

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

DEVELOPMENT OF A STRUCTURED DESIGN AND PROGRAMMING
METHODOLOGY FOR EXPERT SYSTEM SHELLS UTILIZING A VISUAL
PROGRAMMING LANGUAGE; APPLICATION OF STRUCTURED
METHODOLOGY TO THE MK92 MAINTENANCE ADVISOR EXPERT
SYSTEM, PERFORMANCE MODULE PROTOTYPE

by

Lucy Michelle Smith

September, 1994

Thesis Advisor:

Magdi Kamel

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

19950119 023

REPORT DOCUMENTATION PAGE			Form Approved OMB Np. 0704	
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.				
1. AGENCY USE ONLY (Leave blank)		2. REPORT DATE September 1994		3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE: Development of a Structured Design and Programming Methodology for Expert Systems Shells Utilizing a Visual Programming Language; Application of Structured Methodology to the MK92 Maintenance Advisor Expert System, Performance Module Prototype			5. FUNDING NUMBERS	
6. AUTHOR(S) Lucy Michelle Smith, LT, USNR				
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey CA 93943-5000			8. PERFORMING ORGANIZATION REPORT NUMBER	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) Naval Surface Warfare Center, Port Hueneme Division 4363 Missile Way Port Hueneme, CA 93043-4307			10. SPONSORING/MONITORING AGENCY REPORT NUMBER	
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the authors and do not reflect the official policy or position of the Department of Defense or the U.S. Government.				
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.			12b. DISTRIBUTION CODE	
13. ABSTRACT (maximum 200 words) This thesis was initiated as part of a continuing effort to design and implement a diagnostic expert system for the MK92 MOD 2 Fire Control System (FCS) undertaken by the Naval Postgraduate School (NPS) faculty and graduate students. The focus of this thesis is the development of a structured methodology for the design and implementation of an expert system in an expert system shell utilizing a visual programming language. Guidelines for the application of the structured methodology in the Adept expert system shell were developed and these guidelines applied to the initial Performance module prototype of the MK92 MOD 2 FCS Maintenance Advisor Expert System.				
14. SUBJECT TERMS Expert System, Prototype, Structured Methodology, System Design, Knowledge Implementation, MK92 Fire Control System, Software Development, DSOT, Performance			15. NUMBER OF PAGES 156	
			16. PRICE CODE	
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL	

NSN 7540-01-280-5500

Standard Form 298 (Rev. 2-89)

Prescribed by ANSI Std. Z39-18

Approved for public release; distribution is unlimited.

DEVELOPMENT OF A STRUCTURED DESIGN AND PROGRAMMING
METHODOLOGY FOR EXPERT SYSTEM SHELLS UTILIZING A VISUAL
PROGRAMMING LANGUAGE; APPLICATION OF STRUCTURED
METHODOLOGY TO THE MK92 MAINTENANCE ADVISOR EXPERT
SYSTEM, PERFORMANCE MODULE PROTOTYPE

by

Lucy Michelle Smith

Lieutenant, United States Navy Reserve

B.S., University of Rhode Island, 1984

Submitted in partial fulfillment
of the requirements for the degree of

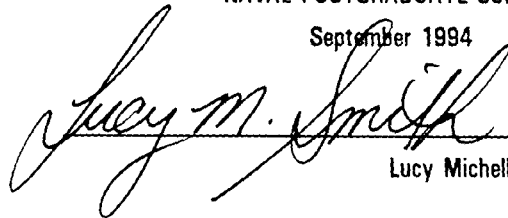
MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL

September 1994

Author:

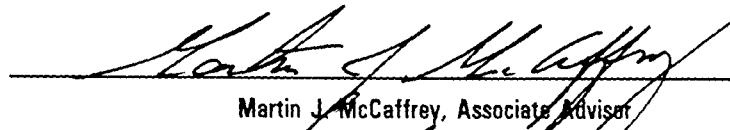


Lucy Michelle Smith

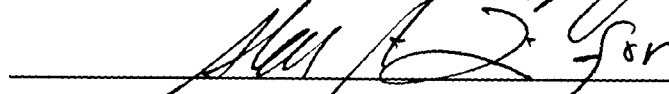
Approved by:



Magdi Kamel, Principal Advisor



Martin J. McCaffrey, Associate Advisor



David R. Whipple, Chairman

Department of Systems Management

ABSTRACT

This thesis is a part of a continuing effort to design and implement a diagnostic expert system for the MK92 MOD 2 Fire Control System (FCS) undertaken by the Naval Postgraduate School (NPS) faculty and graduate students. The focus of this thesis is the development of a structured methodology for the design and implementation of an expert system in an expert system shell utilizing a visual programming language. Guidelines for the application of the structured methodology in the Adept expert system shell were developed and applied to the initial Performance module of the MK92 MOD 2 FCS Maintenance Advisor Expert System prototype developed in an earlier effort.

Accession For	
NTIS CRA&I	<input checked="checked" type="checkbox"/>
DTIC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification _____	
By _____	
Distribution / _____	
Availability Codes	
Dist	Avail and/or Special
A-1	

TABLE OF CONTENTS

I.	INTRODUCTION.....	1
A.	BACKGROUND.....	1
B.	OBJECTIVES.....	2
C.	THE RESEARCH QUESTIONS.....	2
D.	SCOPE AND ASSUMPTIONS.....	2
E.	METHODOLOGY.....	3
F.	THESIS ORGANIZATION.....	3
II.	MK92 MAINTENANCE ADVISOR EXPERT SYSTEM BACKGROUND.....	5
A.	THE MK92 FIRE CONTROL SYSTEM	5
B.	THE MK92 MAINTENANCE ADVISOR EXPERT SYSTEM HISTORY.....	7
C.	THE MK92 DAILY SYSTEM OPERABILITY TEST	10
D.	PROTOTYPE DEVELOPMENT OF THE PERFORMANCE MODULE.....	10
	1. Knowledge Acquisition/Representation.....	10
	2. Hardware/Software Selection.....	13
	3. Knowledge Implementation.....	13
III.	STRUCTURED METHODOLOGIES OVERVIEW.....	16
A.	STRUCTURED METHODOLOGIES.....	16
B.	STRUCTURED DESIGN VS STRUCTURED PROGRAMMING.....	18
	1. Structured Programming.....	19
	2. Structured Design.....	19
C.	STRUCTURE CHARTS.....	21
D.	MODULES.....	21

1.	Black Boxes.....	22
2.	Factoring.....	24
E.	INTRA-MODULAR DESIGN CHARACTERISTICS.....	25
1.	Cohesion.....	25
a.	Functional Cohesion.....	26
b.	Sequential Cohesion.....	26
c.	Communicational Cohesion.....	26
d.	Procedural Cohesion.....	26
e.	Temporal Cohesion.....	27
f.	Logical Cohesion.....	27
g.	Coincidental Cohesion.....	28
2.	Module Size.....	28
F.	INTER-MODULAR DESIGN CHARACTERISTICS.....	29
1.	Coupling.....	29
a.	Data Coupling.....	30
b.	Stamp Coupling.....	30
c.	Control Coupling.....	30
d.	Common Coupling.....	30
e.	Content Coupling.....	30
G.	OTHER GOOD DESIGN/PROGRAMMING CHARACTERISTICS.....	31
1.	Restrictivity/Generality.....	31
2.	Fan-out.....	31
3.	Fan-in.....	32
IV.	ADEPT CONSTRUCTS.....	34
A.	PROCEDURE-BASED PROGRAMMING.....	34

B.	APPLICATIONS.....	37
C.	PROCEDURES.....	37
D.	NODES/ARCS.....	37
1.	Node Styles.....	39
a.	Calculation Node.....	39
b.	Display Node.....	39
c.	Result Node.....	41
d.	Goal Node.....	42
e.	Custom Node.....	43
2.	Node Types.....	43
a.	Start Node.....	43
b.	Work Node.....	45
c.	Case Node.....	45
d.	End Node.....	45
e.	Compound Node.....	46
E.	STATEMENTS.....	48
1.	General Statements.....	48
2.	Literal Statements.....	49
3.	Control Statements.....	50
4.	Compound Statements.....	50
V.	ADEPT STRUCTURED METHODOLOGY.....	51
A.	STRUCTURED METHODOLOGY GUIDELINES FOR ADEPT.....	51
1.	Structure Charts.....	51
2.	Modules.....	52
3.	Intra-modular Design Characteristics.....	55
a.	Module Size.....	55

b.	Cohesion.....	56
c.	Trade-off Between Cohesion and Module Size.....	57
4.	Inter-modular Design Characteristics.....	58
a.	Result and Goal Nodes.....	58
b.	Global Variables.....	58
5.	Other Good Design/Programming Characteristics.....	60
a.	Restrictivity/Generality.....	60
b.	Fan-in and Fan-out.....	61
B.	APPLICATION OF STRUCTURED TECHNIQUES FOR VISUAL DESIGN AND PROGRAMMING TO THE PERFORMANCE PROTOTYPE.....	61
1.	Prototyping.....	62
a.	Advantages.....	62
b.	Disadvantages.....	63
2.	Initial Performance Module Prototype.....	63
3.	Reconstruction of the Performance Prototype.....	65
a.	Structure Charts.....	66
b.	Modules.....	67
c.	Intra-modular Design Characteristics....	72
d.	Inter-modular Design Characteristics....	75
e.	Other Design and Programming Characteristics.....	76
VI.	LESSONS LEARNED AND CONCLUSIONS.....	77
A.	DESIGN AND IMPLEMENTATION ISSUES.....	78
1.	Lack of Adherence to System Development Life Cycle.....	78

2.	Lack of Thorough Application of Structured Design and Programming Methodologies.....	79
3.	Lack of Application of Standard Programming Conventions.....	79
B.	DOCUMENTATION ISSUES.....	80
1.	Initial Documentation.....	80
2.	Prototype Modifications.....	82
3.	Knowledge Documentation.....	83
C.	INSIGHTS.....	85
APPENDIX A.	88
LIST OF REFERENCES.	144
BIBLIOGRAPHY.	146
INITIAL DISTRIBUTION LIST.	147

I. INTRODUCTION

A. BACKGROUND

This thesis is a part of the continuing effort to design and implement a diagnostic expert system for the MK92 MOD 2 Fire Control System (FCS) undertaken by the Naval Postgraduate School (NPS) faculty and graduate students. The MK92 MOD 2 FCS is a complex weapons system based on 1970's technology. Due to the age and complexity of this system, the job of maintaining and repairing the system has been an overwhelming task for the enlisted fire control technician. Often, they are unable to correctly diagnose system failures which results in increased system downtime and therefore a reduction of the ship's operational readiness and war fighting capability. In order to remedy this situation, costly outside technical assistance is frequently required to repair the system.

As manpower levels decrease in the Navy, due to current downsizing trends, the number of senior, experienced technicians decreases as well. This situation complicates matters further and increases the requirement for outside technical assistance. In an effort to reduce costs, system downtime and improve the shipboard technician's ability to better determine, diagnose and resolve problems occurring in the system, the Naval Surface Warfare Center (NSWC), Port

Heuneme Division (PHD) enlisted the help of the Naval Postgraduate School to develop a prototype expert system for the MK92 MOD 2 FCS.

B. OBJECTIVES

The primary objective of this thesis is to develop a structured design and programming methodology for expert system shells utilizing a visual programming language. The secondary goal is to improve upon the initial Performance module prototype, developed in an earlier effort, through the application of this new structured design and programming methodology.

C. RESEARCH QUESTIONS

This research endeavors to answer the following questions:

- Can a structured methodology be developed for expert system shells utilizing a visual programming language?
- Can guidelines be developed for the Adept expert system shell utilizing the new structured methodology?
- How can the application of such a structured methodology improve the Performance module prototype?

D. SCOPE AND ASSUMPTIONS

This thesis focuses on the development of a structured methodology to be applied to system development in expert system shells utilizing a visual programming language. It is assumed that the reader has a basic knowledge and understanding of the predominate structured design and

programming methodologies, the Systems Development Life Cycle (SDLC), expert systems and traditional text-based programming languages. The scope of the application of the developed methodology is limited to the Adept expert system shell, but provides useful insight for its application to fourth generation languages (4GL) and other visual programming environments.

E. METHODOLOGY

The methodology used in this thesis consisted of three distinct stages. In the first stage, a thorough review of structured design and programming methodologies was undertaken. In the second stage, guidelines for the application of the structured methodology in the Adept expert system shell were developed. Finally, the third stage applied those guidelines to the initial Performance module prototype.

F. THESIS ORGANIZATION

This thesis consists of six chapters and one appendix.

Chapter II provides background material on the MK92 MOD 2 Fire Control System (FCS), the history of the MK92 MAES, the elements of the MK92 Daily System Operability Test (DSOT), and the development process of the initial Performance module prototype.

Chapter III reviews the principles of structured design and programming in traditional Computer Based Information Systems (CBIS).

Chapter IV provides an overview of the basic constructs of the Softsell Adept visual programming expert system shell.

Chapter V provides guidelines for the application of traditional structured methodologies, reviewed in chapter three, to the Adept expert system shell. In addition, these guidelines are applied to the initial Performance module prototype.

Chapter VI discusses the lessons learned and insights gained during the development of this thesis.

Appendix A contains a structure chart of the Performance module prototype upon application of the new structured methodology.

II. MK92 MAINTENANCE ADVISOR EXPERT SYSTEM BACKGROUND

This chapter discusses the MK92 Fire Control System (FCS), the history of the MK92 Maintenance Advisor Expert System (MAES), the elements of the MK92 Daily System Operability Test (DSOT), and the development process of the initial Performance module prototype of the MK92 MAES.

A. MK92 MOD 2 FIRE CONTROL SYSTEM

The MK92 MOD 2 FCS can be found on the United States Navy Oliver Hazard Perry class Guided Missile Frigates (FFG 7), as well as Patrol Hydrofoil Missile class (PHM) ships, U.S. Coast Guard High and Medium Endurance Cutters, and the Australian Anzac and FFG 7 class ships.

The MK92 MOD 2 FCS, illustrated in Figure 2.1, is a high performance, multi-purpose complex weapons system. It is part of an integrated, modular system that contains search/track radar, a digital computer, servo system, hydraulic system, amplidynes and other sophisticated electronic components. The function of the system is to use radars, guns and missiles in order to:

1. Locate and track air, surface and shore targets.
2. Calculate the targets future positions relative to the ship's current position.
3. Use the above information to train and launce the gun and missiles that will destroy the designated targets when required.

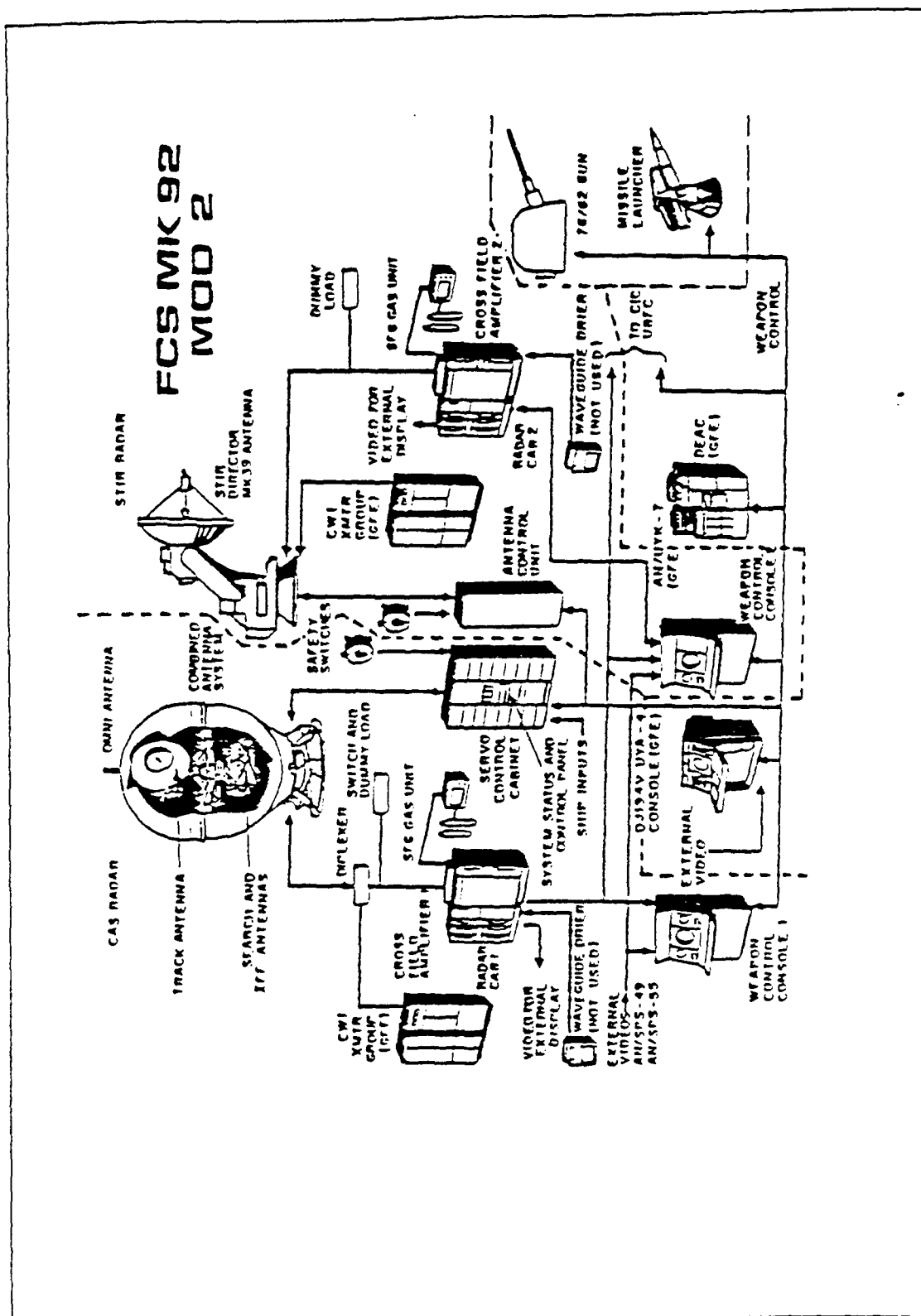


FIGURE 2-1 MK 92 MOD 2 SYSTEM DIAGRAM

A more thorough description of the system can be found in Smith (1993) and Lewis (1993).

B. MK92 MOD 2 MAINTENANCE ADVISOR EXPERT SYSTEM HISTORY

Because of the complexity of the MK92 MOD 2, the job of maintaining and repairing the system has been an overwhelming task for the enlisted fire control (FC) technician, primarily due to a lack of knowledge, experience, and limited technical manuals. Although FC technicians receive intensive training from Naval Training Centers (NTC), textbooks and classroom lectures can not equip technicians properly with the required knowledge to isolate many casualties in such a complex weapons system.

The lack of experience and technical knowledge of the system led to increased equipment downtime and therefore, a reduction of the ship's operational readiness and war fighting capability. In order to remedy this situation, costly outside technical assistance is frequently required to repair the system. This assistance comes from the Naval Sea Systems Command (NAVSEA) where the expert knowledge resides in the experienced civilian technical engineering representative.

In an effort to reduce costs, system downtime and improve the shipboard technician's ability to better determine, diagnose, and resolve problems occurring in the system, the Port Hueneme Division (PHD) of the Naval Surface

Warfare Center (NSWC) initiated, in 1992, the development of a prototype diagnostic expert system. The goal of the development effort was to collect the combined knowledge of the preeminent MK92 MOD 2 expert engineers and represent it in the form of an expert system software program. This program could then be deployed aboard ships for use by FC technicians. Unfortunately, PHD, NSWC personnel ran into difficulties developing the software during the initial program development. It was at this point that PHD requested the assistance of the Naval Postgraduate School (NPS) in the software development of the prototype.

The domain for the MK92 FCS MAES is the MK92 Daily Systems Operability Tests (DSOT), described in detail in the following section. The areas of the DSOT knowledge base that were originally chosen for inclusion in the initial prototype effort were as follows:

- CAS/STIR DSOT Performance Tests
- DSOT Calibration Tests
- CAS/STIR Transmitter RF Power Checks

However, at this time, only expert knowledge for the Calibration and Performance areas of the DSOT have been acquired and implemented into the MK92 FCS MAES. Knowledge to develop the Transmitter RF Power Checks prototype is currently being acquired for later implementation into the prototype in fiscal year 1995.

Although the primary focus of this thesis is the Performance module of the MAES, a brief chronology of events of the project, since NPS involvement, is required in order to provide the reader with an understanding of the proverbial "big picture" and, of course, for completeness. The events are as follows:

1. In 1993, a cost/benefit analysis was conducted by Steven H. Powell (1993) that determined that the development and implementation of a MAES for the MK92 MOD 2 would result in considerable savings to the Navy.
2. In 1993, Claude D. Smith (1993) and Clinton D. Lewis (1993) developed the initial prototype for the Performance module of the MAES.
3. In September 1993, the first prototype of the Performance Module was given to PHD for initial testing and evaluation.
4. In January 1994, the author took over the responsibility of the refinement of and development/application of a structured design methodology to the Performance Module.
5. During 1993/1994, David M. Geick and Steven E. Mikler (1994) developed the initial prototype for the Calibration module of the MAES.
6. During 1993/1994, Susan Tally (1994) had begun the process of developing an integrated parts database system for the MAES. Continuing development of the database is currently being pursued by Janie N. Crawford (1994).
7. In late 1993, John L. McGaha (1994) was assigned the responsibility of maintenance and configuration management of the Calibration module as well as developing implementation procedures for the system.
8. In March 1994, verification, validation and testing of the Calibration module was being conducted by Kent R. Dills and Timothy F. Tutt (1994).

C. THE MK92 DAILY SYSTEM OPERABILITY TEST

The DSOT is a part of the daily maintenance procedures and a sub-system of the MK92 MOD 2 FCS. It is a computerized system that provides a quick, yet comprehensive, assessment of the operational readiness of the system. The DSOT includes injecting simulated targets into the radars in order to determine if all systems are functional, and checking the validity of system responses to preprogrammed input target parameters. Upon completion of the system check, a printout is provided that lists NoGo and out-of-tolerance conditions, if any, for twenty different areas in the system. The two specific sections that the DSOT covers are Performance and Calibration of the MK92 FCS. For the purpose of this thesis, the focus will be on the Performance aspect of the system. For a more technical and detailed description of the DSOT refer to the thesis' by Smith (1993) and Lewis (1993).

D. PROTOTYPE DEVELOPMENT OF THE PERFORMANCE MODULE

Although the development of the Performance module prototype is covered in detail by Smith (1993) and Lewis (1993), a brief overview is provided here for easy reference.

1. Knowledge Acquisition/Representation

Since the MAES project was initiated at PHD, NSWC, it was not necessary for the NPS project team to select the

domain experts. The primary expert assigned to the project was, and still is, Mr. Dorin Sauerbier of Paramax. Due to his extensive experience with the MK92 MOD 2 FCS, he has proven to be an invaluable source of information. The knowledge provided by Mr. Sauerbier is a combination of heuristics from his own experience and that of other experts as well as information provided by the MK92 MOD 2 FCS technical manuals.

The representation of the knowledge, acquired by Mr. Sauerbier, was in the form of handwritten diagnostic trees. As troubleshooting expertise is usually hierarchical in nature, diagnostic trees were the most logical and suitable representation. Figure 2.2 is an actual sample of a diagnostic tree provided by the domain expert. This particular tree represents a sub-division within the FC2 Acquisition module of the system. Note from the figure that the methodology, used by the domain experts, in diagnosing a system failure is done in a series of yes/no steps. There are some areas in the system where there are nodes that have more than two possible decision choices; however, the nodes that have only yes/no choices generally prevail throughout the system.

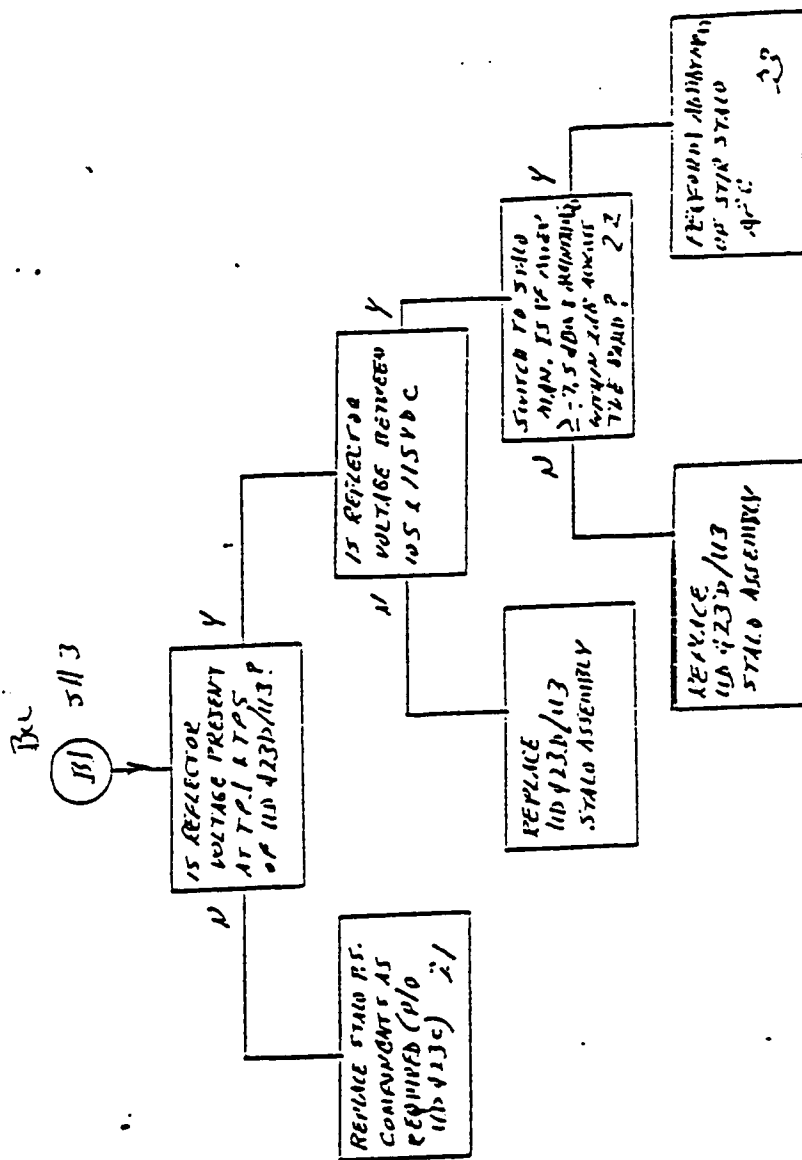


Figure 2.2: Knowledge Document

2. Hardware/Software Selection

Softsell Adept was chosen for the MAES. The choice of software packages was based upon the following criteria:

- It is a procedural based software that was specifically designed for diagnostic expert system development. This matched the MAES knowledge representation.
- It provides a visual programming environment that produces a system that is easy to develop, maintain, and evolve.
- It is a Windows based program that integrates a graphic user interface with an expert system shell and, therefore, eliminates the need for two separate software packages.
- It was used successfully by the Army in the implementation of an M1 tank diagnostic system.

Given the chosen software and the estimated size of the system, a 486 PC was selected as the developmental hardware platform. A final decision has not been made as to what hardware will be used for deployment; however, the choices are confined to either a 486 desktop PC or a 486 laptop/notebook computer.

3. Knowledge Implementation

Knowledge implementation, as defined by Prerau (1990, p. 17), "...is the process of taking the knowledge found during knowledge acquisition and translating it into an operational expert system program by employing the

structures and paradigms that comprise the knowledge representation".

Adept is a procedure-based expert system shell which utilizes a visual programming language that builds applications (programs) as a collection of procedures (sub-programs or modules). During initial prototype development, it became apparent that the knowledge document could be broken down into a series of modules which represented the procedures that a domain expert would follow in the diagnosis of system faults within the major areas and their associated sub-areas of the DSOT. Each module was then implemented as a procedure within Adept. Figure 2.3 is the Adept implementation of the FC2 Acquisition diagnostic tree presented in Figure 2.2. A comparison of Figure 2.2 and Figure 2.3, reveals that the Adept representation is almost identical to the knowledge document from which it was constructed. It becomes apparent why the Adept software was selected as the implementation software.

Appendix A of Smith's (1993) and Lewis' (1993) thesis contain the descriptions and diagrams of the procedures in the application of the initial Performance module prototype.

Chapter III reviews the principals of structured design and programming in traditional Computer Based Information Systems (CBIS).

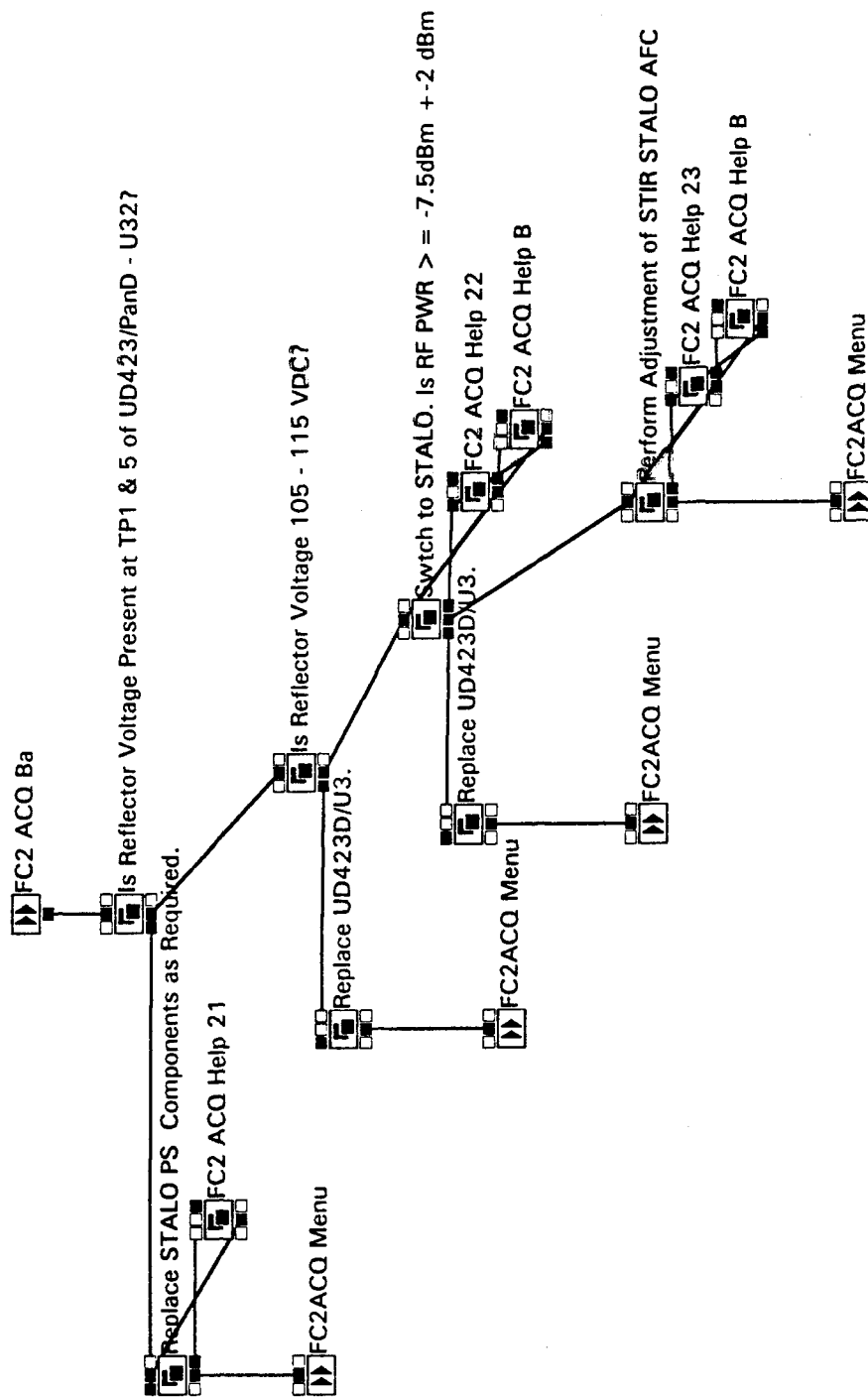


Figure 2.3: FC2 ACQ Ba Procedure

III. STRUCTURED METHODOLOGIES OVERVIEW

This chapter discusses the characteristics of structured design and programming of traditional text-based programming languages in view of applying them to the development of systems utilizing an expert system shell that employs a visual programming language. Specifically, it addresses structured design and programming, structure charts, modularization, inter-modular and intra-modular design characteristics, and other miscellaneous design characteristics.

A. STRUCTURED METHODOLOGIES

Structured is defined, by the Mirriam-Webster Dictionary, as either "something that is made up of interdependent parts in a definite pattern of organization or as an arrangement or relationship of elements in a substance, body or system". Methodology is defined by Aktas (1987, p.33) as "A body of methods, rules, and postulates employed by a discipline". According to both Aktas (1987) and Page-Jones (1988), by applying a specific set of methods, rules and postulates to a problem, one will be able to design a structured, well-defined, maintainable, flexible, and standardized computer based information system (CBIS).

Structured methodologies are used to implement and/or supplement the systems development life cycle (SDLC).

According to Whitten et al (1989, pp. 108-129), the predominant structured methodologies used today are:

- **Structured Analysis** is a technique for system specification that models the flow of data in a new or existing system using a series of flow models. This method compliments/supports the survey, study, definition and selection phases of the SDLC.
- **Structured Design** is the process of factoring a system into a top-down hierarchy of independent modules. The tool used here is the structure chart. This methodology supports the program design portion of the design phase in the SDLC.
- **Structured Programming** is a set of rules by which a programmer designs the logic and code within modules. This supports the construction, delivery, and maintenance/improvement phases of the SDLC.

To date, most of the various structured methodologies that have been developed have focused predominately on the design of the more common types of text-based CBIS.

Primarily, these include Transaction Processing Systems (TPS), Management Information Systems (MIS), and Decision Support Systems (DSS). Currently, there are little to no published approaches pertaining to a structured methodology for use in the development of expert systems, or for that matter any application system, using a visual programming language.

In order to develop such a methodology, we must first understand the characteristics of the methodologies that have been established for the text-based CBIS and then apply those characteristics, where feasible, to expert systems development tools employing visual programming. The purpose of this chapter is to provide the reader with an overview of current CBIS structured methodology characteristics as they apply specifically to design and programming. However, only those characteristics of CBIS structured methodologies that can be applied to visual programming will be emphasized, since a detailed discussion of all the characteristics of structured methodologies would detract more than it would contribute to the purpose of this thesis.

B. STRUCTURED DESIGN VS STRUCTURED PROGRAMMING

Previously, when structured design and structured programming methodologies were developed (and were typically applied to second and third generation languages) there was a rather clear delineation of the roles and responsibilities of each methodology. However, since the advent of higher level languages, such as expert system shells using visual programming, the line of demarcation between the two methodologies has blurred. This is especially true in Adept. Primarily, this is because expert systems utilizing visual programming place an additional layer or interface between the programmer and the traditional style of coding.

In visual programming, many of the traditional programming structures are implicit and lie beneath the visual objects of the interface. Without the actual coding, many of the standard characteristics of structured programming either fade away completely or overlap into structured design.

The constructs of Adept's visual programming language will be discussed in greater detail in the next chapter. The following sections provide a clearer differentiation of the roles of structured programming and structured design.

1. Structured Design

The concept of the structured design methodology is to factor computer programs into a top-down hierarchy of independent modules (Whitten et al, 1989, p. 115). These modules consist of a series of instructions that, if developed properly, serve only one solitary function (this property is known as *cohesion*). Equally as important is the ability of a module to perform its function independent of other modules (this property is known as *coupling*). Structured design supports the program design phase of the SDLC as well as simplifying both the construction and delivery phases via its top-down structure (Whitten et al, 1989, p.115).

2. Structured Programming

Structured programming is a "process-oriented technique for designing and writing programs with greater

clarity and consistency" (Whitten et al, 1994, p. 145).

This technique deals primarily with the logic and code of a program and was originally developed for traditional text-based programming languages. Research has shown that a well structured program can be written using only three control structures. These three control structures, referred to as *restricted control structures* (Whitten et al, 1989, p. 113), are:

- a *sequence of instructions or group of instructions*
- a *selection of instructions or group of instructions based on some decision criteria (this construct is often referred to as the if-then-else or case construct)*
- an *iteration of instructions or group of instructions based on some criteria (this construct comes in two basis forms: repeat-until and do-while)*

These constructs can be used alone or together in any series of combinations. However, there are two main characteristics of employing these constructs. First, each structure must exhibit a single-entry, single-exit property (Whitten et al, 1989, p. 113). In other words, there can only be one point of entry into and one point of exit out of each structure. Second, the program code must be able to be read page by page and from top to bottom with no backward references. Some programmers have incorrectly interpreted this last characteristic to mean that the program code, where applicable, should not include the use of GOTO

statements. Structured programming does not seek to eliminate the use of GOTO statements; rather it endeavors to reduce the undisciplined application of backward referencing.

C. STRUCTURE CHARTS

Structure charts, also known as structure diagrams, are structured design tools used by system designers to represent the shape, size and configuration of a system (Aktas, 1987, pp. 78-80; Page-Jones, 1988, pp. 33-35; Whitten et al, 1989, pp. 115-117). Structure charts are also used to assist in breaking or partitioning a system down into a top-down hierarchy of modules and sub-modules. For example, Figure 3.1 is a partial structure chart of the functions of an Automatic Teller Machine (ATM). Each module represents a specific function that a typical ATM performs. Those functions at the top level of the structure chart represent the main functions. The modules located at the second level represent the subfunctions that must be performed in order for the system to accomplish its primary functions on the first level. As we progress down to the lower levels of the structure chart, the functions of the individual modules become more specific and simplified.

D. MODULES

A module is a series of instructions or program statements that are grouped together to perform one solitary

function. Modules have four basic attributes; input and output flows to other modules, function, mechanics, and internal data (Page-Jones, 1988, p. 35). These are defined as follows:

- Inputs and outputs refer to the parameters received from and returned to other invoking modules.
- The function refers to how the module manipulates the input data in order to produce the output data.
- Mechanics refers to the actual coding within the module which is required in order for the module to perform its designated function.
- Internal data refers to the data workspace of a module to which only that module has access.

A module may perform its designated function alone or it may call upon other modules to assist it in the performance of its assigned task. In Figure 3.1, four modules are called upon to perform specific tasks for the *Checking* module. On the other hand, the modules at the lowest level of the chart do not require other modules to perform their functions.

A module is primarily conceived of in the structured design process. However, it is not developed and tested until the structured programming techniques have been applied to it. The following sections present guidelines for good module design.

1. Black Boxes

All modules within the system should be created as if it were a black box. A module is considered a black box

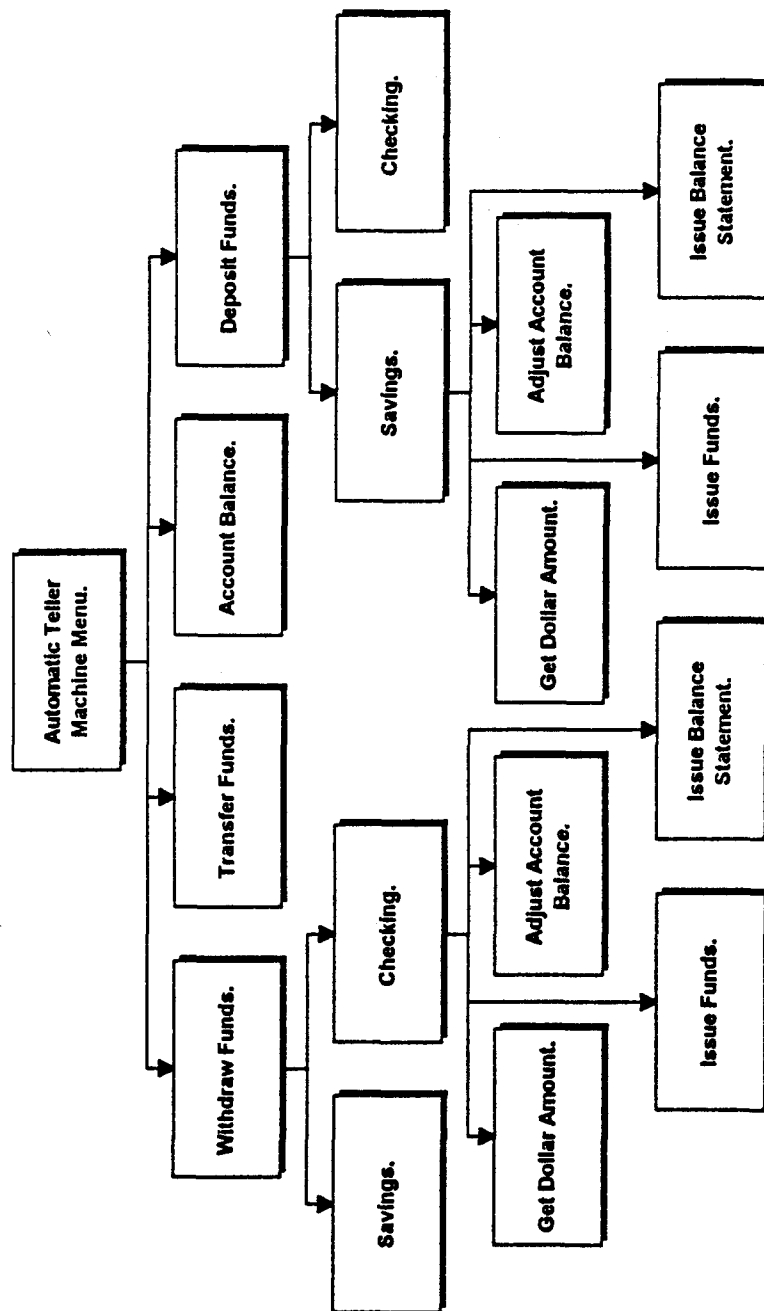


Figure 3.1: ATM Structure Chart

when the function of the module is known and has a well-defined role, but the details as to how the module performs the function is unknown. Black boxes are used in order to reduce a large, complex system into small, useable parts. The use of black boxes serves to make the system easier to design, test, maintain, and understand. They are used in structure charts because the details of the function of each module would only serve to confuse at that point in the design process.

2. Factoring

Factoring is a decomposition method that is used to separate a specific function from one module into a sub-module of its own (Aktas, 1987, p.125). For example, referring to Figure 3.1, the module marked *Savings* (Parent), under the *Deposit Funds* module, has four sub-modules (children) that each perform a specific function. Logically, however, when used in conjunction with one another, they contribute to the performance of the parent module's function. Factoring simplifies the system by promoting reuse and reducing individual module size. Module size will be addressed in Section C of this chapter as one of the intra-modular characteristic. Module reuse is attained by factoring out an activity that is duplicated in several modules. In the ATM example, the function of obtaining a specified dollar amount from the customer was

utilized as a sub-function by more than one of the primary functions of the overall system. Therefore, this function was factored out into a module of its own and called *Get Dollar Amount*. The reason for factoring this module is to reduce redundant code, thereby promoting module reuse and simplifying the system.

E. INTRA-MODULAR DESIGN CHARACTERISTICS

Intra-modular design characteristics refer to characteristics *within* a module. When a systems designer partitions a system into a series of modules, he/she must ensure that the design of the individual modules are functionally cohesive. The designer must be concerned with the module's function, cohesion and size. Intra-modular design characteristics originate as characteristics of the structured design methodology; however, they can not be fully applied and tested until they are used as part of a structured programming methodology.

1. Cohesion

Cohesion refers to the measure of the degree to which a module performs a solitary, well-defined, problem-related and understood function (Aktas, 1987, p. 122). The

following is a brief definition of each type of cohesion from the easiest to the hardest to maintain.

a. *Functional Cohesion*

A module has achieved functional cohesion if it performs one solitary, well-defined and understood function.

b. *Sequential Cohesion*

A module is sequentially cohesive if the instructions/activities within that module must be performed in a specific, sequential pattern in order for the module to perform its function correctly. In other words, the instructions/activities within that module follow a specific order because the output data from one instruction/activity serves as required input data for the next instruction/activity.

c. *Communicational Cohesion*

A module is communicationally cohesive if the activities within the module use the same input or output data.

d. *Procedural Cohesion*

In a procedurally cohesive module, the instructions may be totally unrelated. However, control passes from one activity to another rather than data. Unlike sequential cohesion, the instructions/activities in a procedurally cohesive module are not dependent on each other for completion of their individual functions.

e. Temporal Cohesion

Temporal cohesion is similar to procedural cohesion in that the instructions/activities are unrelated to one another. However, within a temporally cohesive module, the instructions/activities are grouped together because they are all performed at the same time.

f. Logical Cohesion

A module is logically cohesive if the activities of the module are grouped together solely because their functions contribute to a general category of activities and the selection of a specific activity is chosen from outside the module. For example, let's suppose that a module, called *Initialization*, includes the following activities:

- Initialize printer
- Initialize variable_A
- Initialize variable_X
- Initialize device_A

These activities are grouped together, not because they contribute to one solitary function, but rather because each individual activity within the module performs a similar function on different objects. In addition, each time the module is used, not all the functions are performed.

Instead, the choice of which activity to perform is selected by another module and that choice is passed as a parameter

into this module. In this case, the module is logically cohesive.

g. Coincidental Cohesion

A module is considered to be coincidentally cohesive if the instructions/activities within that module are totally unrelated to one another.

A more indepth analysis and description of the various types of cohesion can be found in Page-Jones (1988, pp. 84-96).

2. Module Size

In structured programming, the smaller the module's size the easier it is to implement, maintain and reuse. As stated previously, the optimum cohesion characteristic for a module is functional cohesion. If the module performs one well-defined, solitary function, then it is as small as it can get without loosing cohesion. However, there are trade-offs to achieving functional cohesion. One primary trade-off is programmer comprehension. According to Page-Jones (1988, p. 104) "our ability to understand a module and to find bugs in it depends upon our being able to comprehend the whole module at once". Following that line of thinking, a module that fits on one page and is visible to the programmer in full, is easier to comprehend and debug than a module that is two or more pages. Therefore, a module that achieves functional cohesion and consists of two pages of

code may actually be harder to comprehend and debug than a module that is only sequentially cohesive and consists of only one page of code.

The determination as to which takes precedence, module cohesion or programmer comprehension, is dependent upon the system itself and system designer preference.

F. INTER-MODULAR DESIGN CHARACTERISTICS

Inter-modular design refers to those characteristics that are exhibited *between* modules. Similar to the intra-modular design, these characteristics also initially emerge and are applied as part of structured design but are not fully implemented and/or tested until they are applied as part of structured programming.

Concerns of a system designer when partitioning the system and designing modules is how the modules interact with each other. Of primary concern to the system designer is coupling.

1. Coupling

Coupling is defined as the degree of interdependence between two or more modules in the system (Page-Jones, 1988, p. 58). The lower the degree of coupling, the better the overall system. In order to reduce coupling between modules, a system designer must create connections that are narrow, direct, local, obvious and flexible (Page-Jones, 1988, p. 60). The following list provides definitions of

the various type of coupling from the best to the worst to maintain.

a. *Data Coupling*

Two modules are data coupled if they communicate through the parameters of the modules and the pieces of data are simple and uncomplicated.

b. *Stamp Coupling*

Stamp coupling is said to exist when the information being passed between two modules represent a group of related data items that are essential to the modules.

c. *Control Coupling*

Control coupling occurs when information is passed from one module to another whose sole purpose is to control the internal logic of the receiving module.

d. *Common Coupling*

Common coupling occurs when the data passed between modules is contained in an area that is accessible to other modules. Another name for common coupling is global coupling (Page-Jones, 1988, p.73)

e. *Content Coupling*

Content coupling occurs if one module accesses or refers to the inside of another module. In this case, one module is required to know the inner workings of another module. This defeats the concept of the black box.

A more indepth and detailed analysis of coupling and its characteristics can be found in Page-Jones (1988, pp. 58-79)

G. OTHER GOOD DESIGN/PROGRAMMING CHARACTERISTICS

Although the characteristics, stated previously, play the primary role in the overall quality of systems design, they can not be considered alone. Other characteristics, listed below, play an important role in systems design as well. The following characteristics are predominately structured design techniques. However, the first characteristic, restrictivity/generality, is also applied during structured programming.

1. Restrictivity/Generality

In good design practices, a system designer does not want to make individual modules too restrictive or too general (Page-Jones, 1988, pp. 133-134). A module is too restrictive if it performs a function that is so specific and narrow that it does not promote reusability. A module is too general if it performs a function that is so broad that much of the code is rarely used. That type of coding serves only to complicate and clutter the system.

2. Fan-out

Fan-out refers to the maximum number of children modules that any given parent module can have and still maintain system clarity. The recommended upper limit of fan-out modules has typically been approximately seven.

Page-Jones (1988, pp. 139-140) states that the reason for the number seven is based in human psychology. The optimum number of tasks that a given individual can handle, with little to no errors, is seven. The number of errors begins to increase exponentially as the number of tasks increases beyond seven. This theory has since been transposed to the design of CBISs. Although not a hard and fast rule, if a parent module has more than seven children, the system designer should re-evaluate the modules and determine if an intermediate level should be added.

3. Fan-in

Fan-in refers to the number of parent modules that any given child module can have and still maintain system clarity. In so far as module cohesion and interfaces are not sacrificed, the higher the fan-in, the better. Generally, the greater the amount of fan-in, the more reuse is promoted, which in turn promotes increased programmer comprehension and system simplicity.

The traditional structured methodologies presented in this chapter will be adapted to expert systems utilizing visual programming languages and applied to the Performance module of the MK92 MOD 2 FCS MAES in Chapter V. However, in order for the reader to comprehend and evaluate the application of the structured methodology, an overview of

the fundamental constructs of the Adept expert system shell will be presented in the next chapter.

IV. ADEPT CONSTRUCTS

This chapter provides an overview of the basic constructs of Softsell Adept visual programming as provided in the Adept Reference Manual (Himes et al, 1991). It introduces such constructs as applications, procedures, nodes, statements, variables, constants, operators, functions and objects. In addition, where appropriate, the Adept visual programming constructs will be briefly compared to traditional programming languages and expert system constructs. For a detailed description of the nuances of the Adept constructs or the techniques used in applying them, the reader is referred to the Adept Reference Manual.

A. PROCEDURE-BASED PROGRAMMING

Adept utilizes a creative approach to expert system development by combining visual application programming with expert systems. It is a procedure-based expert system shell that permits the programmer to build expert systems by creating and manipulating objects on a screen.

Prior to the advent of visual programming, most expert system shells utilized a rule-based style of programming. Rule-based programming is generally represented as a series of *IF/THEN* or *Condition/Action* statements which include logical operators such as *AND/OR*. For example, consider the following two simple rules:

```
IF    lightswitch is on
AND   lightbulb is on
THEN  lightbulb is good
```

```
IF    lightswitch is on
AND   lightbulb is off
THEN  lightbulb is defective
```

The first rule concludes that the lightbulb is good if the lightswitch is in the on position and the lightbulb is lit. The second rule concludes that the lightbulb is defective if the lightswitch is in the on position and the lightbulb is not lit. Rule-based programming is easy to understand because it imitates the way in which humans tend to view the relationship between cause and effect (Himes et al, 1991, p. 8). However, expert systems can also be tedious to develop if they contain a large number of rules. As is typical of text-based programming, they can be difficult to debug, maintain and evolve.

Procedure-based programming is a new paradigm, developed at Stanford University, that uses graphical representations of procedures to create expert systems (Himes et al, 1991, p. 9). It was specifically designed in order to "make it easy to model business and technical procedures and then turn those procedures into interactive software applications" on personal computers (Himes et al, 1991, p. 6). Figure 4.1 is an example of how the rule-based example, provided above, would be represented in a procedure-based programming language. This type of programming alleviates

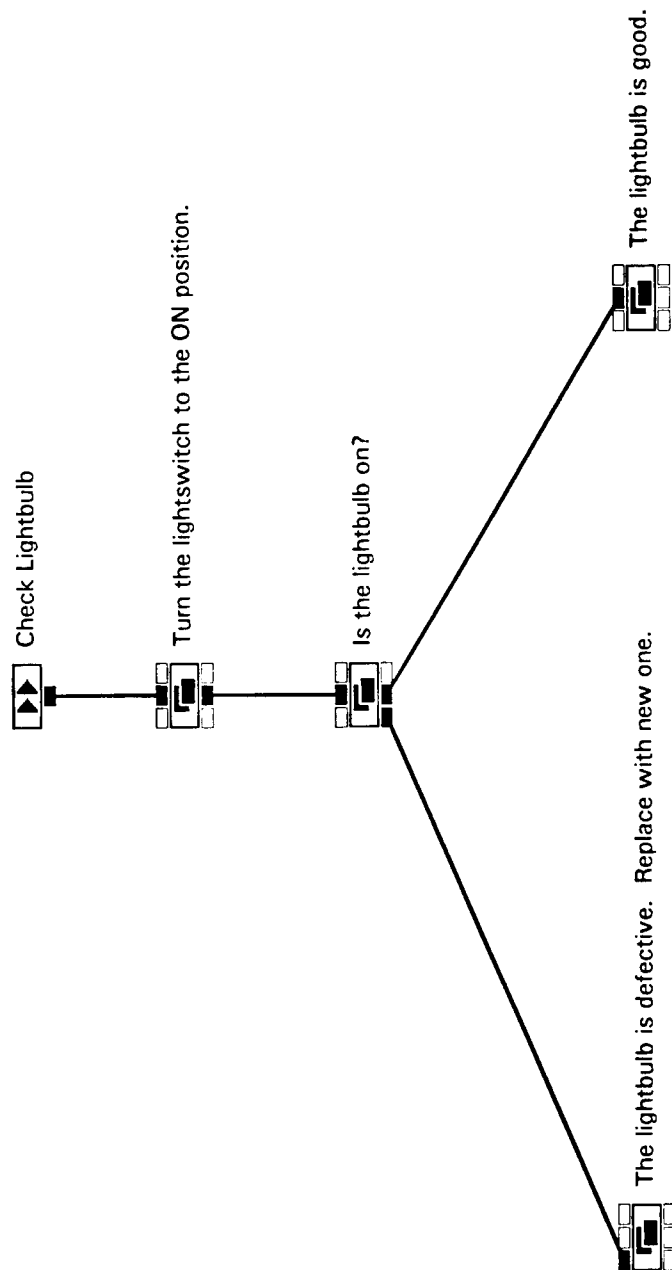


Figure 4.1: Lightbulb Procedure

the developer of an expert system from having to be or become a professional text-based programmer.

The following sections present the main constructs of visual procedural-based programming as implemented in Adept.

B. APPLICATIONS

An application usually consists of several procedures, where each procedure is made up of nodes, displays, variables, and functions. If Adept were to be compared to the ADA programming language, applications would equate to the overall program or system. The current version of the MK92 MOD 2 FCS MAES represents one ADEPT application consisting of approximately 250 procedures.

C. PROCEDURES

Similar to the third generation programming languages, such as PASCAL or ADA, where a system consists of a series of modules that call each other when required, an Adept application consists of a series of programs called *procedures*. A procedure consists of a series of visual objects called nodes. Analogous to the parent/child modular relationships in structured programming, procedures can also be classified as either a parent or a child procedure or both.

D. NODES/ARCS

A node is a graphical object which represents a specific step or series of steps within any given procedure. It is a

visual object that allows end users to design and implement an expert system without having to deal with traditional text-based programming code. Actual programming code lies underneath each node and is transparent to the programmer/user. Since Adept is naturally a hierarchical programming language, nodes can be either a parent node, a child node or both.

The flow of control is determined by evaluating each node in an Adept procedure. This evaluation returns one of three possible values: true, false or unknown. The value returned depends on the style and type of each node.

Generally, a value is returned as a result of:

- User input
- Statement evaluation
- Evaluation of one or more child procedures

Nodes are connected to other nodes via Arcs (refer to Figure 4.1). The purpose of an arc is to establish a logical link between nodes and indicate the sequence of program flows. There are three types of Arcs in Adept that correspond to the three values that a node can evaluate to: true, false and unknown. As nodes are evaluated, control is passed from a parent node to a child node via an arc. If the node is evaluated as true then control passes to the child node that is connected to the parent node by a true arc. Similarly, if a node is evaluated as false or unknown

then control is passed to the node connected by a false or unknown arc, respectively.

In Adept, the task performed by any given node is dependent upon the style and type of node that is used by the programmer.

1. Node Styles

The style of a node determines the function that a node serves in a procedure. There are four basic styles in Adept and a custom style that is defined by the programmer. Each style can be identified by the symbol within the node. Figure 4.2 illustrates each of the basic node styles. The following is a brief description of each of the node styles.

a. Calculation Node

The functions of a calculation style node is to perform mathematical calculations, compare variables or to call a predefined function in Adept. This style of node is evaluated by executing the statement(s) attached to it and then returning a true, false or unknown value.

b. Display Node

Whenever it is necessary to convey information to or obtain information from a user, a display node must be used. This node has a display screen attached to it that can be customized to the program requirements. The evaluation of this style of node is based on the user's input. When the node is executed, the associated display is

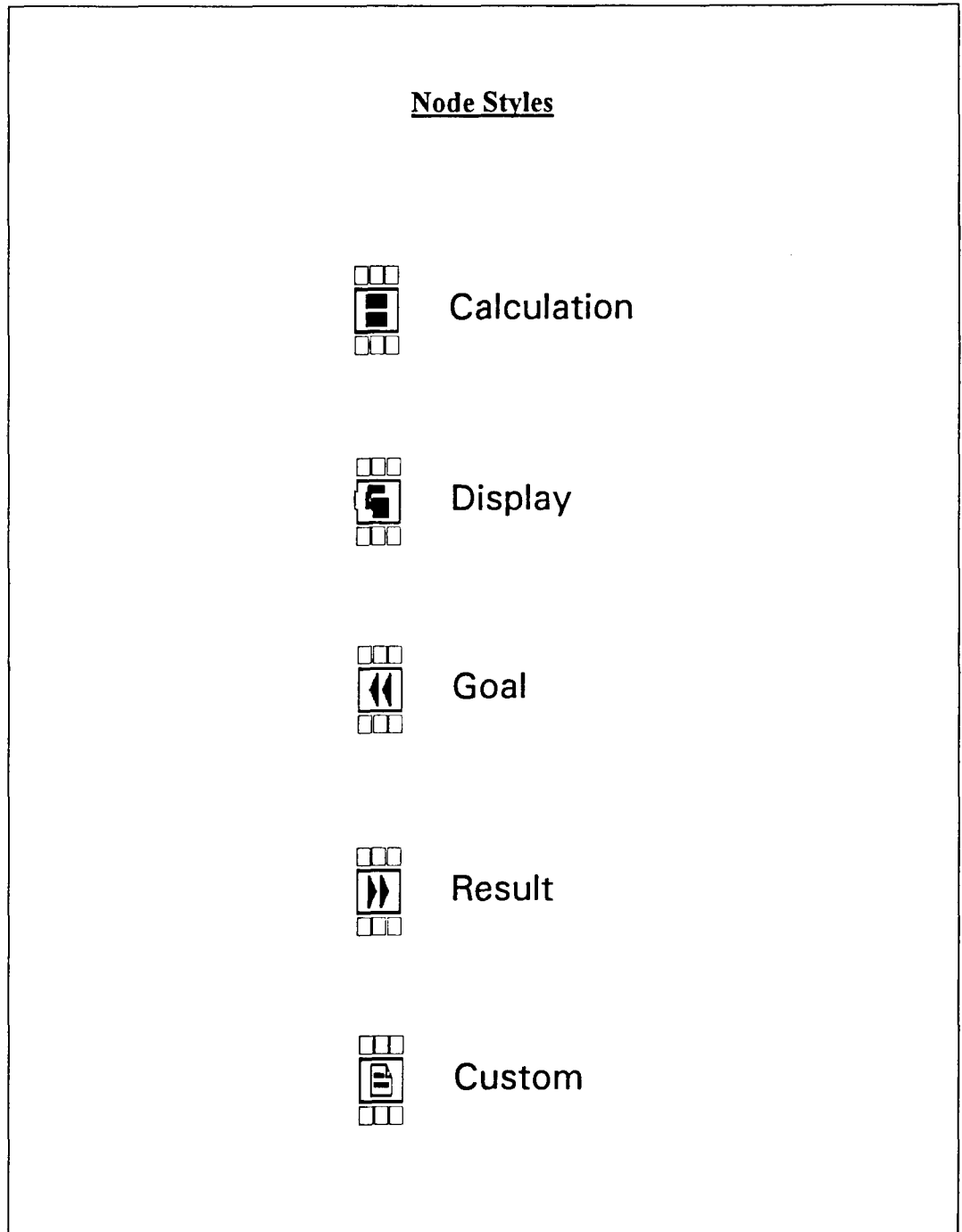


Figure 4.2: Node Styles Diagram

presented to the user and he/she is asked to provide input to the program. Depending on the response, the node evaluates to true, false or unknown, and an appropriate arc path is followed.

c. Result Node

Adept utilizes a result node, which is a one-way link between two or more procedures, to perform the forward chaining aspect of the expert system shell. Forward chaining uses an available set of known facts in order to determine which procedure(s) will be triggered. The procedures that are triggered will in turn derive more facts about the situation and add those to the working memory of the program. This process continues until all facts that can be derived from the initial set of data have been found.

In Adept, when a procedure, or a path within a procedure, has reached a conclusion, a result-style end node is used to transfer control to one or more children procedures who have the matching result-style start nodes. The child procedure(s) is executed only after Adept has evaluated the parent procedure.

It is very easy for a traditional text-based programmer, using Adept for the first time, to confuse the function of a result node with that of the GOTO statement used in traditional programming languages. As such, he/she may attempt to apply traditional structured programming

methodologies for GOTO statements to result nodes. If a programmer were to limit the use of result nodes in the same manner in which GOTO statements are restricted in structured programming methodologies, it would limit the functionality of the forward chaining properties of the Adept expert system shell. It could also negatively impact the functionality and logic of the application.

d. Goal Node

A goal node is used when it is necessary to call upon one or more child procedures to solve a problem and return a value of true, false or unknown to the parent procedure.

In expert system terminology, a goal node is how Adept performs backward chaining. Backward chaining starts with a premise and then looks for a procedure to prove it. The search for the premise continues until all procedures that might prove or disprove the premise have been searched.

Similar to Call statements in traditional text-based programming languages, goal nodes allow Adept to create a two-way link between two or more procedures. When a goal node is reached, processing of the parent procedure ceases and a system call goes out to one or more child procedures that will solve a problem and return a value to the parent procedure. The calling goal node will then be evaluated, using the value returned from the child

procedure. Processing of the parent procedure will continue.

e. Custom Nodes

Occasionally, there is a requirement that can not be supported by the basic node styles. In this event, Adept allows the programmer to design his/her own customized node. This is accomplished by attaching a script to the custom node that specifies how a node is to be evaluated. Script is implemented by using the underlying programming language of Adept. Parameters are attached to the node such that, when executed, they are passed into the node. The script then manipulates and/or evaluates those parameters and returns a true, false or unknown value.

2. Node Types

The type of node used determines the role that the node plays within the procedure. The following is a brief description of the types of nodes within Adept and the role that they serve. Refer to Figure 4.3 for a representation of each node type.

a. Start Node

A start node is used at the beginning of a procedure and has only a true arc handle at the exit point. Adept will execute a procedure if the start node of that procedure evaluates to true.

Node Types



Start



Work



Case



Compound Case



Compound Work



Compound End



Compound Start



End

Figure 4.3: Node Types Diagram

b. Work Node

A work node can be identified as having three arc handles located at the top of the node that indicate the entryway into the node, and three other arc handles located along the bottom of the node that are used as exit points. A work node is used to indicate a decision point within the procedure.

c. Case Node

A case node is similar to a work node in that it has six arc handles and represents a single decision point. However, the true arc exit handle is located to the left of the node. Case nodes can be used alone and, when used in that manner, their role is identical to that of the work node. However, case nodes were originally designed to be used in conjunction with compound nodes in order to represent multiple decision points. A single case node would only be used in the event that a given procedure is complex and the readability of the procedure is of utmost importance.

d. End Node

An end node can be identified as having only three entrance point arc handles. This type of node is used to indicate the end of a procedure. Any given procedure can have one or more end nodes; one at the end of each path in the procedure.

e. Compound Node

A compound node can be any of the types discussed previously and is segmented into a series of three or more nodes, of varying styles, combined together into one large node. Compound nodes can be grouped into two subtypes.

(1) With Logical statements. Compound start, work, or end nodes are used either when a series of steps must be performed in one place or a decision needs to be made based upon the combined results of several decisions (Himes et al, 1991, p. 24). The first node of a compound start, work or end node will have either a programmer defined logical statement attached to it or it will have a default logical statement defined by Adept. The function of the logical statement is to identify how the individual segments, either separately or in combination, of the compound node are to be evaluated in order to determine the result of the overall node. If no programmer defined logical statement is attached to the node, Adept automatically combines the results of each of the individual segments as if they were linked together by logical AND operators.

This Adept construct can be equated to the *if/then* constructs of traditional text-based programming languages which include logical and/or statements. An

example of a compound work node represented in traditional text-based programming would be as follows:

```
IF    the subject is an animal
AND   it has four legs
AND   it's fur has spots
AND   it has a mane
AND   it roars
THEN  it is a lion
```

For a more detailed discussion of the various subtypes and styles of compound nodes, the reader is referred to the Adept Reference Manual.

(2) Without Logical Statements. A Compound case node has no logical statements attached to it and is used to represent a multiple decision point with multiple exit points. In a compound case node, each node within it is evaluated separately, from top to bottom. Similar to the *If/then/else* or *CASE* constructs of traditional text-based programming, if the first node evaluates to true, then control will pass from the compound node to the node connected by the true arc from that node of the compound node. If the node evaluates to false, the next node in the chain is evaluated. If it evaluates to unknown, then Adept exits the compound case node at the unknown arc. This process is continued until either one of the nodes evaluates to true or unknown or all nodes evaluate to false. In the latter case, Adept will exit at the false arc.

E. STATEMENTS

Statements are expressions that describe how Adept is to evaluate or perform a specific problem or task. These statements are located either to the right of a node or within the script of a node and specifically define the role that the node plays in the procedure. The similarity between Statements, in structure and terminology, to some standard programming language instructions will become apparent in the following discussion.

There are four types of statements that Adept uses and their usage is dependent upon the style of node to which it will be attached. There are many combinations of statements and nodes that may be used by a programmer in the design of Adept expert systems. However, for the purpose of this thesis, only a brief overview of these statements are presented. Should a more indepth and detailed analysis of statements and their usage be required, the reader is referred to the Adept Reference Manual.

1. General Statements

As defined by Adept (Himes et al, 1991, p. 100), a general statement "is any expression that contains constants, variables, functions, objects, relational operators, assignment operators, arithmetic operators and logical operators". The following is a description of each

of the components that can be included in a general statement.

- Constants can be text strings, reserved words or numbers.
- Variables are symbols that can take on many values.
- Functions are self-contained units consisting of a set of programming instructions.
- Objects are those components found on the display screen. These include buttons, fields and check boxes.
- Relational operators are operators, such as equal to or less than, which are used to compare two elements.
- Assignment operators are used to assign a specific value to a variable.
- Arithmetic operators are those operators that allow mathematical manipulations to be performed on numerical values.
- Logical operators are used to combine the results of two or more statements in order to obtain an overall result.

2. Literal Statements

As defined in the Adept Reference manual (1991, p. 102), literal statements are expressions that do not contain operators, functions or objects. They contain the programmer's own words or phrases and are used to describe goals, tasks or results. An example of a literal statement would be *Santa Claus is real*.

3. Control Statements

A control statement, as defined by the Adept Reference Manual (Himes et al, 1991, p. 103), "is a reserved word that defines the conditions under which Adept evaluates a group of general statements in a script". Control statements allow the programmer to control when certain tasks are performed based upon specific pre-defined conditions. Some of the reserved words include:

- if/then/else
- while/do
- do/until
- step

4. Compound Statements

These statements consist of a combination of one or more of the previously listed statement types. This type of statement is typically used in custom nodes when the programmer desires to pass more than one parameter into the node.

The next chapter applies the structured methodology presented in chapter three to the Adept constructs discussed in this chapter.

V. STRUCTURED METHODOLOGIES FOR ADEPT

This chapter is comprised of two main sections. The first is a presentation and discussion of guidelines for developing an expert system using the Adept expert system shell. These guidelines are based on the structured methodologies of traditional text-based programming languages as presented in Chapter III. The second is an application of those guidelines to the Performance module of the MK92 MOD 2 FCS MAES developed in an earlier effort.

A. STRUCTURED METHODOLOGY GUIDELINES FOR ADEPT

In order to maintain ease of cross referencing as well as readability, the same format used in chapter three on traditional structured methodologies is utilized in this chapter. As stated in Chapter III, a visual programming interface tends to create an overlap between the structured design and structured programming methodologies. As such, the application of one characteristic of the structured methodology can be contingent upon the application of another.

1. Structure Charts

Since an Adept application represents a system of independent modules called procedures, structure charts should be used to partition the overall system into a top-down hierarchy of modules and submodules. This guideline

reflects a one to one correspondence between traditional text-based programming languages and Adept in the use of structure charts as a tool for the application of a structured design methodology.

2. Modules

As stated in Chapter III, a module represents a series of instructions or program statements that are grouped together to perform one solitary function. In Adept, modules are represented as procedures and the program statements are represented by nodes. An application can be broken down and factored out into a series of procedures, each performing a specific function within the application. As such, each procedure can be developed such that its function is known and has a well-defined role. But the details as to how the procedure performs the function is not known to, or required by, the other procedures in the application. In other words, each procedure can be designed as a black box.

Each procedure can be designed to have only one input flow, mechanics and one function. However, in Adept, a procedure of moderate size can, and typically, have more than one output flow. For example, the procedure in Figure 5.1 has three exit points; one for *FC1 ACQ F* and two for *FC1ACQ Menu*. In a traditional text-based programming language, having more than one exit would be confusing and

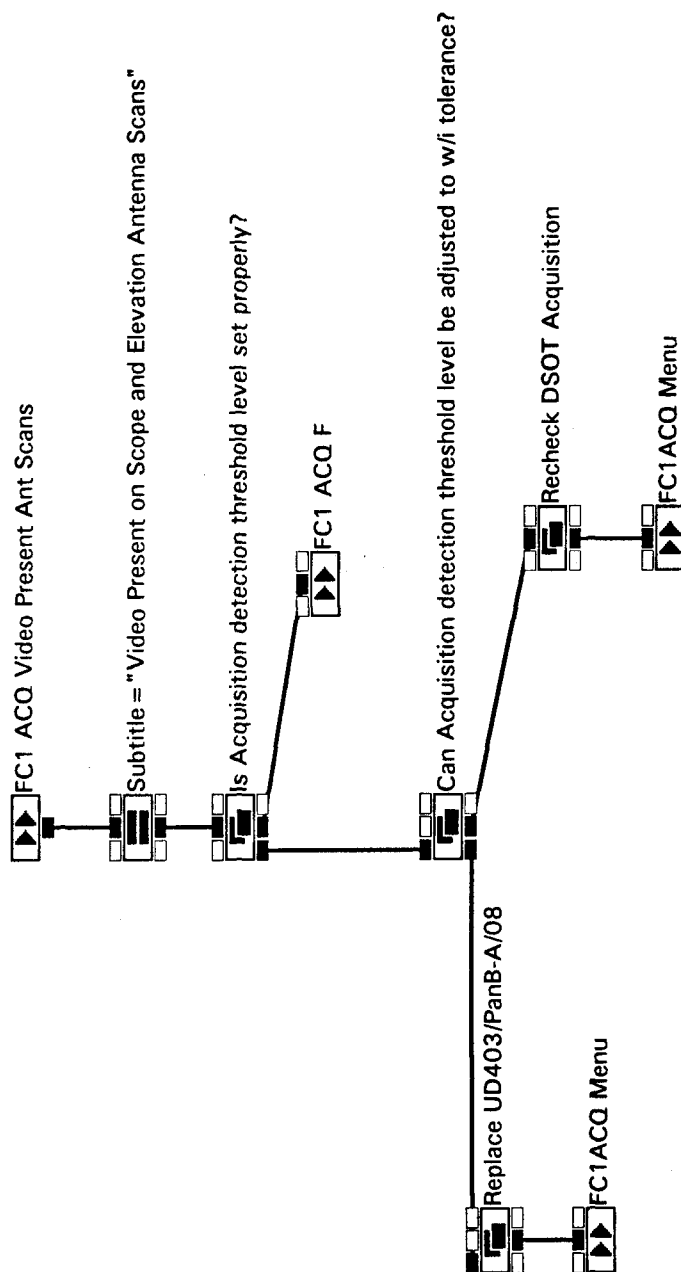


Figure 5.1: FC1 ACQ Video Present Ant Scans Procedure

could cause a decrease in programmer comprehension and an increase in maintenance time. However, in Adept, by the very nature of the visual programming interface, the paths are easy to follow; therefore, making the exit points easy to locate. As such, the general rule that a module should have only one exit point can be relaxed in Adept. In fact, if the single-exit property of structured design and programming were to be applied to Figure 5.1, this procedure would need to be broken out into three distinct sub-procedures. This practice would sacrifice the functionally cohesive property of Figure 5.1 in order to maintain the one-exit property of a module. In addition, excluding the actions to be taken at the end of the paths, each sub-procedure would duplicate coding found in the other sub-procedures; a practice that is to be avoided in the application of structured design and programming methodologies.

As in traditional text-based programming, factoring should be employed, where feasible, in order to reduce redundant coding and increase procedure reuse. If there are series of nodes, representing one function, that are repeated in two or more procedures within an Adept application, then these nodes should be factored out into a procedure of their own. An example of factoring out

redundant nodes is provided in Section B.3.b of this chapter.

3. Intra-modular Design Characteristics

Two characteristics of concern in Adept are module size and cohesion. The following sections discuss these two characteristics as well as the trade-offs between the two.

a. Module Size

The recommended maximum size of a procedure in Adept is approximately 30 nodes. The reasons for this are two-fold.

First, as more and more nodes are added to a procedure, the complexity of the procedure increases, thereby decreasing the readability of the procedure. This, in turn, reduces programmer comprehension, thereby negatively impacting on the maintainability of the procedure.

Second, because Adept is a visual programming language, the readability of a printed procedure decreases as the size and complexity of the procedure increases. The reason is that Adept can only print out large procedures in one of two ways. The first way is to shrink the procedure to such a size that would force it to fit onto one page. In this case, the nodes and statements in the procedure will either become extremely small or they will begin to overlap

onto each other. The second way is to separate the procedure into segments and print out each separately. In this event, the project engineer would then have to paste the individual printouts together in order to follow the logic of the procedure. In both cases, the logic of the printed procedure becomes very difficult to follow.

Therefore, within the confines of the Adept programming language, it may become necessary to sacrifice functional cohesion in order to attain a module which is easy to comprehend, implement, maintain and reuse.

b. Cohesion

A project engineer, programming in the Adept expert system shell, should always attempt to achieve functionally cohesive procedures. That is, the procedures should be developed such that each one performs a solitary, well-defined function. Generally, this objective is relatively easy to obtain. However, as a direct result of the constraints that the Adept programming environment places upon module size, there will be occasions where the highest attainable level of cohesion will be sequential cohesion. This situation should not be a cause for alarm as sequential cohesion is still a very good and acceptable cohesion level.

c. Trade-off Between Cohesion and Module Size

Given the above information, the following two questions need to be addressed:

- When should module size take precedence over the cohesion property of the module?
- When module size is of primary concern, to what degree should cohesion be sacrificed in order to obtain a "good" module size?

First, the reader must take into account that by the very nature of the constructs of Adept, it is very difficult for a procedure to achieve anything less than sequential cohesion. In Adept, all nodes within a procedure are always linked together via arcs as a sequence of steps. The only way for that link to be disrupted is if the project engineer forgot to attach an arc between the nodes. In this case, the program logic is flawed and the error will be discovered when running the application. Given that the achievement of sequential cohesion is still very good, it is the author's opinion, that the characteristic that takes precedence will ultimately depend upon the specific system that is being implemented and the system engineer's preference. A specific example of the trade-off between functional cohesion and module size will be provided in Section B of this chapter.

4. Inter-modular Design Characteristics

When partitioning an application and designing procedures, the system engineer's primary concern is the degree of coupling between procedures. The more independent each procedure is from the others, the lower the degree of coupling and the better the design of the application.

Adept passes information between procedures in two ways. One is via the result and goal style nodes and the other is via the use of global variables.

a. Result and Goal Nodes

As discussed in Chapter IV, result and goal style nodes are used primarily for forward and backward chaining throughout the application. In this case, only one variable is passed between procedures. This variable can take on only one of three possible values; true, false and unknown. This type of data passing is simple and uncomplicated and therefore exhibits good *Data Coupling*.

b. Global Variables

Although the use of global variables is dictated through standard programming conventions and not structured design or programming methodologies, they are mentioned here because they are the only other way in which information is passed from one procedure to another. In traditional text-based programming languages, global variables are those variables whose values are defined for the entire system and

are not passed as parameters from one module to the next. Global variables are typically assigned values that remain constant throughout the system and are used to provide a descriptive way to identify a constant. Standard programming conventions discourage the use of global variables when the assigned value does not remain constant. This is primarily due to the poor maintenance characteristics of global variables. If a global variable's value were to change as the program is executed, and it is not passed as a formal parameter from one module to the next, then an error in the variable can be passed, unchecked, from one module to the next. This, in turn, can complicate the maintenance of an application. This feature holds true in Adept.

The use of global variables constitutes *Common coupling* which in traditional text-based structured design and programming methodologies is highly discouraged. However, as global variables are the only other way in which Adept can pass information, this rule can be relaxed if their use is warranted and they are used carefully. However, in many cases the utilization of global variables can be avoided in favor of local variables. However, when dealing with procedures that have a high reuse rate, the use of global variables whose values change can be a virtual necessity. A specific example is provided in Section B.

5. Other Good Design/Programming Characteristics

The remaining characteristics of the structured methodology to be applied to Adept are restrictivity, generality, fan-out and fan-in. The following discusses these characteristics in more depth.

a. Restrictivity/Generality

The manner in which this characteristic is applied in Adept corresponds exactly to the manner in which it is applied in traditional text-based programming languages. Like a program, a procedure in Adept can be made to be restrictive or general to such a degree that it limits the procedure's capability to be reused within the system. In order to determine if a procedure is too restrictive nor too general, the system engineer must step back and review the application as a whole. He/she must then determine the rate at which an application uses a specific procedure. If a module is used minimally in the application, then he/she may want to review the procedure to ensure that it is neither too restrictive nor general. It should be noted that a procedure which displays restrictive or general properties is not necessarily a bad procedure. There are some cases where the presence of these types of procedures in an expert system is necessary in order to fulfill a specific requirement specified in the knowledge document. The necessity of this type of procedure will ultimately be

determined by the system engineer(s) and the domain expert(s) of the application under development.

b. *Fan-in and Fan-out*

As stated in Chapter III, fan-in equates to module reuse. In Adept, as in traditional text-based programming languages, there can never be too much procedure reuse provided that the procedures exhibit good cohesion and their parameter interfaces are not sacrificed. Fan-out, on the other hand, indicates the number of subordinates stemming from any given procedure. In Adept, fan-out is dictated by module size in addition to the psychological reasons presented by traditional text-based programming languages. Similar to the text-based programs, the general rule for the number of child procedures stemming from any one parent procedure is approximately seven. However, because a procedure's size is restricted within the confines of the Adept programming environment, that number may be increased in order to preserve programmer comprehension. These module size constraints were discussed in a previous section of this chapter.

B. APPLICATION OF STRUCTURED TECHNIQUES FOR VISUAL PROGRAMMING AND DESIGN TO THE PERFORMANCE PROTOTYPE

This section provides a brief description of the prototyping technique of systems development, how that technique was applied to the initial development of the

Performance module and the application of the structured techniques for visual programming in Adept to the initial Performance prototype.

1. Prototyping

As defined by Whitten et al (1989, pp. 124, 414), prototyping is a first full-scale, and usually functional, working model of a system or subsystem. Prototyping is a technique that is typically used when the end users or, in this case, the domain experts can not easily define a complete set of systems requirements in one iteration. As such, as the prototype is developed, the domain experts test the system and recommend modifications to the system based upon requirements, methods and format. This cycle continues until the system is accepted by the users. There are advantages and disadvantages to using the prototyping technique. Some of the advantages and disadvantages are as follows:

a. Advantages

- Encourages the domain experts to become more active in systems development.
- Allows for iterative development which is beneficial to expert system development.
- Provides domain experts with a tangible product.
- Errors are detected earlier in the SDLC.
- Accelerates the definition, design, and construction phases of the SDLC.

b. Disadvantages

- Prototyping tends not to promote a thorough application of structured analysis and design methodologies.
- Systems engineers tend to use the prototype as the sole source of specifications. However, paper specifications are still a necessity when utilizing the prototyping technique (Whitten et al, 1994, p. 492)
- Can lead to "premature commitment to a physical design" (Whitten et al, 1989, p. 417).
- "Overreliance on prototyping tends to lead to development of technological approaches that don't always solve problems and fulfill requirements" (Whitten et al, 1989, p. 417).
- Tends to stifle process improvement as the first solution is often the one implemented.

When using the prototyping technique of system development, it is important to remember that prototyping is not a replacement for standard design and programming methodologies or the SDLC. It is to be used as a complement to structured methodologies and the SDLC.

2. Initial Performance Module Prototype

The Performance module was initially constructed as a first subsystem prototype for the MAES. In the early phases of the prototyping process, the programmers faced a steep learning curve as they were not only required to learn the Adept expert system shell, but also the terminology and the concepts associated with the MK92 MOD 2 FCS. As such, it was determined that the best approach to knowledge

implementation would be to segment the Performance module knowledge document into readable and maintainable modules and represent them in the form of a structure chart (Lewis, 1993, Appendix A and Smith, 1993, Appendix A). The knowledge was then implemented in Adept, exactly as it appeared in the structure chart, using *forward chaining*. Once all the modules of the knowledge document were implemented, the initial prototype was given to the domain experts for review.

Upon review of the Performance module prototype the domain experts were able to determine that modifications were required. The initial knowledge document lacked pertinent information which affected the program logic of the prototype. As such, the domain experts sent revised knowledge documents back to the project engineers for implementation. This cycle continued until May 1994, when it was determined that the Performance module was functional in the sense that it duplicated, to the extent feasible, the procedures that the domain experts would follow in the diagnosis of the MK92 MOD2 FCS Performance module system defaults.

Once the prototype was completed and functional, the project managers and project engineers needed to determine if the current software was suitable to handle implementation of the entire system or if the system should

be reprogrammed in a traditional text-based programming language. Although by no means all inclusive, the following questions were representative of what was considered in the decision process:

- Will the overall system be used on a PC or a mainframe?
- Is the prototype software effective, efficient, and easily maintainable by system programmers?
- Will the software still be effective, efficient, and easily maintained if developed on a large scale?

Upon review of current system requirements, it was determined that the Adept expert system shell would be used to develop the production version of the MAES. It was also decided that the Performance prototype should be restructured by applying sound structured design and programming methodologies.

The following section addresses the reconstruction of the Performance Module Prototype employing the guidelines presented in Section A of this chapter.

3. Reconstruction of the Performance Prototype

In applying a structured methodology to the Performance prototype, the first task was to determine if it would be necessary to discard the entire prototype in favor of total redevelopment or if reconstruction of the current prototype would be a more viable option. Since the prototype fulfilled user specifications and the current

application software was adequate for the MK92 MAES application, the project engineers opted to restructure the current prototype via the application of structured design and programming methodologies.

The purpose of the following discussion is two-fold. First it will provide the reader with an example of how the guidelines of the structured methodology can be applied to an actual system. Secondly, it will serve as the basis by which the Performance prototype will be re-structured. For the ease of readability and cross-referencing between chapters, the format of the following section will be the same as in chapter three. In addition, not all aspects of the reconstruction will be presented here as it would be extremely tedious to mention every single modification to the prototype. Rather, a new structure chart, after application of the new structured methodology, of the Performance prototype is provided in Appendix A.

a. Structure Charts

When a prototyping technique is used in the development of information systems, the process of developing structure charts becomes iterative in nature. As user specifications are re-defined and updated, the structure charts must be updated to accurately reflect the true structure of the evolving prototype.

As stated previously, the project engineers used a prototyping technique in the development of the Performance module of the MK92 MAES. Structure charts were developed in the initial design stages of the prototype and were updated as modifications were made to both the user specifications and the prototype.

Although there are no modifications to the user specifications at this time, the structure charts needed to be modified in order to reflect the changes in the prototype due to the application of the structured design and programming methodologies. These are provided in Appendix A.

b. Modules

The project engineers did a good job in segregating the application into a series of independent procedures which exhibited the black box properties of modularization. However, factoring and module reuse techniques were not fully applied throughout the application.

First, some procedures contained functions that should have been factored out into separate procedures as the use of these functions were required by other modules. Figure 5.2 is such a procedure. Notice that this procedure, called *FC2 ACQ E*, has two start-result nodes; one labeled *FC2 ACQ Ed* and the other labeled

FC2 ACQ E. As designed, those nodes directly below and attached to the *FC2 ACQ Ed* start node may be triggered not only by the *FC2 ACQ E* procedure, but also by other procedures in the application. Since this series of nodes is triggered by other procedures outside of the one in which they are a part, these nodes should be factored out of this procedure and placed in a separate procedure of their own. There are a total of 16 modules that exhibit this trait.

Second, there are several modules within the prototype that perform identical functions and could be collapsed together to increase module reuse. This was one of the primary design and implementation flaws of the prototype. One such example is within the *FC1 Track Bearing and Track Elevation* modules of the prototype. Each of these modules has 11 subordinate modules. When compared against one another, their functions are identical. Specifically, Figure 5.3 is a subordinate module of *FC1 Track Bearing* called *Low XTAL Current* and Figure 5.4 is a subordinate module of *FC1 Track Elevation* called *ELVTN Low XTAL Current*. These two modules are identical with the exception of the following:

- One procedure reviews "Delta B XTALS" while the other reviews "Delta E XTALS".
- The replacement card numbers are different.
- The result nodes refer to different modules.

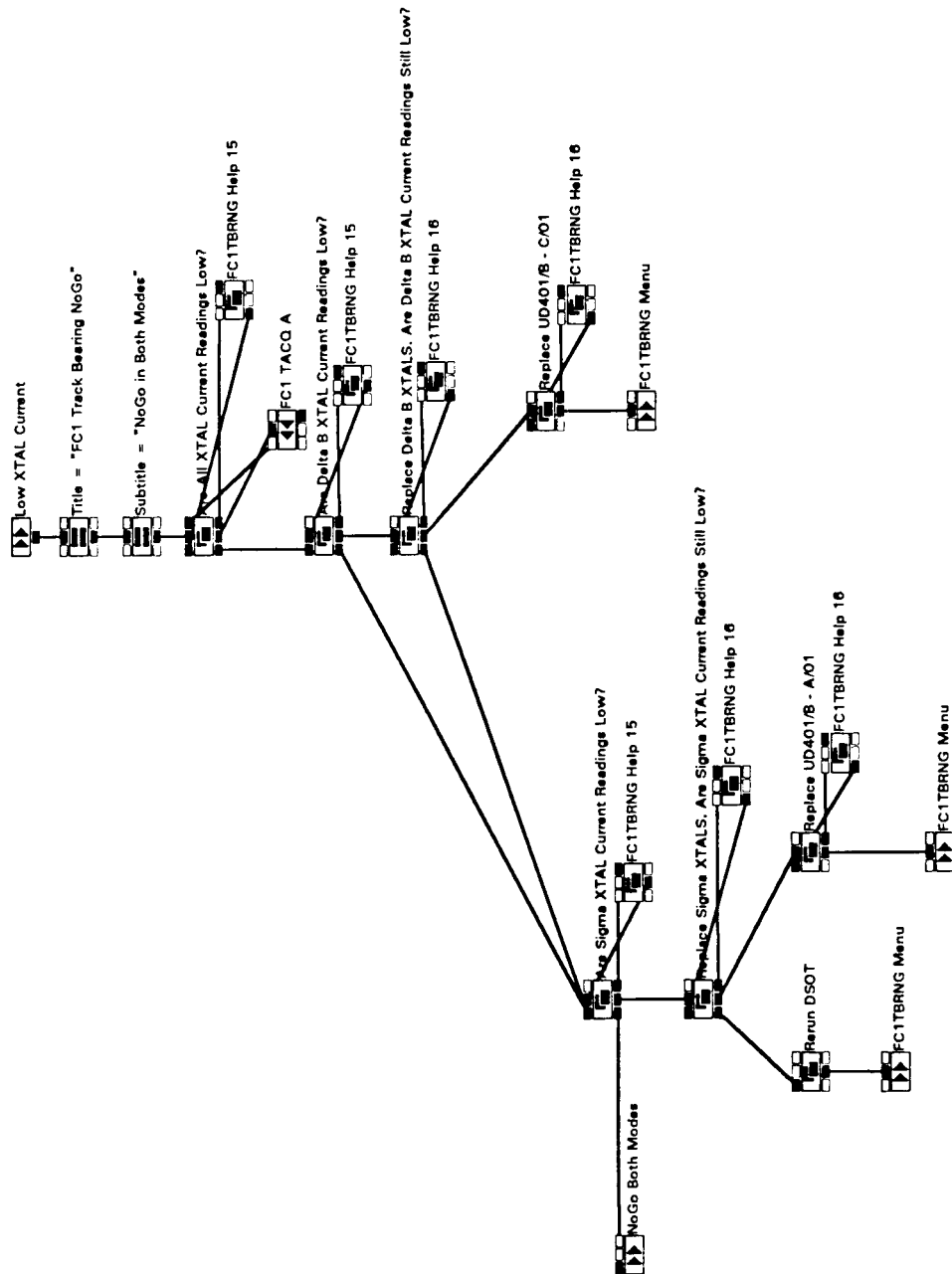


Figure 5.3: Low XTAL Current Procedure

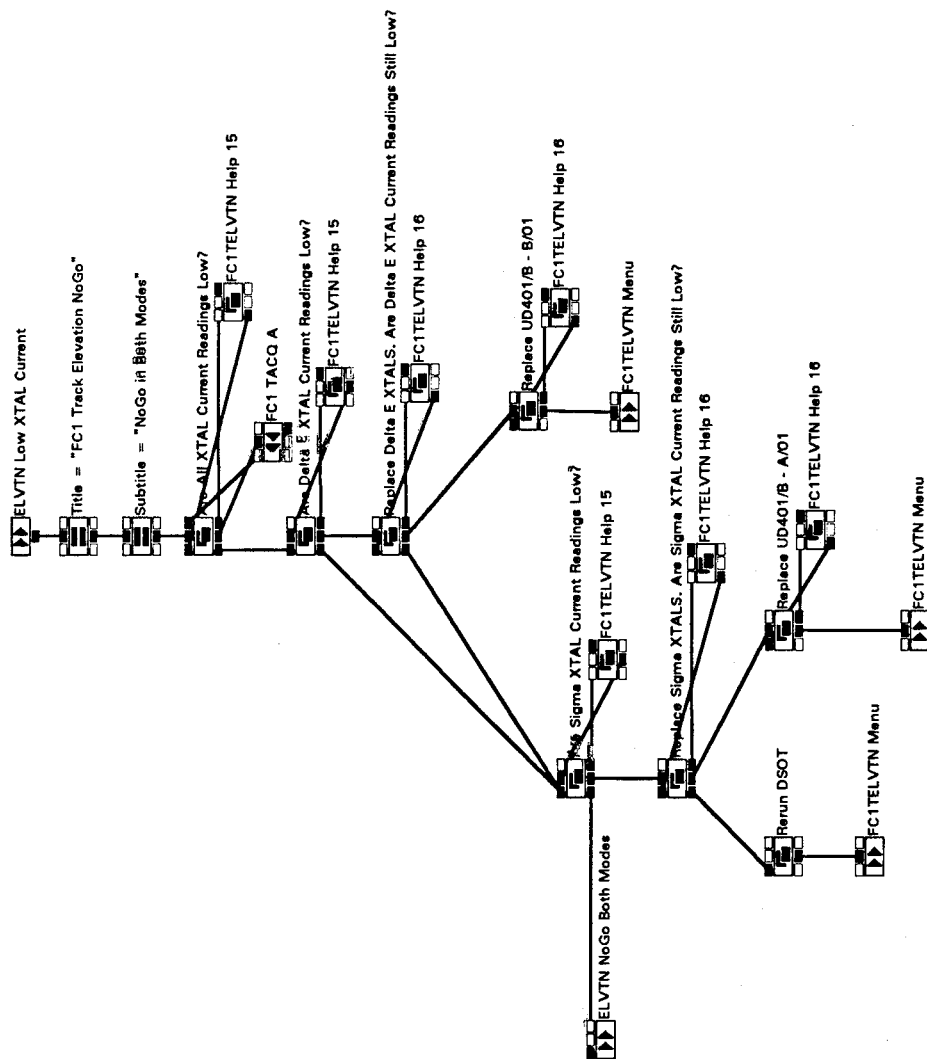


Figure 5.4: ELVTN Low XTAL Current Procedure

These modules can be collapsed together and custom nodes used in order to account for the differences listed above. Collapsing these modules would reduce the overall size of the prototype by 11 procedures. There are several places in the prototype where similar situations occur. Once all of the redundant modules are eliminated, it is estimated that the prototype could be reduced by approximately one third of its original size.

c. Intra-modular Design Characteristics

The two primary intra-modular design characteristics of concern are cohesion and module size. In most cases, functional cohesion was attained in the modules located in the upper levels of the system without sacrificing module size. However, in the lower levels, the project engineers sacrificed functional cohesion for module size. Where this occurred, the lowest level of cohesion attained was Sequential cohesion. For example, Figures 5.5 and 5.6 are two separate procedures within the FC2 Track Elevation module. Figure 5.6 was factored out as a subordinate module to Figure 5.5. The only purpose of factoring out FC2 TELVTN Case 31 into a module of its own was to preserve module size.

Which characteristic should take precedence? In this particular case, the solution is obvious. It would be easiest to factor out those nodes, in FC2 TELVTN Case 3,

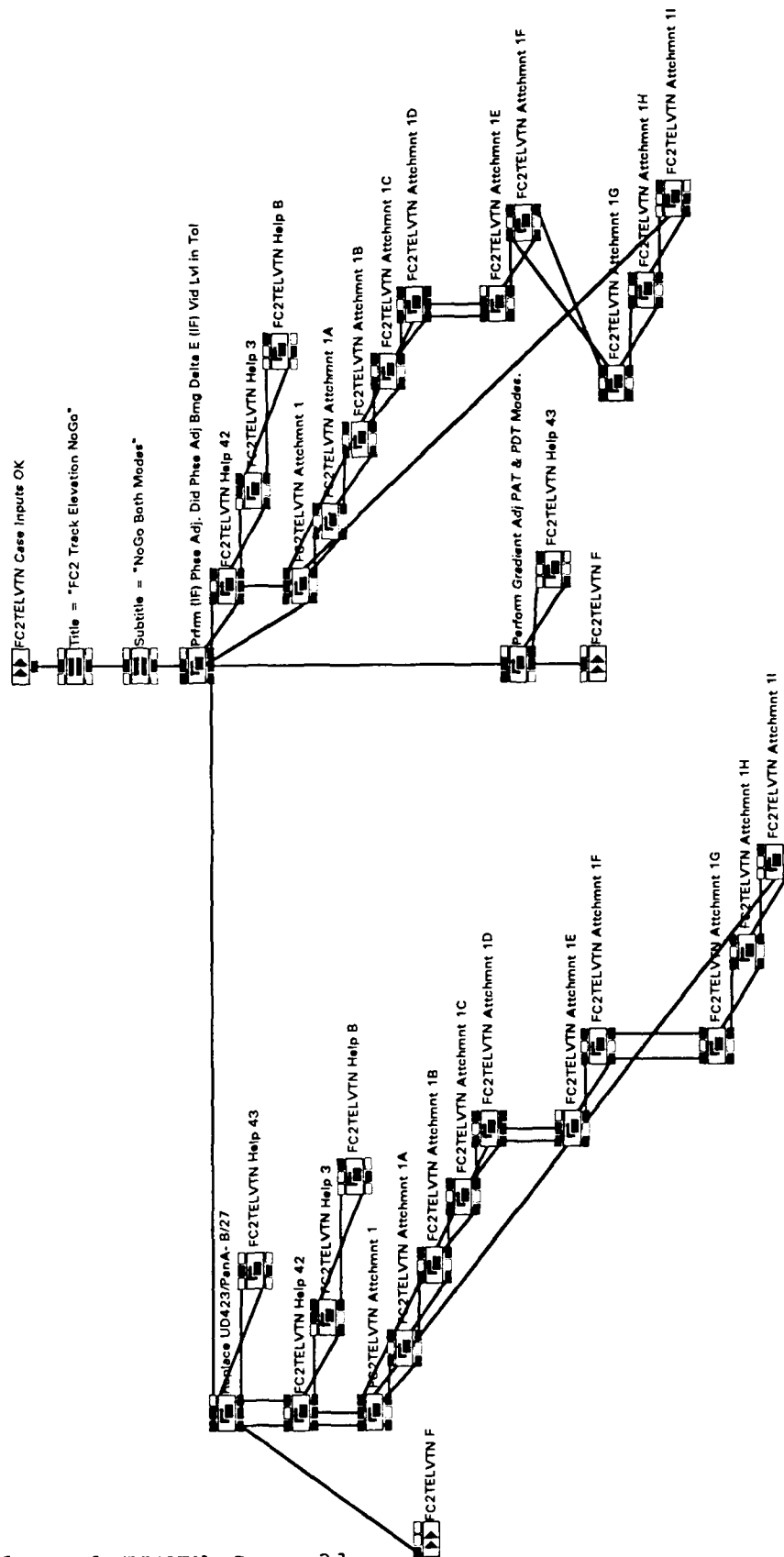


Figure 5.6: FC2TELVTN Case 31

below the compound node and create another module. However, there will be some cases where a cut-off point will not be so obvious and it will be up to the project engineers to decide.

d. Inter-modular Design Characteristics

Throughout the Performance prototype, most procedures exhibited *Data coupling* which is the lowest degree of coupling attainable. There are three pieces of data that are passed from module to module; *Title*, *Subtitle*, and the value (true, false, or unknown) of the last evaluated node of the calling module. The first two pieces of data are passed through the use of global variables. Although the use of global variables should be avoided, given the nature of the constructs of Adept, the project engineers were left with no other viable alternative. Since all modules contain displays that require these inputs, the project engineers would have had to invent new variable names for each module in the prototype in order to avoid the use of global variables. In the procedures that were not used as common modules, the global variables were treated as local variables and redefined at the beginning of the procedure, thereby attaining *Data coupling*. However, the common procedures will have the values of the global variables passed to them from other procedures. As such, these modules only attained *Content coupling*.

e. Other Design Programming Characteristics

As stated in Chapter III, these characteristics include *Restrictivity/Generality*, *Fan-out*, and *Fan-in*. Upon application of the design characteristics mentioned previously, none of the procedures in the Performance prototype will be either too restrictive or too general. This is primarily because of the module size issue presented earlier. The *Fan-in* of the prototype will increase as redundant modules are collapsed together to promote procedure reuse. There was no instance in the prototype where there was too much *Fan-out*. The largest number of child procedures assigned to any given parent procedure in the Performance prototype is six.

The next chapter discusses the lessons learned and insights gained by the author during the development of this thesis.

VI. LESSONS LEARNED AND CONCLUSIONS

This chapter discusses the lessons learned during the implementation of the structured design and programming methodology developed in chapter three and the invaluable insights gained by the author as a result of the work on this thesis. Most of the lessons learned revolve around the Performance module prototype of the MK92 MOD 2 FCS MAES. Uninterestingly enough, and also most frustrating, is that almost every lesson learned during this project was not a new revelation. The problems encountered during this thesis were the same ones that have been identified repeatedly by many information system professionals for years as being common problems found with most information system development projects. The purpose of restating them here is to reinforce what has already been presented in numerous structured analysis, design and programming text books and what has been taught in many courses attended by undergraduate and graduate students alike. This knowledge, however, did not preclude students from repeating the same errors. Only practice provides a deep understanding of these issues and an appreciation of the techniques to avoid them.

The lessons learned can be broken up into two general categories; design and implementation issues and documentation issues.

A. DESIGN AND IMPLEMENTATION ISSUES

Many of the problems encountered with the design and implementation of the Performance module prototype were caused by the following:

- lack of adherence to the analysis and design phases of the SDLC
- lack of a thorough application of a structured design and/or programming methodology
- lack of the application of standard programming conventions.

These issues were addressed in previous chapters; however, they are restated briefly here as to emphasize their importance as valuable lessons learned.

1. Lack of Adherence to System Development Life Cycle

As discussed in Chapter V, one of the purposes for employing the prototyping technique is to accelerate the definition, design, and construction phases of the SDLC, not eliminate them. However, as is typical, the use of prototyping usually tends to promote the incomplete application of the analysis and design phases of the SDLC. The initial Performance prototype was implemented using a piecemeal process. The complete set of user specifications, or in this case the documented knowledge, was not available

for review prior to initiating the actual coding of the system. As such, the overall structure of the prototype suffered greatly.

In order to avoid these types of problems, system engineers should adhere to and follow through on all phases of the SDLC during the development of an information system, regardless of the tools that are used to complement it.

2. Lack of Thorough Application of Structured Design and Programming Methodologies

The use and application of a structured design and/or programming methodology was not thoroughly applied during the initial development of the Performance prototype. As such, the prototype was not well-defined and lacked characteristics that contribute to the overall readability, flexibility, standardization and maintainability of the prototype. Chapter V examined the problems associated with the deficiencies of the initial prototype and how those problems can be rectified through the application of a structured design and programming methodology.

3. Lack of Application of Standard Programming Conventions

The initial prototype did not fully apply the techniques of standard programming conventions. Some of these include the lack of standardized procedure naming conventions and internal documentation. Although these conventions are not considered a requirement of the

structured programming methodology, their application would contribute to the readability, standardization and maintainability of the system. For example, internal documentation is important as it provides the programmer with valuable information about a specific module, thereby improving programmer comprehension and thus contributing to the reduction of maintenance time. As such, standard programming practices should be applied where feasible in order to increase programmer comprehension and the maintainability of the system.

B. DOCUMENTATION ISSUES

The need for documentation standards underscores a common failure of many analysts--the failure to document as an ongoing activity during the life cycle....Most of us tend to post-document software. Unfortunately, we often carry this bad habit over to system's development. (Whitten et al, 1994, p.93).

Although the design and implementation issues, stated previously, contributed to the poor maintenance characteristics of the original prototype, they were not the sole contributor to the problem. The lack of thorough external documentation made the maintenance of the prototype very difficult. The following is a discussion of these issues.

1. Initial Documentation

The documentation of the initial prototype software was limited and that which was available was scattered and

unorganized. As is typical, most of the documentation appeared to have been done after the actual implementation of the knowledge document. The lack of thorough documentation caused the learning curve of the follow on project engineers to increase tremendously. In order to revise the prototype, the project engineers first had to decipher how the prototype was designed/implemented, and to determine if a special methodology had been applied during the design/implementation, and if so, what was it and why was it used? This deficiency was corrected through the organization of the available documentation and the incorporation of additional known information that was pertinent to the last five revisions of the prototype. Beginning with Revision B, binders were developed for each version of the Performance prototype. Each binder contains the following information:

- the knowledge document
- a printout of the Adept procedures
- the knowledge document modification requests provided by the domain experts
- the project engineer's notes regarding the development process, standards used, etc...
- a backup copy of the prototype's programming code on 3.5" diskettes

In addition, all theses, pertaining to the development of the MK92 MOD 2 FCS MAES should be integrated

into one document. All documentation should be actively maintained as requirements dictate.

2. Prototype Modifications

Standard practice during the development and revision of a CBIS is to maintain all documentation that reflects the initial development of the system plus any modifications made to that system thereafter. Such documentation is maintained until the CBIS has been tested, evaluated, and approved for distribution. At which time, only that documentation which describes the final version of the system is required to be maintained. This includes such documentation as structure charts, module interfaces, and data dictionaries.

The initial prototype went through two iterations of revisions prior to submission for testing and evaluation. The available documentation provided no insight as to what the first iteration prototype consisted of nor was there any information on how and why the prototype had been modified. Had this been the final version of the prototype, this would not have been a problem. However, five months after the initial prototype was submitted, it was discovered that the domain experts had at least two more sets of revisions that still needed to be implemented. These revisions were of particular importance as their implementation changed the actual logic of the prototype. The revisions, according to

the domain experts, had been put on hold by the project engineers due to time constraints. None of this was indicated in the documentation and therefore unknown to the project managers. This faux pas extended the scheduled project completion date by an additional six months.

All documentation, pertinent to the development of the MK92 MOD 2 FCS MAES, should be co-located with the prototype until project completion. As such, any additional requests for modifications should be forwarded to the project engineers for incorporation into the established documentation discussed in the previous section.

3. Knowledge Documentation

The initial knowledge documentation, submitted to the NPS project team, was handwritten and fairly clear. However, as modifications were requested and implemented, the original document was scratched over and the modifications handwritten on to the original document. This process continued through all five revisions of the prototype. Eventually, parts of the document have become very difficult to read and interpret. Figure 6.1 is an example of one such knowledge document. In future revisions to the Performance module, or in the development of other modules of the system, it is recommended that the knowledge document be transferred to the computer using graphical software such as VISIO or allCLEAR. As modifications are

[illegible]

84

made, the knowledge document can be modified and a clean copy printed and maintained. This is presently being done by NPS support staff.

In addition, the knowledge document should be factored out into modules that more closely approximate the actual procedures implemented in Adept. Although not of major importance, it would enhance the cross referencing between the knowledge document and the actual prototype.

Finally, a document should be written that describes the knowledge document itself. During the second revision to the prototype, numbers were added to the knowledge document to indicate the association between the specific *Help* screens, listed on a separate document, to specific nodes within the knowledge document. Later in the development process, when new people joined the project and the original project engineers were gone, it was difficult to ascertain the purpose of those numbers.

C. INSIGHTS AND CONCLUSIONS

The development of a new structured design and programming methodology for use with expert system shells using a visual programming language has been challenging. It has provided the author with invaluable insights into the evolving characteristics of fourth generation languages (4GLs) and how their usage impacts the application of traditional structured programming methodologies.

Of particular interest was the correlation between the increasing technology of programming languages and the decreasing need to utilize professionally trained, traditional text-based programmers in the development of some of the more simplistic CBISSs. The reason for this is that as the technology of programming languages continues to improve in order to accommodate a less technically oriented user, the need for the application of traditional text-based structured programming methodology decreases. The degree to which this correlation holds true is contingent upon the specific 4GL used and the size and complexity of the system to be built. In Adept, many of the more tedious aspects of a structured programming methodology, required in the development of traditional text-based programs, fade away completely or overlap into structured design.

One problem encountered during the development of this thesis was the determination of the degree to which this new structured methodology could be applied to other 4GLs employing visual interfaces. Unlike its predecessor's, this structured methodology is based upon 4GLs whose utilization and application have only recently evolved. The authors of traditional text-based structured programming methodologies developed them after years of tedious, grueling, and time-consuming programming experience with many different second and third generation languages and have continued to expand

and refine them as appropriate. The structured methodology developed in chapter five is predicated upon the constructs of the Adept expert system shell only. As such, the structured methodology developed in this thesis may need to be modified in order to accommodate the specific nuances of the 4GL to which it will be applied.

APPENDIX A

STRUCTURE CHARTS AND PROCEDURE FUNCTION DESCRIPTIONS

A. MAIN MENU PROCEDURES

Name: Main Menu (Figure 1)

Description: The first menu in the program. Allows selection of the Performance or Calibration modules of the diagnostic program or exits the program.

Called by: Initiation of the MK92 Fire Control System Maintenance Advisor Expert System.

Calls: Performance and Calibration Menus.

Name: Performance Menu

Description: Allows the selection of FC1, FC2 or FC4 and FC5 Menus.

Called by: Main Menu

Calls: FC1, FC2 or FC4 and FC5

Name: Calibration Menu

Description: Allows the selection of the Calibration procedures.

Called by: Main Menu

Calls: Calls Calibration CAS or STIR

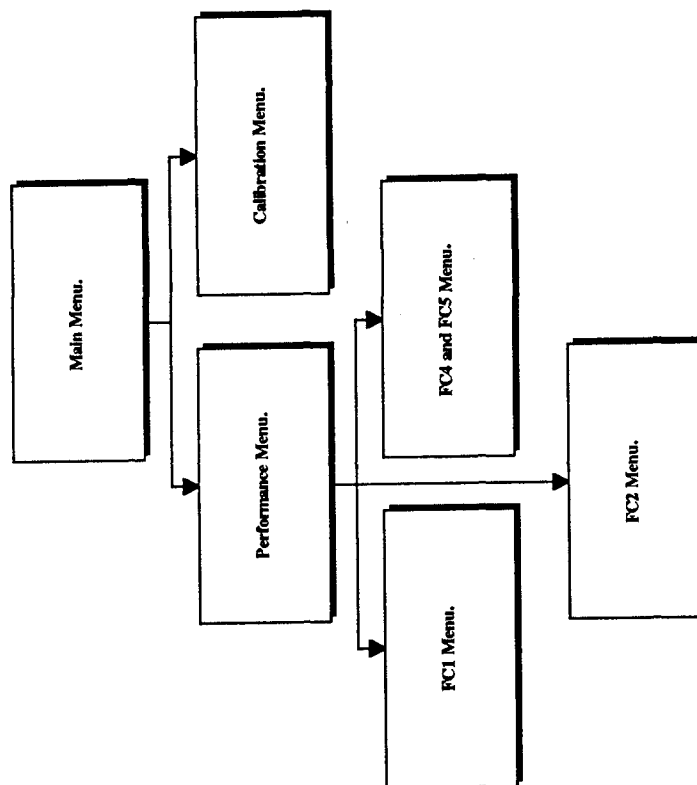


Figure 1: Main Menu

B. FC1 PROCEDURES

Name: FC1 Menu (Figure 2)

Description: Allows selection of FC1 Designation, Acquisition and Track procedures.

Called by: Performance Menu

Calls: FC1 DTRB Menu, FC1 ACQ Menu, and FC1 TBER Menu.

Name: FC1 DTRB Menu

Description: Allows selection of FC1 Designation Track, Range and Bearing procedures.

Called by: FC1 Menu

Calls: FC1 DT Menu, FC1 DR and FC1 DB

Name: FC1 ACQ Menu

Description: Allows selection of FC1 ACQ procedures.

Called by: FC1 Menu

Calls: FC1 ACQ Menu

Name: FC1 TBER Menu

Description: Allows selection of FC1 Track Bearing, Elevation and Range procedures.

Called by: FC1 Menu

Calls: FC1 TBE Menu and FC1 TR Menu

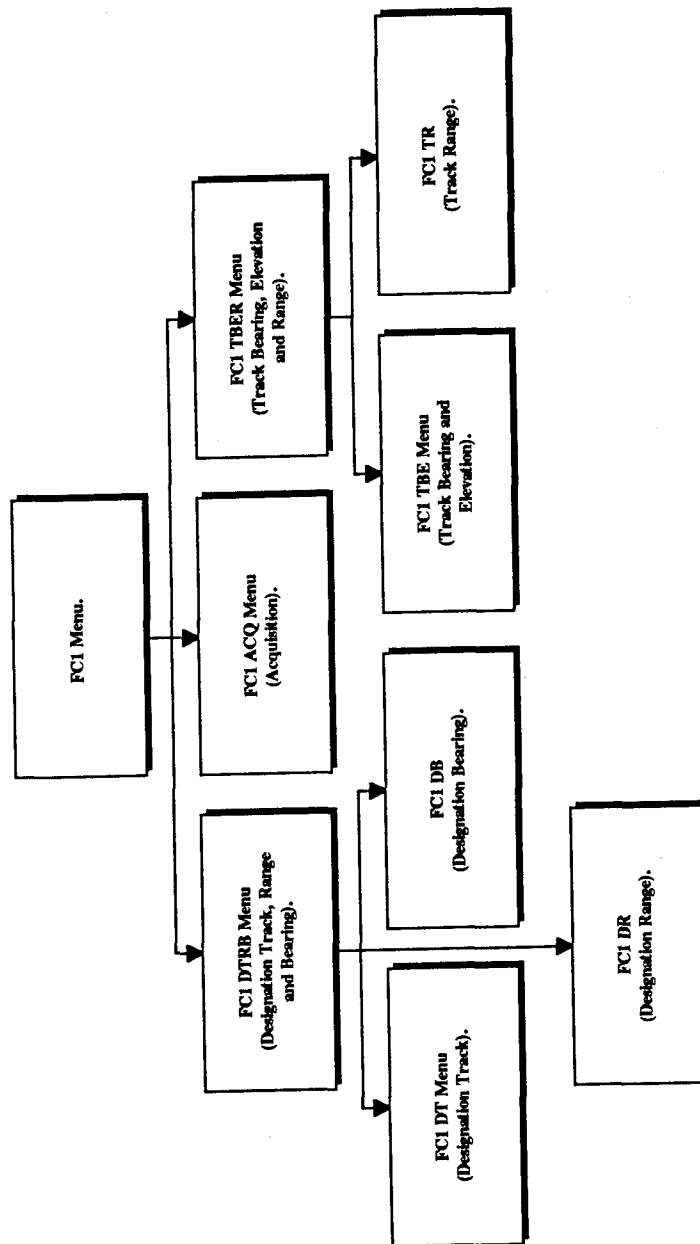


Figure 2: FC1 Menu

C. FC1 DESIGNATION TIME PROCEDURES

Name: FC1 DT Menu (Figure 3)

Description: Allows selection of one of three possible FC1 Designation Time trouble-shooting paths: Case 1 (range reading on TOTE is not the same as range gate position, DT Submenu or No range gate movement.

Called by: FC1 DTRB Menu

Calls: FC1 DT Case 1, FC1 DT Submenu, and FC1 DT No Rng Gte Mvmt A.

Name: FC1 DT Case 1

Description: Allows trouble-shooting of Case 1 procedures.

Called by: FC1 DT Menu

Calls: FC1 DT Case 1A

Name: FC1 DT Case 1A

Description: Continues trouble-shooting Case 1.

Called by: FC1 DT Case 1

Calls: None

Name: FC1 DT Submenu

Description: Allows selection of one of four possible track antenna and range gate movements.

Called by: FC1 DT Menu

Calls: FC1 DT No Trk Ant Mvmt, FC1 DT Trk Ant Mvmt Slow, FC1 DT No Rng Gte Mvmt, FC1 DT Rng Gte Mvmt Slow.

Name: FC1 DT No Trk Ant Mvmt
Description: Allows trouble-shooting of FCS in the event that the track antenna does not move in FC1 designation time.
Called by: FC1 DT Submenu
Calls: FC1 DT No Trk Ant Mvmt A

Name: FC1 DT No Trk Ant Mvmt A
Description: Continues trouble-shooting procedures for no movement of track antenna in FC1 designation time.
Called by: FC1 DT No Trk Ant Mvmt
Calls: None.

Name: FC1 DT Trk Ant Mvmt Slow
Description: Allows trouble-shooting of FCS in the event that the track antenna moves too slowly in FC1 designation time.
Called by: FC1 DT Submenu
Calls: None

Name: FC1 DT No Rng Gte Mvmt
Description: Allows trouble-shooting of FCS in the event that there is no range gate movement in FC1 designation time.
Called by: FC1 DT Submenu
Calls: FC1 DT No Rng Gte Mvmt A

Name: FC1 DT No Rng Gte Mvmt A

Description: This procedure can either be executed as a continuance of trouble-shooting procedures for no movement of the range gate in FC1 designation time or it can be called directly from the main DT Menu as the situation dictates.

Called by: FC1 DT No Rng Gte Mvmt and FC1 DT Menu

Calls: None

Name: FC1 DT Rng Gte Mvmt Slow

Description: Allows trouble-shooting of FCS in the event that the range gate moves too slowly in FC1 designation time.

Called by: FC1 DT Submenu

Calls: None

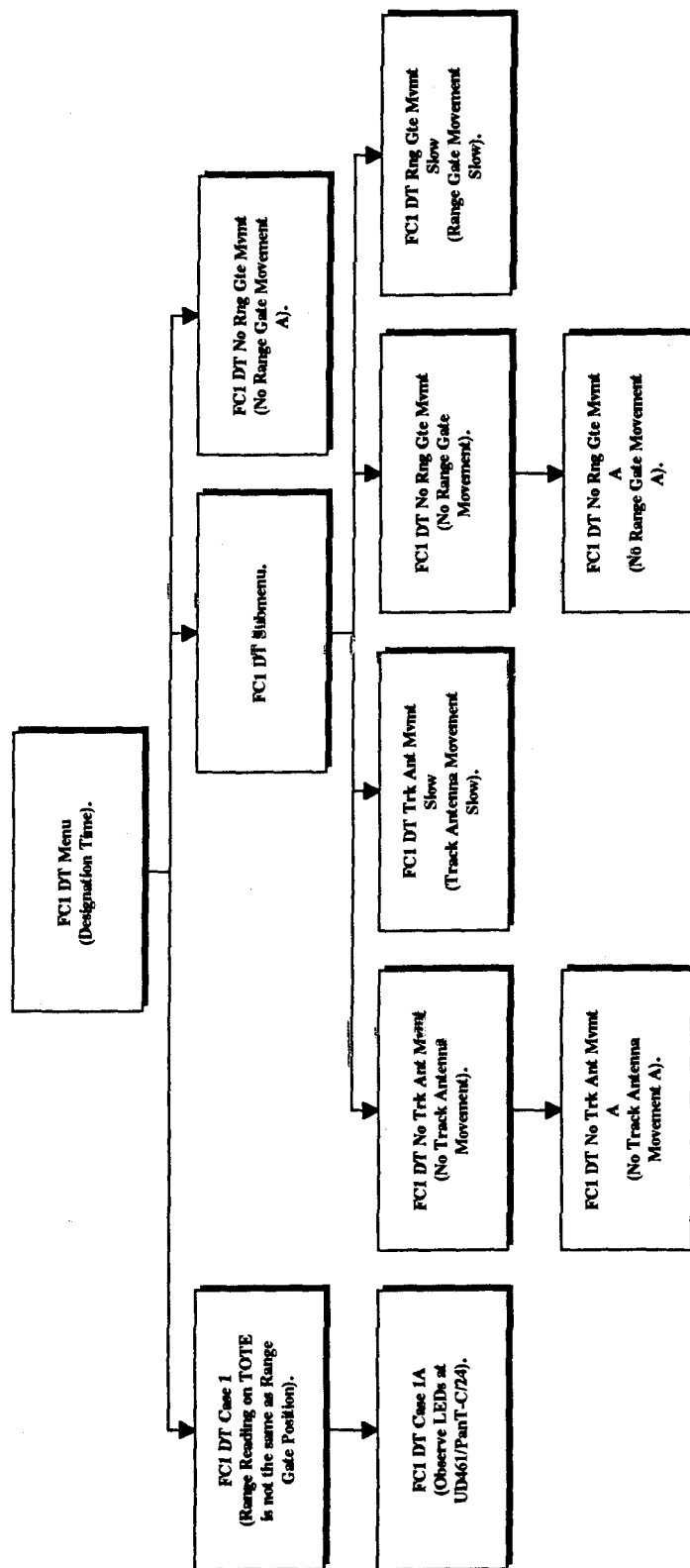


Figure 3: FCI Designation Time Menu

D. FC1 DESIGNATION RANGE PROCEDURES

Name: FC1 DR A (Figure 4)

Description: Allows trouble-shooting of problems associated with FC1 Designation Range.

Called by: FC1 DTRB Menu

Calls: FC1 DR B

Name: FC1 DR B

Description: This procedure is common to and a continuance of both FC1 designation range and FC1 designation bearing trouble-shooting procedures. In this instance, it continues the trouble-shooting procedures for FC1 Designation Range.

Called by: FC1 DR A and FC1 DB

Calls: None

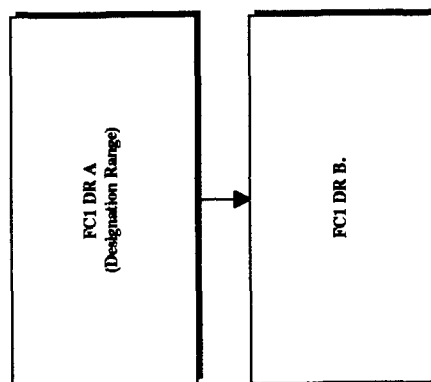


Figure 4: FC1 Designation Range A

E. FC1 DESIGNATION BEARING PROCEDURES

Name: FC1 DB (Figure 5)
Description: Allows trouble-shooting of the FCS designation bearing in FC1.
Called by: FC1 DTRB Menu
Calls: FC1 DR B

Name: FC1 DR B
Description: This procedure is common to and a continuance of both FC1 designation range and FC1 designation bearing trouble-shooting procedures. In this instance, it continues the trouble-shooting procedures for FC1 Designation Bearing.
Called by: FC1 DR A and FC1 DB
Calls: None

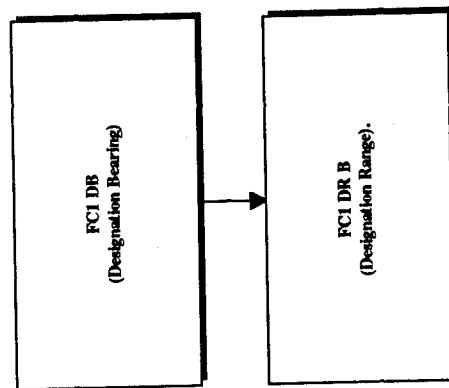


Figure 5: FCI Designation Bearing

F. FC1 ACQUISITION PROCEDURES

Name: FC1 ACQ Menu (Figure 6)

Description: Allows selection of one of four possible trouble-shooting paths of FC1 Acquisition.

Called by: FC1 Menu

Calls: FC1 ACQ Elev Scan Video Present, FC1 ACQ Video Present Ant Scans, FC1 ACQ D and FC1 ACQ E

Name: FC1 ACQ Elev Scan Video Present

Description: Allows trouble-shooting of FC1 Acquisition in the event that the elevation scans and video is present.

Called by: FC1 ACQ Menu

Calls: FC1 ACQ B

Name: FC1 ACQ B

Description: Continues trouble-shooting of FC1 Acquisition elevation scans and video present.

Called by: FC1 ACQ Elev Scan Video Present

Calls: None

Name: FC1 ACQ Video Present Ant Scans

Description: Allows trouble-shooting of FC1 Acquisition in the event that the video is present and the antenna scans.

Called by: FC1 ACQ Menu

Calls: FC1 ACQ F

Name: FC1 ACQ F
Description: Continues trouble-shooting of FC1 ACQ Video Present Ant Scans by checking the video at the video modulator.
Called by: FC1 ACQ Video Present Ant Scans
Calls: FC1 TBE Case 21

Name: FC1 TBE Case 21
Description: This procedure is common to FC1 Acquisition and FC1 Track Bearing and Elevation. In this instance, it continues trouble-shooting procedures of FC1 Acquisition module F.
Called by: FC1 ACQ F
Calls: None

Name: FC1 ACQ D
Description: Allows trouble-shooting of FC1 Acquisition by checking for antenna movement in elevation in WCC (manual mode).
Called by: FC1 ACQ Menu
Calls: None

Name: FC1 ACQ E
Description: Allows trouble-shooting of FC1 Acquisition by checking the crystal current readings and allowing the selection of one of three possible paths; check track receiver LO sample readings, check RF input at UD401/D-U2 or A2 and A7 inputs.
Called by: FC1 ACQ Menu
Calls: FC1 ACQ A, FC1 ACQ C, and FC1 ACQ Eb

Name: FC1 ACQ A
Description: Continues trouble-shooting of FC1 Acquisition
crystal current readings by checking the
track receiver LO sample readings.
Called by: FC1 ACQ E
Calls: FC1 ACQ Aa

Name: FC1 ACQ Aa
Description: Continues trouble-shooting of FC1 Acquisition
track receiver LO sample readings by checking
the TR tubes at UD401).
Called by: FC1 ACQ A
Calls: None

Name: FC1 ACQ C
Description: Continues trouble-shooting of FC1 Acquisition
crystal current readings by checking the A2
and A7 inputs.
Called by: FC1 ACQ E
Calls: None

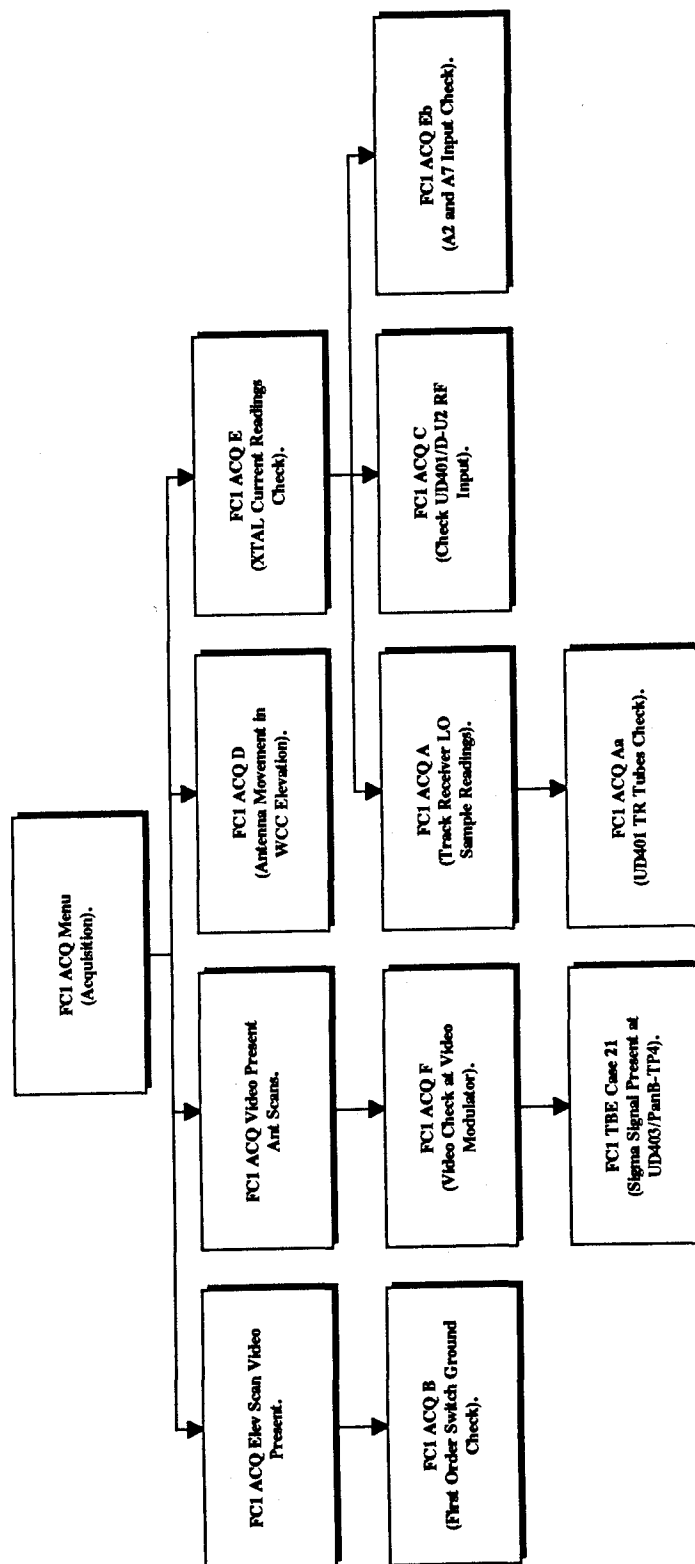


Figure 6: FCI Acquisition Menu

G. FC1 TRACK BEARING AND ELEVATION PROCEDURES

Name: FC1 TBE Menu (Figure 7)

Description: Allows selection of one of three possible track bearing and elevation paths; Nogo PDT mode, NoGo PAT mode or NoGo both modes.

Called by: FC1 TBER Menu

Calls: FC1 TBE NoGo PDT Mode, FC1 TBE NoGo PAT Mode and FC1 TBE NoGo Both Modes

Name: FC1 TBE NoGo PDT Mode

Description: Allows trouble-shooting of FC1 Track Bearing or Elevation in the event that there is a NoGo in PDT mode.

Called by: FC1 TBE Menu

Calls: FC1 TBE C

Name: FC1 TBE C

Description: This procedure is common to all of the three primary Menu selections in FC1 TBE Menu. In all cases it continues trouble-shooting procedures for the appropriate NoGo mode.

Called by: FC1 TBE NoGo PDT Mode

Calls: None

Name: FC1 TBE NoGo in PAT Mode

Description: Allows trouble-shooting of FC1 Track Bearing or Elevation in the event that there is a NoGo in PAT mode.

Called by: FC1 TBE Menu

Calls: FC1 TBE C (described above)

Name: FC1 TBE NoGo Both Modes

Description: Allows trouble-shooting of FC1 Track Bearing and Elevation in the event that there is a NoGo in both PDT and PAT modes. It allows selection of one of three possible paths.

Called by: FC1 TBE Menu

Calls: FC1 TBE Trk Ant Oscillations, FC1 TBE Low XTAL Current and FC1 TBE F

Name: FC1 TBE Low XTAL Current

Description: Allows trouble-shooting of FC1 Track Bearing and Elevation NoGo in both PDT and PAT modes in the event that the crystal current is low.

Called by: FC1 TBE NoGo Both Modes

Calls: FC1 TACQ A

Name: FC1 TACQ A

Description: Continues trouble-shooting of FC1 Track Bearing and Elevation by checking the track receiver LO sample in track acquisition.

Called by: FC1 TBE Low XTAL Current

Calls: FC1 TACQ Aa

Name: FC1 TACQ Aa

Description: Continues trouble-shooting of FC1 Track Bearing and Elevation by checking the TR tube readings at UD401.

Called by: FC1 TACQ A

Calls: None

Name: FC1 TBE Trk Ant Oscillations
Description: Allows trouble-shooting of FC1 Track Bearing and Elevation in the event that there are track antenna oscillations.
Called by: FC1 TBE NoGo Both Modes
Calls: FC1 TBE F

Name: FC1 TBE F
Description: This procedure, and its associated sub-procedures, is a commonly used procedure in FC1 TBE NoGo both modes. It can be selected directly from the FC1 TBE NoGo Both Modes procedure or it can be used to continue trouble-shooting of FC1 Track Bearing and Elevation track antenna oscillations by checking the video level tolerance.
Called by: FC1 TBE Trk Ant Oscillations and FC1 TBE NoGo Both Modes
Calls: FC1 TBE D and FC1 TBE C (described above)

Name: FC1 TBE D
Description: Continues trouble-shooting of FC1 TBE F by checking the continuity of the AGC video level.
Called by: FC1 TBE F
Calls: FC1 TBE Case 2 and FC1 TBE Case 3

Name: FC1 TBE Case 2

Description: This procedure is common in trouble-shooting track antenna oscillations or video level tolerance checks. In both cases, it is used to continue trouble-shooting of video levels in the event that the sigma video level is out of tolerance.

Called by: FC1 TBE F and FC1 TBE D

Calls: FC1 TBE Case 21

Name: FC1 TBE Case 21

Description: Continues trouble-shooting of FC1 TBE Case 2 by checking for the sigma video presence at UD403/PanB-TP4.

Called by: FC1 TBE Case 2

Calls: FC1 TBE E

Name: FC1 TBE E

Description: Continues trouble-shooting of FC1 TBE Case 21 by checking the inputs at UD401/C-B/16.

Called by: FC1 TBE Case 21

Calls: None

Name: FC1 TBE Case 3

Description: Continues trouble-shooting of video levels in the event that the sigma video level is in of tolerance, but the delta video levels are out of tolerance.

Called by: FC1 TBE D

Calls: None

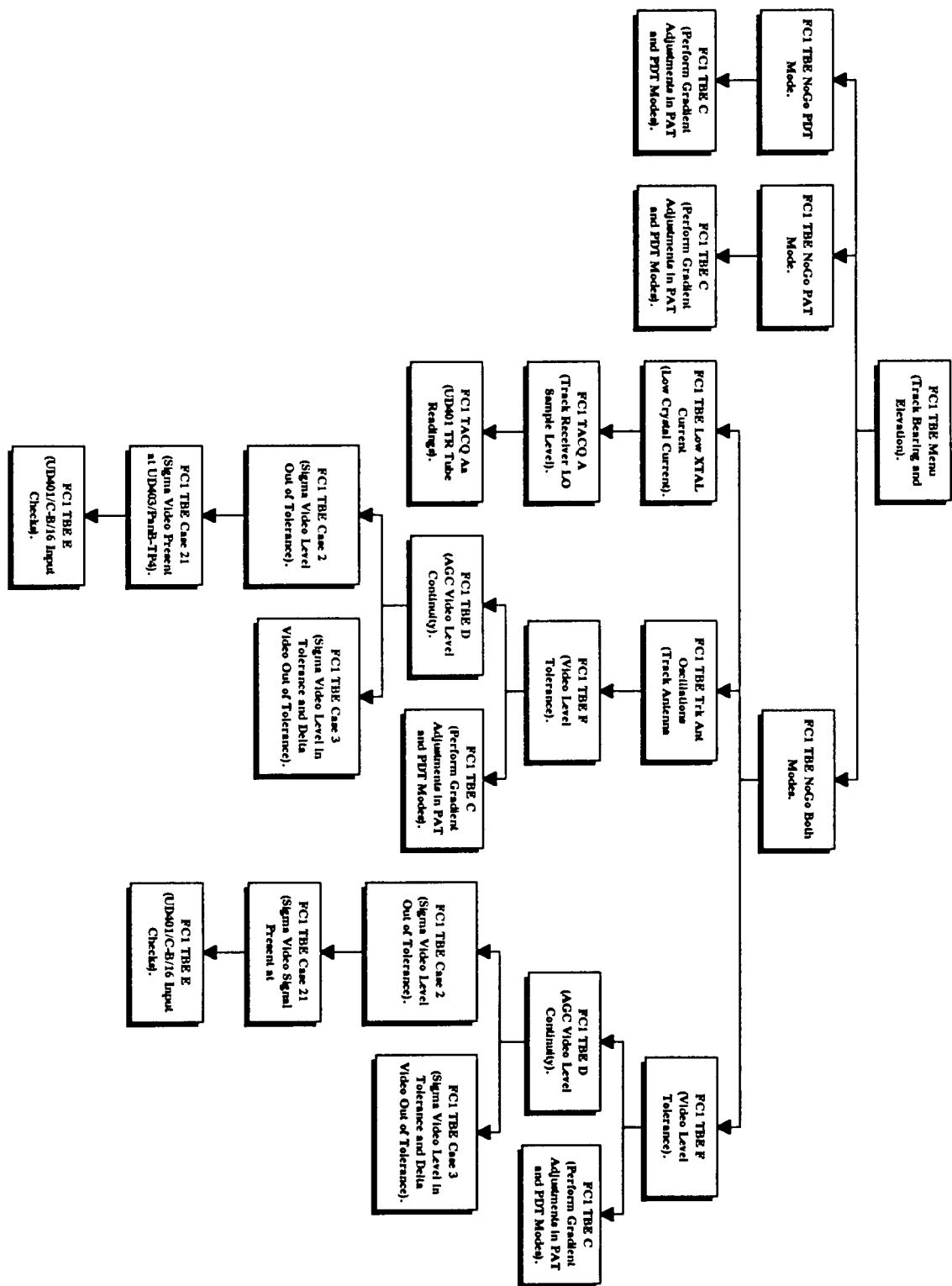


Figure 7: FCI Track Bearing and Elevation Menu

H. FC1 TRACK RANGE PROCEDURES

Name: FC1 TRNG Menu (Figure 8)

Description: Allows trouble-shooting of FC1 Track Range procedures by allowing selection of one of three possible paths: NoGo in PAT mode, NoGo in PDT mode and NoGo in both PAT and PDT modes.

Called by: FC1 TBER Menu

Calls: RNG NoGo PAT Mode Only, RNG NoGo PDT Mode Only and RNG NoGo in Both Modes.

Name: RNG NoGo PAT Mode Only

Description: Allows trouble-shooting of FC1 Track Range in the event that there is a NoGo in PAT mode.

Called by: FC1 TRNG Menu

Calls: FC1 TRNG C

Name: FC1 TRNG C

Description: This procedure is common to all three of the NoGo modules of FC1 Track Range. In this instance, it continues trouble-shooting of a NoGo in PAT Mode via gradient adjustments in PAT and PDT modes.

Called by: RNG NoGo PAT Mode Only, RNG NoGo PDT Mode Only and FC1 TRNG D

Calls: None.

Name: RNG NoGo PDT Mode Only
Description: Allows trouble-shooting of FC1 Track Range in the event that there is a NoGo in PDT mode.
Called by: FC1 TRNG Menu
Calls: FC1 TRNG C (described above)

Name: RNG NoGo Both Modes
Description: Allows trouble-shooting of FC1 Track Range in the event that there is a NoGo in PDT mode.
Called by: FC1 TRNG Menu
Calls: FC1 TRNG D, Trans Micro, RNG Low XTAL Current and Rng Gte Circs

Name: FC1 TRNG D
Description: Continues trouble-shooting of FC1 Track Range NoGo in both modes by checking the sigma video level tolerance.
Called by: RNG NoGo Both Modes
Calls: FC1 TRNG C and FC1 TRNG Sub D

Name: FC1 TRNG C
Description: Continues trouble-shooting of FC1 Track Range NoGo in both modes via gradient adjustments in PAT and PDT modes.
Called by: FC1 TRNG D
Calls: None

Name: FC1 TRNG Sub D

Description: This procedure is common to FC1 TRNG Sub D and Trans Micro. In either event, it continues trouble-shooting of FC1 Track Range NoGo in both modes via checking the sigma LIN video.

Called by: FC1 TRNG Sub D and Trans Micro.

Calls: FC1 TRNG E

Name: FC1 TRNG E

Description: Continues trouble-shooting of FC1 Track Range NoGo in both modes via input checks at UD401/C-B/16.

Called by: FC1 TRNG Sub D

Calls: None

Name: Trans Micro

Description: Continues trouble-shooting of FC1 Track Range via a transmitter or microwave components check and/or replacement.

Called by: RNG NoGo Both Modes

Calls: FC1 TRNG Sub D (described previously)

Name: RNG Low XTAL Current

Description: Allows trouble-shooting of FC1 Track Range NoGo in both modes in the event that there is low crystal current.

Called by: RNG NoGo Both Modes

Calls: None

Name: Rng Gte Circs

Description: Continues trouble-shooting of FC1 Track Range
NoGo in both modes via the range gate circuit
tests.

Called by: RNG NoGo Both Modes

Calls: None

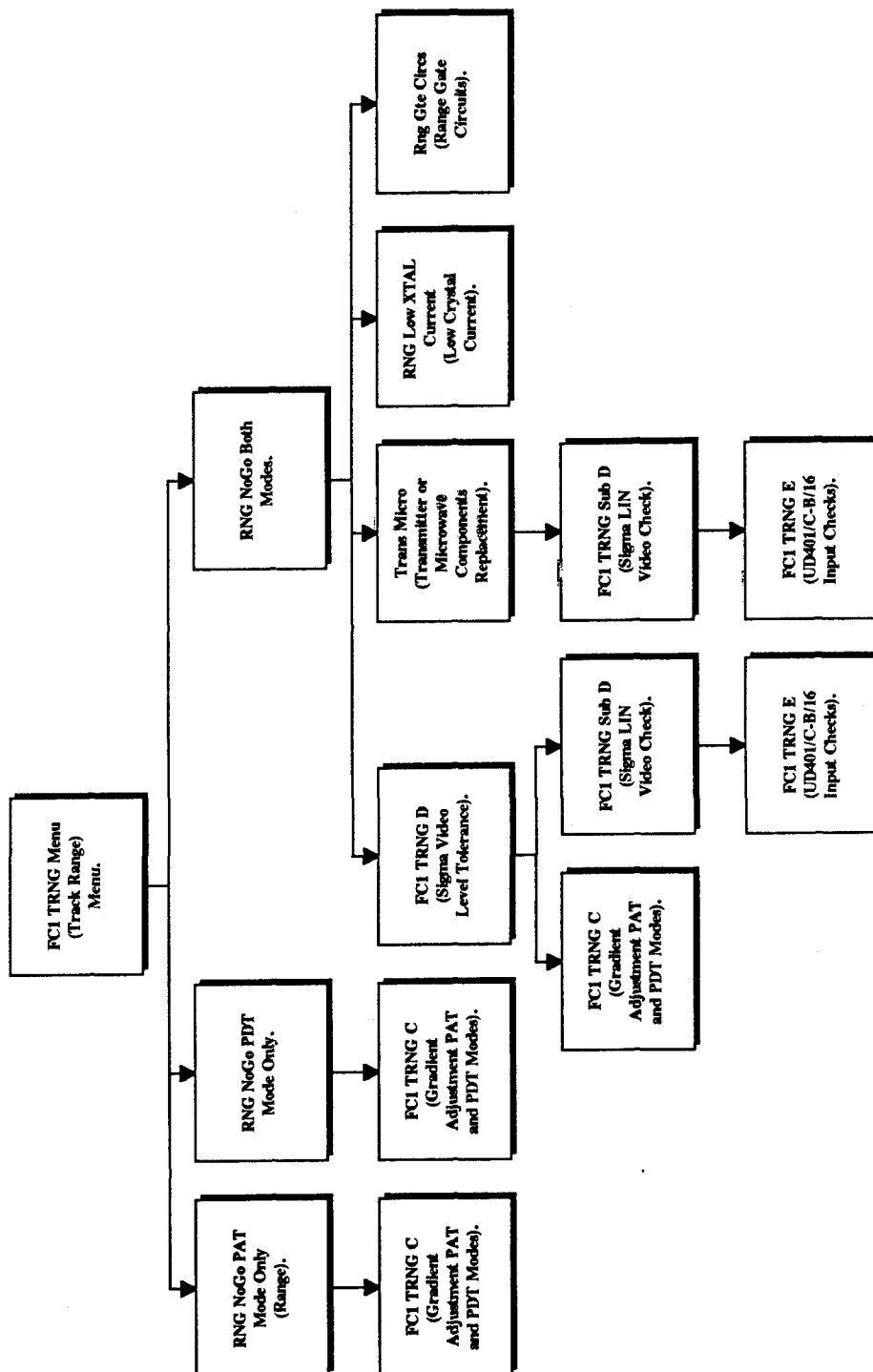


Figure 8: FCI Track Range Menu

I. FC2 PROCEDURES

Name: FC2 Menu (Figure 9)
Description: Allows selection of FC2 Designation, Acquisition and Track procedures.
Called by: Performance Menu
Calls: FC2 DTRB Menu, FC2 ACQ Menu, and FC2 TBER Menu.

Name: FC2 DTRB Menu
Description: Allows selection of FC2 Designation Time, Range and Bearing procedures.
Called by: FC2 Menu
Calls: FC2 DT Menu, FC2 DR and FC2 DB

Name: FC2 ACQ Menu
Description: Allows selection of FC1 ACQ procedures.
Called by: FC2 Menu
Calls: FC2 ACQ Menu

Name: FC2 TBER Menu
Description: Allows selection of FC2 Track Bearing, Elevation and Range procedures.
Called by: FC2 Menu
Calls: FC2 TBE Menu and FC2 TR Menu

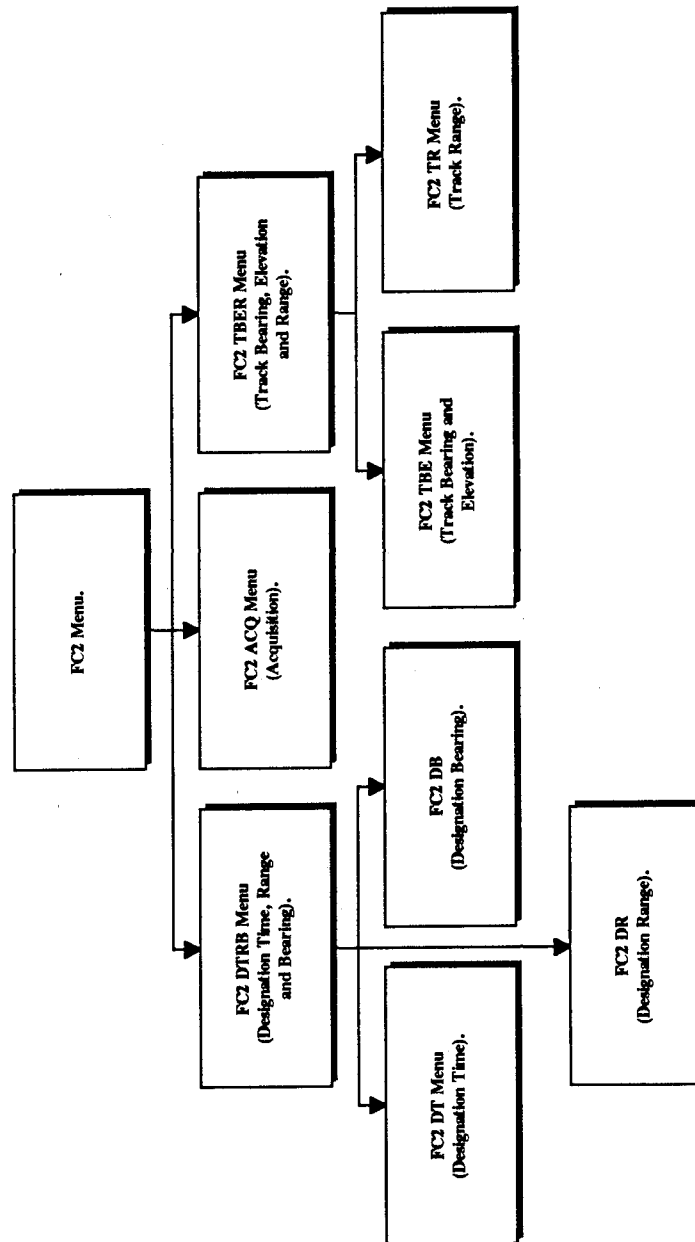


Figure 9: FC2 Menu

J. FC2 DESIGNATION TIME PROCEDURES

Name: FC2 DT Menu (Figure 10)

Description: Allows selection of one of three possible paths; Submenu, range counter LED check and range gate check.

Called by: FC2 DTRB Menu

Calls: FC2 DT Submenu, FC2 DT G and FC2 DT Case 4

Name: FC2 DT Submenu

Description: Allows selection of one of five possible paths; train velocity checks, no range gate movement, range gate movement slow, no track antenna movement, or track antenna movement is slow.

Called by: FC2 DT Menu

Calls: FC2 DT F, FC2 DT No Rng Gte Mvmt, FC2 DT Rng Gte Mvmt Slow, FC2 DT No Trk Ant Mvmt, and FC2 DT Trk Ant Mvmt Slow

Name: FC2 DT F

Description: Continues trouble-shooting for FC2 Designation Time via a train velocity check.

Called by: FC2 DT Submenu

Calls: FC2 DT Fa and FC2 DT Fb

Name: FC2 DT Fa

Description: Continues trouble-shooting of FC2 Designation Time via a check on the Gyro temp low lamp.

Called by: FC2 DT F

Calls: FC2 DT Fc

Name: FC2 DT Fc

Description: This procedure is common to the FC2 DT No Rng Gte Mvmt and FC2 DT No Trk Ant Mvmt. In either case, it continues trouble-shooting of FC2 Designation Time via a DC voltage check at UD417.

Called by: FC2 DT Fa and FC2 DT B

Calls: None

Name: FC2 DT Fb

Description: Continues trouble-shooting of FC2 Designation Time via a DC voltage level check at TP2.

Called by: FC2 DT F

Calls: None

Name: FC2 DT No Rng Gte Mvmt

Description: Allows trouble-shooting of FC2 Designation Time in the event that there is no movement in the range gate.

Called by: FC2 DT Submenu

Calls: None

Name: FC2 DT Rng Gte Mvmt Slow

Description: This procedure is common to FC2 DT Submenu and FC2 DT Case 4. In either case, it allows trouble-shooting of FC2 Designation Time in the event that the movement of the range gates are slow.

Called by: FC2 DT Submenu and FC2 DT Case 4

Calls: None

Name: FC2 DT No Trk Ant Mvmt

Description: Allows trouble-shooting of FC2 Designation Time in the event that there is no track antenna movement. In addition, it allows the selection of one of five possible paths each performing a different type of system check.

Called by: FC2 DT Submenu

Calls: FC2 DT A, FC2 DT B, FC2 DT C, FC2 DT D and FC2 DT E

Name: FC2 DT A

Description: Continues trouble-shooting of FC2 Designation Time no track antenna movement via a check on the OPERATE lamp.

Called by: FC2 DT No Trk Ant Mvmt

Calls: FC2 DT Aa

Name: FC2 DT B

Description: Continues trouble-shooting of FC2 Designation Time no track antenna movement via a check of the antenna in bearing in simulate mode.

Called by: FC2 DT No Trk Ant Mvmt

Calls: FC2 DT Ba and FC2 DT Fc

Name: FC2 DT C

Description: Continues trouble-shooting of FC2 Designation Time no track antenna movement via a fuse check.

Called by: FC2 DT No Trk Ant Mvmt

Calls: None

Name: FC2 DT D
Description: Continues trouble-shooting of FC2 Designation
Time no track antenna movement via a train
brake PD fuse check.
Called by: FC2 DT No Trk Ant Mvmt
Calls: None

Name: FC2 DT E
Description: Continues trouble-shooting of FC2 Designation
Time no track antenna movement via a train
FIELD lamp check.
Called by: FC2 DT No Trk Ant Mvmt
Calls: None

Name: FC2 DT G
Description: Continues trouble-shooting of FC2 Designation
Time via a range counter LED check.
Called by: FC2 DT Menu
Calls: FC2 DT Ga

Name: FC2 DT Ga
Description: Continues trouble-shooting of FC2 Designation
Time via a UD481/PanT LED check.
Called by: FC2 DT G
Calls: None

Name: FC2 DT Case 4

Description: Continues trouble-shooting of FC2 Designation Time via system reset.

Called by: FC2 DT Menu

Calls: FC2 DT Rng Gte Slow (described previously)

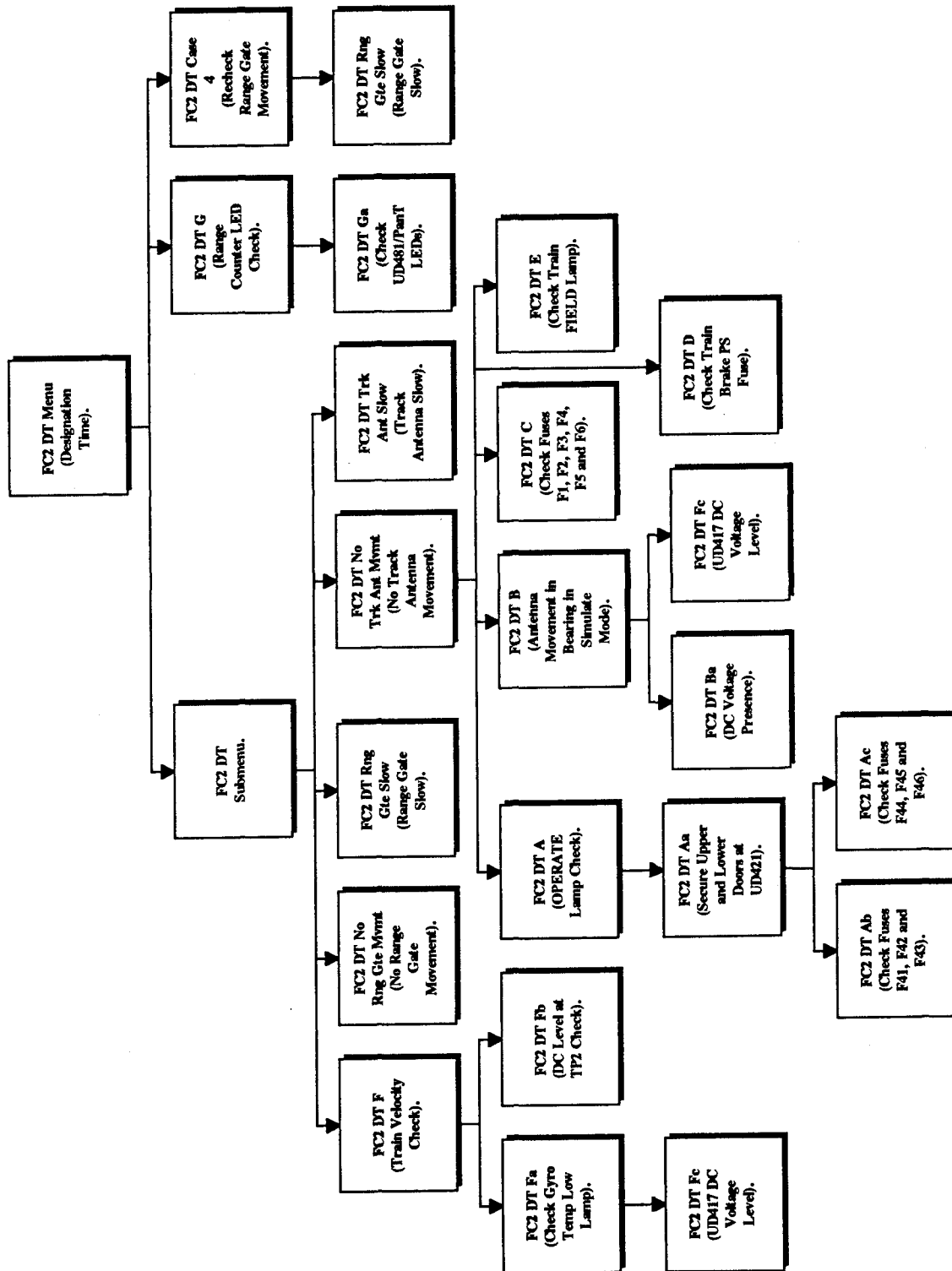


Figure 10: FC2 Designation Time Menu

K. FC2 DESIGNATION RANGE PROCEDURES

Name: FC2 DR A (Figure 11)

Description: Allows trouble-shooting of problems associated with FC2 Designation Range.

Called by: FC2 DTRB Menu

Calls: FC2 DR B

Name: FC2 DR B

Description: This procedure is common to and a continuance of both FC1 designation range and FC2 designation bearing trouble-shooting procedures. In this instance, it continues the trouble-shooting procedures for FC2 Designation Range.

Called by: FC2 DR A and FC2 DB

Calls: None

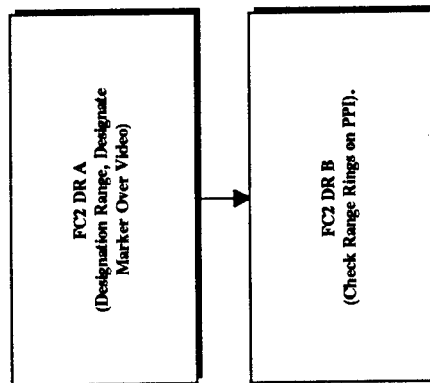


Figure 11: FC2 Designation Range A

L. FC2 DESIGNATION BEARING PROCEDURES

Name: FC2 DB (Figure 12)

Description: Allows trouble-shooting of the FCS designation bearing in FC2.

Called by: FC2 DTRB Menu

Calls: FC2 DR B

Name: FC2 DR B

Description: This procedure is common to and a continuance of both FC1 designation range and FC1 designation bearing trouble-shooting procedures. In this instance, it continues the trouble-shooting procedures for FC1 Designation Bearing.

Called by: FC2 DR A and FC2 DB

Calls: None

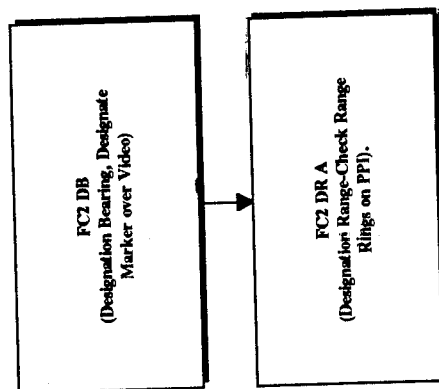


Figure 12: FC2 Designation Bearing

M. FC2 ACQUISITION PROCEDURES

Name: FC2 ACQ Menu (Figure 13)

Description: Allows selection one of four possible trouble-shooting paths; antenna movement in WCC elevation mode, low crystal current readings, weak or no video to detection circuits and acquisition detection at threshold level.

Called by: FC2 Menu

Calls: FC2 ACQ A, FC2 ACQ E, FC2 ACQ Ed, and FC2 ACQ F

Name: FC2 ACQ A

Description: Continues trouble-shooting of FC2 Acquisition procedures in the event that there is no antenna movement in elevation in WCC (manual mode).

Called by: FC2 ACQ Menu

Calls: FC2 ACQ Aa, FC2 ACQ Ac, and FC2 ACQ Ad

Name: FC2 ACQ Aa

Description: Continues trouble-shooting of FC2 Acquisition via a check on the Servo Failure lamp at UD 421.

Called by: FC2 ACQ A

Calls: FC2 ACQ Ab

Name: FC2 ACQ Ab
Description: Continues trouble-shooting of FC2 Acquisition via a check on the elevation brake PS fuse F20.
Called by: FC2 ACQ Aa
Calls: None

Name: FC2 ACQ Ac
Description: Continues trouble-shooting of FC2 Acquisition via a check of the presence of DC voltage at TP5 and TP6.
Called by: FC2 ACQ A
Calls: None

Name: FC2 ACQ Ad
Description: Continues trouble-shooting of FC2 Acquisition via a check of the presence of DC voltage at TP6 and TP7.
Called by: FC2 ACQ A
Calls: None

Name: FC2 ACQ E
Description: Continues trouble-shooting of FC2 Acquisition in the event that the crystal current is low.
Called by: FC2 ACQ Menu
Calls: FC2 ACQ B and FC2 ACQ Ed

Name: FC2 ACQ B
Description: Continues trouble-shooting of FC2 Acquisition via a check on the track receiver LO sample.
Called by: FC2 ACQ E
Calls: FC2 ACQ Ba

Name: FC2 ACQ Ba
Description: Continues trouble-shooting of FC2 Acquisition via a check for reflector voltage.
Called by: FC2 ACQ B
Calls: None

Name: FC2 ACQ Ed
Description: This procedure, and its subordinates, are common to the FC2 ACQ Menu and FC2 ACQ E. In both cases, it continues trouble-shooting of FC2 Acquisition in the event that there is weak or no video to the detection circuits.
Called by: FC2 ACQ E and FC2 ACQ Menu
Calls: FC2 ACQ C

Name: FC2 ACQ C
Description: Continues trouble-shooting of FC2 Acquisition via a check for RF input at UD417/A9-U1.
Called by: FC2 ACQ Ed
Calls: None

Name: FC2 ACQ F
Description: Continues trouble-shooting of FC2 Acquisition via a check of the acquisition detection threshold level.
Called by: FC2 ACQ Menu
Calls: FC2 TBE Case 2 Submenu

Name: FC2 TBE Case 2 Submenu
Description: Procedures utilized in the trouble-shooting of FC2 Track Bearing and Elevation are also used in order to continue trouble-shooting of FC2 Acquisition.
Called by: FC2 ACQ F
Calls: FC2 TBE Case 21

Name: FC2 TBE Case 21
Description: Procedures utilized in the trouble-shooting of FC2 Track Bearing and Elevation are also used in order to continue trouble-shooting of FC2 Acquisition.
Called by: FC2 TBE Case 2 Submenu
Calls: FC2 TBE Case 21

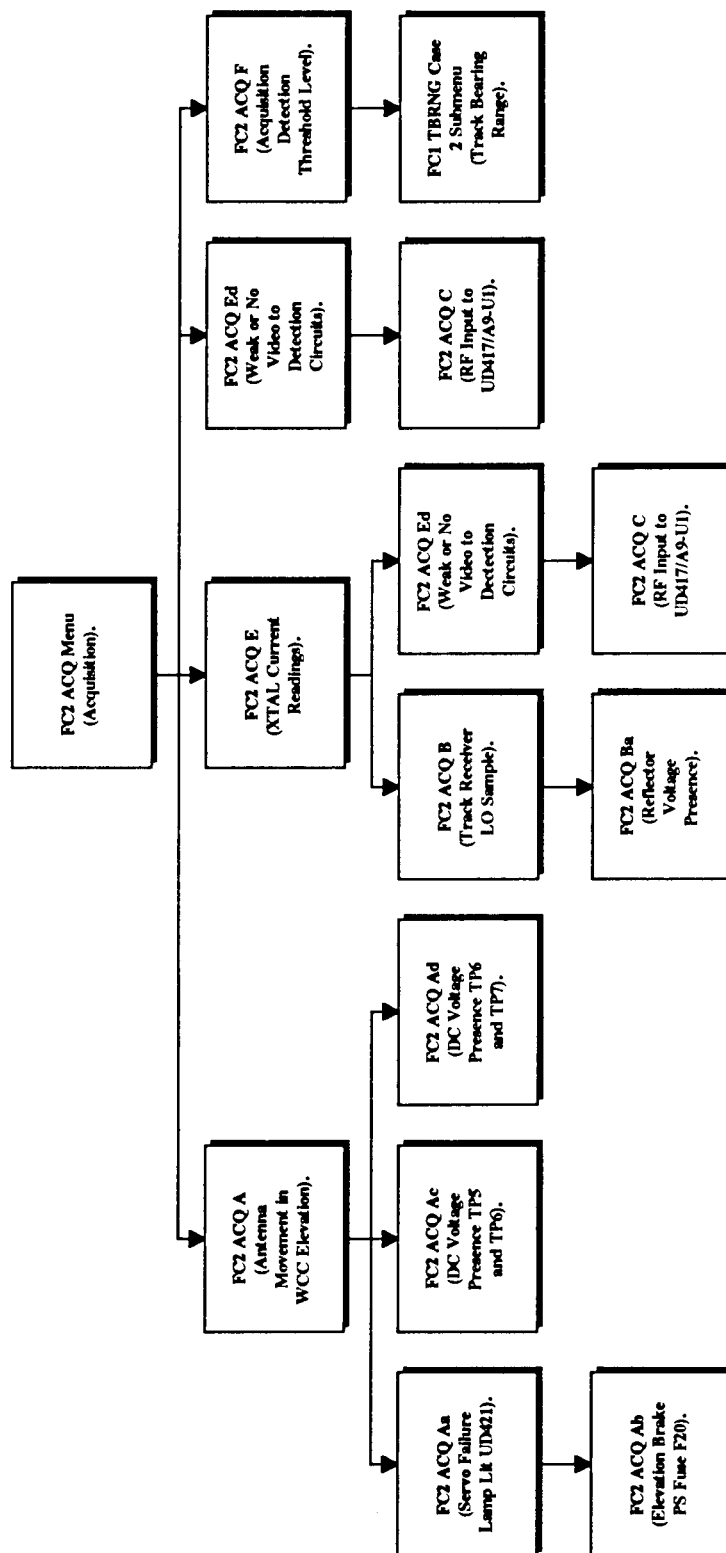


Figure 13: FC2 Acquisition Menu

N. FC2 TRACK BEARING AND ELEVATION PROCEDURES

Name: FC2 TBE Menu (Figure 14)

Description: Allows selection of one of three possible track bearing and elevation paths; Nogo PDT mode, NoGo PAT mode or NoGo both modes.

Called by: FC2 TBER Menu

Calls: FC2 TBE NoGo PDT Mode, FC2 TBE NoGo PAT Mode and FC2 TBE NoGo Both Modes

Name: FC2 TBE NoGo PDT Mode

Description: Allows trouble-shooting of FC2 Track Bearing or Elevation in the event that there is a NoGo in PDT mode.

Called by: FC2 TBE Menu

Calls: FC2 TBE C

Name: FC2 TBE C

Description: This procedure is common to all of the three primary Menu selections in FC2 TBE Menu. In all cases it continues trouble-shooting procedures for the appropriate NoGo mode.

Called by: FC2 TBE NoGo PDT Mode

Calls: None

Name: FC2 TBE NoGo in PAT Mode

Description: Allows trouble-shooting of FC2 Track Bearing or Elevation in the event that there is a NoGo in PAT mode.

Called by: FC2 TBE Menu

Calls: FC2 TBE C (described above)

Name: FC2 TBE NoGo Both Modes

Description: Allows trouble-shooting of FC2 Track Bearing and Elevation in the event that there is a NoGo in both PDT and PAT modes. It allows selection of one of three possible paths.

Called by: FC2 TBE Menu

Calls: FC2 TBE Trk Ant Oscillations, FC2 TBE Low XTAL Current and FC2 TBE F

Name: FC2 TBE Low XTAL Current

Description: Allows trouble-shooting of FC2 Track Bearing and Elevation NoGo in both PDT and PAT modes in the event that the crystal current is low.

Called by: FC2 TBE NoGo Both Modes

Calls: FC2 TBE C (described previously)

Name: FC2 TBE Trk Ant Oscillations

Description: Allows trouble-shooting of FC2 Track Bearing and Elevation in the event that there are track antenna oscillations.

Called by: FC2 TBE NoGo Both Modes

Calls: FC2 TBE F

Name: FC2 TBE F

Description: This procedure, and its associated sub-procedures, is a commonly used procedure in FC2 TBE NoGo both modes. It can be selected directly from the FC2 TBE NoGo Both Modes procedure or it can be used to continue trouble-shooting of FC2 Track Bearing and Elevation track antenna oscillations by checking the video levels.

Called by: FC2 TBE Trk Ant Oscillations and FC2 TBE NoGo Both Modes

Calls: FC2 TBE D and FC2 TBE C (described above)

Name: FC2 TBE D

Description: Continues trouble-shooting of FC2 TBE F by checking the video levels.

Called by: FC2 TBE F

Calls: FC2 TBE Case 2 and FC2 TBE Case 3

Name: FC2 TBE Case 2

Description: This procedure is common in trouble-shooting track antenna oscillations or video level tolerance checks. In both cases, it is used to continue trouble-shooting of video levels in the event that the sigma video level is out of tolerance.

Called by: FC2 TBE F and FC2 TBE D

Calls: FC2 TBE Case Submenu

Name: FC2 TBE Submenu
Description: Continues trouble-shooting procedures for a NoGo in both modes via a sigma video level checks.
Called by: FC2 TBE Case 2
Calls: FC2 TBE Case 21

Name: FC2 TBE Case 21
Description: Continues trouble-shooting of FC2 TBE Case 2 via a IF Phase adjustment.
Called by: FC2 TBE Case 2 Submenu
Calls: FC2 TBE E

Name: FC2 TBE Case 3
Description: Continues trouble-shooting of video levels in the event that the sigma video level is in of tolerance, but the delta video levels are out of tolerance.
Called by: FC2 TBE D
Calls: FC2 TBE Case 31

Name: FC2 TBE Case 31
Description: Continues trouble-shooting of video levels via additional IF Phase adjustments.
Called by: FC2 TBE Case 3
Calls: None

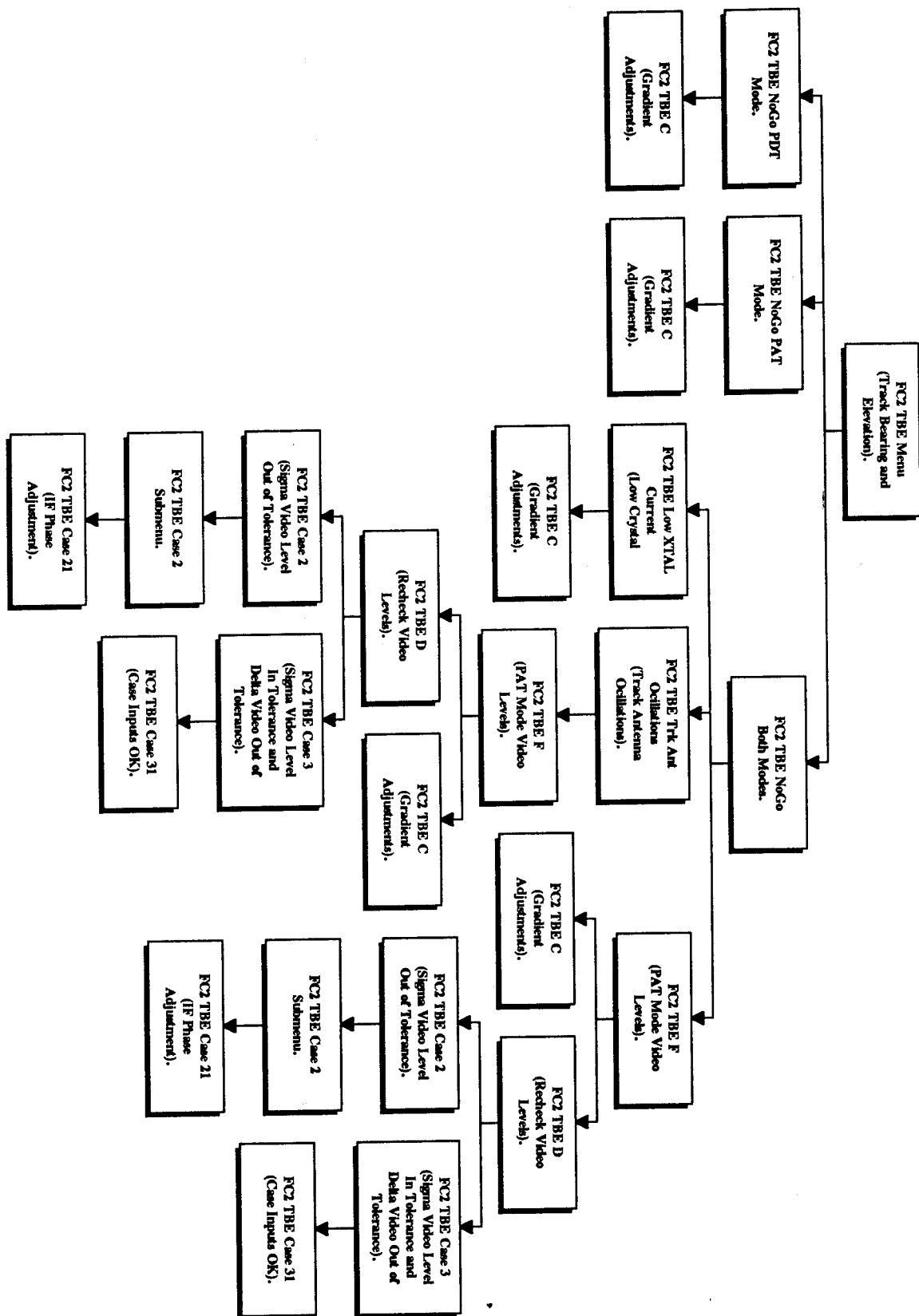


Figure 14: FC2 Track Bearing and Elevation Menu

O. FC2 TRACK RANGE PROCEDURES

Name: FC2 TRNG Menu (Figure 15)

Description: Allows trouble-shooting of FC2 Track Range procedures by allowing selection of one of three possible paths: NoGo in PAT mode, NoGo in PDT mode and NoGo in both PAT and PDT modes.

Called by: FC2 TBER Menu

Calls: FC2 TRNG NoGo PAT Mode Only, FC2 TRNG NoGo PDT Mode Only and FC2 TRNG NoGo in Both Modes.

Name: FC2 TRNG NoGo PAT Mode Only

Description: Allows trouble-shooting of FC2 Track Range in the event that there is a NoGo in PAT mode.

Called by: FC2 TRNG Menu

Calls: FC2 TRNG C

Name: FC2 TRNG C

Description: This procedure is common to all three of the NoGo modules of FC2 Track Range. In this instance, it continues trouble-shooting of a NoGo in PAT Mode via gradient adjustments.

Called by: FC2 TRNG NoGo PAT Mode Only, FC2 TRNG NoGo PDT Mode Only and FC2 TRNG D

Calls: None.

Name: FC2 TRNG NoGo PDT Mode Only
Description: Allows trouble-shooting of FC2 Track Range in the event that there is a NoGo in PDT mode.
Called by: FC2 TRNG Menu
Calls: FC2 TRNG C (described above)

Name: FC2 TRNG NoGo Both Modes
Description: Allows trouble-shooting of FC2 Track Range in the event that there is a NoGo in PDT mode. In addition, it allows selection of one of four possible trouble-shooting paths.
Called by: FC2 TRNG Menu
Calls: FC2 TRNG Low XTAL Current, FC2 TRNG Gte Circs, FC2 TRNG Trans Micro and FC2 TRNG F

Name: FC2 TRNG Low XTAL Current
Description: Continues trouble-shooting of a NoGo in both modes in the event that there is low crystal current.
Called by: FC2 TRNG NoGo Both Modes.
Calls: None

Name: FC2 TRNG Gte Circs
Description: Continues trouble-shooting of a NoGo in both modes via range gate servo-jump tests.
Called by: FC2 TRNG NoGo Both Modes.
Calls: None

Name: FC2 TRNG Trans Micro

Description: Continues trouble-shooting of a NoGo in both modes via a transmitter and/or microwave test.

Called by: FC2 TRNG NoGo Both Modes

Calls: FC2 TRNG F

Name: FC1 TRNG F

Description: This procedure, and its associated sub-procedures can be called directly from the FC2 TRNG NoGo Both Modes procedure or called as a result of a check on the transmitter and/or microwave components. In either event, it continues trouble-shooting of a NoGo in both modes via a check of the video levels.

Called by: FC1 TRNG NoGo Both Modes and FC2 TRNG Trans Micro

Calls: FC1 TRNG C (described previously) and FC2 TRNG D

Name: FC2 TRNG D

Description: Continues trouble shooting of a NoGo both modes via a recheck of the video levels.

Called by: FC2 TRNG F

Calls: FC2 TRNG Case 2

Name: FC2 TRNG Case 2
Description: Continues trouble-shooting procedures of a NoGo in both modes via a sigma video level tolerance check.
Called by: FC2 TRNG D
Calls: FC2 TRNG Case 21

Name: FC2 TRNG Case 21
Description: Continues trouble-shooting procedures of a NoGo in both modes by checking sigma video level tolerance in conjunction with IF Phase adjustments.
Called by: FC2 TRNG Case 2
Calls: None

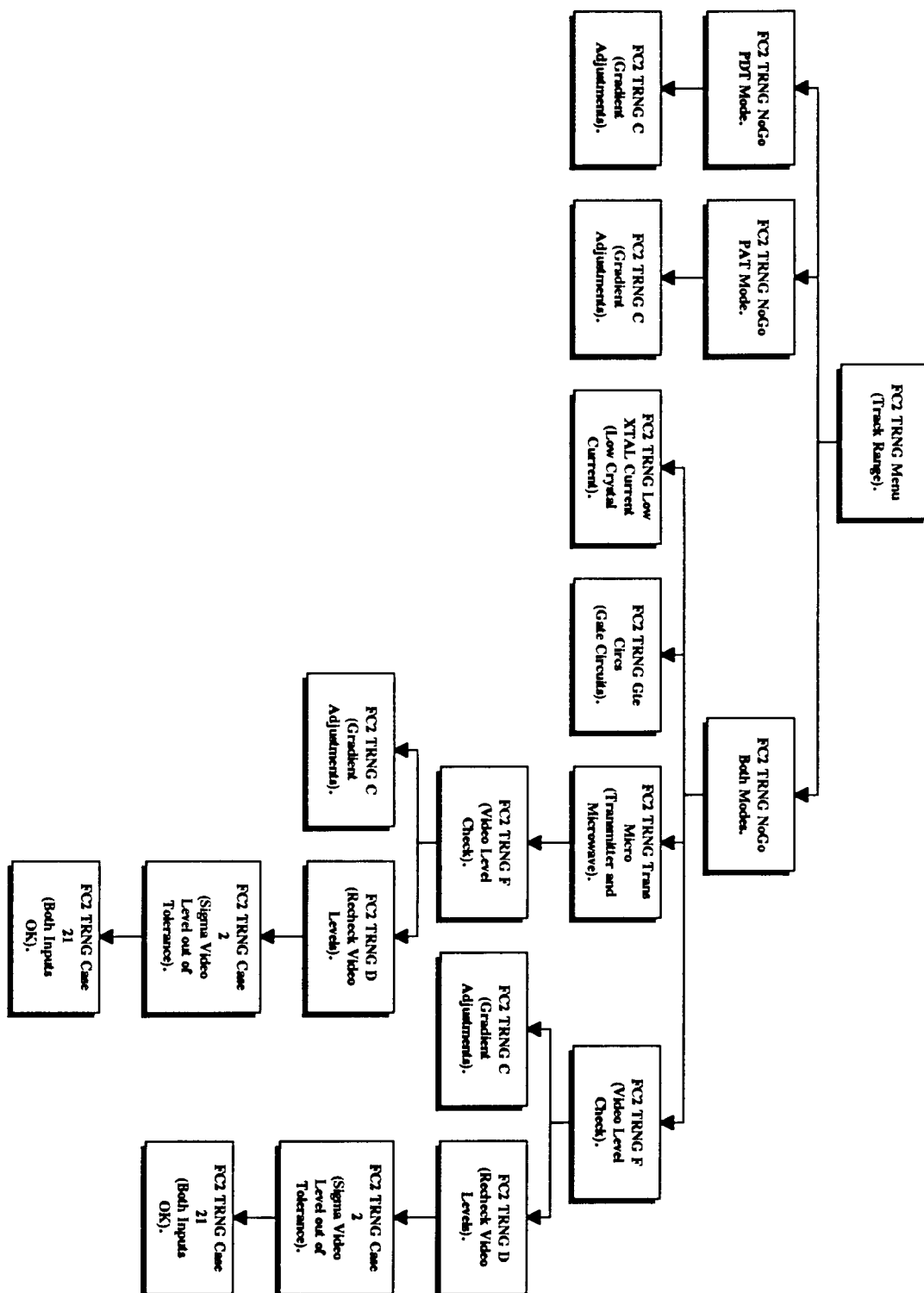


Figure 15: FC2 Track Range Menu

P. FC4 AND FC5 PROCEDURES

Name: FC4 and 5 Menu (Figure 16)

Description: Allows selection of FC4 and FC5 trouble-shooting procedures for designation time, track bearing and track range.

Called by: Performance Menu

Calls: FC4and5 DT Menu, FC4and5 TR and FC4and5 TB

Name: FC4and5 DT Menu

Description: Allows selection of one of three possible designation time trouble-shooting procedures.

Called by: FC4and5 Menu

Calls: FC4and5 DT, FC4 DT Only and FC5 DT Only

Name: FC4and5 DT

Description: Continues trouble-shooting procedures for FC4 and FC5 Designation Time in the event of a NoGo.

Called by: FC4and5 DT Menu

Calls: None

Name: FC4 DT Only

Description: Continues trouble-shooting procedures for FC4 Designation Time in the event of a NoGo.

Called by: FC4and5 DT Menu

Calls: None

Name: FC5 DT Only
Description: Continues trouble-shooting procedures for FC5 Designation Time in the event of a NoGo.
Called by: FC4and5 DT Menu
Calls: None

Name: FC4and5 TR
Description: Continues trouble-shooting procedures for FC4 and FC5 Track Range in the event of a NoGo.
Called by: FC4and5 Menu
Calls: None

Name: FC4and5 TB
Description: Continues trouble-shooting procedures for FC4 and FC5 Track Bearing in the event of a NoGo.
Called by: FC4and5 Menu
Calls: None

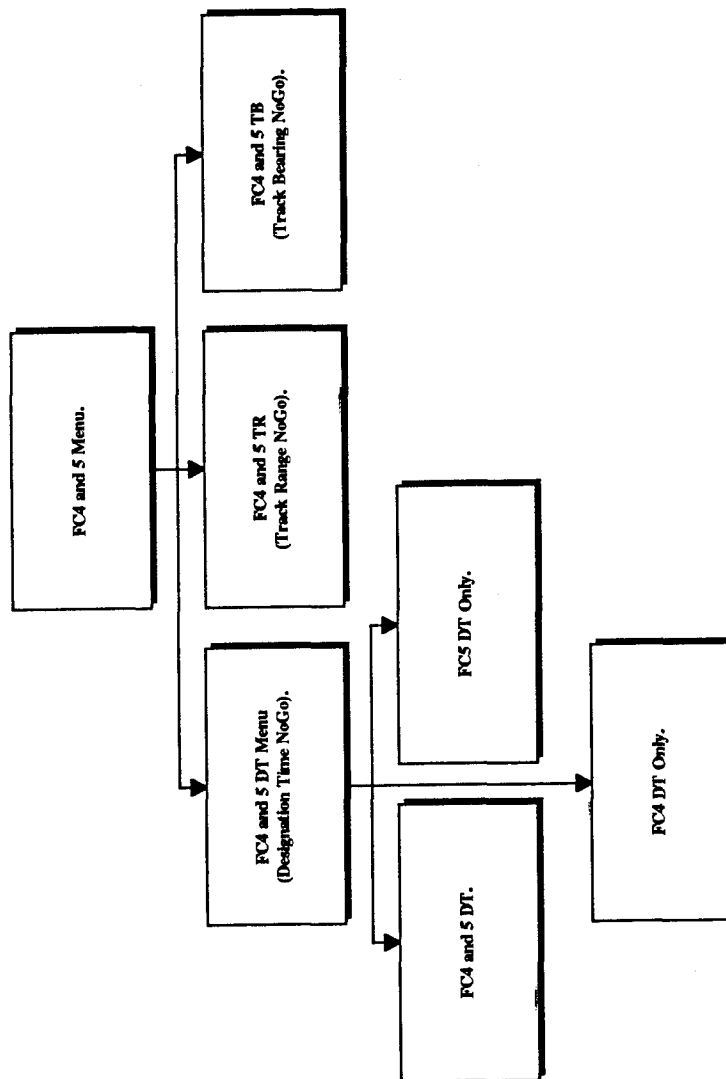


Figure 16: FC4 and 5 Menu

LIST OF REFERENCES

- Aktas, A.Z., *Structured Analysis & Design of Information Systems*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1987.
- Crawford, J.N., *Design and Implementation of a Prototype Database System in Conjunction with the Maintenance Advisor Expert System for the MK92 Fire Control System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.
- Dills, K.R. and Tutt, T.F., *Verification and Validation of the MK92 MOD 2 Fire Control System Maintenance Advisor Expert System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.
- Geick, D.M. and Mikler, S.E., *Design and Implementation of the Calibration Module of the MK 92 Prototype Maintenance Advisor Expert System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1994.
- Himes, A., and Sperry, S., *Symbolic Adept Reference*, Version 2.1, Symbolic Corporation, 1991.
- Lewis, Clinton Dean, *Development of a Maintenance Advisor Expert System for the MK 92 MOD 2 Fire Control System: FC-1 Designation - Time, FC-1 Track - Bearing, Elevation and Range, and FC-2 Track - Bearing, Elevation and Range*. Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.
- McGaha J.L., *Implementation of the Production Version of the Performance and Calibration Modules of the MK92 MOD 2 Fire Control System Maintenance Advisor Expert System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.
- Meisch, P.J., *Applying Multimedia to the MK92 MOD 2 Maintenance Advisor Expert System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1994.
- The New Merriam-Webster Dictionary*, Merriam-Webster Inc., Springfield, Massachusetts, 1989.
- Page-Jones, Meilir, *The Practical Guide to Structured Systems Design*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1988.

Powell, Steven H., *Economic Analysis of the MK 92 MOD 2 Maintenance Advisor Expert System*, Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

Prerau, D.S., *Developing and Managing Expert Systems: Proven Techniques for Business and Industry*, Addison-Wesley Publishing Company Inc., Menlo Park, California, 1990.

Savitch, W.J. and Petersen, C.G., *Ada; An Introduction to the Art and Science of Programming*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1992.

Smith, C.D., *Development of a Maintenance Advisor Expert System for the MK 92 MOD 2 Fire Control System: FC-1 Designation - Time, Range, Bearing; FC-1 Acquisition; FC-1 Track - Range, Bearing; FC-2 Designation - Time, Range, Bearing, FC-2 Acquisition, FC-2 Track - Range, Bearing; and FC-4 and FC-5*. Master's Thesis, Naval Postgraduate School, Monterey, California, September 1993.

Tally, S. ., *Design and Implementation of a Prototype Database for Part Information to Support the MK92 Fire Control System Maintenance Advisor Expert System*, Master's Thesis, Naval Postgraduate School, Monterey, California, March 1994.

Whitten J.L., Bentley L.D., and Barlow V.M., *Systems Analysis & Design Methods*, Richard D. Irwin, Inc., Homewood, IL; Boston, MA, 1989.

Whitten J.L., Bentley L.D., and Barlow V.M., *Systems Analysis & Design Methods*, Richard D. Irwin, Inc., Burr Ridge, IL; Boston, MA; Sydney, Australia 1994.

BIBLIOGRAPHY

Kamel, M.N. and McCaffrey, M.J., *Engineering Development Model FCS MK 92 Maintenance Advisor Expert System*, Research project, to be published.

MRC 4820 D-1; FC2 MK92 MOD2 Maintenance Requirement Card.

SW271-C2-MMO-060/(U) MK92 MOD 2; Maintenance Manual for Fire Control System MK92 MOD2 (Trouble Isolation).

Yourdon, Edward, *Techniques of Program Structure and Design*, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1975.

INITIAL DISTRIBUTION LIST

- | | |
|--|---|
| 1. Defense Technical Information Center
Cameron Station
Alexandria, VA. 22304-6145 | 2 |
| 2. Library, Code 52
Naval Postgraduate School
Monterey, CA. 93943-5002 | 2 |
| 3. Naval Sea Systems Command, Code 91W3
2531 Jefferson Davis Hwy.
Washington, D.C. 22242-51603 | 1 |
| 4. Naval Sea Systems Command, Code 91W3
2531 Jefferson Davis Hwy.
Washington D.C. 22242-51603
Attn: Ed McGill | 1 |
| 5. Naval Sea Systems Command, Code 91W3
2531 Jefferson Davis Hwy.
Washington D.C. 22242-51603
Attn: FCC Stein | 1 |
| 6. Commander, Code 4K32
Naval Surface Warfare Center
Port Hueneme Division
4363 Missile Way
Port Hueneme, CA. 93043-4307
Attn: Henry Seto | 3 |
| 7. Professor Magdi Kamel, Code SM/Ka
Naval Postgraduate School
Monterey, CA. 93943-5000 | 2 |
| 8. Professor Martin J. McCaffrey, Code SM/Mf
Naval Postgraduate School
Monterey, CA. 93943-5000 | 2 |
| 9. LT Lucy M. Smith, USNR
6992 Old Brentford Rd
Alexandria, VA. 22310 | 1 |