained by factoring the bubble chart into a tree structure. The resulting design is then evaluated.

The criteria for evaluating the design are coupling and cohesion. Coupling is a measure of the relationship between modules, and cohesion is a measure of the relationships between modules, and cohesion is a measure of the relationships among elements in the same module. The objective is to minimize coupling and to maximize cohesion. Other



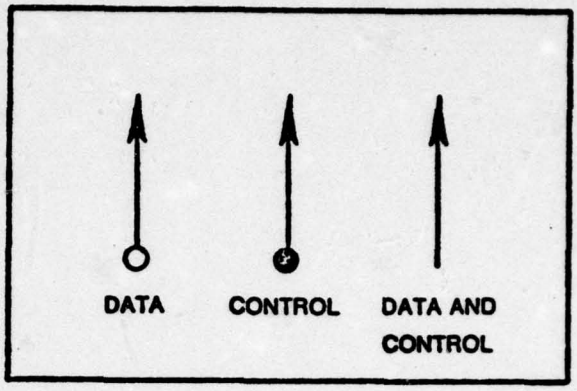
B-1 Information Flow

criteria to be considered are the scope-of-effect and the scope-of-control. Scope-of-effect is the collection of all modules containing any processing that is conditional upon a particular decision. Scope-of-control is a module and all of its subordinates. The objective is to have the scope-of-effect and the scope-of-control coincide.

Structured Chart Syntax

Structured charts consist of modules connected either to modules and/or to data bases by information flow lines or by connection lines. Figure B-1 illustrates the basic syntax for connecting modules and data bases. The solid line represents a normal connection. A dashed line represents a transfer of control which takes place automatically,

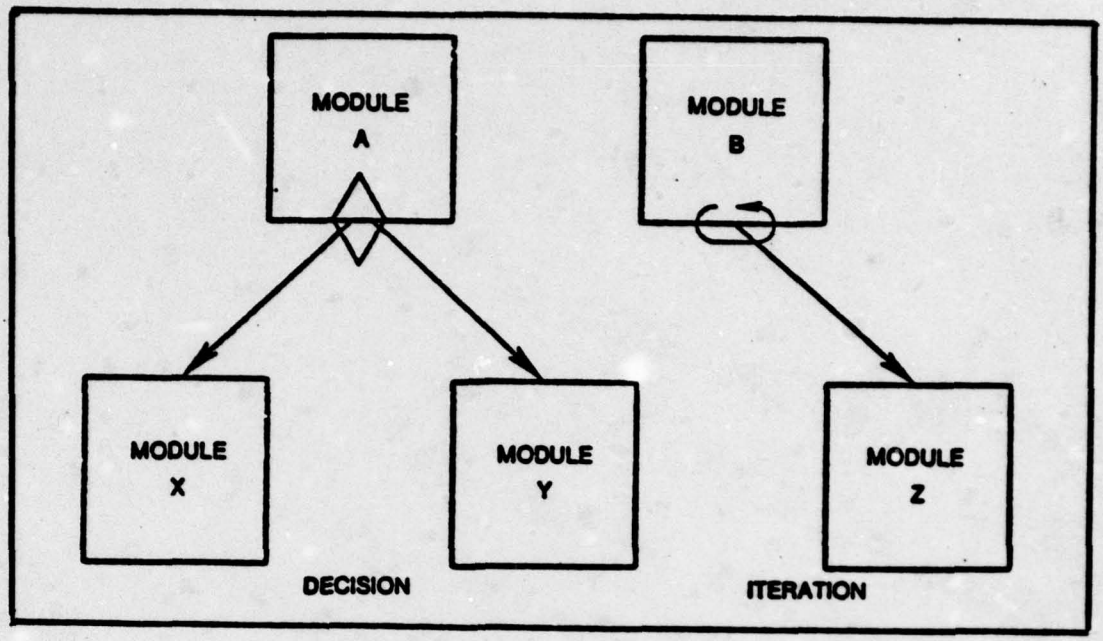
asynchronously or concurrently with established processes. This includes program interrupts and parallel subroutine calls. (Note that the data base is designated by a rectangle with curved sides.) Connections are ordered left-to-right as they emerge from the referencing moduel in the same order that they would usually be accessed. The arrows adjacent to any connection indicate the direction of the flow of information.



B-2 Information Arrow Types

The three types of arrows used are illustrated in Figure B-2. An arrow with a small circle on its tail always denotes data. An arrow with a dot (point) on its tail always denotes control. A plain arrow denotes either control, data or both.

Conditional access to intermodular connection is shown by enclosing the point(s) of reference in a diamond (decision symbol) as illustrated in Figure B-3. When intermodular references are used repeatedly within an iterative procedure (loop), the beginnings of the connection(s) are encompassed by a half loop as shown in Figure B-3.



B-3 Decision and Iteration

Appendix C

Database Record Descriptions and Data Item Definitions

ORG

ORG CODE	ORG NAME	ORG ADDR

Purpose: Identifies a Government organization responsible for computer resource acquisition

Figure C-1 ORG Record

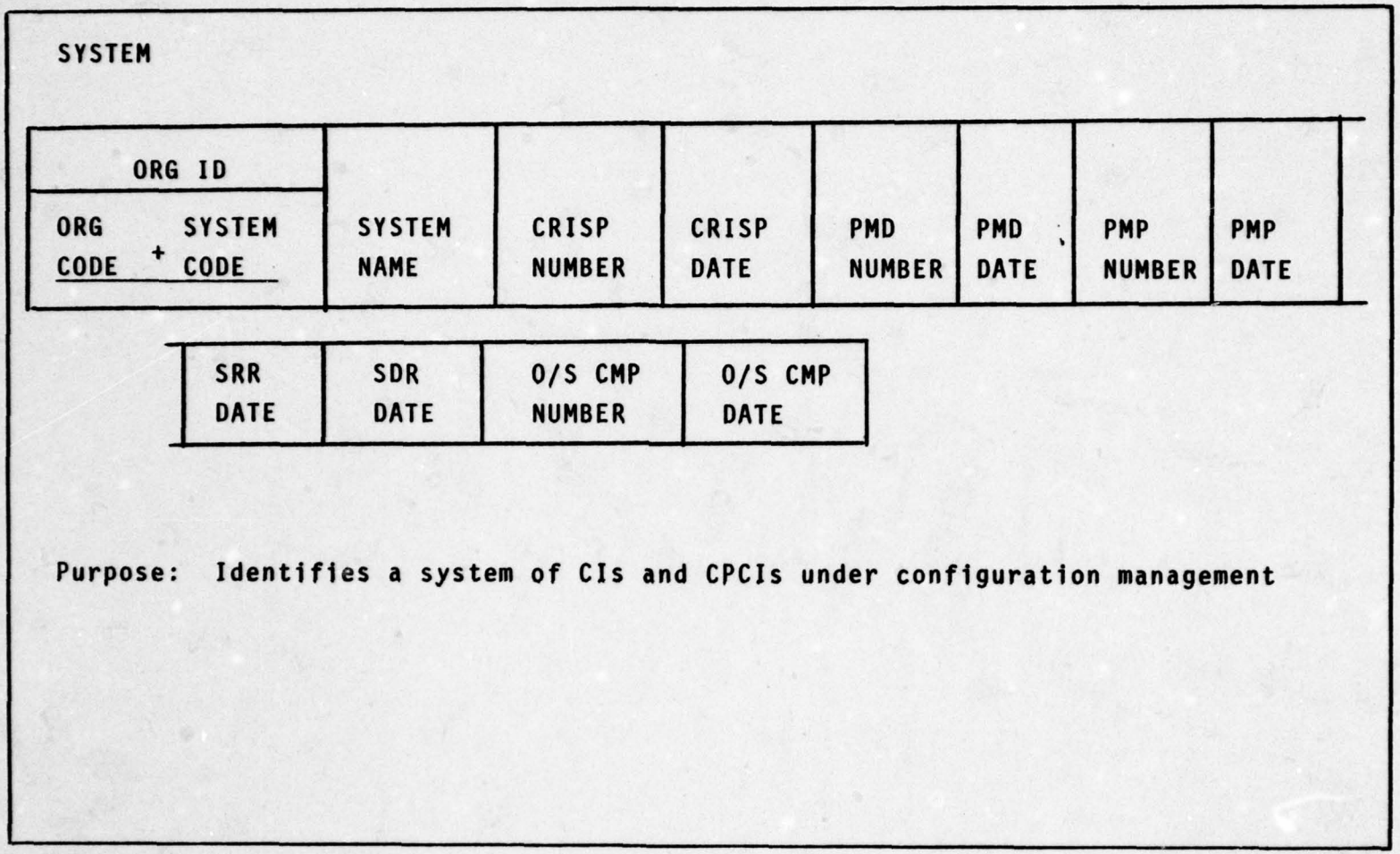


Figure C-2 SYSTEM Record

MANUFACTURERS

MANUFACTURER CODE	MANUFACTURER NAME	MANUFACTURER ADDR

Purpose: Identifies a manufacturer involved in a Government computer resource acquisition program

Figure C-3 MANUFACTURERS Record

PROCUREMENT INSTRUMENT

ORG CODE +	FISCAL YEAR +	PROCUREMENT INSTRUMENT TYPE +	PROCUREMENT INSTRUMENT SERIAL NUMBER	PROCUREMENT INSTRUMENT DATE

Purpose: Identifies a Government procurement instrument

Figure C-4 PROCUREMENT INSTRUMENT Record

CPCI

CPIN	CPCI NAME	CURRENT CPCI REV	CBLI SERIAL NUMBER	CPDP NUMBER	CPDP DATE	PDR DATE	CDR DATE	FCA DATE	PCA DATE	FQR DATE

Purpose: Identifies a computer program configuration item

Figure C-5 CPCI Record

OP-MODE

OP-MODE ID	
CPIN + OP-MODE CODE	OP-MODE NAME

Purpose: Identifies an operating mode for a group of computer program configuration item modules

Figure C-6 OP-MODE Record

CLASS

CLASSIFICATION ID			
OP-MODE ID	+	CLASSIFICATION CODE	CLASSIFICATION NAME

Purpose: Identifies a functional classification for program modules that support a particular CPCIs operating mode

Figure C-7 CLASS Record

SUB-CLASS

SUB-CLASSIFICATION ID		
CLASSIFICATION ID	+ SUB-CLASSIFICATION CODE	SUB-CLASSIFICATION NAME

Purpose: Identifies a functional sub-classification for program modules within a particular classification

Figure C-8 SUB-CLASS Record

FUNCTION

FUNCTION ID	
SUB-CLASSIFICATION ID + FUNCTION CODE	FUNCTION NAME

Purpose: Identifies a function performed by a group of program modules

Figure C-9 FUNCTION Record

MODULE

MODULE ID			
FUNCTION ID + MODULE CODE	MODULE NAME	MODULE DESCRIPTION	LANGUAGE

Purpose: Identifies and describes a CPCI program module

Figure C-10 MODULE Record

MODULE-REV

MODULE-REV-ID						
MODULE ID +	MODULE REV CODE	MODULE REV PRIORITY	MODULE REV DATE	SOURCE MEDIA TYPE	SOURCE MEDIA NUMBER	SOURCE LENGTH

OBJECT MEDIA TYPE	OBJECT MEDIA NUMBER	OBJECT MEDIA LENGTH	SOURCE MEDIA ADDR	OBJECT MEDIA ADDR	MODULE REV REMARKS

PURPOSE: Describes each revision of each module of a CPCI

Figure C-11 MODULE-REV Record

ACTIVE-ITEM

ITEM TYPE	+	ITEM IDENTIFICATION

Purpose: Identifies an item to be reported in the Red Flag and Activity section of the Computer Program Change Report

Figure C-12 ACTIVE-ITEM Record

CPCI-REV

CPCI-REV-ID	
CPIN + CPCI REVISION CODE	CPCI DISTRIBUTION DATE

Purpose: Identifies a revision to a CPCI

Figure C-13 CPCI-REV Record

DOC-TYPE

DOC TYPE CODE	DOC NAME

Purpose: Identifies a type of maintainable document

Figure C-14 DOC-TYPE Record

MAINT-DOC

DOC ID				
DOC TYPE CODE	+ DOC BASE NUMBER	+ DOC VOLUME NUMBER	DOC TITLE	DOC REVISION LEVEL

Purpose: Describes a maintainable document

Figure C-15 MAINT-DOC Record

MAINT-DOC-REV

DOC ID + DOC REV	DOC DUE DATE	DOC DISTRIBUTION DATE	DOC REMARKS	DOC REV CHANGE

Purpose: Identifies a revision to a maintainable document

Figure C-16 MAINT-DOC-REV Record

DOC-CONFIG

DOC-CONFIG-ID			
DOC REV ID	+	CN SERIAL NUMBER	CN DATE

Purpose: Completely identifies a maintainable document

Figure C-17 DOC-CONFIG Record

BASE-ECP

ECP BASE ID	
SYSTEM ID	+ ECP BASE NUMBER

Purpose: Identifies a set of related ECPs

Figure C-18 BASE-ECP Record

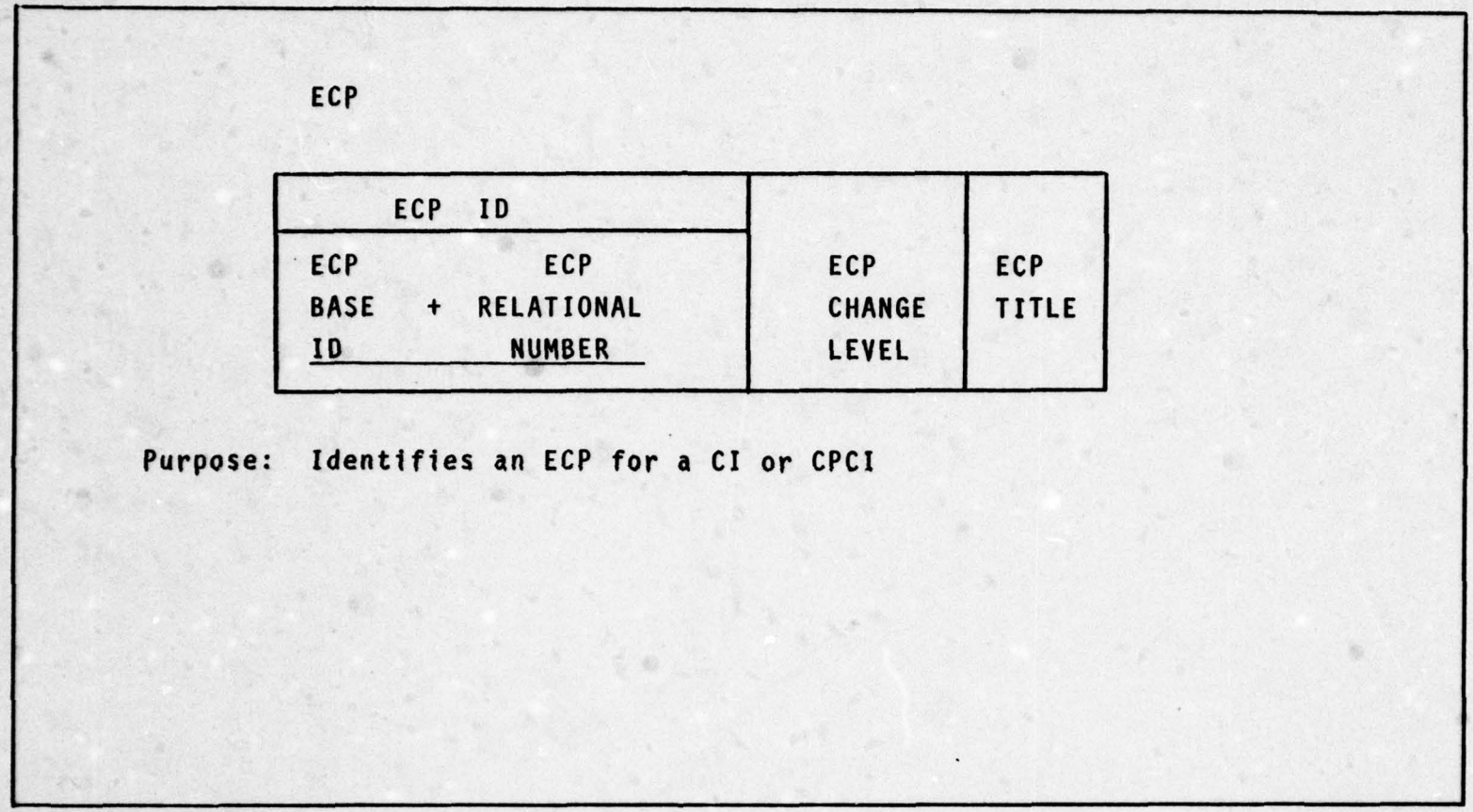


Figure C-19 ECP Record

ECP-REV

ECP REV ID					
ECP ID + ECP REVISION CODE	ECP CLASS	ECP TYPE	ECP STATUS	ECP REQUESTED AUTHORIZATION DATE	ECP JUSTIFICATION CODE

ECP PRIORITY	ECP CHANGE INCORP LEVEL	ECP DESC	ECP SOLUTION	ECP REMARKS

Purpose: Describes a revision to an ECP for a CI or CPCI

Figure C-20 ECP-REV Record

MILESTONES

MILESTONE CODE	MILESTONE NAME

Purpose: Identifies a type of milestone

Figure C-21 MILESTONES Record

CPCI-REV/DOC-CONFIG

CPCI-REV ID	+	DOC-CONFIG ID

Purpose: Provides the link between:

a. CPCI/revisions and their configuration identification documents; and,

b. A document and the CPCI revision it describes

Figure C-22 CPCI-REV/DOC-CONFIG Record

ECP-REV/DOC-CONFIG

ECP-REV ID	+	DOC-CONFIG ID

Purpose: Provides the link between:

a. An ECP/revision and a configuration identification document that it affects; and,

b. A configuration identification document and the ECP/revisions that are affecting it

Figure C-23 ECP-REV/DOC-CONFIG Record

ECP-REV/MILESTONES

ECP REV ID + MILESTONE CODE	ECP MILESTONE SCHED DATE	ECP MILESTONE ACTUAL DATE	ECP MILESTONE COMMENT

Purpose: Identifies the development schedule for each computer program engineering change proposed by a contractor

Figure C-24 ECP-REV/MILESTONES Record

MODULE-REV/CPCI-REV

MODULE-REV ID	+	CPCI-REV ID

Purpose: Provides the link between:

1. A module revision and the CPCI revisions in which it is incorporated; and,

2. A CPCI revision and all the module revisions that comprise it

Figure C-25 MODULE-REV/CPCI-REV Record

ECP-REV/DOC-CONFIG

ECP-REV ID	+	DOC-CONFIG ID

Purpose: Provides the link between:

1. An ECP revision and the documents that it affects; and,
2. A document and the ECP revisions that are affecting it

Figure C-26 MODULE-REV/DOC-CONFIG Record

MODULE-REV/ECP-REV

MODULE-REV ID	+	ECP-REV ID

Purpose: Provides the link between:

a. A program module revision and the ECP revisions that generated it; and,

b. An ECP revision and the program module revisions it caused

Figure C-27 MODULE-REV/ECP-REV Record

Table C-I

Data Item Definitions

Data Item	Description
CBLI Serial Number	Line item number on a contract
CDR Date	Date critical design review accomplished
Classification Code	Manufacturer's code to identify class of functions performed by program modules
Classification ID	Data aggregate that identifies a specific class of module functions for an organization
Classification Name	Name for a classification identifier
CPCI Distribution Date	Date a revision to a CPCI is distributed
CPCI Name	Name of a CPCI
CPCI Rev ID	Data aggregate that identifies a specific revision to a CPCI
CPCI Revision Code	Code to identify revisions to CPCI
CPDP Date	Date computer program development plan was distributed
CPDP Number	Unique number assigned to a computer program development plan
CPIN	Unique number assigned to a computer program configuration item
CN Date	Date of delivery of a change notice
CN Priority	Code that identifies the priority assigned to delivering a change notice
CN Serial Number	Sequence number assigned to a change notice

CRISP Date	Distribution date of computer resources integrated support plan
CRISP Number	Unique number assigned to a computer resource integrated support plan
DOC Base Number	Unique number assigned to a document or set of related documents
DOC Config ID	Data aggregate that uniquely identifies the configuration of a specific CPCI support document
DOC Distribution Date	Distribution date of a revision to a maintainable document
DOC Due Date	Due date of a revision to a maintainable document
DOC ID	Data aggregate that uniquely identifies a maintainable document (part of an item's configuration identification)
DOC Priority	Code that identifies the priority assigned to distributing a document
DOC Remarks	Narrative for a revision of a document
DOC Rev Change Level	Code that identifies the number of change notices issued against a revision of a document
DOC Rev ID	Data aggregate that uniquely identifies a revision to a specific maintainable document
DOC Revision Level	Code that identifies the last revision of a document that was issued
DOC Title	Title assigned to a specific document or set of documents
DOC Type Code	Code that uniquely identifies a type (class) of documents
DOC Type Name	Name of a type of document
DOC Volume Number	Identifier of a volume of a document (Note: The volumes of a document are treated as individual members of a set of documents)

ECP Base ID	Data aggregate that uniquely identifies a group of related ECPs for a specific system
ECP Base Number	Identifier of a group of related ECPs
ECP Change Incorp Level	Organizational level incorporating the change
ECP Change Level	Code that identifies the number of times an ECP has been revised
ECP Class	Number that identifies the change class to which the ECP has been assigned: I or II
ECP Description	Brief narrative identifying why the change is required
ECP ID	Data aggregate that uniquely identifies a proposed change to a CPCI or CI
ECP Justification Code	Code that identifies the reason for a proposed change
ECP Milestone Actual	Date that an ECP milestone was reached
ECP Milestone Comment	Brief narrative associated with an ECP milestone
ECP Milestone Name	Name of a milestone for an ECP
ECP Milestone Sched	Date by which an ECP milestone should be reached
ECP Priority	Code that identifies the priority of an ECP
ECP Relational Number	Number that is used to distinguish between members of a group of related ECPs
ECP Remarks	Narrative associated with a unique ECP revision
ECP Revision Code	Code that identifies a revision to an ECP

ECP Rev ID	Data aggregate that uniquely identifies an ECP revision
ECP Solution	Narrative describing a proposed change
ECP Status	Code that identifies the current status of an ECP
ECP Title	Title of an ECP
ECP Type	Code that identifies an ECP as preliminary or formal
FCA Date	Date that the functional configuration audit was accomplished
Fiscal Year	Number that identifies a fiscal year
FQR Date	Date that the formal qualification review was accomplished
Function Code	Manufacturer supplied code to identify a function performed by a CPCI module
Function ID	Data aggregate that uniquely identifies a function performed by a group of modules supporting a specific CPCI
Function Name	Name assigned to a function identifier
Item Identification	Code that uniquely identifies a status accounting data item
Item Type	Code that identifies a status accounting data item type
Language	Code for the programming language in which a module is written
Manufacturer Addr	The address of a computer resource supplier
Manufacturer Code	Code that uniquely identifies a Government supplier
Manufacturer Name	Name of a Government supplier

Milestone Code	Code that uniquely identifies a milestone
Milestone Name	Name assigned to a particular milestone
Module Code	Manufacturer supplied code to identify a program module
Module Description	Brief description of the purpose of a program module
Module ID	Data aggregate that uniquely identifies a program module being acquired by a program office
Module Name	Name assigned to a program module
Module Rev ID	Data aggregate that uniquely identifies a revision to a CPCI program module
Module Rev Code	Code that identifies a revision to a program module
Module Rev Date	Date that a revision to a program module was distributed
Module Revision Priority	Code that identifies the priority assigned to delivering a revision to a program module
Module Rev Remarks	Narrative for comments about a program module revision
Object Length	The number of bytes required to store a program module in object form
Object Media Address	Logical storage device addr of a program module a in object form
Object Media Number	Identifier of a storage device containing a program module in object form
Object Media Type	Type of storage media containing a program module in object form
Operating Mode Code	Manufacturer supplied code to identify an operating mode in which a CPCI can execute

Op Mode ID	Data aggregate that uniquely identifies an operating mode for each of an organization's CPCIs
Organization Address	Address of a Government organization procuring a system
Organization Code	Code that uniquely identifies a Government organization procuring a system
Organization Name	Name of a Government organization procuring a system
O/S CMP Date	Date operational/support configuration management procedures were distributed for a system being acquired
O/S CMP Number	Unique number assigned to an operational/support config mgmt procedures document
PCA Date	Date physical configuration audit accomplished
PMD Date	Date program management directive was distributed
PMD Number	Unique number assigned to a program management directive
PMP Date	Date program management plan was distributed
PMP Number	Unique number assigned to program management plan
Procurement Instrument Serial Number	Sequential number assigned to a Government procurement instrument
Procurement Instrument Type	Type of agreement between the Government and the supplier
SDR Date	Date system design review accomplished
Source Length	Number of bytes needed to store a program module in source form
Source Media Addr	Logical storage device addr of a program module in source form

Source Media Number	Identifier of a storage device containing a program module in source form
Source Media Type	Type of storage media containing a program module in source form
SRR Date	Date system requirements review accomplished
Sub-classification code	Manufacturer supplied code to identify a sub-class of functions performed by a program module
Sub-classification ID	Data aggregate that identifies a specific sub-class of module functions for an organization
Sub-classification Name	Name for a sub-classification identifier
System Code	Code that identifies an acquisition program for computer resources
System ID	Data aggregate that uniquely identifies a system being acquired
System Name	Name for a system being acquired

Vita

Alexander Schuster was born in New York, New York on 8 August 1951. Upon being graduated in 1969 from Cardinal Hayes High School, Bronx, New York, he entered Manhattan College, Bronx, New York and enrolled in their four year Air Force ROTC program. He was graduated in May 1973 after having earned a Bachelor of Science degree in Mathematics, and he was subsequently commissioned as a Second Lieutenant in the USAF in July. Before coming to the Air Force Institute of Technology in August 1977, he served for four years at HQ SAC, Offutt AFB, Nebraska, where he worked as a programmer, systems analyst, and configuration manager of software for the Force Management Information System.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GCS/EE/79-2	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design for an Automated Status Accounting System for Software Configuration Management		5. TYPE OF REPORT & PERIOD COVERED MS Thesis 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Alexander Schuster Captain USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, OH 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Simulator System Program Office ASD/SD24T Wright-Patterson AFB, OH 45433		12. REPORT DATE March 1979 13. NUMBER OF PAGES 245
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLAS 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

Approved for public release; IAW AFR 190-7

JOSEPH P. HIPPS, Major, USAF
Director of Information

16 MAY 1979

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Computer Program Design
Configuration Status Accounting
Software Engineering
Configuration Management

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This report documents a design effort for an automated system to record and report the configuration status of software for Air Force embedded computer systems. The work included requirements analysis, software design, and data-base design. Because of the flexibility given to program managers in tailoring their reporting requirements and in selecting the data "integator", only the requirements of a single AFSC program office were presented in detail. However, the requirements of other offices were considered as well. Current software engineering techniques were used to derive the design. They are highly

Next Page

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

recommended for use on other software development projects.