

AD-A056 517

AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 1/3
SOFTWARE STRUCTURE DESIGN TECHNIQUES APPLIED TO EMBEDDED COMPUT--ETC(U)
DEC 77 R J DE SANTO
AFIT/GE/EE/77-14

UNCLASSIFIED

NL

1 of 2
AD
A056 517



AD No. _____
DC FILE COPY

AD A056517

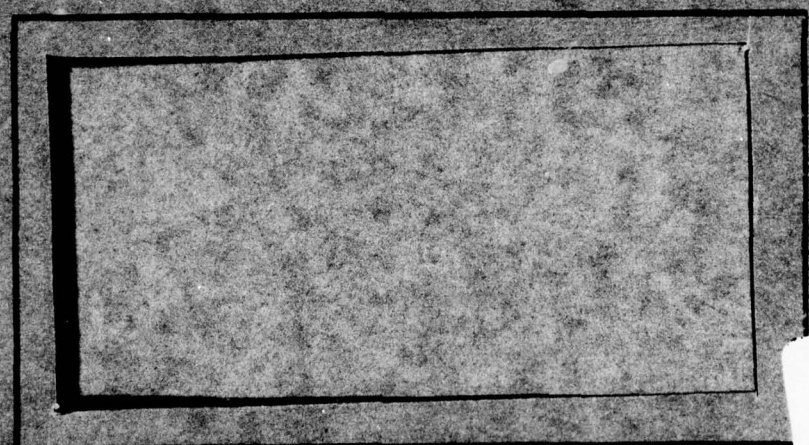
LEVEL *II*

Q1

AIR FORCE INSTITUTE OF TECHNOLOGY



AIR UNIVERSITY
UNITED STATES AIR FORCE



This document has been approved
for public release and sale; its
distribution is unlimited.

DDC
RECEIVED
JUL 20 1978
F

SCHOOL OF ENGINEERING

WRIGHT-PATTERSON AIR FORCE BASE, OHIO

78 07 07 011

AFIT/GE/EE/77-14

LEVEL II

1

AD A056517

AD No. _____
DDC FILE COPY

DDC
RECEIVED
JUL 20 1978

6 SOFTWARE STRUCTURE DESIGN
TECHNIQUES APPLIED TO EMBEDDED
COMPUTER SYSTEM SOFTWARE,

THESIS

14 AFIT/GE/EE/77-14 Robert J. De Santo
Captain USAF

9 Master's Thesis,

11 Dec 77

12 194p.

Approved for public release; distribution unlimited.

78 07 07 011

012225.

cc

SOFTWARE STRUCTURE DESIGN TECHNIQUES
APPLIED TO EMBEDDED COMPUTER
SYSTEM SOFTWARE

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

by

Robert J. De Santo, B.S.E.

Captain

USAF

Graduate Electrical Engineering

December 1977

Approved for public release; distribution unlimited.

Preface

This report summarizes an investigation which examined the structured design methodology applied to an operational flight program (OFP). Several design techniques are successfully applied to a particular OFP. Hypothetical software modifications illustrate the advantages of structured design OFP software over a current software implementation. Software life-cycle effects of structured designed OFPs are briefly described. An OFP familiarization and design methodology is developed for avionics software engineers/contract monitors. The report is written for a reader who possesses a basic knowledge of software development and who understands the structured design methodology.

I wish to express my special thanks to Captain James B. Peterson for his advice and leadership as my thesis advisor. A special thanks also goes to my thesis sponsor, Mr. Ajmel Dulai, for his patience and vast knowledge of avionics concepts. I also wish to express my appreciation to Dr. Gary Lamont and Professor Charles Richard for their advice and support, and to Mrs. Joyce Clark for the excellent job typing the final version of this thesis. Finally, I wish to thank my wife, Pat, and our children for their patience, understanding and love throughout this thesis endeavor.

Robert J. De Santo

ADDITIONAL	1 - 1st Edition	☒	☐	☐
NTIS	2 - 2nd Edition	☐	☐	☐
DDC	3 - 3rd Edition	☐	☐	☐
UNANNOUNCED				
JUSTIFICATION				
BY	DISTRIBUTION/AVAILABILITY CODES			
Dist.	SPECIAL			
A				

Contents

	Page
Preface	ii
List of Figures	vi
List of Tables	vii
Abbreviations	viii
Abstract	ix
I. Introduction	1
Background	1
General	1
Design Approaches	1
Typical Software Problems	2
Purpose	3
Scope	4
Preliminary Rationale	5
Assumptions	6
Development Plan	6
II. Toward the Structuring of Computer Software . . .	8
Introduction	8
The Problem/Coding Syndrome	8
Requirements Engineering	11
Architectural Philosophy	12
The Technique of Structured Design	13
Summary	15
III. The PAVE TACK Operational Flight Program . . .	17
Introduction	17
General	17
Subsystem Interfaces	18
Executive Structure	20
Summary	22

Contents

	Page
IV. Application of Design Techniques	23
Introduction	23
Initial Approach	23
Objective	23
Application of Transform Analysis	24
Revised Approach	28
Objectives	28
Finite-State Technique	28
Transform Analysis Technique	32
Transform/Transaction Analysis Technique	47
Comments	52
Summary	52
V. Comparison of Modification and Life-Cycle Effects Associated with OFP Software	54
Introduction	54
IDS Image Derotation Task Comparison	54
Current Software	54
Revised Design	55
Sample Modification	55
Comment	61
Slant Range Modification Comparison	61
Sample Modification	61
Current Software	61
Revised Design	62
Short Term Effects of Structured Designed OFPs	62
Long Term Effects of Structured Designed OFPs	63
Summary	63
VI. A Software Engineering OFP Design Methodology	65
Introduction	65
General	65
Requirements for Methodology Development	66
Design Methodology	67
Phase I	68
Phase II	69
Phase III	72
Limitation	73
Summary	73

Contents

	Page
VII. Results, Conclusions, and Recommendations	75
Introduction	75
Results	75
Conclusions	76
Recommendations	78
Bibliography	80
Appendix A: Brief Discussion of Operational Flight Program Concepts	84
Appendix B: Bubble Chart Symbol Descriptions	86
Appendix C: Structure Chart Symbol Descriptions	87
Appendix D: Symbols and Notation for Use with Finite- State Technique	89
Vita	91

List of Figures

Figure		Page
1	Software Life-Cycle	9
2	PAVE TACK OFP Subsystem Interfaces	19
3	Mainloop State Diagram by Mode	31
4	IDS Image Derotation First-Cut Bubble Chart	35
5	IDS Image Derotation First-Cut Structure Chart	37
6	Alternate Afferent Branch	43
7	Revised Afferent Branch	43
8	IDS Image Derotation Intermediate Structure Chart	45
9	IDS Image Derotation Intermediate Bubble Chart	46
10	Bubble Chart for "Perform Target Operations"	49
11	Bubble Chart for "Perform Search Functions"	49
12	Partial Structure Chart for "Perform Tracking Operations"	51
13	Simplified Flowchart for IDS Image Derotation	56
14	Modified Flowchart for IDS Image Derotation	58

List of Tables

Table		Page
I	Mainloop State Transitions by Mode	31
II	First Cut Structure Chart Parameters	39

Abbreviations

ADP	Automatic Data Processing
AFSC	Air Force Systems Command
C	Cue (Mode)
CALCS	Calculations
DIWX	Discrete Input-Word x
DIW2	Discrete Input Word Two
DIW4	Discrete Input Word Four
EO	Electro-Optical
HIPO	Hierarchy Plus Input-Process-Output
HORZ-NAT	Horizontal Natural
HOS	Higher Order Software
IDS	Infrared Detecting Set
IPO	Input-Process-Output
LOS	Line-of-Sight
OFF	Operational Flight Program
PDR	Preliminary Design Review
S	Search (Mode)
T	Track (Mode)
WFOV	Wide Field of View

Abstract

Constantine's structured design methodology is applied to a real-time (flight) software program. A modified finite-state technique is successfully applied to an operational flight program (OFP) mainloop. Transform and transaction analysis are successfully applied to mainloop tasks. Each technique is demonstrated to show avionics software engineers/contract monitors "how to get started" on a design. Two hypothetical software modifications illustrate advantages of structured design over the current software implementation. Short and long term effects of structured designed OFPs are briefly described. An OFP familiarization and design methodology is developed for avionics software engineers/contract monitors.

Structured design techniques are beneficial to the software engineer/contract monitor during initial understanding of OFP design requirements from a draft Part I specification. Design alternatives may be considered and the software producers interpretation of design requirements may be verified. This effort was sponsored by the Aeronautical Systems Division (ASD/ENALA) located at Wright-Patterson Air Force Base, Ohio.

SOFTWARE STRUCTURE DESIGN TECHNIQUES APPLIED TO EMBEDDED COMPUTER SYSTEM SOFTWARE

I. Introduction

Background

General. In recent years, sophisticated and expensive Air Force Computer programs (software) used in avionics systems have been relatively inflexible, difficult to repair, and have required extensive updates to keep pace with state-of-the-art improvements in computer technology and changing operational requirements. Normally, in a software development program, deadlines and cost are the major concerns. As a result, the importance of the design phase is greatly de-emphasized. There is a great tendency to start coding software at the beginning of the project. Since the design phase is, in many cases, "cut short," errors result which remain unnoticed until after system implementation. In addition, there is usually little concern about future repair and update (maintenance) requirements, and the efficiency and ease with which that software maintenance can be performed.

Design Approaches. Some avionics software designs have been accomplished with a "bottom-up" approach by developing software before addressing functional interface and integration problems. Recently, more successful designs have been accomplished with a

"top-down" approach in which the "top" is assumed to be a firm, fixed requirements specification and hardware architecture. It had been assumed that the data structure was already established. In some cases, these types of assumptions change, and the designer must then back-track to update his design.

More recently, a software design methodology called "structured design" has received much attention. This methodology, which produces a modularized software design, allows easier system implementation and maintenance than previous approaches permitted. Cost effectiveness is usually improved with the use of this design method, and program complexity is reduced (Ref 40). There has been an attempt to apply this design methodology to a real-time system (Ref 20). At the time of this writing, the author is unaware of any efforts to apply this methodology to operational flight software.

Typical Software Problems. Avionics software, like automatic data processing (ADP) software, is not without its share of problems. In fact the typical problems associated with both types of software are very much alike. In the avionics software, there are similar indications of high project costs and low or poor software reliability (Ref 10:477, 479; and 36:228). However, there are other problems with avionic software which need to be discussed.

Recently, it has been suggested that avionics software is being used like a "band-aid." That is, the lack of early specified, firm, explicit requirements contributes to the idea that avionics software

can be used to "fix" all the marginal hardware performance (Ref 10:477). In general, engineering management should be performed on avionics software to the level normally expected on any hardware project (Ref 10:478).

The uncertainty of software module interfaces in a large program causes considerable anxiety for managers. Problems frequently occur here, and module design or debugging may require several iterations. Designing a set of "milestones" may help monitor project status, but it appears to be beneficial if a hierarchical structure chart of the modules is constructed as early as possible. Module interfaces should also be defined. This approach helps in evaluating design alternatives (Ref 10:478).

Normally, most avionics software lacks the flexibility of being reuseable. Software procurement for avionics usually results in new software for each new flight computer. This redevelopment procedure is extremely costly. Most software can not be readily changed to meet the everchanging threat environment. It is unmaintainable by the government, and difficult to maintain even by the original vendor (Ref 36:228-229).

Purpose

The purpose of this report is to present the results of an investigative effort to apply the structured design methodology to part of an operational flight program. Computer software developed for avionics

software systems by this design method should be easier to implement and understand. Complexity should be reduced, and the software should be less difficult to maintain and more cost-effective with respect to life-cycle costs. A generalized approach or methodology for the design of avionics operational flight software will be developed. It will include the use of structured design. The objective of the methodology will be to provide Air Force avionics software engineers/contract monitors with an additional method to understand the requirements definition for preliminary design and management of operational flight software.

Scope

In order to more realistically investigate structured design applied to avionics operational flight software, it would be helpful to investigate a real system. The PAVE TACK operational flight software (see Chapter III) draft Part I specification will be used. Due to the complexity of the PAVE TACK requirements and the length of time permitted for this study, the structured design methodology will not apply to the entire PAVE TACK flight program software.

Since structured design is well documented (Ref 34; 40), this study will be limited by purposefully halting the design process at certain points with the specific purpose of showing the software engineer "how to get started" on a design. This approach allows several design techniques to be used. These techniques are known

and well documented. Therefore, the techniques will be demonstrated, not taught.

The draft and final PAVE TACK preliminary design (Part I) specifications will be used. No attempt will be made to modify either specification or produce any program code. Testing of any redesigned software and access to the PAVE TACK computer hardware will not be required.

Preliminary Rationale

It was desirable to have a clear and concise operational flight program (OFP) design (Part I) specification at the start of this investigative effort. The purpose of this specification is to supply the designer with a comprehensive and unambiguous software requirements definition for the OFP. Thus, all appropriate and necessary information would be available to the software designer.

It was anticipated that a reasonable learning curve would be experienced during the initial phases of the design investigation. There was previous personal experience with defining and designing from written software requirements, but no experience with avionics software and no prior knowledge of OFP concepts. In addition, there was no previous personal experience with AFSC Part I specifications per se.

Finally, it was not known whether or not structured design could be applied to an OFP or any part of an OFP. If structured design

techniques could be used, then another aid would exist which would allow avionics software engineers/contract monitors to cope with the problems of interpreting OFP software design specifications. In addition, it would aid in making better decisions and suggestions about design alternatives.

Assumptions

The reader should be familiar with the basic concepts of an OFP. However, if this is not the case, Appendix A briefly discusses basic concepts. This brief discussion on flight software concepts (Appendix A) was included since many people are interested in software design, but avionics flight software is quite different from normal data processing software applications. It is assumed that the reader understands the structured design methodology, its techniques, and its principles.

Finally, any statements made by the author concerning flight software, PAVE TACK, structured design, Air Force specifications, or anything associated with this research effort is strictly the author's interpretation of that information and bears no official judgment on behalf of the Air Force. The author bears sole responsibility for this research and report effort.

Development Plan

Chapter II briefly discusses some problems associated with Air Force software development, concepts of a software architecture for

flight program managers, and presents a description of the structured design methodology.

Chapter III presents a brief description of the PAVE TACK operational flight program.

Chapter IV discusses the initial approach taken for this investigation, difficulties encountered, and how the approach was revised and applied to achieve meaningful results.

Chapter V presents a discussion on the effects of two hypothetical software modifications to certain parts of the designs from Chapter IV and the current PAVE TACK software. Short and long term effects associated with OFP's designed using structured design techniques are also discussed.

Chapter VI presents an OFP design methodology developed for avionics software engineers and contract monitors. It is a generalized design approach which consists of three phases: (1) understanding the flight program requirements; (2) use of a technique for understanding flight program tasks and task sequencing; and (3) use of structured design techniques for task design.

Finally, Chapter VII presents results and conclusions of this investigation. Some recommendations are also made for future efforts.

II. Toward the Structuring of Computer Software

Introduction

This chapter briefly discusses certain problems associated with Air Force software development. Some concepts for flight program managers about software architecture are also presented. Finally, a description of a recent software design technique which improves the architectural design of software and aids in reducing life-cycle costs is presented. The information discussed in this chapter can be applied to software in general (e.g. algorithm development).

The Problem/Coding Syndrome

From the very beginning, a person learning to program a computer is taught to understand the problem to be solved, devise a solution algorithm, and finally, code that algorithm in a suitable programming language. Since this methodology is taught at a basic level, it seems to stay with a person forever. For many years this problem/coding tradition has been a part of the software life-cycle.

The software life-cycle is well known by software professionals (Ref 1:4). However, as shown in Fig. 1, the requirements definition phase and a major portion of the design phase have been bypassed. The design phase consists of preliminary or system level design (Ref 22:14), and detailed design. In Fig. 1, the shaded area within the design phase represents only part of the detailed design. See

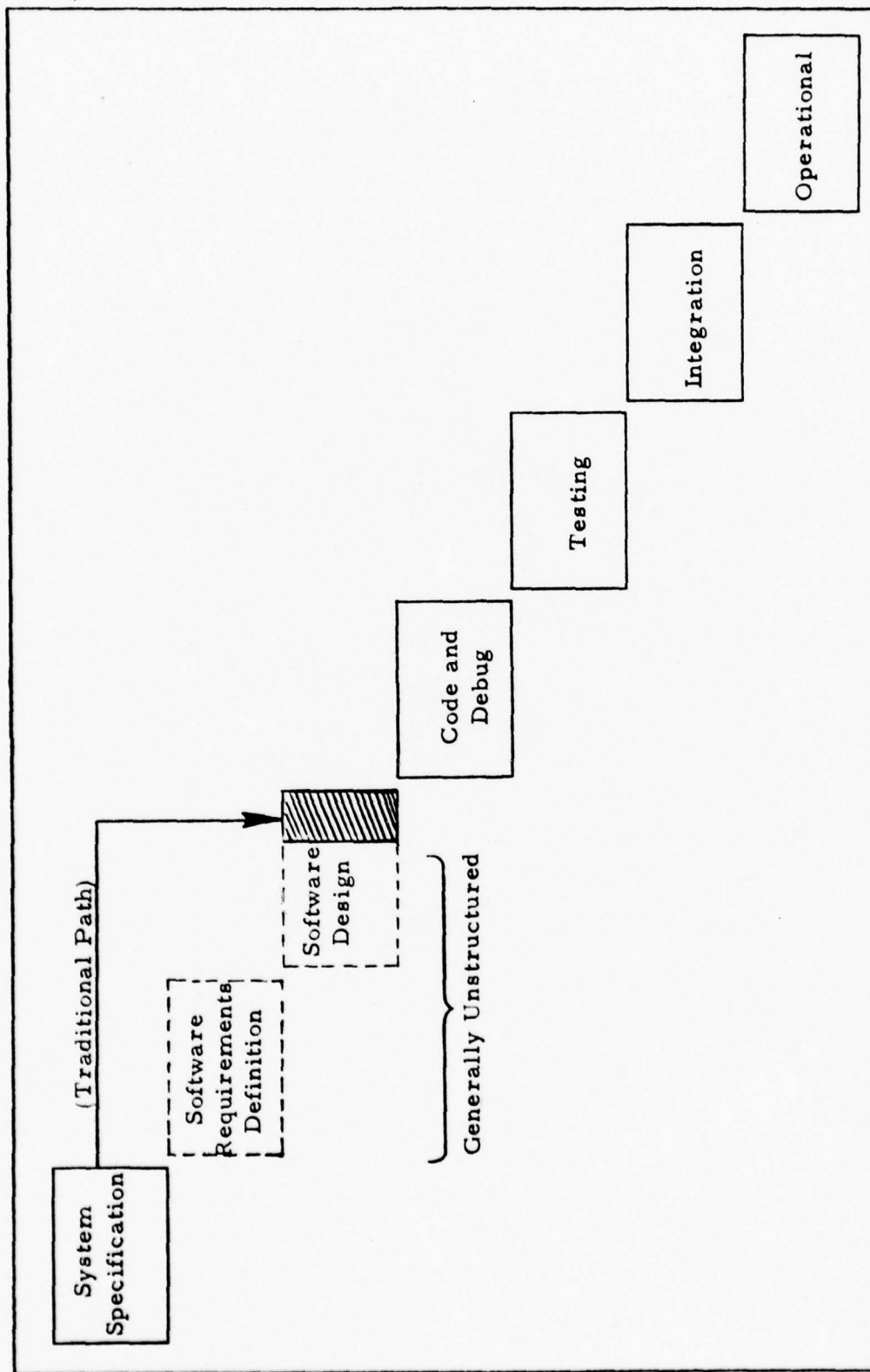


Fig. 1. Software Life-Cycle

reference 12 for further details. The importance of the software requirements definition ("what") and design ("how") cannot be over-emphasized in any Air Force software development efforts.

The software requirements and design phases must be accomplished so that the end-item software system satisfies the user's requirements, is more easily understood (reduced complexity), and is maintainable. These attributes will help to significantly decrease total software life-cycle costs. There is almost a direct relationship between the requirements/design phases and the operational phase.

Software maintenance occurs in the operational phase (Ref 1:6; 12:25) of the software life-cycle. Latent software errors are generally discovered at this time (Ref 1:6). This category of errors (i.e. latent) is usually due to poor requirements definition, poor design effort, or both. Generally, the later the errors are found the more costly they are to correct with respect to effort and time (Ref 12:5). It is well known that software life-cycle costs are high and are continuing to increase. Therefore, software development can no longer be taken lightly.

The Air Force has been involved with the procurement of aeronautical system software for many years, but recently has indicated that maintenance and support of weapon system software throughout the life-cycle may be accomplished by the Air Force if deemed cost effective. Since new weapon systems have a long lead-time and technology (i.e. software and hardware) and mission requirements are

constantly being improved, it is very probable that an active modification and enhancement program for any particular avionics software will be generated.

Requirements Engineering

The purpose of the requirements definition phase in the software life-cycle is to completely and unambiguously document an interpretation of the software requirements necessary to satisfy the system specification. This phase of the software life-cycle can be considered the most critical (Ref 1:5). A Part I software specification (design) contains the requirements information peculiar to the design and development of a particular software system (Ref 2:15).

The requirements represent a complete analysis of "what" (not "how") one is trying to accomplish with the software (Ref 22:14) or what the software product will do, and serves as a basis for common agreement among all parties concerned (Ref 12:5). A well detailed description of "how" these requirements are to be achieved is accomplished in the design phase of the software life-cycle.

In general, software development is divided into many physical tasks to be accomplished by many people, and as a result, it may be extremely difficult to maintain the "integrity" of the entire system (Ref 13:42-44). An excellent place to start maintaining system integrity is with the use of a complete and concise software requirements definition. As with any problem which is to be solved, a better

solution is achieved if the problem structure is more clearly and concisely defined, described, and understood.

Architectural Philosophy

The word architecture implies organization. At first glance, a well organized software systems architectural representation might resemble an organization chart for some large corporation. This architectural representation also implies a management hierarchy (Ref 40:25) which is closely associated with the different information levels within the organization or structure. In a software system, this hierarchical structure is represented by the purposeful ordering of the pieces or parts (modules) which make up the system.

A major concern of the software engineer or designer should be the ordering of the modular structure in such a way as to reduce complexity and ease the processes of coding, debugging, testing, integration, and operation so that software life-cycle costs will be reduced. The design phase is extremely important. Within this phase, the preliminary design should result in an identification of all modules, their functional descriptions, their interfaces, the hierarchical structure of those modules, the associated data structure relationships, and any necessary performance requirements (Ref 22:15).

A recent report (Ref 38:5-6) indicates that the software for an operational flight program used in a sophisticated electro-optical sensor system was designed in such a way that its architecture will

not allow easy modification to meet changing requirements. In order for the software maintenance and support operations to successfully meet changing requirements, the necessary internal software design information must be easily transferable, understandable, and complete (Ref 8:23-25).

It would be highly desirable to use an appropriate design and self-documenting methodology to provide this information. It should be conducive to providing an improved software architectural design which can more easily meet changing requirements and, in addition, help reduce total software life-cycle costs.

The Technique of Structured Design

Recently several software design techniques have received the attention of software engineers. Hierarchy plus input-process-output (HIPO) by IBM (Ref 17), higher order software (HOS) by Draper Labs (Ref 16), top-down by Wirth (Ref 39), information hiding by Parnas (Ref 27; 28), Jackson's method (Ref 19), composite design by Myers (Ref 26), and structured design by Constantine (Ref 40) are design methodologies currently in use. The latter, structured design by Constantine, will be briefly discussed.

Structured design is a set of general program design considerations and techniques used to make coding, debugging, and modification easier, faster, and less expensive by reducing complexity (Ref 20:Sec I, 1). The major ideas of this methodology resulted from

nearly ten years of research by Mr. Larry L. Constantine. He approached his research from a cost standpoint and observed that some designs were cheaper than others, were easier to implement, and allowed faster and easier modifications (Ref 34).

A major objective of structured design is to develop a software system of modules arranged in a hierarchical manner such that individual portions can be evaluated, implemented, modified, or changed without affecting the rest of the system (Ref 29:42). For an optimal design, certain principles and rules are used. The principles of Coupling (intermodular connection relationship) and Cohesion (intramodular strength relationship) aim for low or loose coupling between modules, and high individual module cohesion (Ref 24:13). In addition, rules for module scope-of-effect and scope-of-control aid the software designer in achieving this objective. A module's scope-of-control is that module plus all subordinate modules. The scope-of-effect (of a decision) is the collection of all modules containing any processing that is conditional upon a decision. For any given decision, the scope-of-effect should be within the scope-of-control (Ref 40:240).

A data flow diagram or bubble chart is used (see Appendix B). The bubble chart defines the required order of data transformation and, according to Constantine, should not show control flow, timing, looping or machine dependency. The bubble chart does not indicate the modular structure (Ref 20:Sec I, 5). The focus is upon major

streams of data as they flow from external input to external output, and the transformation processes in between (Ref 24:15). A structure chart is developed from the bubble chart and represents the hierarchical relationships of the modules and procedural or control information (see Appendix C). These charts are used in a heuristic manner to document design refinement and evaluate the system architecture with respect to the principles and rules previously discussed.

The functional orientation of structured design makes it particularly appropriate for software systems which might experience changes in function (Ref 24:16), possibly like those associated with operational flight software in a real-time embedded system. Modifications which involve additional, augmented, or deleted functions are relatively straightforward (Ref 24:16) where structured design has been used.

Summary

This chapter expressed a concern for software requirements and design engineering with a goal toward purposeful software system structuring and the use of a method for accomplishing that structuring. The traditional approach of many software practitioners (and managers) in going directly from the problem (system specification) to the coding phase is highly undesirable and costly. The need for a complete, unambiguous, written, and mutually agreed upon requirements definition is highly desirable. A software architecture which

easily allows future changes, is easily understood, and aids in reducing software life-cycle costs associated with a system is also desirable. Present day Air Force software is very expensive. Finally, Structured Design by Constantine is a software design methodology that makes coding, debugging, and modification easier, faster, and less expensive. These attributes are highly desirable for Air Force computer software. Prior to discussing the application of design techniques, the PAVE TACK OFP will be briefly described.

III. The PAVE TACK Operational Flight Program

Introduction

This chapter briefly describes the operational flight program for a current Air Force system named PAVE TACK. General information, subsystem interfaces and the executive structure are discussed.

General

PAVE TACK is a name given to an advanced day/night electro-optical, pod-mounted attack and surveillance system intended to improve weapon delivery capability. The PAVE TACK pod can be centerline mounted or located on any stores station of a particular fighter aircraft. In addition to supporting avionics and power conversion equipment, the pod contains a digital computer (Ref 25:50). Within the computer the OFP performs significant mission functions.

The PAVE TACK OFP interfaces with the pod to perform target search and tracking functions. It also interfaces with the aircraft computer (serial digital interface) for the input of navigation and aim-point data and the output of navigation correction updates. Computations are performed which in turn generate signals to the PAVE TACK stabilized sight subsystem to assist in acquiring and tracking targets. Laser information (e.g. slant range), operator signals (e.g. tracking control handle), and electro-optical sightline angular data are

processed by the OFP to provide data for updating aircraft navigation and weapon delivery functions (Ref 7:Sec III, 1).

The OFP resides in the flight computer memory. Under control of the stored OFP, a conversion unit accepts analog data from the pod and avionics systems and converts this data for use by the digital computer. After processing the data, some outputs of the computer are converted to appropriate analog signals. Other data is available or produced in the form of "discretes" which are special words in computer memory. Each word consists of bits, each of which represent a specific status (e.g. inhibit laser, system failure, hand-control switch, and so forth). For further details on the OFP, the reader may refer to references 4, 5, 6, 7, 15, 18, and 38.

Subsystem Interfaces. It is necessary to synchronize the OFP functions with those being performed by subsystems, and there are a number of different interfacing loops involved. The three interfacing loops are the operator loop, the pod/OFP loop, and the aircraft computer/OFP loop. The functional subsystem interfaces for the OFP may be seen in Fig. 2.

PAVE TACK's primary function is target acquisition and tracking. In the operator loop, this function is controlled by the aircrew. Analog thumb tracker inputs (i.e. the tracking control handle) allow the aircrew to control the line of sight (LOS) of the forward looking infrared (FLIR) detector and laser in the pod. Feedback to the aircrew is through a CRT display which is controlled by software and

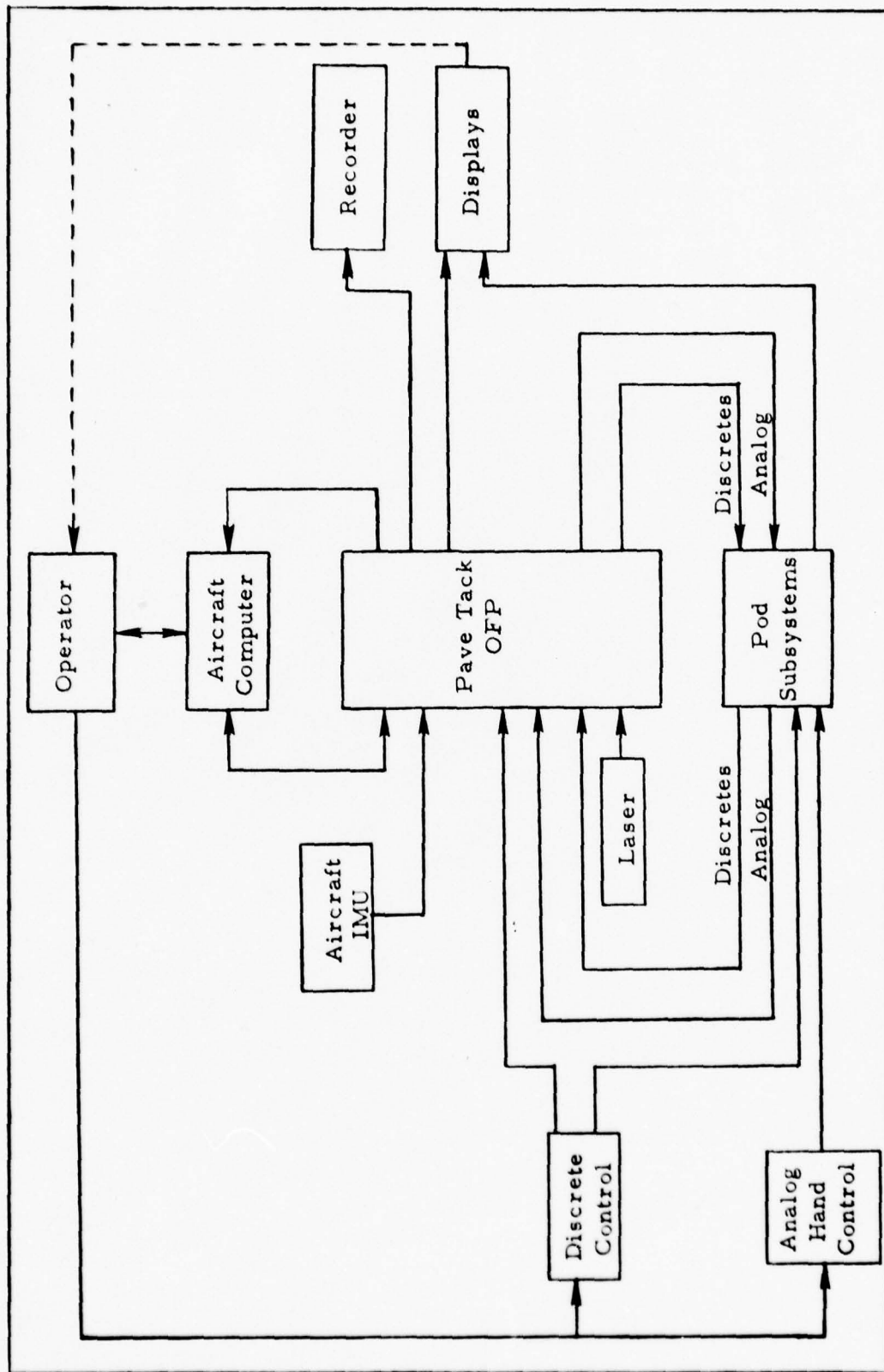


Fig. 2. PAVE TACK OFF Subsystem Interfaces (Ref :28)

hardware. The interactive pod/OFP loop exists between the pod subsystems and the OFP. The pod provides analog LOS position and rate data to the OFP, while the OFP provides analog LOS position control as a function of errors and control inputs. Hardware and software parameters maintain loop stability. The primary purpose of the aircraft computer/OFP loop is to provide aircraft computer navigation and aimpoint data to the OFP. This loop is interconnected with the operator interface through the aircraft computer.

Executive Structure. The executive handles the timing and task sequencing of the entire OFP. In addition, the executive is responsible for assuring that the program resumes without "impurity" where it left off after a timing or data interrupt has occurred.

The executive control module of the PAVE TACK OFP consists of three main routines: the power interrupt module, the level-1 interrupt routine, and the level-0 interrupt routine. The power interrupt module performs the initial condition settings for the OFP after the occurrence of the power-up interrupt or after the completion of a system built-in test (BIT).

The level-1 routine controls the basic timing of the OFP program execution. A programmable timer (counter) is set to generate a level-1 timer interrupt every 8.33 milliseconds. Whenever this timer interrupt is generated, the executive control portion of the level-1 routine is entered. Processing consists of a mainloop cycle of 25 milliseconds, a fastloop cycle of 8.33 milliseconds, and slow

cycle of 100 milliseconds and 1 second. The fastloop performs the rate aiding and image derotation angle extrapolation computations in order to provide the rate aiding signals and derotation angle output at the rate of 120 hertz. The slow cycles include a call to a subroutine which flashes the laser transmitter-on indicator on the CRT at the rate of five times a second whenever the laser transmitter is on. The other portion of the slow cycle consists of the CRT display output processing.

The main processing of the PAVE TACK OFP is performed every 25 milliseconds when the executive calls the mainloop module. The start of analog data input conversion is initiated by the executive immediately before the execution of the mainloop. The mainloop may be interrupted by the level-1 timer interrupts. The executive checks to see whether the slow cycle processings are required, calls the slow cycle processings or sets the flags for slow cycle processings when required, and returns to the processing interrupted. After a cycle of the mainloop processing is completed, control is returned to the executive.

The executive also calls the one hertz processes when required and starts or continues a BIT software program which is performed on a time available basis. This program is called the running BIT program, and it performs limited self-tests of the CPU, memory, input/output, and pod subsystems. The level-0 interrupt routine services analog data conversion and laser input processing when the

signal converter interrupts are generated. The OFP executive control program is a synchronizing executive. The level-0 interrupt routine services asynchronous external inputs at a higher priority level than the executive program.

Summary

This chapter presented a brief description of an operational flight program for a current Air Force pre-operational avionic system called PAVE TACK. General information, subsystem interfaces and the executive structure were discussed.

IV. Application of Design Techniques

Introduction

This chapter describes the results of applying several design techniques to the PAVE TACK OFP. The initial approach attempting to apply the structured design methodology is discussed. Included are some of the problems that were encountered. Finally, the revised design approach, as a result of the problems encountered during the initial approach, is described, and several design techniques are discussed.

Initial Approach

Objective. For the initial approach, the structured design methodology was to be applied to the entire PAVE TACK OFP requirement. After a reasonable amount of requirements familiarization time, the draft Part I software specification (Ref 4) was used, as it should be, for the basis of the software design. It was anticipated that this would be the same design starting point for an avionics software engineer/contract monitor. The operation of the OFP, developed with structured design, would be a "black box" equivalent to the current software. However, the internal software architecture of the new design would be different, and the benefits of this type of design were previously discussed in Chapter II. Transform analysis was used to start the design process.

Application of Transform Analysis. Constantine defines Transform Analysis, or transform-centered design, as a strategy which identifies the primary processing functions (transform center) of the desired software system, the high-level inputs (afferent data elements) to those functions, and the high-level outputs (efferent data elements) from those functions (Ref 40:254). The concepts behind the strategy of a structured design were briefly described in Chapter II. The major steps of transform analysis are:

1. Restate the "problem" as a data flow diagram (bubble chart).
2. Identify the afferent and efferent data elements on the bubble chart.
3. Perform first level factoring to obtain an initial high-level structure chart.
4. Using this structure chart, factor the afferent, transform, and efferent branches.
5. Complete the final structure chart by noting any departures from the structure chart obtained in the previous step.

Departures refer to changes resulting from a design trade-off analysis of scope-of-effect, scope-of-control, the principles of coupling and cohesion, and any a priori knowledge of particular "real-world" effects on a proposed design. As mentioned in step one, the transform analysis process starts with the development of a bubble chart.

Attempting to start the initial development of a system-level bubble chart for the PAVE TACK OFP was not easy. The draft Part I specification (Ref 4) did not necessarily indicate how data for the OFP was related to, or oriented (e.g. task, operator selected mode, etc.) for use by the desired software. A system-level bubble chart should consist of all major inputs, all major outputs, and all appropriate data transformations between the two types of data. However, due to lack of written direction (draft Part I specification), the complexity of the design problem slowly became apparent. It might be possible to orient the data for the bubble chart by task (or mission related function), data classes (discrete or analog), mode (search), interrupt levels or some combination of these. Upon closer examination, it appeared as though a task and/or mode data orientation might be the best direction to take.

However, a change in applying transform analysis to the entire OFP had to be made in order to allow the study to proceed. With respect to the entire OFP, processing outside the mainloop appeared to involve too much timing and control. A basis in applying a bubble chart is to avoid timing and control. Therefore, it was decided that a bubble chart would not be directly applicable to the executive and other associated functions outside of the OFP mainloop. As a result, the decision was made to apply the transform analysis steps to the OFP mainloop since most of the operational airborne functions are performed there (Ref 38:A1).

Several unsuccessful attempts were made to create a mainloop bubble chart. One particular attempt is worth noting. Realizing that the mainloop performed several functions, the idea of generating subsets of bubble charts (bubble chunks), which would later be combined, seemed appealing. Bubble chunks were created for several sub-functions such as ACQUIRE, SNOWFLOW, TERRAIN MONITOR and NAVIGATION cueing. The bubble chunks were combined to form the bubble chart associated with sightline control during the search/acquire function. Although the idea of using bubble chunks is a valid technique (Ref 20:Sec II, 26-31), it did not work well in this case. The result was a very poor, if not invalid, bubble chart. The bubble chart was extremely cluttered, had interfering, crossing data flow lines (not allowed), was inconsistent in several areas (e.g. undesirable differences in data stream names), and tended to show some control (not allowed). Later the reason for this result will become apparent.

After much time was consumed experiencing several such "blind alleys," the final Part I software specification (Ref 5) became available and proved to be helpful. The OFP functional summary now listed the major tasks to be performed (Ref 5:Sec I, 1-2) rather than a description of the OFP "modules" (Ref 4:2). Normally a complete description (including interfaces) of OFP mainloop modules would exist after the preliminary design as a result of an entire application of, for example, the design techniques presented in this

chapter. The following major tasks associated with the OFP main-loop were extracted (verbatim) from the functional summary list (Ref 5:Sec I, 1-2):

1. Provide signals to the PAVE TACK sightline control electronics to assist in acquiring and tracking targets.
2. Provide velocity trim and navigation update data to the aircraft computer. (Clarifying Note: The navigation update actually consists of velocity and range trim data.)
3. Inhibit laser operations under specified conditions.
4. Provide image derotation and focusing signals to the AN/AAQ-9 Infrared Detecting Set.

The major tasks listed above served as the first vague indication of actual mainloop requirements as per a Part I specification.

The final Part I specification also gave very brief summaries of the mainloop major tasks listed above (Ref 5:Sec I, 2-7). The summaries were helpful with respect to task familiarization. It was reasonable to expect, from a designer's point of view, some written description of any precedence relationship associated with the set of tasks comprising the mainloop. It was obvious from studying the specification that some tasks or actions were required prior to others (e.g. input specific data at start of mainloop, or perform coordinate transformation of inertial-to-sightline information). However, precedence information of a level of detail sufficient for this design study was not found.

A final attempt was made to apply transform analysis to the OFP mainloop. Major functions were analyzed. During the attempt to generate a bubble chart, it was realized that no afferent or efferent data elements could be found between the major tasks. In fact, each major task actually functioned as an input-process-output (IPO) scheme. That is, each major function investigated for the mainloop could be considered a small application program in itself. This explained why previous bubble chart attempts (e. g. using bubble chunks) did not succeed. With this realization, it became obvious that the approach to the structured design investigation would have to be modified.

Revised Approach

Objectives. For the revised approach, several design techniques were applied to the PAVE TACK OFP mainloop requirement. The final Part I specification (Ref 5) was used in the same manner previously discussed (initial approach objective). The design techniques were applied to several sections of the OFP mainloop, but procedures related to each technique are carried out only far enough to show the avionics software engineer/contract monitor "how to get started" on a design. Finally, a software design is subjective, and naturally, may vary from one designer to another.

Finite-State Technique. For the purposes of this study, the finite-state technique is defined as a method that uses a state

diagram to represent a specific algorithm's control flow. The state diagram is made up of nodes and directed links which represent control of the OFP mainloop operational airborne functions or tasks. The nodes are represented by circles. The directed links (i.e. edges) between nodes are represented by an arc.

The technique, as used here, has been slightly modified from current finite-state automata methods (Ref 24:22-27; 11:221-239). In this study, each node represents a task. Since each task within the OFP mainloop has previously been defined as an IPO scheme, no data is passed between tasks at this level (state diagram) of observation. Proper sequencing of control from one OFP mainloop task to another is of prime importance at this stage of design. In this respect, the technique permits a better understanding of the algorithm's control structure by using an appropriate "level of observation" and reducing superfluous detail.

For the PAVE TACK OFP mainloop, a particular control relationship (i.e. search, cue or track mode selected by the operator) was known (Ref 5:Sec III, 65; Sec XX, 9-10) at the start of each mainloop execution. This relationship was used to direct control to the next task in the mainloop. This is possible because the maximum mainloop execution rate is forty hertz, and the mode does not change during any particular mainloop cycle. Although the mainloop is pre-emptable, task execution resumes when the request causing an interrupt is satisfied. For example, an interrupt generated for

laser data input pre-empts an "in-process" task in the mainloop. After the laser data has been "processed," the interrupted task in the mainloop resumes its processing. The control relationship, defined above, works in a satisfactory manner for this application of the finite-state technique since the mainloop can be considered a closed sub-task system (mainloop is part of the executive). That is, the mainloop contains a unique initial task (adjust raw input data for OFP use) and a unique terminal task (adjust raw input data for OFP use) and a unique terminal task (adjust processed data for output). Obviously, execution of the mainloop is considered complete when all applicable tasks, depending on the control relationship, have been processed including the initial and terminal tasks.

The control relationship previously discussed, and results of further analysis of the mainloop requirements (Ref 5:Sec III, 65-68) were used to apply the finite-state technique. The resulting state diagram is shown in Fig. 3, and a description of next-state transitions by mode (search, cue or track) are shown in Table I. This type of table may be helpful during the coding phase, but can be quite large depending on the particular OFP requirements. In Fig. 3, node one is the initial task and node eight is the terminal task of the OFP mainloop. Node six (Perform Target Operations) is in an abstract form in order to simplify the state diagram. It consists of many sub-tasks or functions, some of which will be shown later in this chapter (see Transform/Transaction Analysis).

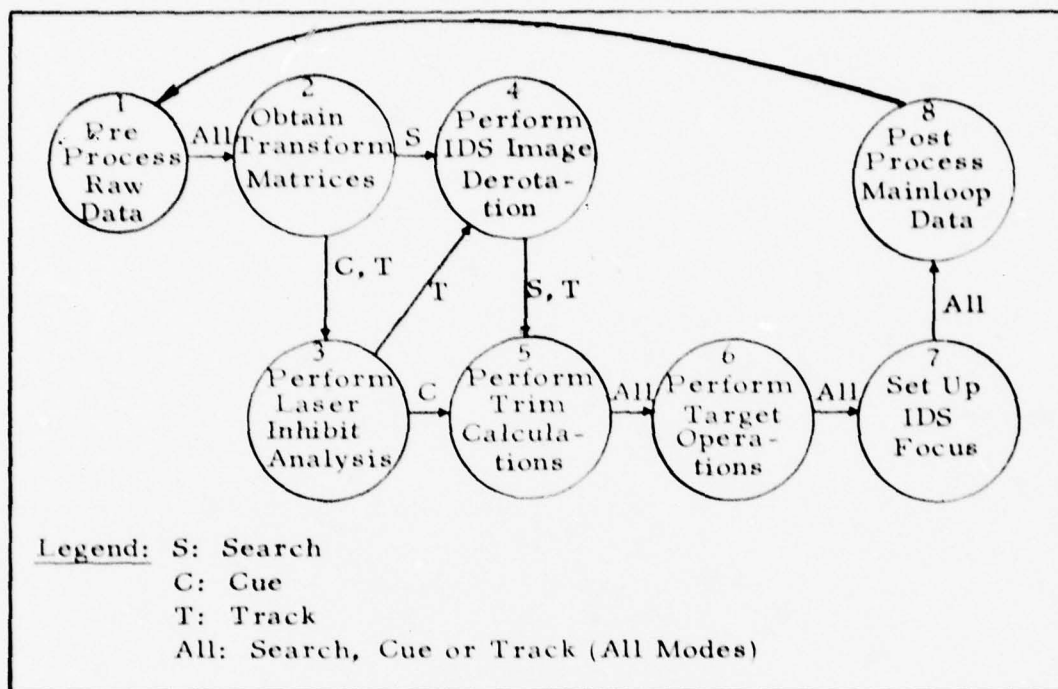


Fig. 3. Mainloop State Diagram by Mode

Table I
Mainloop State Transitions by Mode

Present State \ Next State	(Mode)		
	Search (S)	Cue (C)	Track (T)
1	2	2	2
2	4	3	3
3	-	5	4
4	5	-	5
5	6	6	6
6	7	7	7
7	8	8	8
8	1	1	1

If larger, more complex task systems are required, the designer can use a more rigorous approach. A stricter task notation for chains and precedence may be used. If the designer requires this type of detail, any good text that uses graph theory applied to operating systems may be used. See reference 14.

The state diagram may be refined in any appropriate manner, and verified by checking the tasks of the diagram against the functional design requirements until the requirements are met. Desired levels of detail, clarity and completeness may be achieved. It is important that the state diagram directly reflects the algorithm structure and meets the written design requirements. Once the software designer completes this step (application of the finite-state technique), the technique could be applied (if warranted) to another single "task" within the state diagram just developed (e.g. Perform Target Operations). Developing state diagrams at more than one level allows design refinement. Each successive level would obviously contain a greater amount of detail. In addition, the techniques which follow can also be applied to a single task until the entire OFP mainloop design is complete.

Transform Analysis Technique. For the revised approach, Constantine's version of transform analysis (Ref 40:254-300) was modified. The technique was extended to include iterations between the bubble and structure charts (Ref 20:Sec II, 1-36). The modified transform analysis technique permits successive design refinement.

The major steps are:

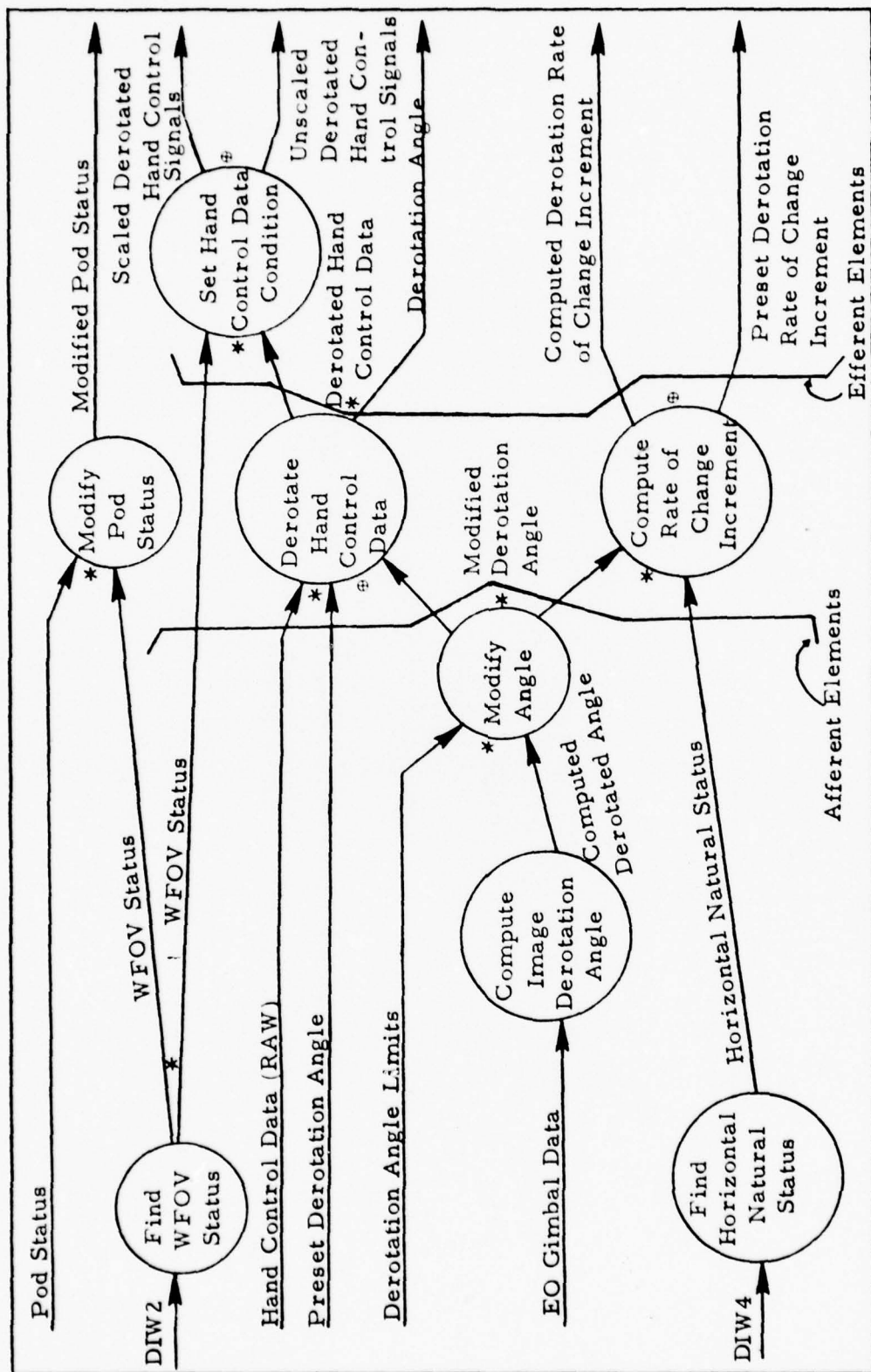
1. Restate the "problem" as a data flow diagram (bubble chart).
This is called the "first-cut" bubble chart.
2. Identify the afferent and efferent data elements on the bubble chart.
3. Develop a fully factored, "first-cut" structure chart.
4. Using the basic principles of coupling, cohesion, scope-of-effect, and scope-of-control, analyze the first-cut structure chart.
5. Using the results of the analysis, rearrange any modules as necessary, and recheck the analysis as necessary. When the analysis is complete, the resulting structure chart is called the "intermediate" structure chart.
6. From the intermediate structure chart, develop a new (revised) bubble chart. This is referred to as the intermediate bubble chart.
7. Revise the transform, and afferent and efferent data elements on the bubble chart as necessary. This is called the "final" bubble chart.
8. Lastly, develop the final structure chart from the final bubble chart.

In any analysis of a bubble chart, it should be remembered that the bubble chart may be considered a graphical representation of the problem's requirement definition. Obviously each bubble chart

should be checked against the written requirements for misinterpretation of those requirements or any other errors (e.g. an incorrectly named data element). The final structure chart represents the design, and the chart should reflect the problem structure.

Transform Analysis will be applied to the "Perform IDS Image Derotation" task in the mainloop state diagram shown in Fig. 3. Some of the OFP mainloop tasks are trivial (e.g. "Setup IDS Focus"), while others are more complex (e.g. "Preprocess Raw Data"). However, the infrared detector set (IDS) Image Derotation task is simple to understand. Operator hand control data and electro-optical (EO) gimbal data are used to generate certain derotation data which permits the IDS to perform the image derotation function. When a target has been acquired (with respect to the aircraft using PAVE TACK), the target image can be displayed (in the cockpit) in a normal and upright manner, and thus enhance recognition of a target. This type of image display is accomplished if the operator selected the "Horizontal Natural" display mode. If this type of image display is not selected, the target image is displayed such that the vectored component of aircraft velocity is upward relative to the display (Ref 5:SecI, 6).

After an extensive analysis of the requirements for the IDS image derotation task, a first-cut bubble chart was completed, and the afferent and efferent data elements were identified as shown in Fig. 4. The placement of the afferent/efferent "cuts" may vary



slightly from one designer to another. The central transform for this problem lies between the afferent/efferent cuts. Within the identified central transform, two main functions are performed. Raw hand control data is derotated by using the derotation angle which is either a preset, or calculated and modified angle. The modified derotation angle is derived by applying predetermined angle limits to the computed angle. The computed angle is obtained from the electro-optical gimbal data (roll, pitch and yaw sine/cosine gimbal information). For the second function, the derotation angle rate-of-change increment is computed using the modified derotation angle and the status of the horizontal-natural mode selected by the operator. The horizontal-natural status and the wide-field-of-view (WFOV) status are obtained from specific discrete input words (DIWX) in memory. The WFOV status is used to update the PAVE TACK pod status. The WFOV status may also be used to scale (a magnification function used for cockpit display) the derotated hand control data.

The bubble chart, Fig. 4, can be used to verify the designer's interpretation of the problem by comparing it to the requirements definition (Part I specification). If necessary, the bubble chart may be revised to conform to the problem requirements as they are understood at this point in time. When the designer is satisfied with the first-cut bubble chart, the first-cut structure chart can be developed.

The first-cut structure chart shown in Fig. 5 was developed directly from the first-cut bubble chart with the data and control

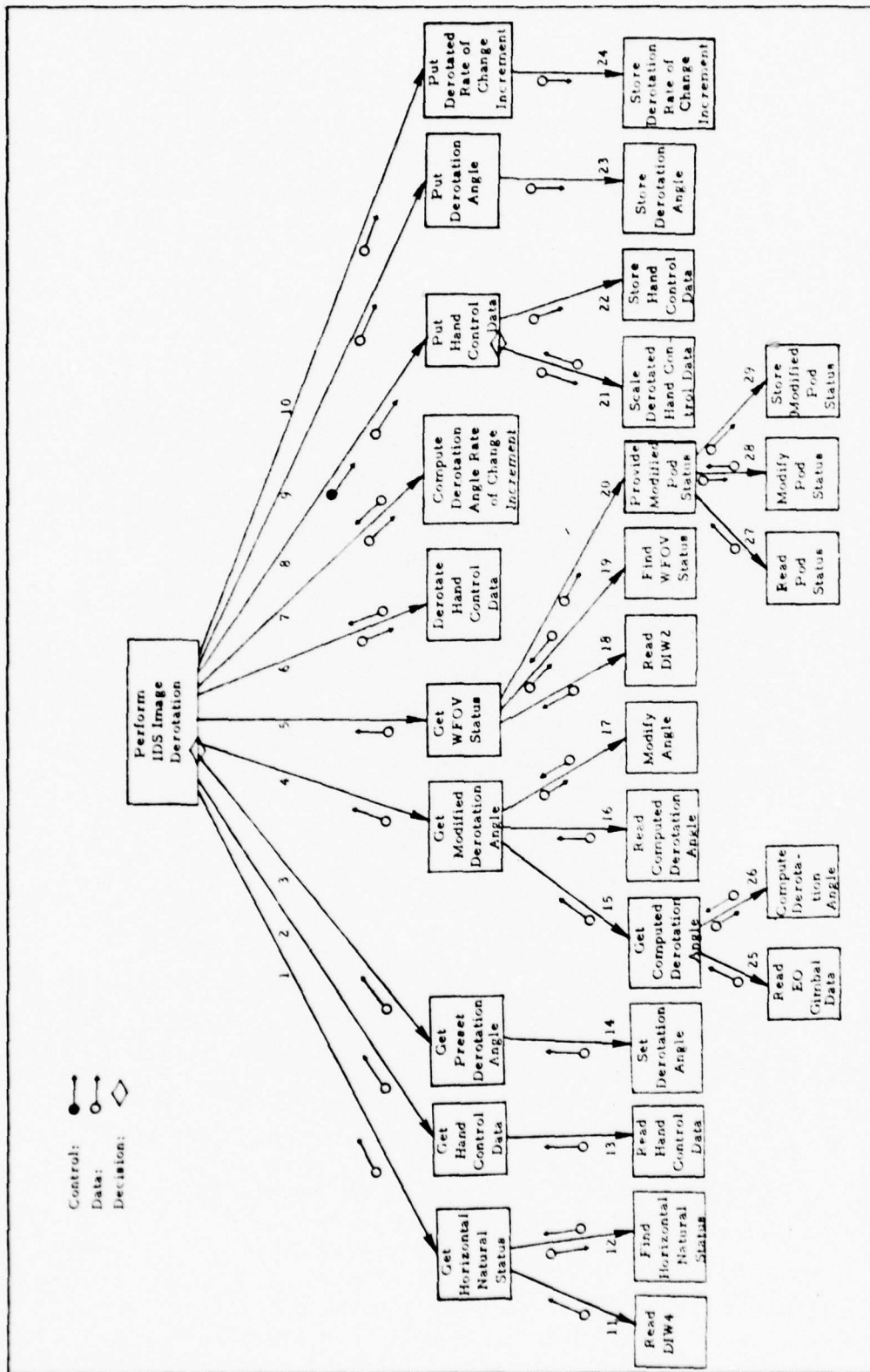


Fig. 5. IDS Image Derotation First-Cut Structure Chart

arrows added. Any decisions within the hierarchical structure are also shown. The modules are annotated with descriptive phrases rather than single names. Table II lists the input and output parameters of the first-cut structure chart. Any control parameters are underlined.

The software engineer/contract monitor may not want to use a descriptive phrase for each module. As an alternative, singular verbs can be used which characterize each module's function (Ref 40: 273-277). For example, the following names could be applied to the second-level modules (by number) of Fig. 5:

- | | |
|-------------------|------------------|
| 1. GETHORZSTAT | 6. DEROTHANCONTL |
| 2. GETHANCONTLDAT | 7. COMPANGROCI |
| 3. GETSETANG | 8. PUTHANDAT |
| 4. GETMODANG | 9. PUTANGDAT |
| 5. GETVIEWSTAT | 10. PUTINCREDAT |

The designer has the choice of using either method of module identification. Regardless of the method chosen, its use should be consistent for each structure chart.

The first-cut structure chart was analyzed. The structure in Fig. 5 is fully factored, and has a depth of four. The fan-in for all modules of the second through fourth levels is one. The fan-out or span-of-control of modules in the same levels ranges from one to three. Note that the fan-out for the level-one module is ten, which might be considered slightly high. This in turn makes the

Table II
First Cut Structure Chart Parameters

Module	Input	Output
1	---	Horizontal-Natural Status
2	---	Hand Control Coordinates
3	---	Preset Derotation Angle
4	---	Modified Derotation Angle
5	---	WFOV Status
6	Hand Control Coordinates, Preset Derotation Angle, Computed Derotation Angle	Derotated Hand Control Coordinates
7	Modified Derotation Angle, Horizontal-Natural Status	Computed Rate of Change Increment, Preset Rate of Change Increment
8	WFOV Status Flag, Derotated Hand Control Coordinates	---
9	Derotation Angle	---
10	Computed Rate of Change Increment, Preset Rate of Change Increment	---
11	---	DIW4
12	DIW4	Horizontal-Natural Status
13	---	Hand Control Coordinates
14	---	Preset Derotation Angle
15	---	Computed Derotation Angle
16	---	Derotation Angle Limits

Table II (Continued)

Module	Input	Output
17	Computed Derotation Angle, Derotation Angle Limits	Modified Derotation Angle
18	---	DIW2
19	DIW2	WFOV Status
20	WFOV Status	---
21	Derotated Hand Control Coordinates	Scaled Derotated Hand Control Coordinates
22	Scaled Derotated Hand Control Coordinates	---
23	Derotation Angle	---
24	Computed Rate of Change Increment, Preset Rate of Change Increment	---
25	---	EO Gimbal Angular Data
26	EO Gimbal Angular Data	Computed Derotation Angle
27	---	Pod Status
28	WFOV Status, Pod Status	Modified Pod Status
29	Modified Pod Status	---
Note: Control Information is underlined.		

scope-of-control eleven for the first level module. The scope-of-control is four for modules four, five and twenty; three for modules one, eight and fifteen; and two for all applicable remaining modules. There are two decisions shown on the structure chart. One decision is in the first level module and involves modules three and four. The data (derotation angle) obtained from these modules is used for processing in modules six and seven which are also invoked by the first level module. Therefore, the scope-of-effect for this decision is within the scope-of-control. The other decision is located in module eight (second level), and its scope-of-effect is within the scope-of-control for that module. Finally, note that modules two and three are truly afferent, and modules nine and ten are truly efferent by definition.

To complete the analysis, coupling and cohesion of all modules was examined. The procedures used were applied to all modules on the first-cut structure chart. The analysis of module coupling (Ref 40:116-142; 26:33-53) required the use of Fig. 5 and Table II. For example, modules one, eleven and twelve are data coupled. On the other hand, module eight is control coupled. Finally, the cohesion of all modules in Fig. 5 was examined. The analysis required familiarization with cohesion type-definitions (Ref 40:143-186; 26:19-31). A sentence describing the purpose of each module was written. Each sentence was analyzed (Ref 40:171-173; 26:28-30). For example:

1. Module one: The purpose of this module is to obtain the "current" horizontal-natural status.
2. Module eleven: The purpose of this module is to read discrete input word four (DIW4) from memory.
3. Module twelve: The purpose of this module is to find the horizontal-natural status embedded within DIW4.

From these "simple" sentences, it is obvious that each of these modules is functionally cohesive. Examining each module's coupling and cohesion in this analysis enables the designer to investigate alternative structures in an attempt to produce an intermediate structure chart.

An example of investigating structure chart alternatives is illustrated in Figures 6 and 7. Figure 6 shows a section or one afferent branch that was easily obtained from Fig. 5. Note that the decision in the first level module of Fig. 5 is "high" in the structure. It was desirable to move that decision further down into the structure (a lower level). By moving modules three and four down to the third level subordinate to a "new" single module ("OBTAIN DEROTATION ANGLE") in the second level, the afferent branch in Fig. 6 is obtained. A problem created by this alternate afferent branch is that the superordinate module is control coupled to the first level module in Fig. 5. It is not necessarily desirable to go from data coupling to control coupling, but in this case since the "Horizontal-Natural Status" is passed to the "OBTAIN DEROTATION ANGLE" module to

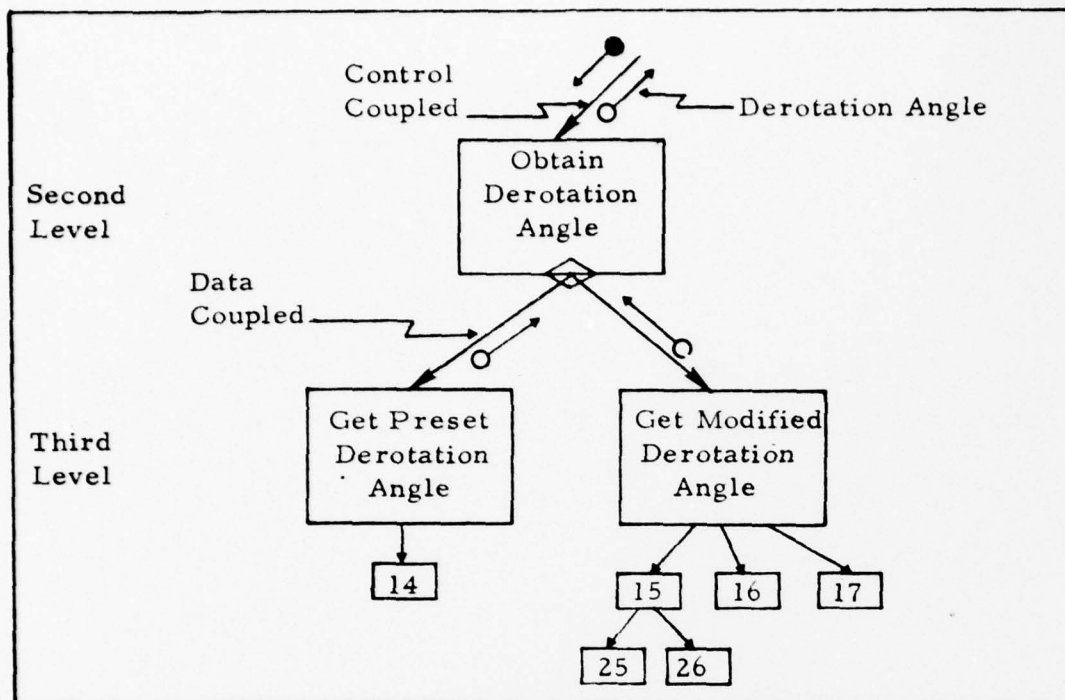


Fig. 6. Alternate Afferent Branch

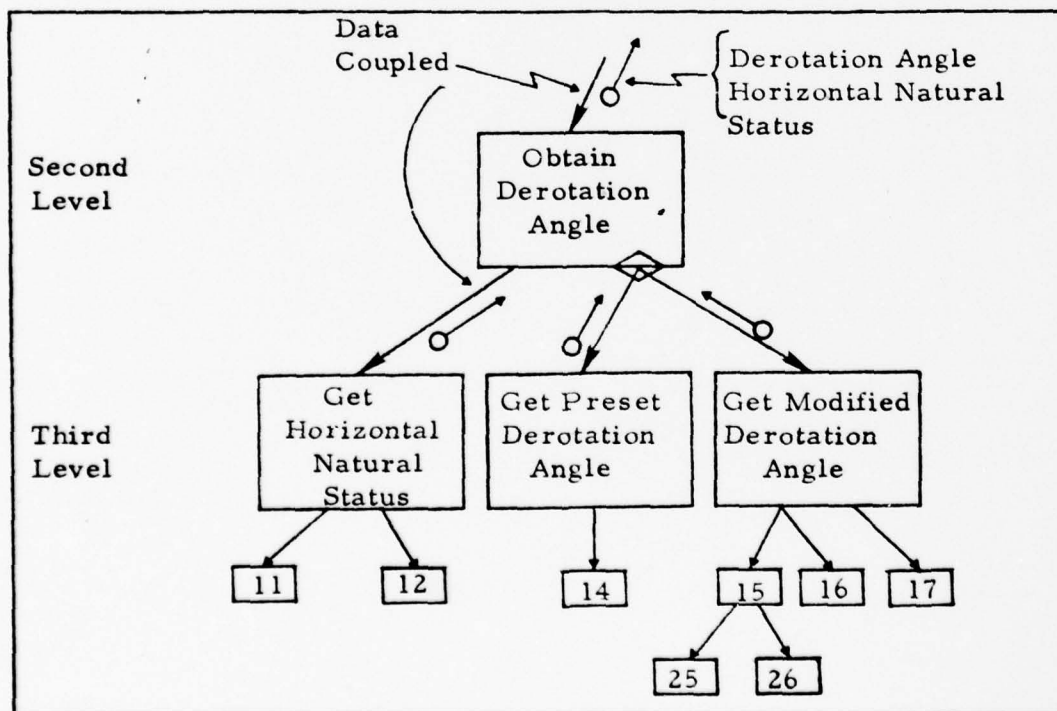


Fig. 7. Revised Afferent Branch

affect the decision, the control coupling problem can be remedied.

The revised alternate afferent branch is shown in Fig. 7. Module one ("GET HORZ NAT STATUS"), an afferent module from Fig. 5, is moved to the subordinate position as shown in Fig. 7. The "OBTAIN DEROTATION ANGLE" module is now data coupled. The modules on the second and third levels of Fig. 7 are still functionally cohesive. Using this method of developing alternative branches for the structure chart is useful since smaller sections of the structure chart are easier to understand and analyze. It helps the designer to develop the intermediate structure chart, which is the next suggested design step. However, to arrive at an acceptable *intermediate structure chart*, the designer must perform an analysis such as that performed on the first-cut structure chart.

After several alternate revised first-cut structure chart branches were analyzed, an intermediate structure chart was completed. For simplicity, the abbreviated intermediate structure chart (no data or control arrows and no module numbers) is shown in Fig. 8. An analysis of the intermediate structure chart was accomplished in a similar manner as that described for the first-cut structure chart. The completion of this structure chart permits the development of the next bubble chart.

An intermediate bubble chart was developed as shown in Fig. 9. It was verified against and shown to satisfy the final Part I specification. A rule-of-thumb (Ref 20:Sec II, 10) suggests that if the sum of

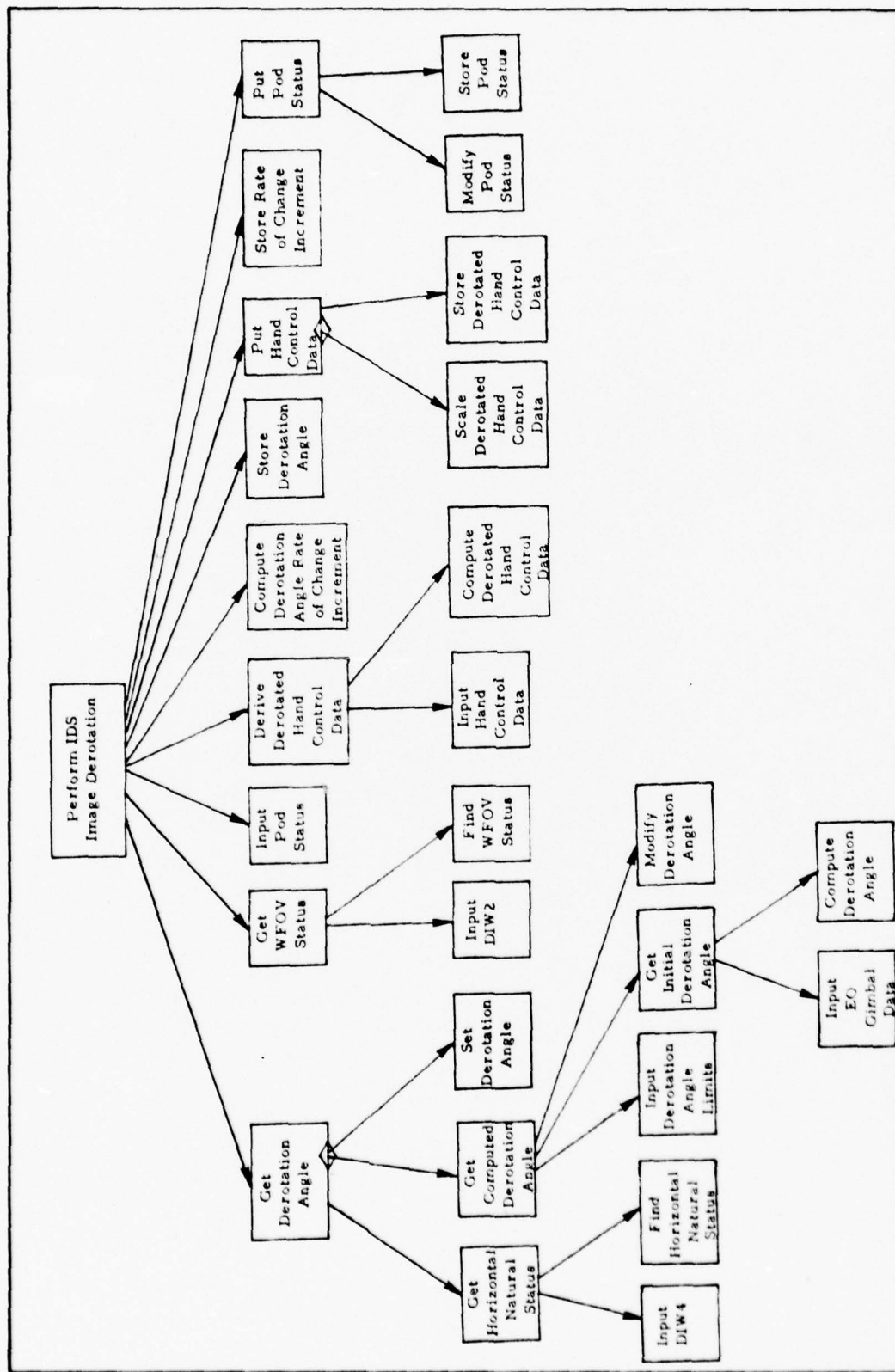


Fig. 8. IDS Image Derotation Intermediate Structure Chart

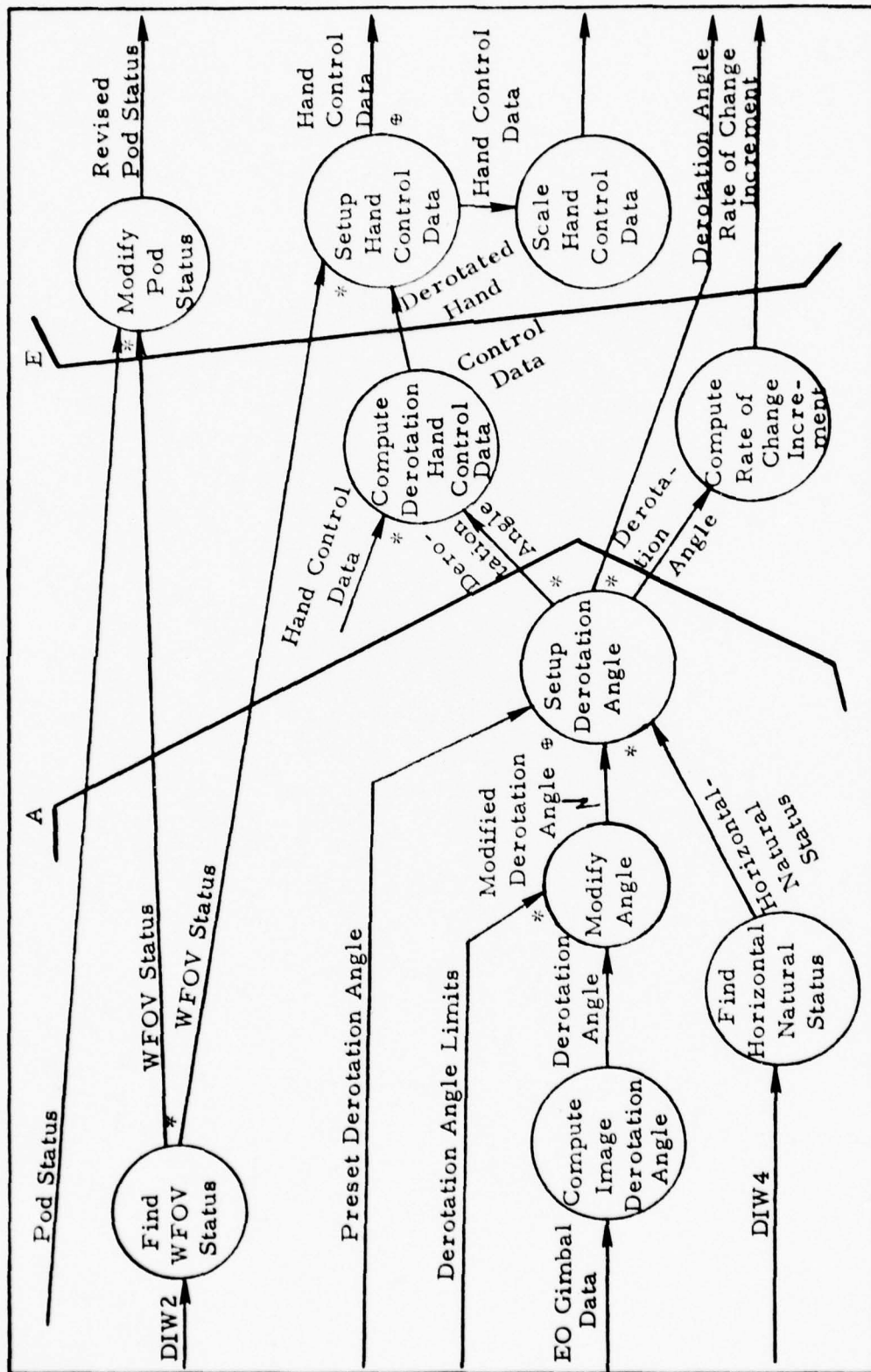


Fig. 9. IDS Image Derotation Intermediate Bubble Chart

the conceptual data streams cut by the afferent-transform and efferent-transform lines is at a minimum, then a "better" subsequent structure chart can be developed. Note that this sum for the first-cut bubble chart in Fig. 4 is ten. The sum for the intermediate bubble chart in Fig. 9 is eight. If the design process were to continue at this point, the intermediate bubble chart would be analyzed. If appropriate, the afferent-transform and efferent-transform lines may be moved. For example, if the "Pod status" data stream in Fig. 9 were to be moved to the right of the efferent "cut," the sum of cuts would drop to six.

Transform/Transaction Analysis Technique. Transform analysis was applied to the "Performed Target Operations" task (node six) from Fig. 3. Note that this particular task was shown at an abstract level in order to reduce complexity of the state diagram. This task actually consisted of many sub-tasks. Figure 10 shows the bubble chart for this abstract task. The "level of observation" for Fig. 10 was purposefully kept high for this discussion. This reduced superfluous detail at this point. Note that the transform "Examine Selection Data" is actually a transaction center as defined by Constantine (Ref 40:301-302). This indication allows the use of a supporting strategy to transform analysis called transaction analysis.

Transaction analysis (Ref 40:301-329) is a technique that uses the characteristics of a transaction center to map or develop a modular software structure while permitting the designer to also use the

principles of coupling, cohesion, scope-of-control and scope-of-effect. A transaction center must be able to obtain the transaction, determine its type, dispatch on that type, and complete the processing of each transaction (Ref 40:303). The major steps of the transaction analysis strategy are (Ref 40:307-308):

1. Identify the source of the transaction.
2. Specify the appropriate transaction-centered organization.
3. Identify the transactions and their defining actions.
4. Note potential situations where modules can be combined.
5. For each transaction, or cohesive collection of transactions, specify a "transaction" module to completely process it.
6. For each action in a transaction, specify an "action" module subordinate to the appropriate transaction module(s).
7. For each detailed step in an action module, specify an appropriate "detail" module subordinate to any action module that needs it.

The use of transaction analysis does not necessarily produce a structure with exactly four levels of processing. Depending on the "problem" to be solved, fully factored transactions may only require a single transaction-level module, or as many as nine or ten levels (Ref 40:308).

Transform analysis was also applied to the "Perform Search Functions" transform of Fig. 10. This increased understanding of the function. The resulting bubble chart is shown in Fig. 11. Note

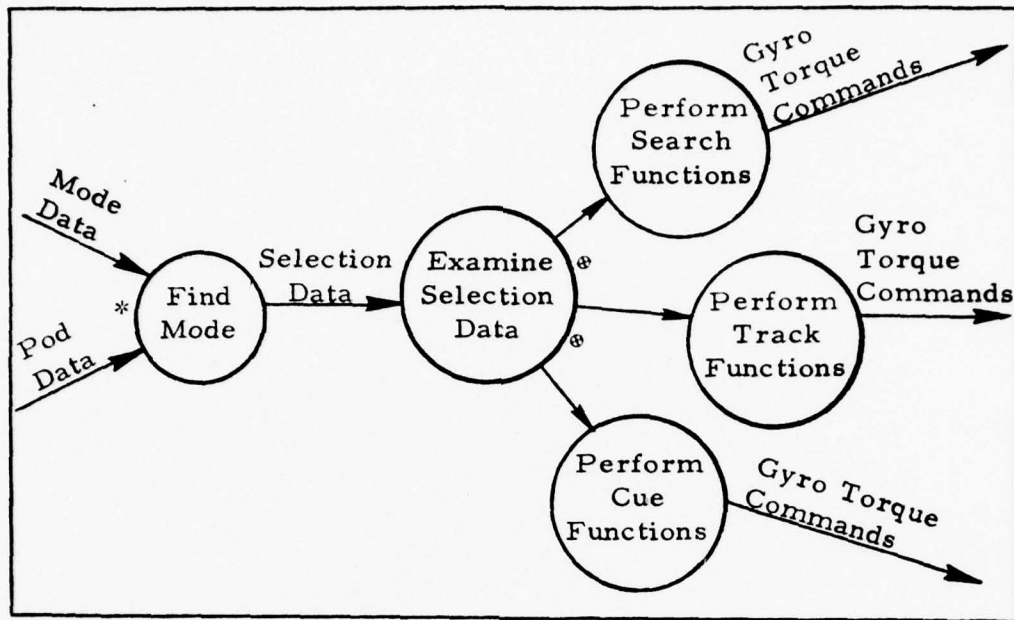


Fig. 10. Bubble Chart for "Perform Target Operations"
(Ref Fig. 3)

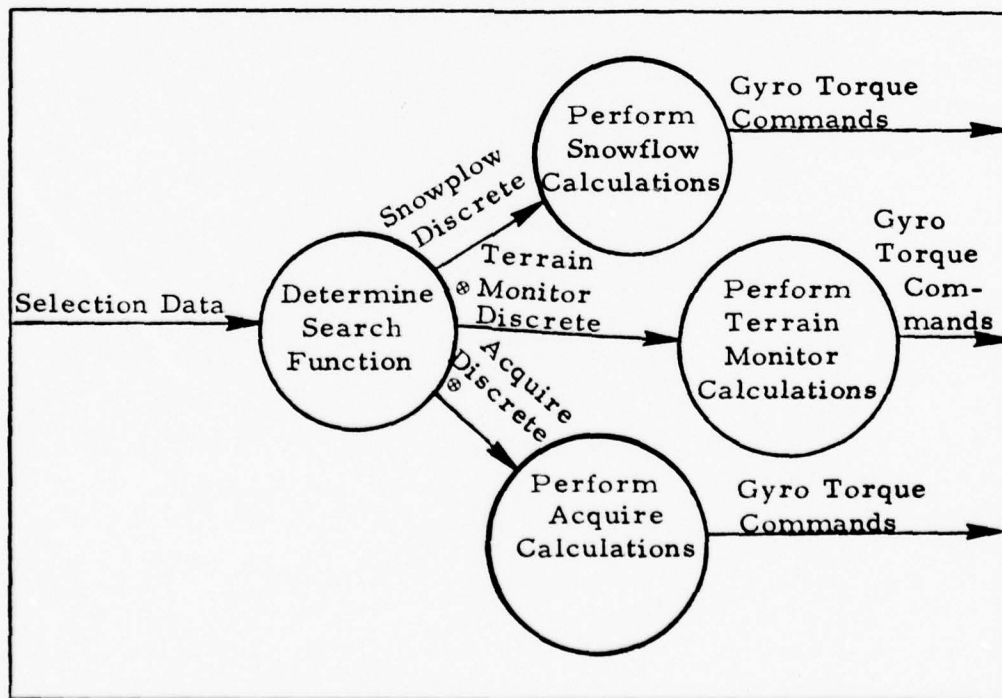


Fig. 11. Bubble Chart for "Perform Search Functions"
(Ref Fig. 10)

that another transaction center existed. Since Fig. 10 was developed at a higher "level of observation," application of the transaction analysis technique would have resulted in a "pancake" structure chart. It would have consisted of only a transaction level (second level).

However, application of the transaction analysis steps to the bubble charts from Figures 10 and 11 lead to the development of the partial structure chart shown in Fig. 12. The structure chart demonstrates the result of using the transaction analysis technique. If the design process were to continue at this point, transform analysis would be applied to the "Track" and "Cue" transforms in Fig. 10. Transaction analysis would be used where applicable.

The completed "Perform Search Functions" branch of the structure chart shown in Fig. 12 is "classical" (Ref 40:304). However, there are no fixed limits on the number of levels in a structure chart developed as a result of using the transaction analysis technique. The "detail level" shown in Fig. 12 resulted from further analysis of the requirements associated with the bubble charts of Figures 10 and 11. Note that the modules outlined with dashed rectangles were placed in their appropriate position in Fig. 12 to add some perspective, and show their relationship with respect to the search function previously expanded. The "Calculate Rate Aiding" module was included since it was associated with the cue and track functions. Finally, OFP slant range data was required for each function (i.e. Search, Cue and Track). Therefore, the slant range

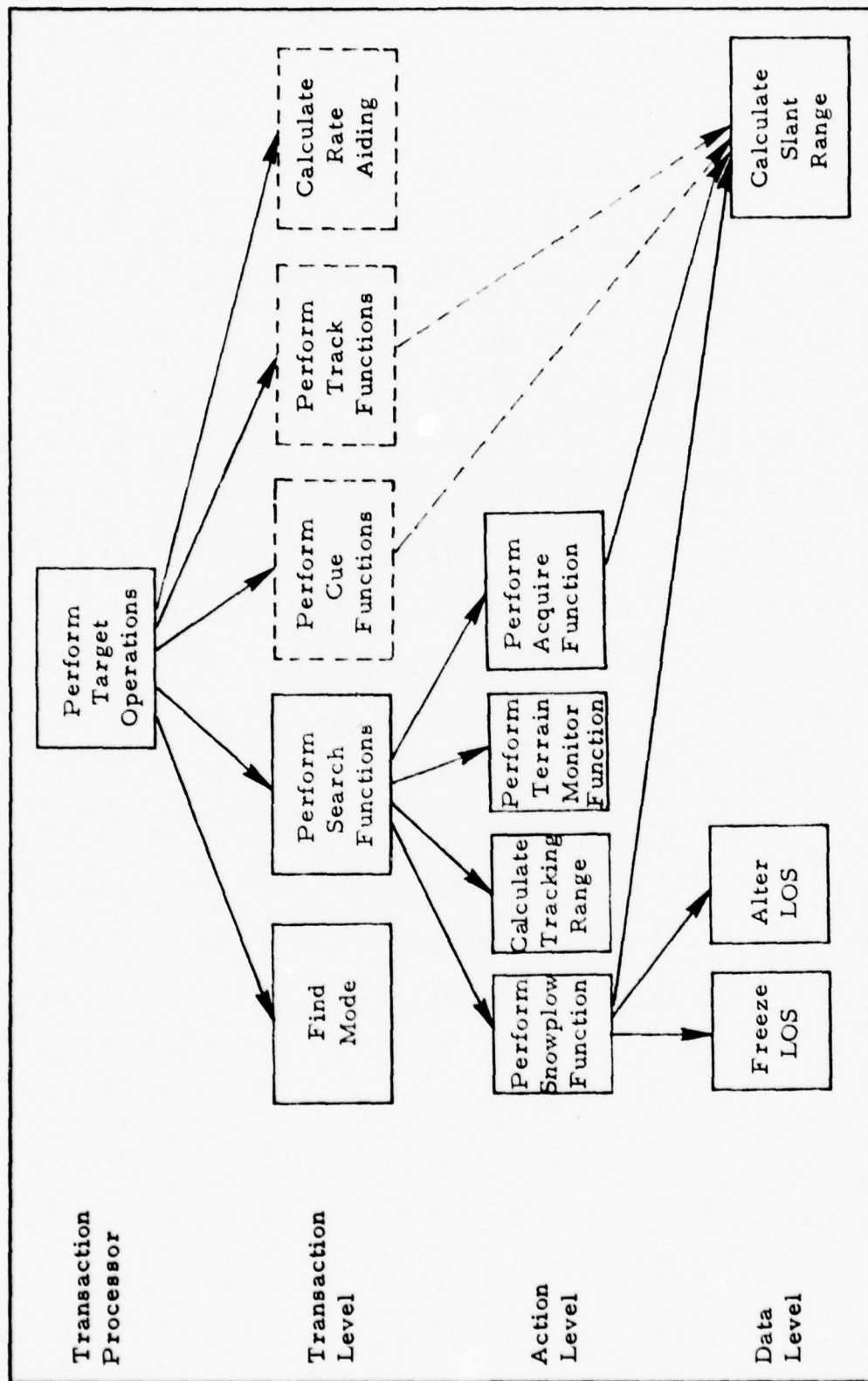


Fig. 12. Partial Structure Chart for "Perform Tracking Operations"

module was placed at the "detail level" as shown in Fig. 12. It was now accessible by the other functions. The position of that module was significant. Fan-in was increased, and perhaps maximized. In addition, the module remained highly cohesive. If the module was coded, the increased fan-in would reduce coding requirements (Ref 40:236, 308).

Comments. At this point, it would be helpful to remember several things. Software design, regardless of the technique used, is generally subjective, and one designer's results may obviously vary from another designer's results. When using the transform or transaction analysis techniques discussed in this chapter, a table of input and output parameters for each level (e.g. first-cut) of structure chart should be developed. Each structure chart should be fully analyzed. Several alternate structures should be developed, if appropriate, for comparison. Finally, a comparative analysis of alternate structures would allow the designer to "find," with confidence, the "best" structure chart for a particular level (e.g. the intermediate structure chart level).

Summary

The initial and revised approaches for the application of several design methodologies were discussed. For the initial approach, the unsuccessful attempt to apply transform analysis to the PAVE TACK OFP was discussed. As a result, the approach

was modified.

In the revised approach, several design techniques were successfully applied to the PAVE TACK mainloop and discussed. Each use of a technique was demonstrated far enough into the design process to show the software engineer/contract monitor "how to get started" on a design. Use of an OFP Part I specification as a basis for design allowed the software engineer/contract monitor a "place" to start. The design techniques allow further understanding of the OFP requirements. They also allow verification of requirements. Finally, by using these techniques, the software engineer/contract monitor is in a better position to discuss a particular OFP design, or make constructive suggestions on an in-process OFP design.

V. Comparison of Modification and Life-Cycle Effects

Introduction

This chapter discusses two hypothetical software modifications applied to a particular task and module of the PAVE TACK OFP mainloop. New designs, from the previous chapter, effected by these modifications are briefly examined and compared to the current OFP software. Finally, some short and long term affects associated with OFPs designed by structured design techniques are discussed.

IDS Image Derotation Task Comparison

Current Software. The IDS Image Derotation task is currently implemented as a single module (Ref 6:61-62). In Chapter IV this task was shown as an IPO scheme. Also, the task was viewed at a certain "level of observation" which permitted easier understanding. However, since the current software equates this task with that of a module, then by definition, the "level of observation" is lower, and the task itself will be examined as a module.

The current "module" for image derotation has certain undesirable characteristics. Coupling and cohesion were analyzed by examining the OFP product specification (Ref 6:61-62, Sec X, 89-90, Sec XXX, 289-291). This single module is common coupled due to its various reads and stores to a shared global data structure. It

has procedural cohesion since several functions are grouped within the module for algorithmic reasons. This was immediately obvious when the coding for the module was examined and compared to the module's flowchart.

Revised Design. In Chapter IV, transform analysis was used to develop a modular structure for the IDS Image Derotation task. The resulting intermediate structure chart in Fig. 8 displays the characteristic of functionality within the structure. The modules are minimally coupled and maximally cohesive as a result of transform analysis. These characteristics reduce complexity, and permit easier, faster modification of software when necessary.

Sample Modification. The proposed modification is hypothetical, and affects the current software and revised design in different ways. The modification requires that the derotation angle's computed rate of change increment also be used in the computations for derotation of the hand control data. Figure 13 shows a simplified flowchart of the current image derotation "module." Portions of the flowchart effected by this modification are denoted with an asterisk. The computations for the derotation angle rate of change increment are located at (C) on Fig. 13. To accomplish the modification, the hand control data computations at (A) must be placed immediately after (C). Since the flowchart segment at (B) uses the computed hand control data from (A), segment (B) must be placed immediately after the new location of (A). The revised flowchart is shown in Fig. 14.

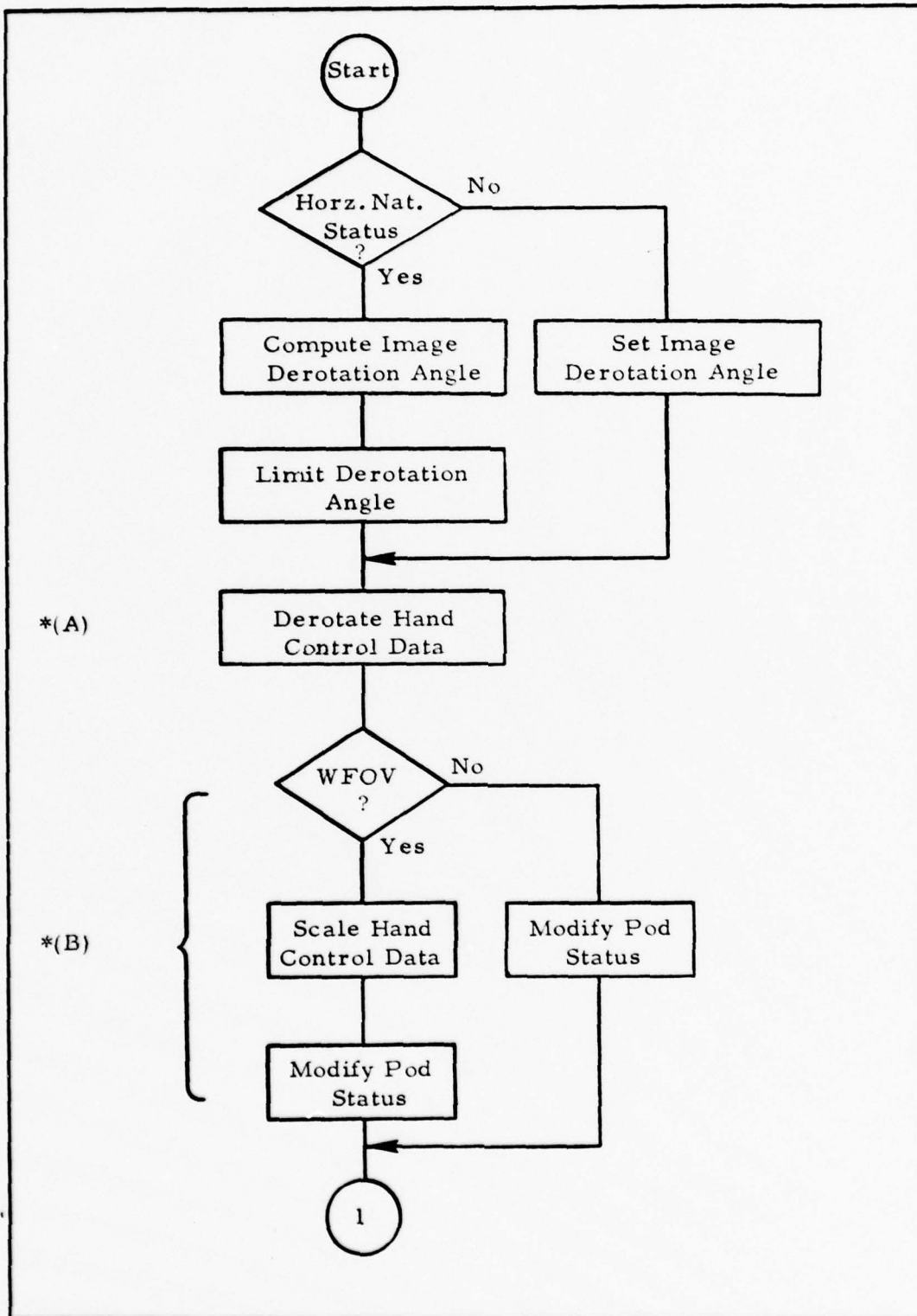


Fig. 13. Simplified Flowchart for IDS Image Derotation
(Ref 6:Sec X, 89-90)

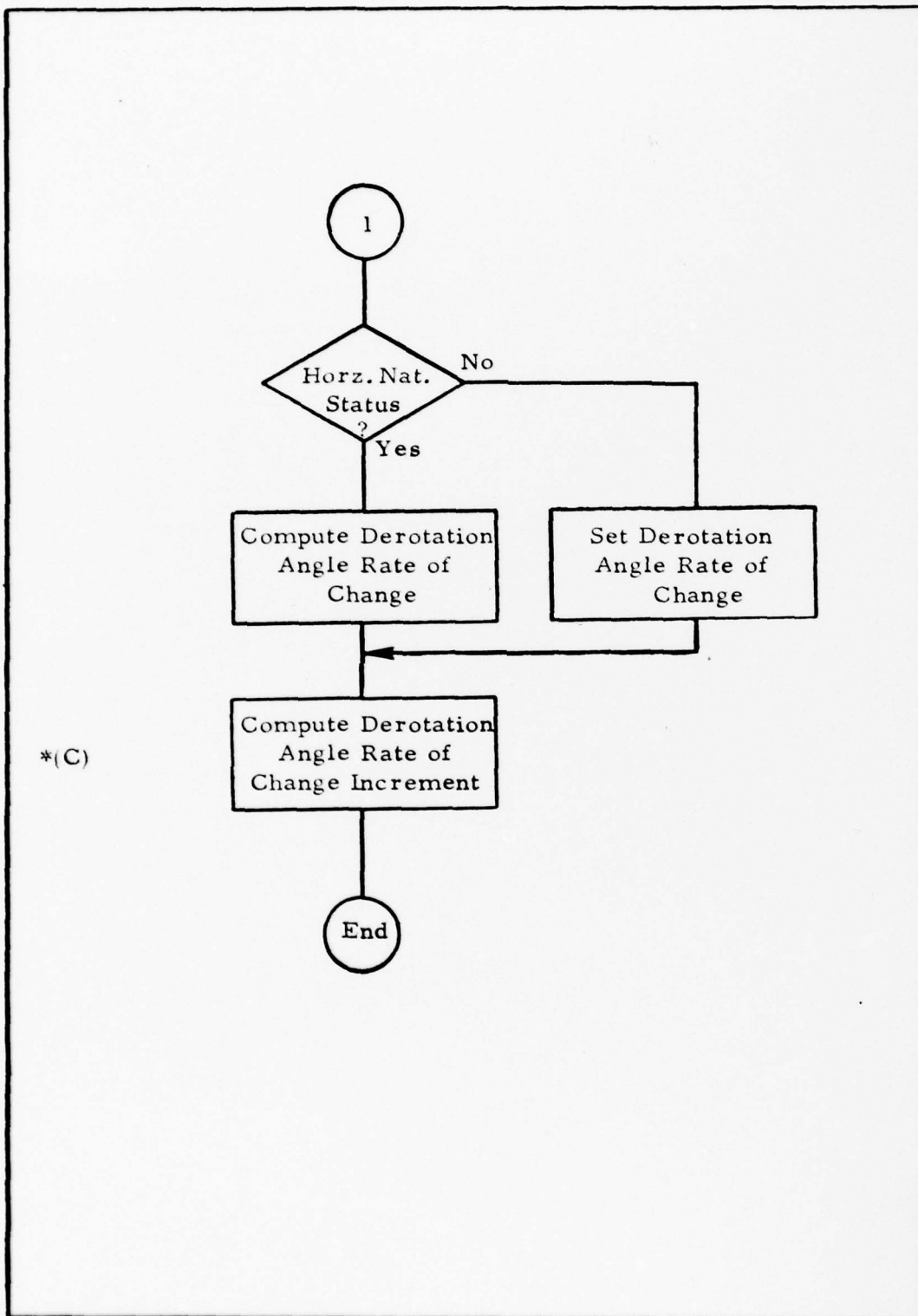


Fig. 13 (continued)

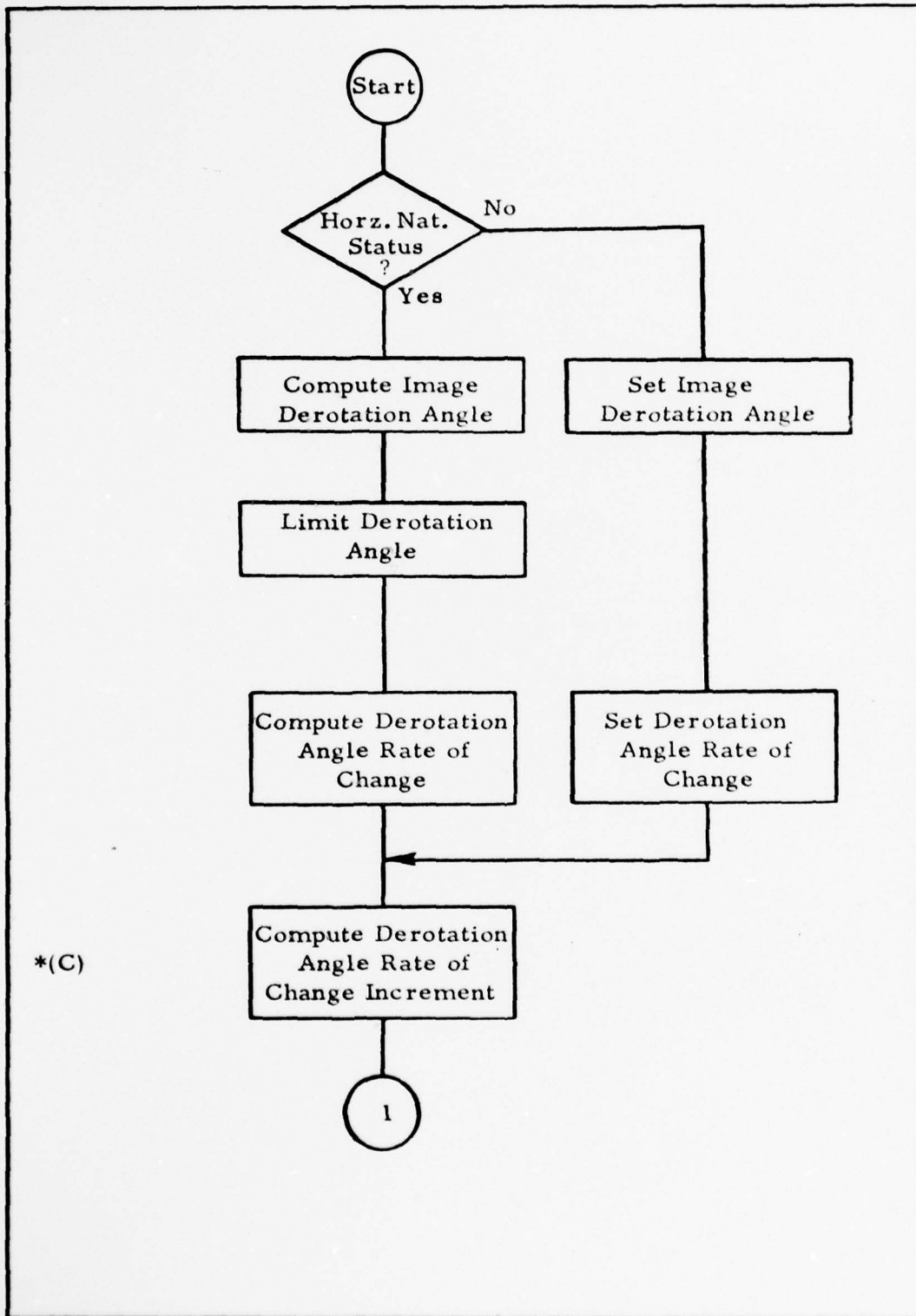


Fig. 14. Modified Flowchart for IDS Image Derotation

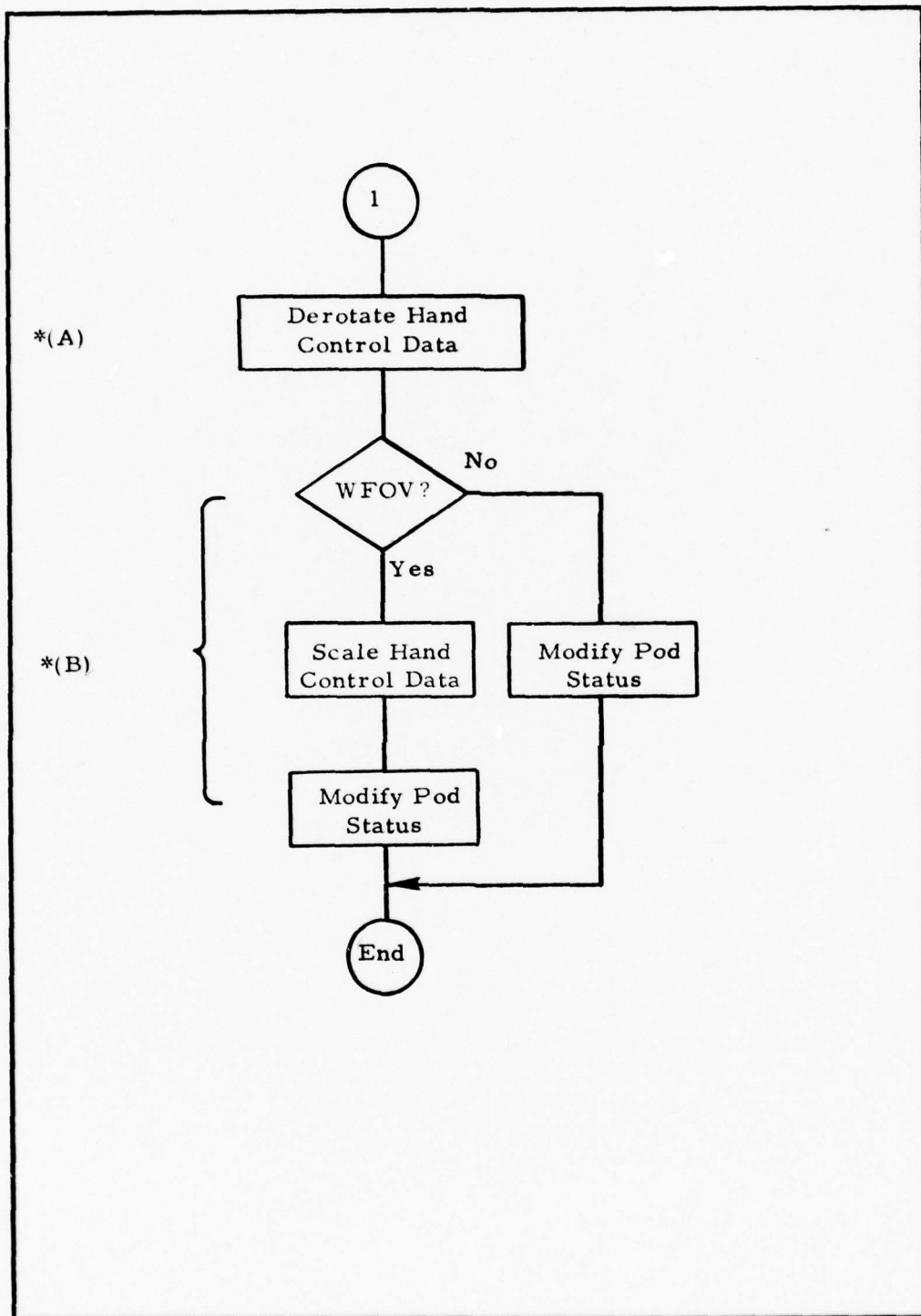


Fig. 14 (continued)

After actual coding of the module, complete retesting of all effected functions within the module would have to be accomplished. In addition, the module's input/output interface would have to be reverified. Finally, the time required to accomplish this modification, including recoding, retesting, implementation, and documentation revision, would obviously add some fixed cost to the total PAVE TACK software life-cycle costs.

For the revised design (Chapter IV), the implementation of this modification would be much easier, and less costly. In Fig. 8 the "Derive Derotation Hand Control Data" module (module four) and the "Compute Derotation Angle Rate of Change Increment" module (module five) are located at the second level in the structure chart. Presently, module four is invoked prior to module five. To affect the required modification, a simple change within the superordinate module would be made to invoke module five prior to module four. This change would permit the rate of change increment data to be passed to module four which computes the derotated hand control data. The only other change would be to incorporate the rate of change increment into the hand control data computations. Since this module would remain data coupled and functionally cohesive, it can be easily retested, and the input/output interface quickly redefined and verified. Finally, since the implementation of this modification is easier and faster in the revised design than with the current "module," the resulting costs are much less. Obviously, the impact

on the total life-cycle costs is less.

Comment. Although this hypothetical modification may seem trivial, the effort and resulting costs of more complex modifications will follow the same trend. That is, modifications will generally be easier, faster, and cheaper due to the previously discussed implicit characteristics normally associated with a software architecture developed with the use of the structured design methodology.

Slant Range Modification Comparison

Sample Modification. This modification is also hypothetical, and also affects the current software and revised design in different ways. The modification requires a change to the algorithm used to derive slant range. At this point, it does not matter what the actual change is, but rather that there must be a revision to the algorithm for slant range.

Current Software. In the current PAVE TACK OFP, slant range is derived in at least seven different modules. It is not difficult to understand that the recoding effort would take time, and in fact, would be somewhat redundant. All effected modules would have to be retested, and each input/output interface reverified against the written requirements. The total cost associated with recoding, retesting, implementation and documentation updating would be added to the PAVE TACK total life-cycle costs.

Revised Design. As shown in the partial structure chart in Fig. 12, there is a single module for slant range calculations. It is accessible by any other module requiring a slant range calculation. After modifying this module, retesting and input/output interface reverification would be accomplished. The module would still remain highly cohesive. It is obvious that the recoding, retesting, implementation, and documentation updating would take less effort and time with the revised design than in the current software. Finally, the modification costs would be significantly less.

Short Term Effects of Structured Designed OFPs

Although this study investigated structured design techniques applied to only part of the PAVE TACK OFP mainloop, certain characteristics associated with this type of software design cause short term effects worth noting. Some of the effects are:

1. Reduced complexity in design allows easier understanding of that design.
2. The goal of low coupling between modules and high cohesion within modules permits easier and less complex coding, testing, and integration.
3. Use of the structured design methodology has an overall effect on the reduction of total system design errors.
4. Since the software architecture of a system would be well defined, management can refine their software development

plan as well as revise development cost estimates.

5. The affects mentioned above would contribute toward a less complicated design review.

Long Term Effects of Structured Designed OFPs

In a similar manner, the same type of characteristics associated with structured designed software also cause long term effects.

Some of the effects are:

1. A total system is more easily maintainable and manageable.
2. The total system is more flexible to changing requirements.
3. Less total time and effort are consumed in making revisions to software.
4. All effects (short and long term) would contribute toward the reduction of total life-cycle system costs.

Summary

It was obvious from the two hypothetical modifications that the functionally modular structures produced (Chapter IV) by using structured design techniques responded easily and effectively to changing requirements. Modifications were discussed for the IDS Image Derotation task and the slant range algorithm. Finally, some short and long term affects associated with OFP software designed by structured design techniques were discussed. Characteristics and benefits of structured designed software were previously discussed.

A generalized OFP familiarization/design methodology includes some techniques which may be used to achieve these characteristics and benefits, and is described in the next chapter.

VI. A Software Engineering OFP Design Methodology

Introduction

This chapter describes a suggested software engineering OFP design methodology developed for Air Force avionics software engineers/contract monitors. The need for such a methodology is briefly discussed as well as the objective and goals established to develop it. The methodology consists of three phases: (1) understanding the flight program requirements; (2) use of a technique for understanding flight program tasks and task sequencing; and (3) use of structured design techniques for flight program tasks. Each phase is discussed.

General

Software engineers monitoring the development of avionics flight software (e.g. operational flight programs) often rely completely on the software developers' choice of design. They accept the software end-item without much input to the producer about design during preliminary design of the software architecture. A good software design requires discussion of design alternatives between the developer and user. This interaction should result in a mutually agreeable design which also satisfies the particular flight software written requirements definition (Part I specification). In order for the software engineer to effectively participate in and

monitor the development of an OFP, it would be beneficial to have a software engineering methodology which would permit a more efficient and effective evaluation of OFP design alternatives. The methodology could also be used to verify the software developers interpretation of the written requirements definition.

Use of this methodology by a software engineer/contract monitor permits establishment of various stages of familiarization. This important effect allows two primary uses of the methodology. First, the user of this methodology can gain a greater understanding of the OFP requirements definition (draft Part I specification). Second, the user may develop a separate OFP design for familiarization. The latter not only permits greater understanding of requirements, but more importantly, allows more detailed and thorough discussions between the software engineer/contract monitor and the software producer about OFP design alternatives.

Requirements for Methodology Development

To help develop a useful and meaningful methodology, an objective and accompanying goals were established. The objective was to develop an OFP design methodology or approach which would assist or aid software engineers/contract monitors (the methodology user in this case) in the initial understanding and/or design of OFP software from a draft Part I specification. It was desirable to include the finite-state technique and the techniques of structured design as

demonstrated in Chapter IV.

A reachable set of goals were developed. The goals were:

1. The approach should aid the user from a practical point of view if it is to be useful.
2. It should be simple enough to allow a free-flow development of ideas for the design, yet comprehensive enough to be useful.
3. It should be understandable in order to facilitate ease of use and maximize information transfer to the user.
4. It should be flexible in order to permit use on different OFP requirement definitions.
5. It should allow hardware independence during design.

Design Methodology

The design methodology may be applied in three phases. Phase one consists of understanding the problem to be solved. Phase two suggests the steps to be taken for application of the finite-state technique to OFP tasks. Finally, phase three will complete the preliminary design by application of the structured design techniques. The software engineer/contract monitor may stop at the end of any phase. Use of each successive phase lowers the "level of observation" applied to the "problem" and increases understanding of the overall OFP design.

Phase I. In phase one, there are certain, and probably obvious, steps that can be taken by the user to understand the "problem" to be solved. In this case the problem is to understand the requirements for the development of a viable OFP design. A top-down approach is suggested for understanding those requirements.

The understanding "process" is started by achieving a thorough understanding of the software system specification. Mission and technical requirements of the OFP are explained in the specification as well as functional areas, programming language requirements, design standards, and so forth. The system specification establishes the initial functional baseline for the system (Ref 2:15). To further decompose the problem and increase understanding, the next suggested step in phase one is to obtain the written requirements definition.

It will be necessary to gain a detailed understanding of the written requirements definition. The draft Part I software specification contains the particular written requirements for the design, development, functional performance, test and qualification of the OFP (Ref 2:15). The draft Part I specification will be used as the "basis for design" in the remaining phases of this design methodology.

Finally, familiarization with any written assumptions and limitations associated with the OFP is suggested. They may be required or useful during the design process. The assumptions and limitations can usually be obtained from either the system or draft Part I

specification, or both. Implementation and hardware information is "nice to know," but inclusion of such information during the preliminary design should be avoided. This type of information will be useful later during the detailed design (e.g. algorithm implementation, timing, etc.). The preliminary design process should remain free of biases or limitations since the resulting OFP design may be unduly affected. If deeper understanding of the OFP is desired, or the software engineer/contract monitor must develop a high-level ("level of observation") OFP design, the initial steps are suggested in phase two.

Phase II. In phase two a basic sequence of steps are suggested for the application, where applicable, of the finite-state technique as discussed in Chapter IV. In Chapter IV a particular OFP mainloop contained tasks represented as an IPO scheme for a particular "level of observation." Typically, the IPO relationship will hold for OFP mainloop tasks as well as some tasks (e.g. built-in-testing) found in OFP executives. Therefore, the software engineer/contract monitor can use the suggested steps in this phase for a "first interpretation" of the requirements definition and preliminary design.

The finite-state technique is applied to the OFP major tasks in the same manner as discussed in Chapter IV. In addition the software engineer/contract monitor should write a brief description of each major task in the OFP. Each task description should represent an individual rewritten interpretation of the requirements from the

draft Part I specification. It is important at this point in time that the designer's interpretation of task requirements and the task state diagram be committed to paper. In addition to the notation described in Chapter IV for use with this technique, part of Appendix D contains some additional task description notation which may be helpful.

Abstraction can be used where appropriate to simplify the state diagram. Abstraction was demonstrated in Chapter IV (i. e. "Perform Target Operations").

The written interpretations of tasks and the validity of the state diagram should be verified against the requirements definition (i. e. draft Part I specification) for any misinterpretations. Where appropriate, the system specification may be used. However, the draft Part I specification should contain all information necessary for design. If it doesn't, it may be inconcise, incomplete or ambiguous. Appropriate corrective action(s) should be taken to correct the draft Part I specification if this problem exists.

A viable task state diagram and accompanying task descriptions may need revision (e. g. an error exists in the diagram). Correct or modify the task state diagram as necessary. Clarify written task descriptions where appropriate. When these steps are completed, the user is ready to proceed to the final steps in this phase.

At this point, if time permits, it is worthwhile to initiate a "reader cycle." The written OFP design interpretation can be given

to another knowledgeable person for a critique. The reader should (also) use the draft OFP Part I specification to verify the software engineer/contract monitor's design interpretation. Comments should be written by the reader, and no form of communication should take place between the two people concerned before or during the critique process.

At the completion of the reader's critique, existing discrepancies should be discussed with the reader. The software engineer/contract monitor should correct or modify any valid discrepancies noted. When these steps are complete, two products of this phase should exist: a final task state diagram and clear, concise, unambiguous written task descriptions.

Phase two is now complete. The designer may stop at this point or proceed to phase three. This decision can depend on many factors. However, the two most important factors are the depth of understanding needed and the level of familiarization desired. Completion of phase three in this methodology should result in a total understanding of the OFP preliminary design requirements. The software engineer/contract monitor should in fact end up with a viable OFP preliminary design. However, a prime objective at the completion of phase three would be a discussion of OFP design alternatives or suggestions about an "in-process" design with the software producer.

Phase III. In this phase, the structured design methodology is applied to individual OFP tasks (state diagram nodes). The structured design methodology can be used as discussed in Chapter IV. The transform analysis technique is applied to each "defined" task. Either the Constantine or Hughes version of this technique may be used by the software engineer/contract monitor. If transaction centers are revealed, the transaction analysis technique is used. This technique was also discussed in Chapter IV. The structured design process continues until the final task structure charts are completed, including their supporting information.

At the completion of phase three the software engineer/contract monitor should have a detailed understanding of the OFP requirements plus a final documentation package to support that understanding. The information contained in the final documentation package is easily transferable to another person. It should include:

1. If applicable, a task state diagram (from phase two) and concise written descriptions for each task shown on the diagram.
2. Final bubble charts which directly reflect the requirements definition for each defined task.
3. A final, fully annotated structure chart for each task which represents the hierarchical software structure developed from its associated final bubble chart.

4. A written functional description of each module of each final task structure chart.
5. A final module input and output parameter table for each final task structure chart.
6. A final, full analysis of module coupling, cohesion, scope-of-effect and scope-of-control for each final task structure chart.

Obviously, the software engineer/contract monitor can use the final task state diagram and final task structure charts together for an overall system representation and viewpoint. The final documentation package can now be used for technical and management in-house discussions and briefings as well as backup information for technical discussions of design alternatives with a software producer.

Limitation

Use of the finite-state technique in this methodology was limited to non-concurrent processes. The technique was discussed in Chapter IV and successfully applied to the PAVE TACK OFP mainloop. The mainloop processing was sequential. Concurrent processes are beyond the scope of this study.

Summary

This chapter presented a description of a suggested software engineering methodology developed to help software engineers/contract monitors evaluate OFP design alternatives more efficiently and

effectively. The general need for such a methodology was briefly discussed as well as the objective and goals established to develop this methodology. The design methodology or guideline consisted of three phases and each phase was described. Use of this methodology also permits the software engineer/contract monitor to verify the OFP software developers interpretation of the written requirements definition.

VII. Results, Conclusions, and Recommendations

Introduction

This chapter highlights the results and conclusions of this study. Objectives for this investigation were met. Recommendations are made for future avionics efforts.

Results

A modified finite-state technique was successfully applied to the PAVE TACK OFP mainloop. A state diagram was developed. Each node of the state diagram represented an OFP mainloop task. The diagram displayed proper sequencing of the mainloop tasks. This technique was limited to non-concurrent processes. The task state diagram directly reflected the mainloop control structure.

The transform analysis and transaction analysis techniques were successfully applied to separate OFP mainloop tasks. Initially transform analysis did not work when applied to the entire mainloop since each separate task functioned as an input-process-output scheme. Data were not passed directly between tasks. Therefore, attempts to create a mainloop bubble chart produced invalid results. However after revising the approach, separate task bubble charts and structure charts were successfully developed and analyzed. Coupling, cohesion, scope-of-effect and scope-of-control were examined. During an application of transform analysis, transaction analysis was

applied when transaction centers were discovered. A structure chart was developed and analyzed.

Two hypothetical modifications were made to certain parts of the redesigned OFP. The modifications were easily made and consumed little time. Modification costs were reduced with structured design software when compared to the current PAVE TACK OFP software.

A generalized OFP familiarization and design methodology was proposed. It provides guidance for using the finite-state and structured design techniques. A software engineer/contract monitor participating in or monitoring the design and development of an OFP can use a draft Part I specification and these techniques to gain an understanding of design requirements. The software producers interpretation of an OFP's requirements can also be verified at the preliminary design review (PDR).

Conclusions

The bubble chart used in the transform analysis technique is a graphical representation of the requirements definition. It helps the user of the technique to understand the requirements. In addition, it can be used to verify the requirements of the (draft) Part I software specification. Inconcise written descriptions of OFP tasks and inconsistent terminology used to describe different tasks are readily discovered. Areas of superficial or extremely detailed OFP

information are easily detected with the use of a bubble chart.

The OFP requirements definition in a Part I specification must be complete, concise and unambiguous. Requirements written in "natural" English satisfy familiarization needs, but not OFP design needs. The OFP requirements should not be so loosely written that the software producer's "experience" will be expected to provide the missing information. MIL-STD-483 and MIL-STD-490 (Ref 3:Sec II, 28-32) are reasonable guidelines for development of a Part I specification, but specific methodologies should be specified to the software producer for requirements analysis and design. This will help ensure complete, concise and unambiguous software specifications, and a "better" OFP design.

Due to the scope of this study, a partial conclusion was drawn about the techniques used in this study applied to an OFP executive. No attempt was made to show conclusively that the finite-state technique could be applied to develop a total OFP executive. Also, the structured design techniques were not applied to OFP executive tasks. Concurrent processes were not considered in this study. However, based on the successful results of applying these techniques to an OFP mainloop not involving concurrent processes, these techniques can be applied in the same manner to an OFP executive of the same class (non-concurrent) of processes.

Use of the OFP familiarization and design methodology suggested in this report should permit early interaction with the software

producer concerning OFP design alternatives. In choosing the proper alternative, consideration should also be given to the software maintenance aspect due to the changing requirements normally associated with OFPs. Use of this methodology by the software engineer/contract monitor will obviously depend on personal time available and the project schedule. The availability of a draft Part I specification prior to PDR will also affect "how much" of the methodology can be used on a particular OFP.

Recommendations

Requirement analysis and design methodologies or techniques should be specified in the statement of work written by the software engineer/contract monitor. The goal is to obtain a complete, concise and unambiguous requirements definition and the "best" OFP design. Requirements analysis/definition methods such as Structured Analysis (Ref 30; 31; 32; 33), Problem Statement Language/Problem Statement Analyzer (PSL/PSA) (Ref 35) or SREM (Ref 9) are suggested. Structured Design (Ref 20; 21; 40) or Composite Design (Ref 26) are suggested for preliminary OFP design in addition to the Finite-state technique discussed in this report. If Structured Analysis is used for requirements analysis, a method exists that permits an easy transition to the structured design techniques (Ref 23:71-93). Finally, any deviations in the basic OFP design that changes the task design (structure charts) to facilitate OFP

implementation should be described by the software producer in the Part II specification.

The requirements analysis and design methods previously suggested obviously will not solve all the requirements and design engineering problems. However, use of these methods will significantly reduce the problems. Better OFP designs will be the result. The required effort and time will be less for OFP modifications to meet changing requirements, and thus, total software life-cycle costs will be reduced.

Bibliography

1. Aeronautical Systems Division. Management Guide to Avionics Software Acquisition. ASD-TR-11. Volume I. Wright-Patterson AFB, Dayton, Ohio, LOGICON, June 1976.
2. -----. Management Guide to Avionics Software Acquisition. ASD-TR-11. Volume II. Wright-Patterson AFB, Dayton, Ohio, LOGICON, June 1976.
3. -----. Management Guide to Avionics Software Acquisition. ASD-TR-11. Volume III. Wright-Patterson AFB, Dayton, Ohio, LOGICON, June 1976.
4. Aeronutronic Ford Corporation. Computer Program Development Specification for PAVE TACK Operational Flight Program, AS 125720, Part I of Two Parts, (Draft Copy), 30 April 1975.
5. -----. Computer Program Development Specification for PAVE TACK Operational Flight Program CPCI, AS125612C, Part I (Final), 29 September 1975.
6. -----. Computer Program Development Specification for PAVE TACK Operational Flight Program, AS125720, Part II of Two Parts, 24 June 1977.
7. -----. PAVE TACK Operational Flight Program User's Manual (Preliminary), U-6205, 23 February 1976.
8. Air Force Avionics Laboratory. Operational Software Concept (Phase Two). AFAL-TR-75-230, Fort Worth, Texas, General Dynamics, January 1976, (ADA 021327).
9. Alford, Mack W. "A Requirements Engineering Methodology for Real-Time Processing Requirements," IEEE Transactions on Software Engineering, 60-69 (January 1977).
10. Archibald, W. R., and E. S. Hinton. "Critical Needs in Avionic System Software," IEEE 1974 NAECON Proceedings, Dayton, Ohio, 475-481 (May 1974).
11. Barnes, B. H. "A Programmer's View of Autonomy," ACM Computing Surveys, 4(4):221-239 (December 1972).

12. Boehm, B. W. Software Engineering. Redondo Beach, California: TRW, 1976.
13. Brooks, Frederick P., Jr. The Mytical Man-Month. Massachusetts: Addison-Wesley, 1975.
14. Coffman, E. G., Jr., and P. J. Denning. Operating Systems Theory. New Jersey: Prentice-Hall, Inc., 1973.
15. Ford Aerospace & Communications Corp., Critical Item Product Fabrication Specification for Digital Computer CP-12921 AV-26, Part I Specification No. AS 125694B of 20 December 1976, and Part II Specification No. AS 125694, 20 December 1976.
16. Hamilton, M., and S. Zeldin. "Higher Order Software--A Methodology for Defining Software," IEEE Transactions on Software Engineering, 9-32 (March 1976).
17. HIPO--A Design Aid and Documentation Technique, IBM, GC20-1851-0, October 1974.
18. IBM Federal Systems Division. PAVE TACK TC-3 Computer/Software Critical Design Review. Briefing Charts, 10 June 1975.
19. Jackson, M. Principles of Program Design, Academic Press, New York, 1975.
20. Jensen, E. "Structured Design." 1975 IR&D Structured Design Methodology, 2: Hughes Aircraft Company, April 1976.
21. -----, "The Mystique of Structured Design." 1975 IR&D Structured Design Methodology, 3: Hughes Aircraft Company, April 1976.
22. Jensen, Louis K. "On the Education of Software Development Managers," COMPCON 77 (Fall): 12-16 (September, 1977).
23. Marvin, Kenneth L. Structured Analysis and Design of a Sateilite Simulator. MS Thesis. Wright-Patterson AFB, Ohio: Air Force Institute of Technology, September 1977.
24. McGowan, Clement L., and John R. Kelly. A Review of Some Design Methodologies, TPO53, Softech, Inc., September 1976.

25. Miller, Barry. "Tests Set for Attack/Surveillance System," Aviation Week and Space Technology, 47-51 (February 2, 1976).
26. Myers, Glenford J. Reliable Software through Composite Design. New York: Mason/Charter Publishers, 1975.
27. Parnas, D. L. "A Technique for Software Module Specification with Examples," Communications of the ACM, VOL. 15, NO. 5, 330-336 (May 1972).
28. -----. "On the Criteria to Be Used in Decomposing Systems into Modules," Communications of the ACM, VOL. 15, NO. 12, 1053-1058 (December 1972).
29. Reifer, Donald J., and Stephen Trattner. "Software Specification Techniques: A Tutorial," Compcon 76 (Fall), 39-44 (September, 1976).
30. Ross, D. T., and K. E. Schoman, Jr. "Structured Analysis for Requirements Definition," IEEE Transactions on Software Engineering, 6-15 (January 1977).
31. Ross, D. T. "Structured Analysis (SA): A Language for Communicating Ideas," IEEE Transactions on Software Engineering, 16-34 (January 1977).
32. Softech, Inc. Structured Analysis Reader Guide. Waltham, Massachusetts: May 1975.
33. -----. An Introduction to Structured Analysis and Design Technique. Waltham, Massachusetts: November 1976.
34. Stevens, W. P., G. J. Myers, and L. L. Constantine. "Structured Design," IBM Systems Journal, 2: 115-139 (1974).
35. Teichroew, Daniel, and E. A. Hershey III. "PSL/PSA: A Computer-Aided Technique for Structured Documentation and Analysis of Information Processing Systems," IEEE Transactions on Software Engineering, 41-48 (January 1977).
36. Trainor, W. Lynn. "Software for Avionics: An Overview," IEEE 1975 NAECON Proceedings, Dayton, Ohio, 227-233 (May 1975).
37. -----. "Support Software Systems for Avionics--A Tutorial," IEEE 1977 NAECON Proceedings, Dayton, Ohio, 998-1003 (May 1977).

38. TRW Defense and Space Systems Group. PAVE TACK Avionics Integration Support Facility Requirements Analysis Report on Life Cycle Requirements for OFP Support. Redondo Beach, California: TRW, May 1977.
39. Wirth, N. "Program Development by Stepwise Refinement," Communications of the ACM, VOL. 14, NO. 4, 221-227 (April 1971).
40. Yourdon, E., and L. L. Constantine. Structured Design. New York: Yourdon, Inc., December 1975.

Appendix A

Brief Discussion of Operational Flight Program Concepts

Avionic software can be categorized as either mission or support software. Mission software executes within a flight computer (on-board computer) and performs mission-related functions. A frequently used analogous term is "embedded" software. Support software aids in the design and production of avionic systems (Ref 37:998).

Mission software can be further divided into two sub-categories. The first is real-time functions, which are performed with an OFP, and the second is system testing which is performed with an operational test program (Ref 37:999).

The OFP contains functions which are executed in a flight computer in real-time (or near real-time), and perform the primary aircraft mission functions such as executive control, navigation, guidance, targeting and display processing (Ref 37:999). A combination of these functions is not normally found in ADP software. Also, the OFP is often used to close avionic control loops. These may range from relatively slow control, such as range update of navigation parameters, to rapid control in a critical aircraft/missile control loop function which must be completed in a few milliseconds. The requirement may exist for displaying all pertinent information

for aircrew evaluation and reaction in real-time where aircrew safety is directly involved, and as a result, the software becomes considerably more integrated and complex. The OFP can also be used to perform continuous checking of specified hardware parameters. Finally, the requirement may exist for absolute software program correctness so that, conceivably, the OFP might be used to control the release of a nuclear weapon. It is easy to see that functional requirements are quite different from normal ADP (Ref 10:476).

Inputs to an OFP generally originate from sensors via analog-to-digital converters, and control actions are automatically taken without a stored record of the input which caused them (some applications require a stored "snap-shot" of certain system or program parameters on a flight recorder). This type of closed-loop operation can depend upon a variety of hardware devices, each one having many failure modes not necessarily predictable. Therefore, recovery and real-time control requirements exist, and the software must continue to operate in a reliable manner regardless of any hardware problems that may occur (Ref 10:476).

AD-A056 517 AIR FORCE INST OF TECH WRIGHT-PATTERSON AFB OHIO SCH--ETC F/G 1/3
SOFTWARE STRUCTURE DESIGN TECHNIQUES APPLIED TO EMBEDDED COMPUT--ETC(U)
DEC 77 R J DE SANTO
UNCLASSIFIED AFIT/GE/EE/77-14

NL

2 of 2
AD
A056 517



END
DATE
FILMED

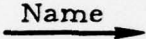
9-78

DDC

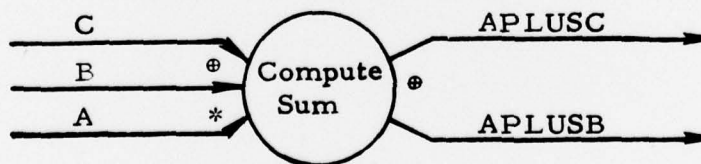
Appendix B

Bubble Chart Symbol Descriptions

The following symbols are used in this report and are described as follows (Ref 40:54-62, 565-566):

<u>Symbol</u>	<u>Description</u>
	The circle represents a transform of data from one form to another. "Name" identifies the transform process and is a verb or verb phrase.
	The labeled arrow represents a data stream either entering an initial transform, leaving a final transform, or connecting one transform to another. "Name" identifies the data stream and is a noun or a noun phrase.
*	The asterisk represents the conjunction operator which is used to denote the "AND" function of data streams.
⊕	The ring-sum symbol represents the disjunction operator which is used to denote the "EXCLUSIVE-OR" function of data streams.

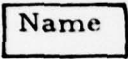
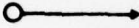
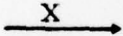
Example:



Appendix C

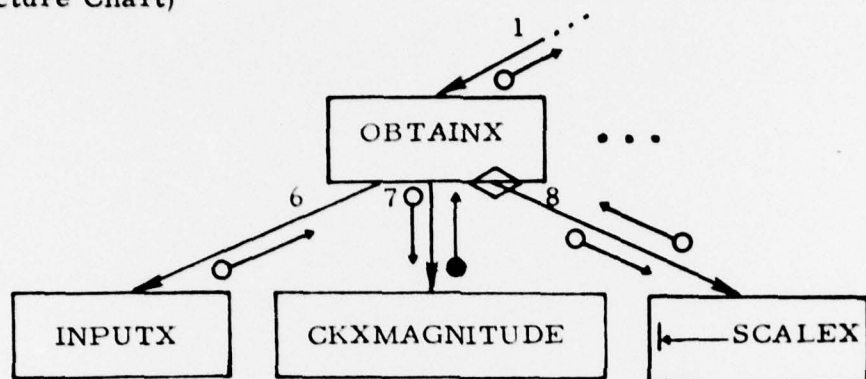
Structure Chart Symbol Descriptions

The following symbols are used in this report and are described as follows (Ref 40:547-552, 589-699):

<u>Symbol</u>	<u>Description</u>
	A simple rectangle represents a module. "Name" is the module name and is a verb. A descriptive verb phrase may be used in place of a single name.
	An arrow with a dot on its tail denotes control information.
	An arrow with a small circle on its tail denotes data.
	A diamond denotes a conditional decision embedded in a module.
	A plain labeled arrow represents the super-ordinate-subordinate reference to a module. "X" may be used to represent the Arabic number identifier of the subordinate module.

Example:

(Structure Chart)



(Table)

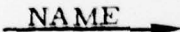
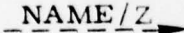
Parameters

Module	Input	Output
1	---	XVALUE
6	---	XVALUE (RAW)
7	XVALUE (RAW)	<u>OKFLAG</u>
8	XVALUE (RAW)	XVALUE (SCALED)

Appendix D

Symbols and Notation for Use with Finite-State Technique

The following symbols and notation may be used to extend the finite-state technique (discussed in Chapters IV and VI):

<u>Symbol or Notation</u>	<u>Description</u>
	The circle represents a node or task on the task state diagram. "X" is an Arabic number which denotes the node identifier. "PROCESS" is a verb or verb phrase that identifies the task process. This task is preemptable.
	Same description as above except that the double circle represents a non-preemptable task.
	This directed line or arc represents a transfer of control from one task to another. "NAME" identifies the condition of transfer.
	This dashed line or arc also represents a transfer of control from one task to another, but in addition, control information may be passed to the next task. "Z" represents the control information identifier.
$T_i(I, P, O, t)$	A 4-tuple may be used to identify certain information to be associated with or contained in a written description of task number i . "I" and "O" are identifiers, each of which represents a set of numbers. Each set contains Arabic numbers that identify the major input and output data associated with task i after task i is initiated and before it terminates. "P" is an identifier which represents the written description of the task i process. "t" is the time allowed (in some convenient time unit) for completion of task i processing. The 4-tuple

notation for each task can be annotated onto the task state diagram. Some convenient method may be used to associate the 4-tuple notation with the task or node representing the task (e.g. a "squiggle arrow," ). Note that a 5-tuple, $T_i(I, P, O, t\text{-min}, t\text{-max})$, could be used for task i if a minimum and maximum time for task completion are applicable.

Vita

Robert J. DeSanto was born in Hackensack, New Jersey on January 10, 1943. He graduated in June 1961 from the Bergen County Vocational and Technical High School, Hackensack, New Jersey. After working as an electronics technician, he enlisted in the U.S. Air Force in September 1963. He attended the University of Minnesota at Duluth, the University of Albuquerque (New Mexico) and the University of New Mexico. He was selected by AFIT for the Airman Education and Commissioning Program, and attended Arizona State University, College of Engineering, starting in January 1970. He majored in digital systems and computer science, and received his BS in Engineering in August 1972. On November 30, 1972 he graduated from OTS, and was commissioned a Second Lieutenant. He worked as a Computer System Design Engineer at Air University (AU), Maxwell AFB, Alabama with the additional duty of technical advisor to the Commandant, Air War College (AWC). He was also a Liaison Officer (between AWC and AU) for educational computer applications before coming to the Air Force Institute of Technology in June 1976 for a Master of Science degree in Digital Systems Engineering. Captain De Santo is a member of Eta Kappa Nu, ACM and the IEEE Computer Society.

Permanent Address: 98 MacArthur Drive
Saddle Brook, NJ 07662

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFIT/GE/EE/77-14	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE STRUCTURE DESIGN TECHNIQUES APPLIED TO EMBEDDED COMPUTER SYSTEM SOFTWARE		5. TYPE OF REPORT & PERIOD COVERED MS Thesis
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Robert J. De Santo Captain USAF		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Air Force Institute of Technology (AFIT-EN) Wright-Patterson AFB, Ohio 45433		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS ASD-ENAIA Directorate of Avionics Engineering Wright-Patterson AFB, OH 45433		12. REPORT DATE December 1977
		13. NUMBER OF PAGES 103
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		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 190-17 <i>Jerral F. Guess</i> JERRAL F. GUESS, Captain, USAF Director of Information		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number)		
Structured Design	PAVE TACK	Mainloop Tasks
Transform Analysis	Finite-State Technique	Design Methodology
Transaction Analysis	Embedded Software	
Avionics Software Design	OFP	
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Constantine's structured design methodology is applied to a real-time (flight) software program. A modified finite-state technique is successfully applied to an operational flight program (OFP) mainloop. Transform and transaction analysis are successfully applied to mainloop tasks. Each technique is demonstrated to show avionics software engineers/contract monitors "how to get started" on a design. Two hypothetical software modifications		

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

illustrate advantages of structured design over the current software implementation. Short and long term effects of structured designed OFPs are briefly described. An OFP familiarization and design methodology is developed for avionics software engineers/contract monitors. Structured design techniques are beneficial to the software engineer/contract monitor during initial understanding of OFP design requirements from a draft Part 1 specification. Design alternatives may be considered and the software producers interpretation of design requirements may be verified. This effort was sponsored by the Aeronautical Systems Division (ASD/ENALA) located at Wright-Patterson Air Force Base, Ohio.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)