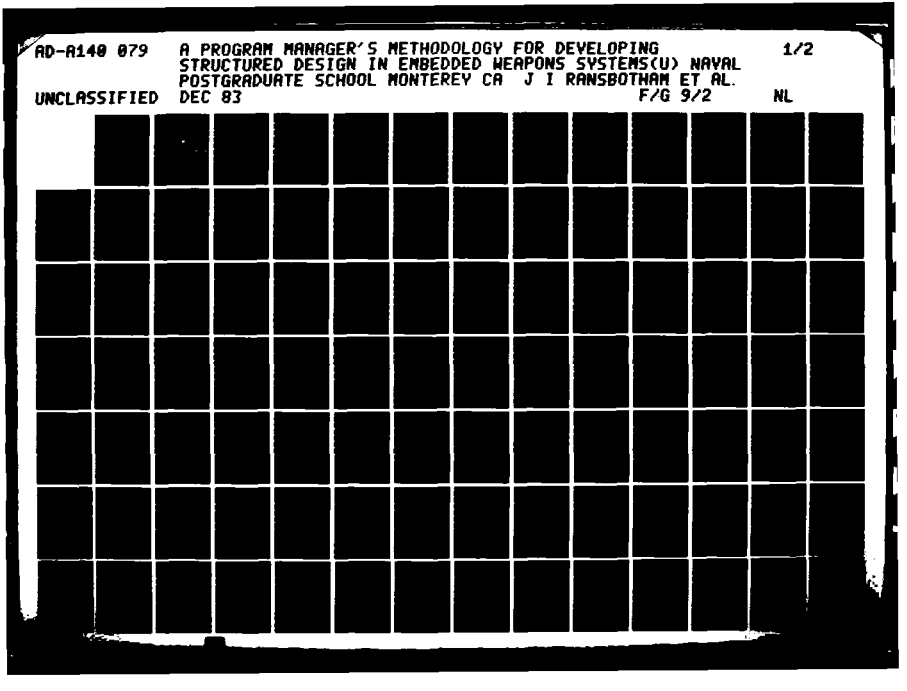
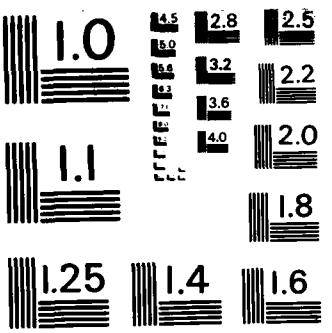
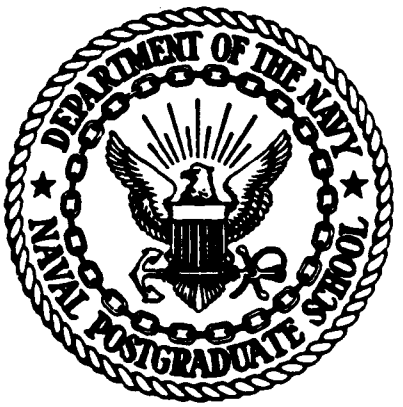

AD-A140 079
A PROGRAM MANAGER'S METHODOLOGY FOR DEVELOPING STRUCTURED DESIGN IN EMBEDDED WEAPONS SYSTEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA J I RANSBOTHAM ET AL.
1/2
UNCLASSIFIED
DEC 83
F/G 9/2
NL



NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

A PROGRAM MANAGER'S METHODOLOGY
FOR DEVELOPING STRUCTURED DESIGN
IN EMBEDDED WEAPONS SYSTEMS

by

James I. Ransbotham, Jr.
and
Donald F. Moorehead, Jr.

December 1983

Thesis Advisor: Ronald W. Modes

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A140 079	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Program Manager's Methodology for Developing Structured Design in Embedded Weapons Systems		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis December 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) James I. Ransbotham, Jr. and Donald F. Moorehead, Jr.		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93943		12. REPORT DATE December, 1983
		13. NUMBER OF PAGES 143
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited

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

18. SUPPLEMENTARY NOTES

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

Methodology, Embedded Weapons Systems, Structured Design

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

This thesis demonstrates a methodology to be used by a Program Manager to allow him to procedurally monitor the design development of an embedded weapons system. The methodology consists of a unique combination of several software engineering strategies integrated to form a powerful management tool. The primary objective of the methodology is to provide an algorithmic procedure which stresses simplicity (Continued)

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

S/N 0102-LF-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ABSTRACT (Continued)

at all levels of abstraction. Further, the system must be capable of generating good system specifications, good documentation, and fully understandable products. Sample products from the implementation of the methodology on the HARPOON Shiboard Command-Launch Control Set (HSCLCS) are provided for illustrative purposes.

Approved for public release; distribution unlimited.

A Program Manager's Methodology for Developing Structured Design in Embedded Weapons Systems

by

James I. Ransbotham, Jr.
Lieutenant Commander, United States Navy
B.S., Georgia Institute of Technology, 1972

and

Donald F. Moorehead, Jr.
Lieutenant Commander, United States Navy
B.S., U.S. Naval Academy, 1975

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1983

Authors: James I. Ransbotham

Donald F. Moorehead

Approved by: Ronald W. Hobbs
Thesis Advisor

Ronald B. Kurtz
Second Reader

Dennis K. Hooper
Chairman, Department of Computer Science

K. T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

This thesis demonstrates a methodology to be used by a Program Manager to allow him to procedurally monitor the design development of an embedded weapons system. The methodology consists of a unique combination of several software engineering strategies integrated to form a powerful management tool. The primary objective of the methodology is to provide an algorithmic procedure which stresses simplicity at all levels of abstraction. Further, the system must be capable of generating good system specifications, good documentation, and fully understandable products. Sample products from the implementation of the methodology on the HARPOON Shipboard Command-Launch Control Set (HSCLCS) are provided for illustrative purposes.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGMENTS

The authors wish to thank the following people. LCDR Ron Modes, our advisor, for supplying the necessary guidance and support to see us through the difficult times. LCDR Ron Kurth, our second reader, for his insight and timely encouragement. Finally, and most importantly, to our wives, Marti and Shirley, for supporting the successful completion of our work.

I. INTRODUCTION

A. BACKGROUND

Project Management within the Navy involves the coordination of a complex set of managerial and technical responsibilities. The complexity is the result of such factors as the diversified areas in which a Program Manager must become competent and the size and complexity of modern weapons systems. The task is aggrevated and the problems magnified by several factors including schedule limitations and resource scarcity (human, monetary, procedural management tools, etc). Because the current institutionalized procedures are inadequate, a Program Manager has insufficient tangible guidelines to organize a project in a way which will counter and mitigate complexity.

As a consequence, most projects suffer increasing inefficiency which is paralleled by a rise in disorganization. This is a sure result of uncontrolled complexity. One of the more notable areas of inefficiency is in the process of specifying the desired system. Our current "methodology" all too often generates nebulous and inaccurate system specifications. This situation begins a snowball effect of increasing ambiguity as contractors, bidding on the project, attempt to design a system to meet specifications which may not be complete or correct. Therefore, contractors are forced to react to the assumed meaning of poor specifications rather than acting toward generating a clear, logical, and correct design. This approach to generating specifications generally results in the contractor's proposals not meeting the user's real need. Hopefully, problems are discovered early; the later they surface, the higher the

cost to correct them. At best, however, these undetected flaws cause the needless loss of much time and money (after the project is given to a contractor) regardless of when discovered.

To summarize, complexity is inherent but controllable in all projects. We currently do very little in attempting to control it. The resulting disorganization leads to time and money losses mainly due to poor specifications.

B. PURPOSE

This thesis presents a procedural methodology for an embedded weapons system's specification development and design documentation, answering the need defined in the previous section. The method is abstracted from a case study of the Harpoon Shipboard Command-Launch Control Set system development initialized by Sentman and Maroney [Ref. 1] and refined by Olivier and Olsen [Ref. 2]. It is our intention to show that by using this methodology, complexity will be reduced and the following improvements to embedded weapons system procurement will be realized:

1. better specifications generated,
2. better evaluation of contractor's proposals,
3. increased efficiency within the project manager's office,
4. better pass down information available to the project manager's relief, and
5. development costs lowered.

C. SCOPE OF THE METHODOLOGY

The methodology discussed in this thesis is intended to apply to the development of all embedded computer systems for tactical weaponry. The possibility for a broader scope exists since the underlying principles are widely

applicable. However, further generalizing of the methodology is not appropriate at this time since the case study only addressed a tactical weapons embedded computer system.

Figure 1.1 shows the placement of the Software Engineering Methodology within the initial weapons system procurement phase. Figure 1.2 details the general flow of control within the Software Engineering Methodology. This figure also shows that while the Contractor Support Services (CSS) Contractor develops the specifications and other products, the Program Manager lends guidance to and approves the final products of this process. The guidance supplied is of a managerial and not a technical nature. Since our handling of the methodology is concerned with the technical issues of how the procedures should be performed, the thrust of our discussion will be aimed at the CSS System Development block of Figure 1.2.

D. METHODOLOGY OVERVIEW

It was our determination that the system design methodology, while generally only an abstraction from the case study, must possess several broad traits in order to meet the objectives stated in the Purpose, Section B. Where these traits were not innate in the abstracted procedures, the methodology was refined to encompass them. These traits are introduced below. The Conclusions portion of this thesis, Chapter 5, discusses why each of these traits is necessary and how they permeate the methodology.

1. Simplicity. Simplicity of the methodology and in the understanding of its goals and products is necessary. Unless a system is simple, it has great potential to become part of the complexity problem rather than part of its solution.

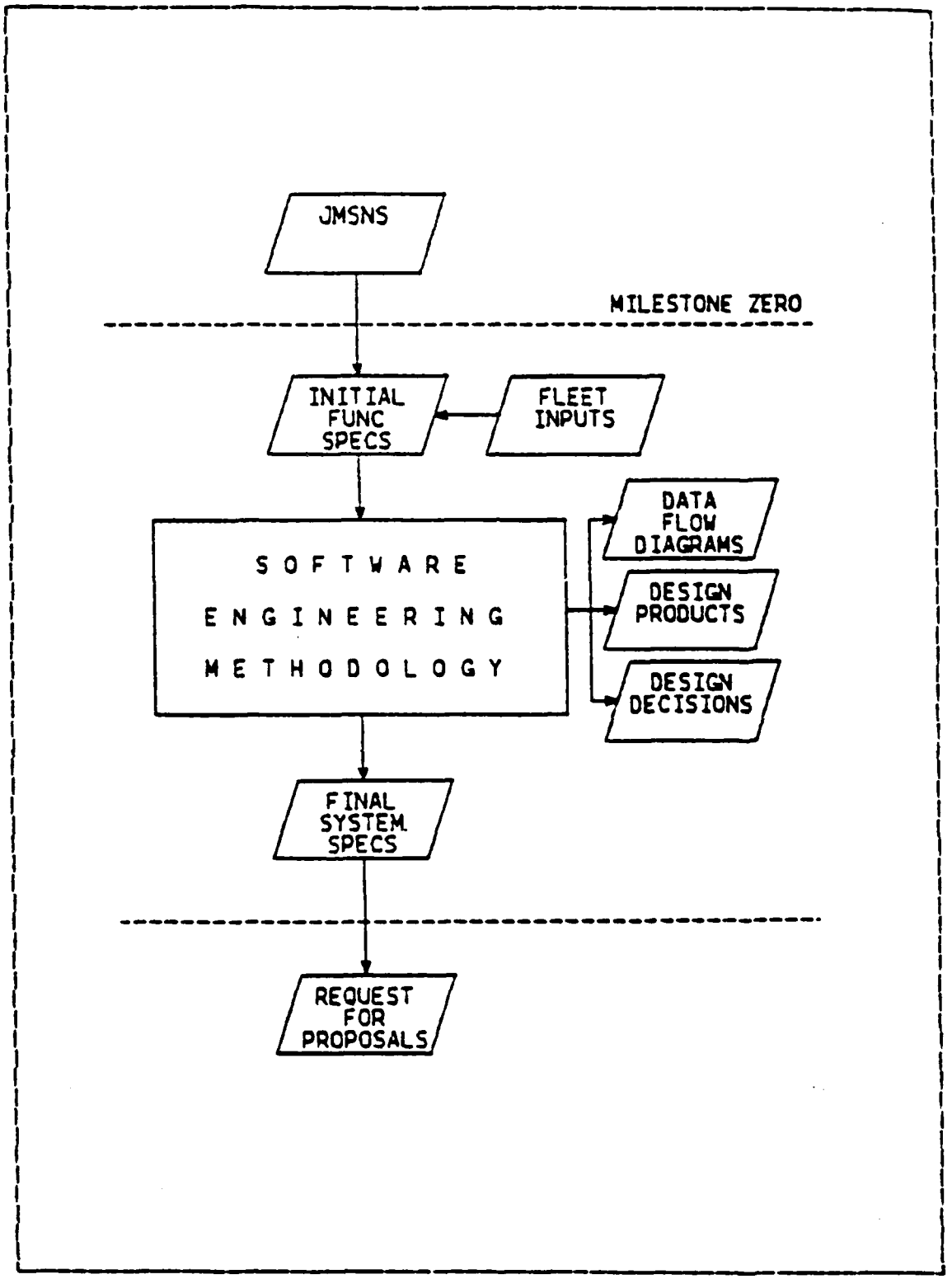


Figure 1.1 Program Management High Level Flow Chart.

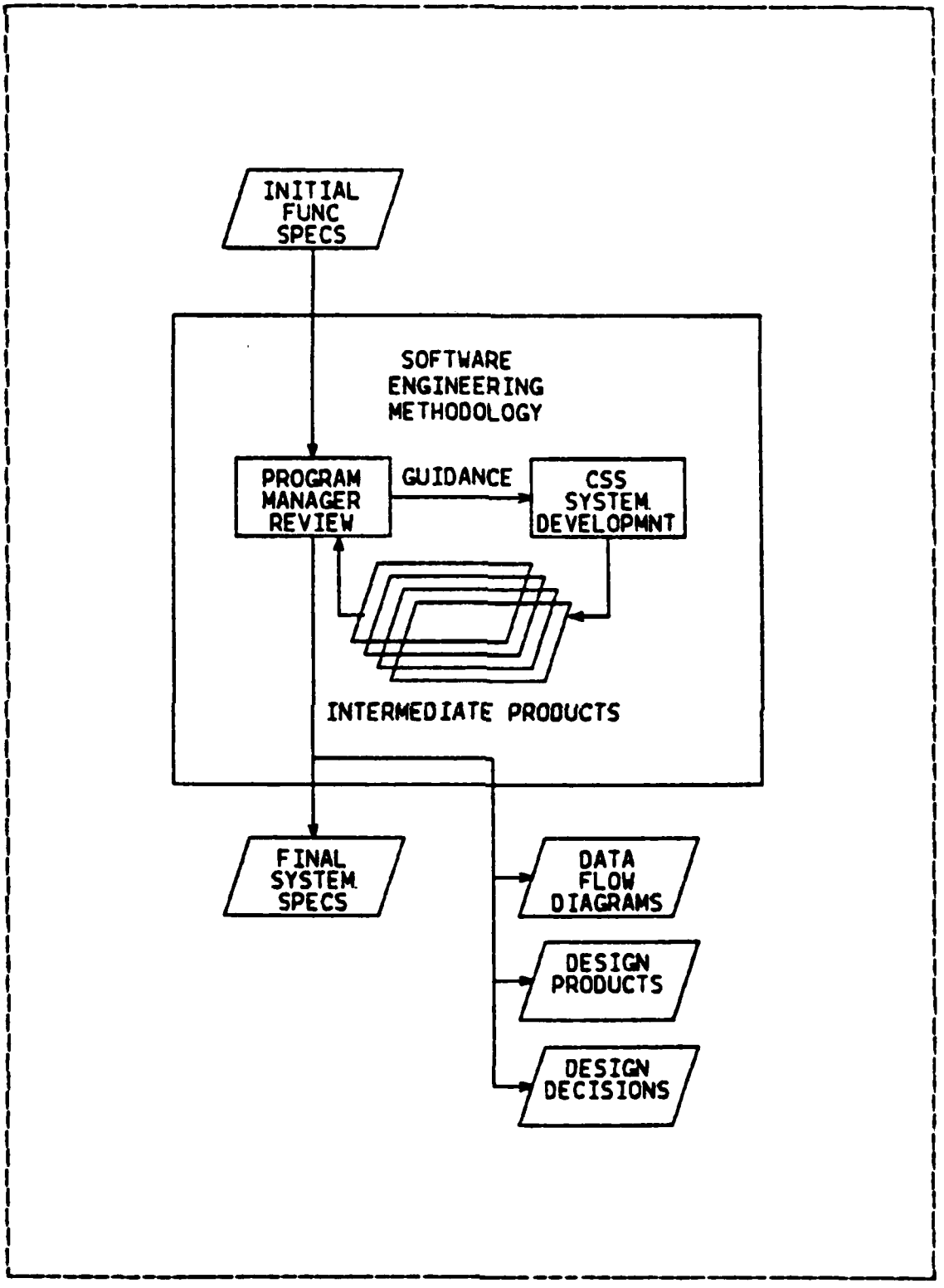


Figure 1.2 Detail of the Software Engineering Methodology.

2. Generator of Good System Specifications. The methodology must produce firm, finely-tuned, and in-house system specifications. Note that the term in-house refers to the project being directly supervised by the Program Manager regardless of where the actual work is performed. To be most effective, however, the actual work should be done in the same general location as the Program Manager (i.e. the same office, office building, or group of buildings). This assumes that it is necessary to have physical closeness of the Program Manager and the project designer in order to achieve their continual and effective communication.
3. Generator of Good Documentation Products. The methodology must produce products which serve as a proper passdown to reliefs of the Program Manager and his staff memebers. If design decisions and system specifications are not properly documented, corporate knowledge will surely be lost upon job turn-over.
4. Generator of Understandable Products. The methodology must produce products which require little formal training to understand and use. Also it must be couched in terminology easily absorbed by the average Program Manager.

To ensure that these broad system traits are achieved, the methodology must yield products which possess several specific features, inter alia understandability, reliability, efficiency, and modifiability. These are the major goals of software engineering design methods. To achieve these generic goals, the software must adhere to many structure principles. Ross, Goodenough, and Irvine [Ref. 3] provide the following list of required principles:

1. Modularity. The modularity principle defines how to structure a software system appropriately.
2. Abstraction. The abstraction principle helps identify essential properties common to superficially different entities.
3. Localization. The localization principle highlights methods for bringing related things into physical proximity.
4. Hiding. The hiding principle highlights the importance of making nonessential implementation information inaccessible. It enforces constraint on access to information.
5. Uniformity. The uniformity principle ensures consistency.
6. Completeness. The completeness principle ensures nothing is left out.
7. Confirmability. The confirmability principle ensures that information needed to verify correctness has been explicitly stated.

The methodology must meet the goals and objectives detailed above and must possess the listed traits. It must also adhere to all of the principles of software engineering design strategies. Only by religious adherance to these criteria can the complexity of designing a tactical weapons system be significantly reduced.

There is one fundamental premise of this methodology imperative to its success: the system software development must hold top priority with hardware issues being deferred until the system specifications are completed. In other words, the software decisions must drive the hardware selection. This premise has been reiterated and substantiated by numerous case studies performed in recent years among them Barry Boehm's "software first machine" [Ref. 4]. In view of the fact that the amount of computer development money spent

on software is several times the amount spent on hardware, this is a logical prioritization of project emphasis.

The basis for the above premise is that, in order to meet the goals of reliability, modifiability, maintainability, and to a large degree portability in software, it must be procedurally developed independent of and without regard for the hardware on which it will execute. A major source of frustration and inefficiency for programmers and maintainers of current tactical weapons system software is that the hardware is ingrained in and an inflexible part of the system. Consequently, all modifications to the software must be couched in the limitations of the system hardware, limitations which often require that software modifications disregard all principles of software engineering. If the reverse process, that of allowing the hardware to drive the software, is used, these hardware deficiencies are quickly realized. When this occurs, the potential for maintaining the desired goals specified above is greatly reduced.

Holding off on the hardware specification until the methodology is completed is not an unrealistic proposal. This is especially true in light of the high frequency of hardware change and upgrade which most weapons system projects experience. The basic idea is simple: it is relatively easy to find shelf hardware to implement a software system while the difficulty of achieving the design goals listed above on a specified piece of hardware is at best unpredictable.

A standard argument against having the software drive the hardware is that there are many hardware systems purchased (one per platform) but only one software system. This basically implies that cost savings are more a function of hardware than software. This argument could be valid if no modifications to the software, which destroy its structure, were required. But the probability of achieving this

over the system's life cycle is incredibly small. If the structure is destroyed, the future system costs, even in discounted or constant dollars, would invariably be many times the initial cost savings in hardware.

Prior to initiating the procedures of the methodology, the Program Manager along with his staff must become familiar with the current project documents and the specific purpose and mission of the weapons system. The first step is to become intimately familiar with the Broad Specifications detailed in the Life Cycle Management Milestone Zero documentation, the Justification for Major System New Start (JMSNS). These broad system requirements are developed based on a projected mission need by Department of the Navy planners. In-house refinement of these Broad Specifications due to changing needs, technical advancements, and inputs from the fleet (the user group) produces a set of Initial Functional Specifications. Next the Initial Functional Specifications are used as the input to the methodology to design the proposed system utilizing the principles of software engineering. Again Figure 1.1 provides a graphic representation of this flow.

Three disjoint items are pertinent to the overall view of the methodology in this stage of the system development. First, the system design is most likely being performed by a Contractor Support Service (CSS) firm. This is because the Program Manager nor his staff have the time and in many cases the ability to perform these tasks. Second, this CSS firm is effectively part of the Program Office. It should not be thought of as a separate entity but rather as a technical representative augmenting the Program Manager's staff. This closeness ensures that the Program Manager's desired system will be generated. Third, the products produced by the system are generated and updated iteratively (see Figure 1.2). This continual refinement of the products ensures

good documentation of the perceived system. These items of note must be fully comprehended by the Program Manager in order to most effectively utilize the methodology.

There are four output products generated and refined by using this methodology: a detailed set of system specifications (the final Refined Specifications), complete Data Flow Diagrams and Hierarchy Charts, the designed system in ADA System Design Language (SDL) with Module Descriptions, and Design Decision Documentation (see Appendices A-E which show these products for the HSCLCS design). Collectively they provide all the documentation required to perpetuate the corporate memory of the project and to give a complete picture of the proposed system. Individually they provide the following functions:

1. System Specifications. These are the detailed specifications delivered to project bidders responding to the request for proposals. The higher the level of refinement of the specifications when entering this phase of weapons system development, the better the chances are that bidders will develop sound system proposals to meet the real need.
2. Data Flow Diagrams (DFD) and Hierarchy Charts. These products provide a graphic display of the system by illustrating the system functional operation. Using only the functions to be performed and the input and output data needed to perform these functions, DFD's and Functional Hierarchies are simple to generate and use.
3. Design in ADA SDL With Module Descriptions. The design provides a procedural-level illustration of the system. It documents how the required functions shown in the DFD's are transposed into a hierarchy of procedures, fuctions, and tasks for data manipulation in order to perform these functions.

4. Design Decision Documentation. While most design decisions appear in other documents (i.e. the specifications, design, etc.), some are not feasibly includable in other products. The Design Decision Documentation provides a place to store pertinent facts and parameters of the system.

Thus far in this section we have dealt with the necessary goals, principles, and requirements of the Software Engineering Methodology box of Figure 1.1 and not the mechanics of the system. This is because the high-level view of the methodology must be one of achievement of design objectives and not in the procedures necessary to produce documents. Whether or not these objectives are met will be the subject of Chapter 5, Conclusions. However, to provide a proper overview of the methodology details Figure 1.3 is included as an illustration of the iterative product formulation phase. The detailed discussion of this flow and its subgoals is the sole subject of Chapter 4.

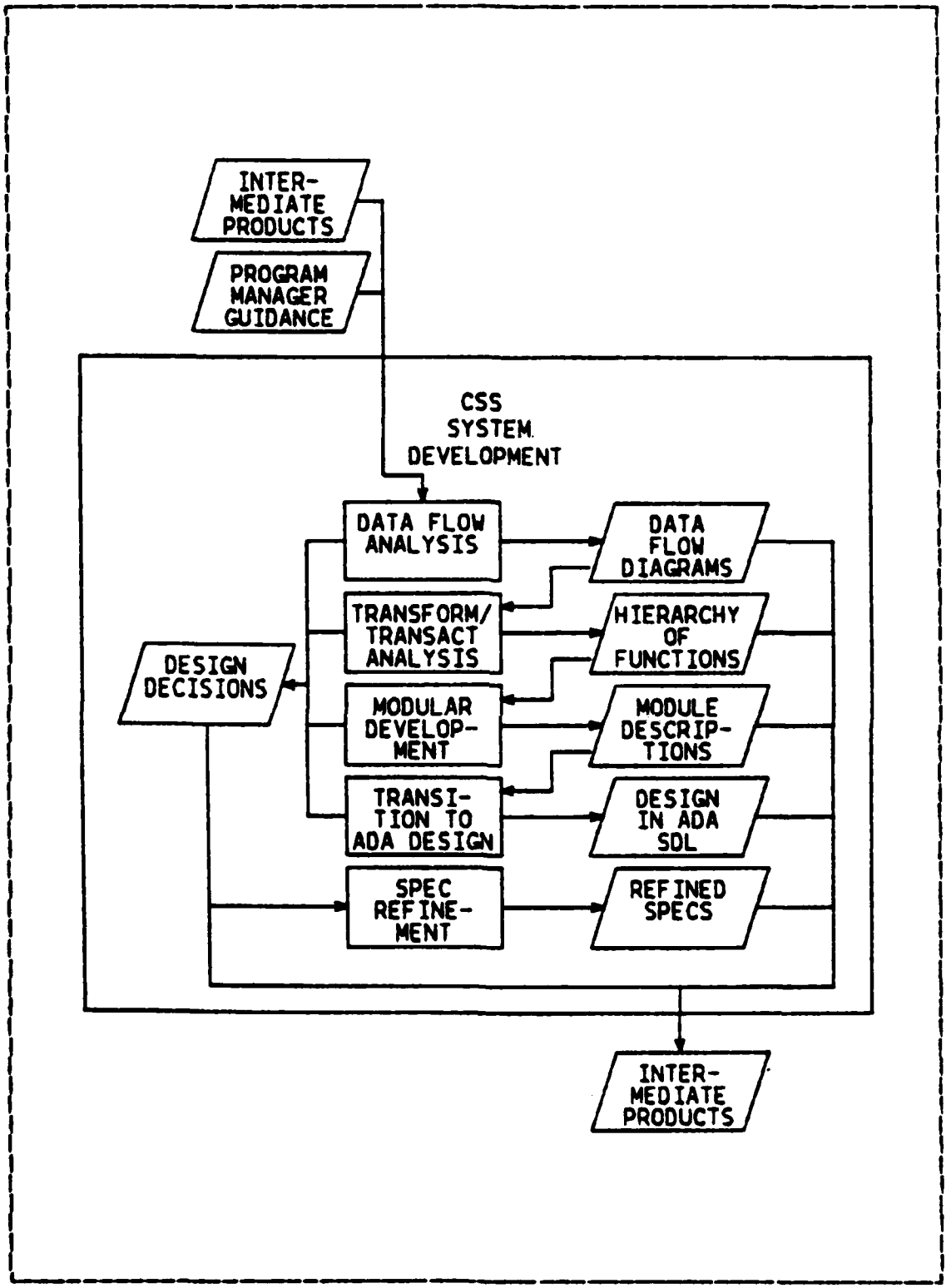


Figure 1.3 Detail of the CSS System Development.

II. BACKGROUND OF THE HARPOON CONTROL SET DESIGN

Recently when the missile subsystem of the HARPOON Weapon System was upgraded to include two new block enhancements, the existing HARPOON Shipboard Command-Launch Control Set (HSCLCS) was rendered inadequate to support the design features of the new blocks of missiles. Upon examination by analysts, it was decided that the existing HSCLCS software was not modifiable and a new design effort was necessary. The new design would need to not only cover the recent missile changes but also be flexible enough to be modified to support anticipated technical achievements in the near future. This chapter will introduce the basic facets of the HARPOON Weapon System and provide background on the work done in two previous theses, [Ref. 1] and [Ref. 2], toward redesign of the HSCLCS.

A. EXISTING HARPOON WEAPON SYSTEM

The HARPOON Weapon System (HWS) has been developed to fulfill the requirements of the Navy's anti-ship mission. The HWS is currently deployed on surface ships, submarines and aircraft. The HWS provides over the horizon anti-ship capability in all weather, day or night. The HWS is comprised of the missile, launcher, and command-launch subsystems. The ship-launched HARPOON employs either onboard or third party sensor data for targeting information. The missile is a "launch and forget" weapon, since no ship control or information is needed after launch.

For surface ships, the HWS control and data processing functions are provided by the HSCLCS which has three modes of operation: normal, casualty and training. In the normal mode the major functions of the HSCLCS are:

1. Distribution of power to various HWS equipment.
2. Selection and application of missile warmup power.
3. The ability to conduct various automatic and manually initiated tests which confirm the operability of the system.
4. Distribution of ship motion data from ship equipment.
5. Selection, transfer, processing and display of target data.
6. Coordination of the selection of the tactical missile mode and type of fusing.
7. Selection of the launcher cell containing the intended missile to be launched.
8. Initialization of the selected missile and the supervision of the exchange of data between missile and other HWS equipment.
9. Control of all missile firing activities.

These functions are implemented and integrated by the HARPOON Weapon Control Indicator Panel (HWCIP) and the HARPOON Weapon Control Console (HWCC).

The HWCC contains most of the HARPOON system-unique command and launch subsystem equipment, including the Data Processor Computer (DPC), the Data Conversion Unit (DPU) and the HWCC life support equipment. Together these components perform data processing and conversion among various data types and provide interfacing with existing sensor and ship's equipment.

The WCIP provides visual status information to the operator during formulation of the fire control problem, and additionally provides manual controls for the operator. The existing WCIP is shown in appendix E.

The DPC is a 16-bit microcomputer with 15K of EPROM. The DPC uses an assembly language program to provide the following:

1. Launch envelope parameter validation.
2. Missile command generation for implementation of missile control parameters including ship's attitude, search pattern orders, engine starting, flight termination range, altimeter setting, and various selectable flight trajectory and maneuvering modes.
3. Pre-launch testing.
4. Pre-launch sequencing and timing.
5. Data formatting and transfer synchronization.

The DCU processes all digital and analog signal conversions as required by installed hardware. The DCU also provides interfacing of target data inputs from the Naval Tactical Data System (NTDS) Slow Interface. Ship motion parameter data is also converted in the DCU.

B. PROBLEMS ASSOCIATED WITH EXISTING HSCLCS

Since the existing software of the present HSCLCS is written in assembly code and is heavily hardware dependent, the maintenance cost in the face of periodic missile changes is relatively high. Also several different hardware configurations exist for the different firing platforms.

The HSCLCS also has numerous deficiences in engagement planning as the operator cannot fully control the features of the new block missiles. In fact, the operator has no automated assistance in engagement planning in the current system, and there is no display of the tactical situation at the WCIP. The current firing solution does not have environmental factors included unless the operator considers them manually. On some platforms NTDS was intended to provide the services mentioned in most of these deficiencies but the location of the WCIP has inhibited this effort and indeed many HARPOON platforms do not have NTDS!

C. HARPOON WEAPON SYSTEM CONSTRAINTS

The constraints in this section are for the most part technically oriented. Managerial constraints are to be determined by competent authority at a later date. The upgrade of the HSCLCS must be able to support the new block missiles as well as the old ones since the old missiles will be in the fleet for some time.

The implementation of the upgrade must continue to provide all previous fuctions as well as interfacing with NTDS. The existing launcher hardware will remain the same and the physical size of the HSCLCS must be the same.

While the DPU hardware configuration cannot change, the DPC software is subject to change as necessary to implement the upgraded HSCLCS. Although the current software is in assembly language, this is not a requirement for the upgrade. System reliability of the upgrade must meet or exceed existing standards for the HSCLCS.

D. SYSTEM DEFINITION FOR HSCLCS UPGRADE

A detailed discussion of the system definition for the upgrade can be found in [Ref. 1]. It is summarized below.

The hardware of the system will change significantly. The existing DPC will be replaced with a commercially available CPU with additonal memory. The WCIP will be modified to include a display which shows the current tactical situation and programmable software keys to control both the display and engagement planning features which will be incorporated into the DPC software. A hook and cursor similar to those in NTDS will also be provided at the WCIP for the operator. A display processor will be attached to the WCIP. The DCU hardware will remain the same however the software must be changed to accommodate new inputs from NTDS and environmental data.

The software upgrade of the DPC which is the major part of the HSCLCS focused upon by this thesis is to eliminate the existing deficiencies mentioned in the section B of this chapter. Specifically, a software plan must be developed which produces adequate software that provides required capabilitities and is flexible enough in design to be modified in the future with minimum amount of blood and tears.

E. STATE OF THE UPGRADE

The software upgrade of the HSCLCS has been the subject of the two previously referenced theses. The initial thrust of the first thesis by Maroney and Sentman was to develop a software plan, Figure 2.1, and complete the first three phases. Emphasis was placed on good software engineering techniques. A systems requirements analysis was conducted which produced revised system specifications and laid the foundation for the preliminary design. Data flow diagrams and subsequent transform analysis techniques described in [Ref. 5] were used. ADA was chosen as the system design language in anticipation of its proclamation as the standard DOD SDL and because it lends itself so well to the modularity concepts necessary for modular design.

The second thesis by Olsen and Olivier continued the software development by deriving a preliminary design from the products of Maroney and Sentman. To continue the plan, a final design must be completed along with detailed documentation. This final design process is described in the methodology chapter.

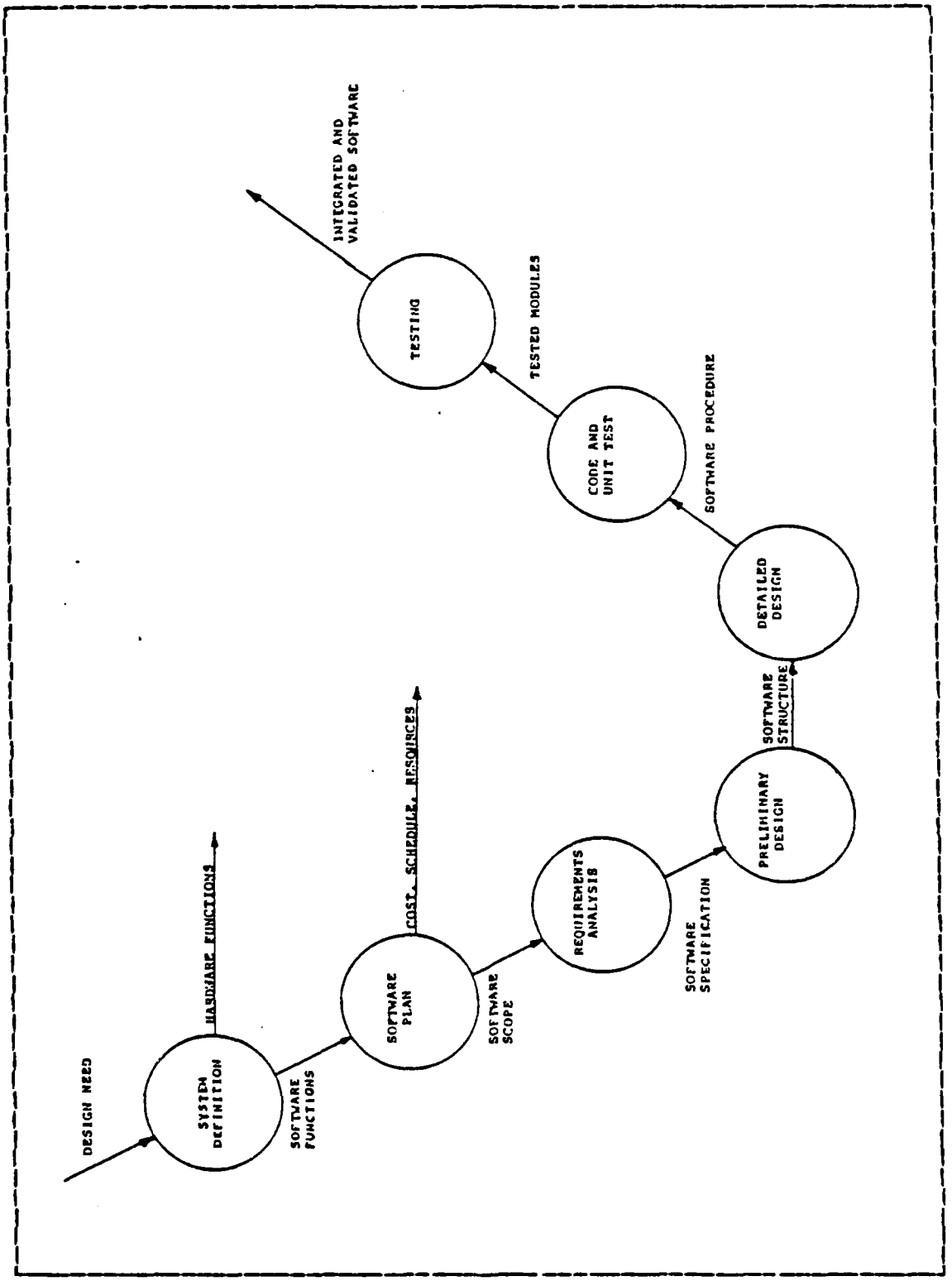


Figure 2.1 Software Plan from Reference 1.

III. SOFTWARE ENGINEERING SNAPSHOT

The need for good software engineering techniques has become increasingly evident in the past decade with the exponential growth of software development and maintenance costs. Since necessity is the mother of invention, the number of new software engineering methods and techniques has also grown exponentially. The major contributors to the methodology of this thesis, Pressman, De Marco, and Booch, all have derived systems for software design using their own particular styles. In this chapter we will briefly discuss those styles and also comment on some other software design methodologies.

Structured design was first publicized by Yourdon and Contantine [**Ref. 6**]. It was developed to be used as the transition tool between Structured Analysis and actual implementation. Composed of various concepts, measures, rules of thumb, and analysis techniques, this method with early development by De Marco is the basis for the Pressman design methodology.

In [**Ref. 7**], De Marco describes the life cycle of a software project from requirements analysis to specifications. After an initial survey of systems requirements, a data flow analysis is conducted using data flow diagrams. The next step involves creating a data dictionary from the data identified in the data flow analysis. At this point in De Marco's methodology, the data flow diagrams are translated into a set of specifications using a subset of English called Structured English. Structured English is a specification language that makes use of a limited vocabulary and a limited syntax. The vocabulary consists of imperative verbs, terms defined in the Data Dictionary, and

certain reserved words for logic formation. The mapping of the data flow analysis to the Structured English specifications is fairly algorithmic but uses several heuristics that will not be discussed here. De Marco also explains the desired traits of a design based on the specifications generated, but does not include a procedure for realization of the design.

Pressman, [Ref. 5], elaborates on all phases of the software life cycle and gives several different approaches to design such as data flow oriented design and data structure oriented design. In both of these areas he carries the software development process through the preliminary design phase but does not address specification generation. The data flow analysis of Pressman resembles that of De Marco but his transform/transaction analysis which leads to module hierarchy charts contibutes significantly to design realization.

The object oriented design methodology of Booch [Ref. 8] concerns the development of design after some sort of data analysis has been conducted. Booch does not indicate a preference as to whether data flow diagrams or any other kind of analysis identifies the objects in a project as long as the method is complete. After objects are identified and given attributes, this methodology develops a system design by stepwise refinement of a simple prose description of the system. This prose eventually is transformed into ADA system design language. No guidance for conversion of the ADA SDL to structured system specifications is given in this methodology.

There are several system analysis and design tools that have been implemented but have not gained wide-spread use. SADT (a trademark of SOFTECH, Inc) is a system analysis and design technique developed within the Yourdon organization that is used as a tool for system definition, software

requirements analysis, and system design. The methodology encompasses technical tools and a well-defined organizational harness through which the tools are applied. An automated requirements analysis tool is SREM [Ref. 5], where elements, attributes, relationships, and structures (all parts of the Requirements Statement Language (RSL)) are combined to form the details of the requirements specification. SREM was initially designed for embedded computer systems and requires a software support package called REVS which uses computer graphics and reports on information flow. Still another automated tool is CADSAT (Computer-Aided Design and Specification Analysis Tool) which with PSL/PSA provides an analyst with several capabilities. These include:

1. description of information systems, regardless of application area,
2. creation of a data base containing descriptors for the information system,
3. addition, deletion, and modification of descriptors, and
4. production of formatted documented and various reports on the specification.

CADSAT does not present a panacea but it does provide benefits that include documentation quality, easy cross reference of documents, easy modification, and reduced maintenance costs. The major disadvantage of most of these automated systems is that they require a considerable amount of training in order to be used effectively. However, the concept of automated design is here to stay because the benefits far outweigh the disadvantages.

The methods described above are only a few of the many ways that software development is being conducted today. The design tools such as decision tables, flow charts, HIPO-charts, structured flow charts, and program listings

abound. It is outside the scope of this thesis to discuss in detail all of the methodologies, but each one is based on the design principles outlined in this thesis. If each methodology produces results with the desired characteristics, only through extensive experience can one judge the relative efficiency of the methodologies. Since software engineering is still at the fledgling stage, we can only hope that these methodologies will mitigate the software crisis.

IV. DESIGN METHODOLOGY

The methodology for refining embedded computer weapons systems specifications, which is the subject of this chapter, is required to possess an algorithmic form and logical design at all levels. By levels we mean the levels of abstraction from which the methodology can be viewed. For example, an outsider to the project office would view the methodology as a "black box" which inputs broad specifications and fleet criteria and outputs final design specifications and refined design products (see Figure 1.1). The Program Manager would be heavily involved in the iterative refinement of the system specifications and products and consequently would see the methodology as a generation and refinement tool. His "black box" would be the Contractor Support Services (CSS) System Development block of Figure 1.2. Finally, the CSS Contractor would view the methodology as an algorithm for production. This algorithmic flow is shown in Figure 1.3. These are proper abstractions for the methodology; they optimally map the responsibilities of each of the individuals into their required level of concern for detail.

This chapter is concerned with introducing a methodology at the CSS Contractor level which embraces all of the goals and principles and proper trade-offs of Software Engineering design. This level can be viewed as the bottom of the abstraction hierarchy because it is the lowest level at which the entire design is still within view. It is our belief that if this level of the design methodology is well-structured and simple then the entire hierarchy will be so. This hypothesis will be further developed in Chapter 5.

The methodology, at the level specified above, was conceived and tuned using the following pair of guiding rules: it must have a simple, sequential form and it must support a data transform driven design. By data transform driven design we mean that the products of design must project how a datum is interrelated to other data and how data is transformed as processes act upon it. The reasons for these basic requirements are the subject of the two subsequent paragraphs. The achievement of the first requirement is best revealed by an illustration; Figure 4.1 serves this purpose. Notice on this diagram that the flow is characterized by singular inputs and outputs with a processing block between them. This by definition is the simplest form of sequential flow, thus the first rule is satisfied. Figure 4.1 additionally shows that the first step of the methodology or what we will henceforth refer to as the first methodology function is to manipulate the specifications into Data Flow Diagrams. This function, data flow analysis, strictly follows De Marco's procedure [Ref. 7], a procedure which fully incorporates the criteria for data transform driven design listed in the definition above. It follows that the second rule is additionally satisfied.

There are several strong reasons for requiring a methodology with simple, sequential flow. For example, the usage of such a methodology is straight-forward and easily grasped. Further, this type of flow tends to be highly logic rather than heuristic oriented. But the chief reason we wanted simple, sequential flow was to have a structure which readily supported our methodology model. This model views the system as a series of functional mappings, e.g. data flow analysis is a function mapping specifications into a hierarchy of Data Flow Diagrams (see Figure 4.1). The use of the word function is not intended to imply that the

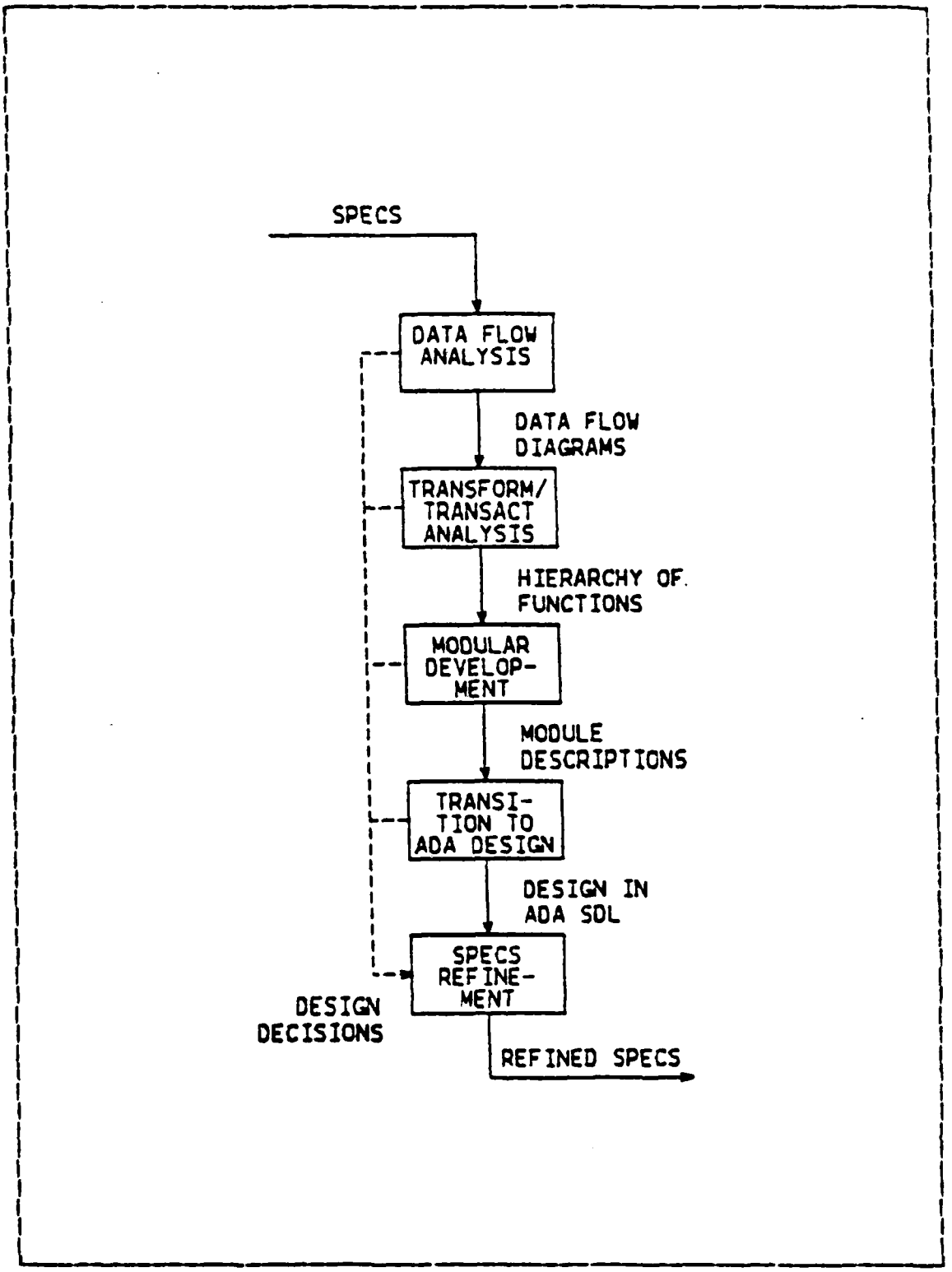


Figure 4.1 Methodology Sequential Flow.

products, i.e. the Data Flow Diagrams, produced by the methodology are themselves unique; the mapping is not one-to-one. However, we suggest that each of our methodology functions map their input product into a small set of output products which is a realistic partition of all possible output products. By realistic partition we mean an equivalence subset of the output products which contains only those products having all of the desired structure principles but which omits those grossly inefficient representations of the solution. The benefit of this terminology is it enables the reader to view the methodology from a familar technical vantage. Using the terminology we introduce our hypothesis that these functions retain the properties of the input products by transmitting them to the output products. In other words the methodology functions are designed to ensure that the good initial structure is carried forward throughout the methodology.

The main reason for requiring the methodology to use data driven design was based on the fact that real-time systems (all applications of our methodology will be real-time systems) are easiest to design this way. Shooman [Ref. 9] supports this hypothesis. We decided on data flow design because the graphical nature of the data flow model supports DeMarco's [Ref. 7] belief that all products of analysis functions should be graphic.

The procedures of the methodology represent the compilation of related work performed by several distinguished pioneers in the field of software engineering. But the overwhelming majority of contributions came from three individuals: De Marco; Pressman; and Booch. While each of these men see the problem in the same basic light, they have channeled their research efforts into different facets of the problem. The De Marco contribution consists of a method for transforming system specifications into a set of structured

products, Data Flow Diagrams, which represent a graphic solution to the specification requirements. Pressman details a procedure, transform/transaction analysis, for creating an abstracted hierarchy of context-independent modules, a Function Hierarchy, from Data Flow Diagrams. Booch, claiming to have achieved object oriented design [Ref. 8], contributes a method for developing a final design given a Function Hierarchy. It will be shown later that the Booch procedure is in fact an object oriented design technique. Figure 4.2 illustrates the specific areas of methodology coverage by each of the authors. Fortunately for our purposes, these areas of specialization correspond to one or more of the specific functions in our methodology such that all of them (except Specifications Refinement which is our contribution) have been significantly researched. Thus only the structural interfaces between the various contributors need to be specified before reducing the methodology to a series of independent functional units (see section B).

The effort required to structurally interface between the contributors is minimal. On the surface this may appear puzzling in light of the complexity normally encountered when synthesizing a complete product from disjoint pieces. But because each of the contributors used the same generally accepted product formats at the interface points, these problems were not present. No interface is required between the De Marco and Pressman portions of the Methodology. This is because Pressman uses all of the rules of De Marco to produce Data Flow Diagrams, the input to his transform/transaction analysis. Consequently, we can view this situation as if De Marco and Pressman "collaborated" on the interface. Nor is an interface between Pressman and Booch required. The portion of Booch's method we use requires only a function hierarchy as input. Since this is the output product of Pressman, no structural interface specified by the methodology is needed.

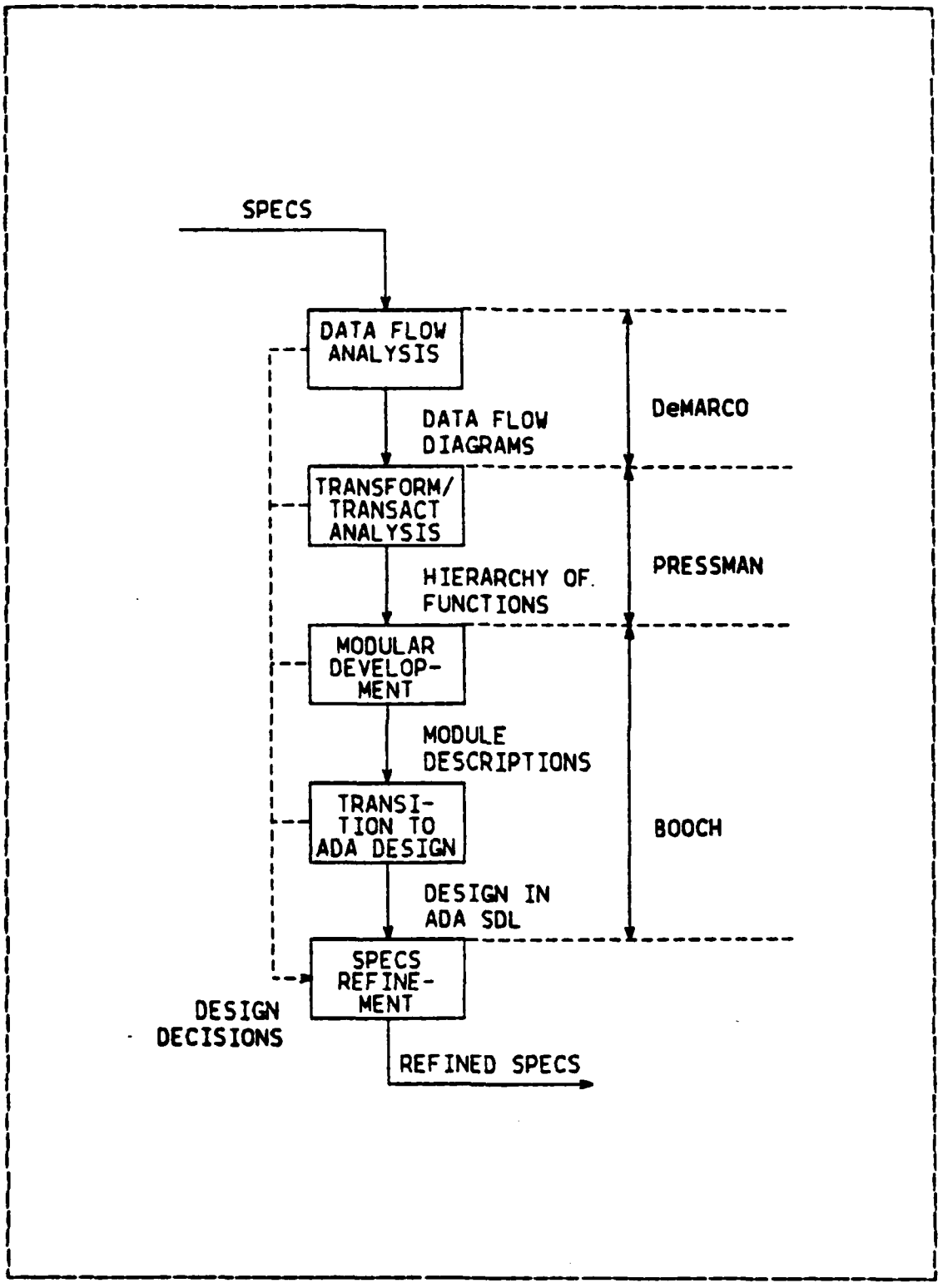


Figure 4.2 Contributors to the Methodology.

A. METHODOLOGY CRITERIA

1. Goals and Principles

The goals for the software produced by the methodology (understandability, reliability, efficiency, and modifiability) are generally accepted by software engineers as those of primary importance. In general, this list encompasses all of the relevant attributes necessary to ensure that software will realize its minimum life-cycle cost. These goals are defined as follows:

1. Understandability. Understandability is that potential for software to project a clear and logical meaning. It is achievable in all systems regardless of the complexity if both the structure and the level of abstraction are appropriate for the proposed application. It must be stressed that both of these properties are needed. Having merely a formatted structure yields a legible but complex product. In order to realize any of the other goals, understandability is paramount.
2. Reliability. Reliability is the ability of the software to function, under all conditions, as the specifications intended. It can be thought of as freedom from anomolies as well as the absense of blatant mistakes. Reliability also encompasses error recovery, the ability of the program to continue processing in the event of non-catastrophic system failure. Achievement of total reliability is extremely difficult to prove even in a system strictly adhering to software engineering principles. It is impossible to prove software reliability under lesser conditions.
3. Efficiency. Efficiency, as a structure-driving goal, is wrong. However, blatant inefficiency makes a

system impractical. The efficiency balance must be achieved by first adhering to all other goals and then screening for gross inefficiencies which can be corrected by encapsulating and modifying inefficient modules. This is supported by Belady and Lehman [Ref. 10] who state that glob l optimization is not a practical objective, but that by locally optimizing, global sub-optimization can be achieved. Thus efficiency should be deferred until a solid system structure is established.

4. Modifiability. Modifiability is a broad term which encompasses the ability to easily change software for enhancements or errors, for performance tuning, and for subsetting. The achievement of modifiability is difficult because the effects of change are very hard to predict. Thus modifiability, more than any other goal, universally requires the strict adherence to all of the software engineering principles.

To meet these design goals, the principles addressed in Chapter 1 (modularity, abstraction, hiding, localization, uniformity, completeness, and confirmability) are the primary attributes required of the methodology products. It seems apparent from our readings that among the seven principles, modularity and abstraction are uniformly accepted as the dominant requirements of all software. This is not surprising considering that these software qualities, which logically reduce large problems into manageable subproblems, are the most effective reducers of complexity. These two principles are highly coupled; one abstracts to reduce complexity by modularizing and modularizes by performing a series of logical abstractions. Thus they should be thought of as iterative subprocesses of some higher level generic design process. A more detailed description of the

requirements and specifications to benchmark the achievement of modularity and abstraction are given below:

1. Modularity. As stated above, it is nearly impossible to address modularity as a stand-alone principle. In its simplest form, however, modularity can be considered achieved when the solution to the problem is reduced to a hierarchy of separately addressable modules. In order for this hierarchy to approach the optimal solution, though, it must have a good balance between two inversely proportional measures: the degree of module complexity and the degree of interface complexity.
2. Abstraction. Abstraction, too, is not an independent concept. It can be considered achieved when the problem has been iteratively expanded (or stepwise refined) such that each of the abstraction levels has a solution representation which captures the essense of the system at this level, but specifies no unnecessary complicating details. These levels of abstraction provide an intellectually graspable view of the problem's solution.

Of the remaining principles required of the methodology the most important ones are completeness, independence, and hiding. While the presentation of these principles may tend to imply that they are of second echelon order, this is not true. Rather they complete the system of interwoven requirements of the methodology. The reason these principles are presented separately is because unlike modularity and abstraction these concepts are not universally accepted in name or in their definition by the contributors. Yet each of them is either directly stated or indirectly supported as method requirements. For example, Pressman stresses module independence, a concept which

requires modularity, abstraction, and completeness as prerequisite principles. Thus Pressman must indirectly support these structural concepts. Further, he requires the simplicity of module interface in his independence concept. This is actually a loose form of the hiding principle. The key point, however, is that his method builds a structure which allows hiding to be efficiently appended to the set of principles across the interface with the Booch method. From a broad scope this implies that the method embraces a more stringent set of principles at each method interface ultimately yielding a design which adheres to all of the necessary structure principles. This idea is developed in the next subsection. The specifications for achievement of these three additional concepts are given below:

1. Completeness. Completeness, a principle stressed by De Marco, is a critical property of the products of our methodology. Its criticality is especially apparent when performing the first function, data flow analysis. It is mandatory to ensure that each system specification is appropriately captured in at least one Data Flow Diagram. If the first procedure of the methodology produces a complete set of Data Flow Diagrams then all subsequent steps will have a good, graphical representation of the requirements by which to benchmark. Thus achievement of completeness requires the assurance that each methodology function carries forward all of the information from the input product into the output product.
2. Independence. Independence, the chief principle stressed by Pressman, becomes an important concept when developing the Function Hierarchy. The degree of module independence can best be qualitatively measured by first measuring the levels of cohesion and coupling of the modules. Cohesion is the measure

of module single-mindedness [Ref. 5]. The highest cohesion, which is the goal state for maximizing independence, is achieved when every module has a single function. Coupling is the measure of module interconnection and interdependence [Ref. 5]. The lowest coupling is realized when the interfaces between modules are simplest. Low coupling is also required to achieve modular independence. Thus independence is achieved when the design products have modules which address a specific subfunction of requirements and has a simple interface when viewed from other modules.

3. Hiding. Hiding, a principle developed by Parnas and highly stressed in the Booch method, implies the prerequisite achievement of completeness, modularity, abstraction, and independence. An expansion of the requirements of independence that distinguishes hiding as a more powerful concept is that these single function modules must have a simple interface, the interface must be the only part of the module visible to other modules, and how the function is accomplished within the module must be hidden [Ref. 11]. This invisibility of module internal information takes us one step beyond what these other four principles provide: design decision encapsulation. Therefore, achievement of hiding requires a conscious effort by designers to delay design decisions until the latest possible time and when decisions are made they must be encapsulated and concealed in the structure of the design.

Tim Rentsch has boldly defined the requirements of the nebulous procedure termed object oriented design [Ref. 12]. He states that the essense of this concept is an

adherence to the principles of abstraction, information hiding, decision encapsulation, and modularity. Using his definition we can conclude two interesting facts. First, the Booch method, as Booch himself claims, is object oriented design. Second, our methodology, because of its strong adherence to the five major structure principles, is also an example of object oriented design. As the software "buzz word" of the 1980's, object oriented design will undoubtedly be a must in DOD software by the 1990's.

2. Principle Set Synthesis

Now that all of the design concepts required in the methodology have been formally presented, it is necessary to show how they are related to the methodology functions. This includes determining the point at which each of these principles becomes an active concept in the design. The synthesis idea of this subsection refers to the fact that all of the individual principles are not uniformly visible throughout every function of the methodology. They have a point at which they become necessary and are thereafter carried forward in the principle set. This idea that concepts once incorporated in the design are thereafter ingrained in its structure is justified in Section C of this chapter.

To realize a principle at the optimum time in the design, the structure must be capable of supporting the inclusion of the new concept. A rather simple way of viewing this requires one to visualize the principle to be added as needing a set of prerequisite traits. For example, the prerequisites for independence are completeness, abstraction and modularity. Thus, if the current structure of the design contains the prerequisite traits then the structure will be capable of supporting the new principle.

Because the set of principles behaves in the manner stated above, the structure requirements become increasingly more stringent as the design is refined. This is the desired effect because the ultimate objective of the methodology is to produce a design which encompasses all of the software traits but maintains its flexibility as long as possible.

The initial principle set for the methodology contains the concepts of abstraction and completeness. It is easy to see abstraction as a necessity because each of the functions iteratively refines its products and the refinement process is based on levels of abstraction. Completeness across all of the interfaces requires no explanation; without all of the parts, the design could not be correct. At the first interface, the DeMarco/Pressman junction, the structure must be able to support the addition of modularity. The fact that modularity is required at this point in the design is no surprise considering that the purpose of the Pressman function is to modularize. The second and subsequent iterations of the module heirarchy continually refine the design structure to achieve low module coupling and high module cohesion. When a satisfactory trade-off between coupling and cohesion is made, independence of modules is achieved thus appending independence to the set of principles. With the set of principles now containing all the prerequisites, the Pressman/Booch interface structure is capable of supporting hiding. Figure 4.3 illustrates the synthesis of the principle set.

B. METHODOLOGY COMPONENTS

1. Data Flow Analysis

Data flow analysis is the first facet of the evaluation and synthesis phase of the software requirements

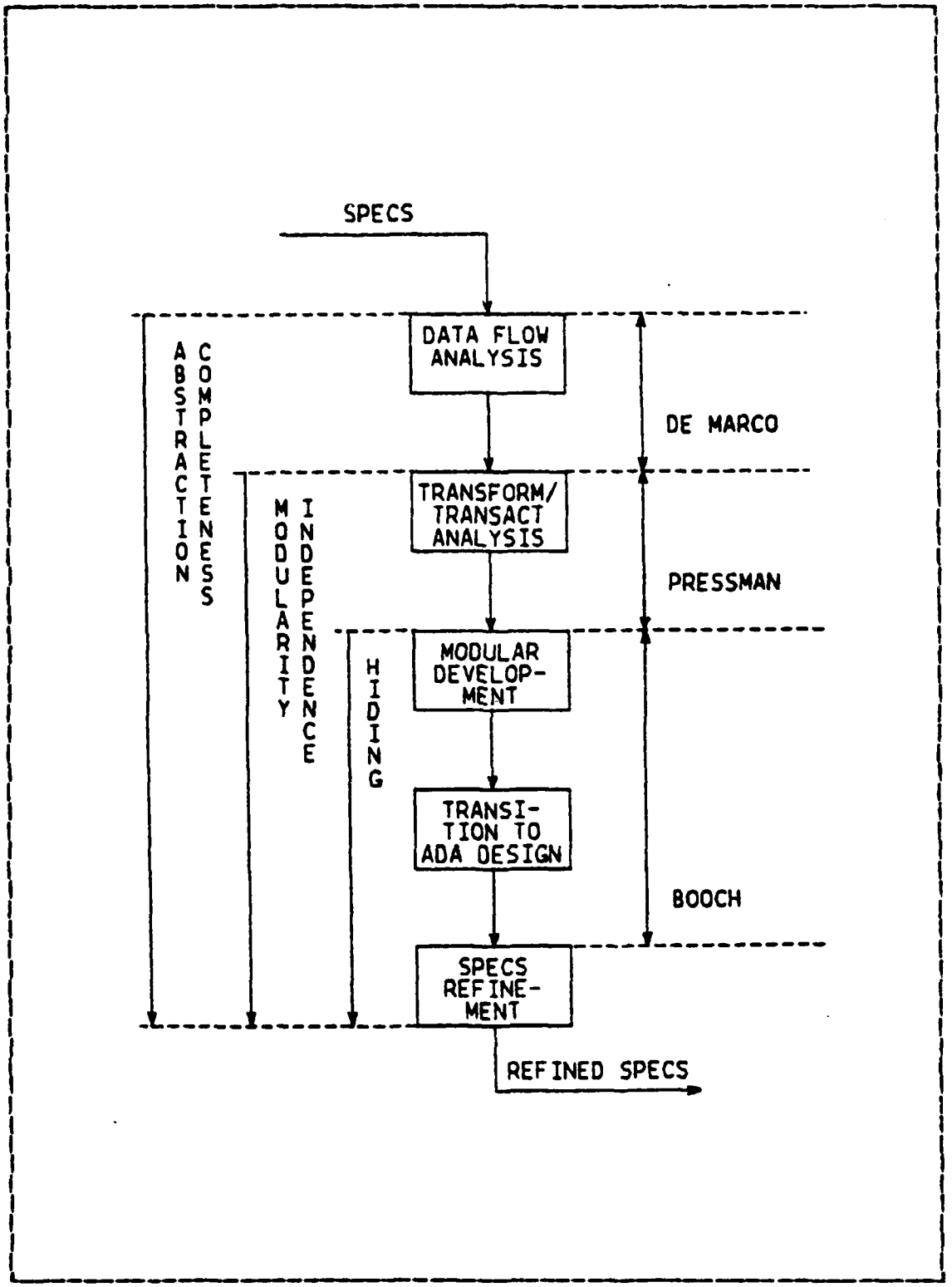


Figure 4.3 Illustration of the Principle Set Synthesis.

determination process. By examining the data flow we get the big picture on what the entire system receives as input and produces as output and the path that data follows in the system to be designed. Data flow is our analysis start point because we do not want to get bogged down in specific areas of a system trying to define functions which may not be clear in the initial analysis. Data flow, on the other hand, is usually much easier to identify than flow of control, which in most large scale projects is very complex. The primary tool we will use to examine the data flow will be the Data Flow Diagram (DFD). In this section we will briefly describe how to build a DFD summarizing the methods detailed in [Ref. 5] and [Ref. 7] and also what the DFD can give to the Program Manager. We will also introduce a set of example DFD's from the HSCLCS system that will be used as a case study to illustrate the methodology components throughout the chapter.

a. Data Flow Diagram Definition

The data flow diagram is a graphical aid for depicting the data flow of the software system being designed. A complete understanding of the DFD is imperative to the understanding of the design methodology described in this paper. The most significant characteristics of DFD's are:

1. The diagrams are graphic.
2. They produce natural partitions in a system.
3. They are multidimensional.
4. They emphasize the flow of data.
5. They de-emphasize the flow of control.

Data flow diagrams are made up of four basic elements:

1. Data flows represented by an arrow or vector from the source of the data to the destination.
2. Processes represented by circles or "bubbles".
3. Stored information (e.g data bases or files) represented by two horizontal parallel lines with a meaningful label.
4. Data sources and sinks represented by boxes.

Data flow can be broadly defined as information flowing between two processes or between a process and a source or a sink. There are several general rules concerning data flow.

1. Data flow names are hyphenated and capitalized.
2. No two data flows have the same name.
3. Choose names that describe the data explicitly but be concise.
4. Data flow should not represent a flow of control.
5. Data flow is not considered a process activator.

Processes invariably show some amount of work performed on data. More explicitly, a process is a transformation of incoming data flow into outgoing data flow. Each process bubble should be numbered and given a unique descriptive name.

Sources and sinks increase the readability of the DFD by showing where the net inputs to the system come from and where the net outputs go to. Sources and sinks differ from files and data bases in that they are considered to be out of the context of the system. Thus, they show how the internal system relates to the outside world. Figure 4.4 is the source/sink diagram for the Harpoon System Command-Launch Control System (HSCLCS).

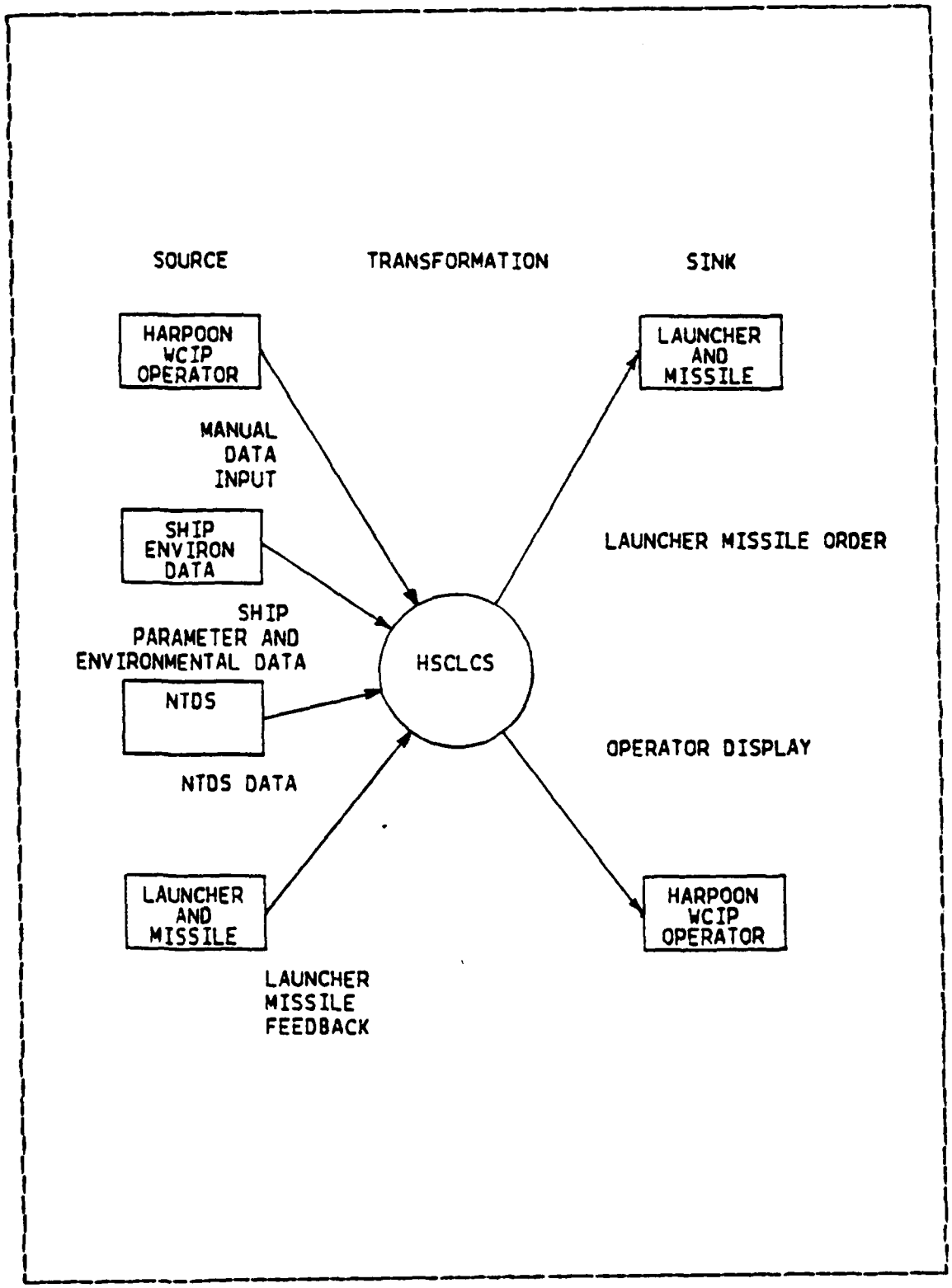


Figure 4.4 HSCLCS Source/Sink Diagram.

b. DFD Construction

The first point to keep in mind during the data flow analysis is not to try to learn everything at one time about the whole system. Think top down by conceptualizing the high level data flow first, defering the development of the low level data flow. Especially avoid addressing any implementation details at this time and be flexible enough in your thought process to start over from scratch if road-blocks are encountered. Remember the data flow analysis process is iterative.

The primary input to the data flow analysis is the Broad Specifications of the system to be designed. Direct liaison with the Program Manager and prospective users may also provide additional information. A key point to remember during each phase of the methodology is that decisions concerning design that are not specifically addressed in the Broad Specifications must be documented at the point of the decision. These design decisions will later be used to update the Broad Specifications.

To start the process, identify all net input and output data flows and list them around the border of your working paper. This step is important because it is at this point that you define the context or scope of the analysis to be conducted. Data flow outside of the scope defined here will never be addressed again.

Filling in the DFD is the next step of the process. What you try to do is put lines in your diagram depicting data flow and try to connect them with circles or "bubbles" where a data transformation occurs. You can start from the inputs, outputs or in the middle whichever is the most obvious development for you. Insure flow of data goes from left to right for ease of reading and avoid looping back to the left. If a loop appears necessary, duplicate

the process bubble that is looped to in order to keep the data flow moving right. Do not cross lines and defer naming the bubbles until later. When all of the data flows are connected then examine each bubble to determine if some data flow occurs within a bubble to achieve the bubble output. If so, then break down the bubble into subprocesses and create lines for the new data flows discovered. If your working paper is getting flooded with lines at this point, it may be time to consider a leveled DFD approach.

Basically with the leveled DFD the first sheet of working paper contains the set of lines and bubbles that were thought of on the first cut while subsequent sheets contain the internal development of bubbles that were determined to contain internal data flow. The leveled DFD system enforces top-down data analysis for large systems which in turn naturally induces modularity in system design development. Figure 4.5 is an example of a first cut system DFD. For convention purposes the bubble which spawned the internal data flows will be called the parent and the bubbles that result are called children. For numbering clarification a child is always given a unique number which is prefixed by the parents bubble or process number. As a correctness check, always be sure the inputs and outputs of the children correspond to those of the parent and vice versa. It is also wise to only expand one bubble at a time to insure continuity of thought. Data bases and files accessed or modified by a bubble should appear on the high level diagram with the parent and the appropriate lower level diagram with the child. To be sure, upon further analysis a child may develop children of its own and in this way various levels of a system would be created. Figure 4.6 shows how one bubble of the HSCLCS was decomposed to form new levels. Note that this particular example does not balance parent and child inputs and outputs; so further refinement is required to capture the correct data flow.

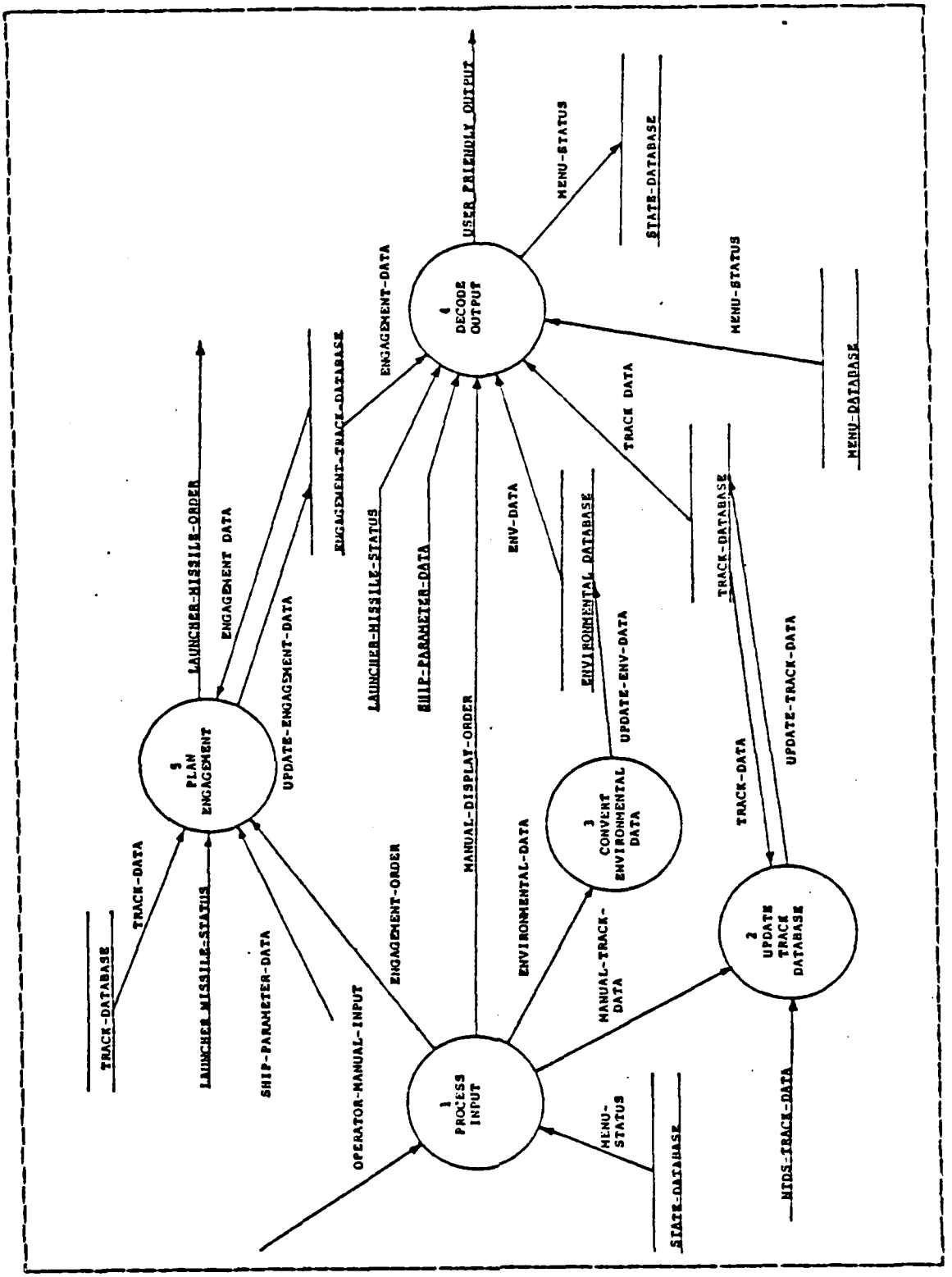


Figure 4.5 HSCLCS System Flow Diagram.

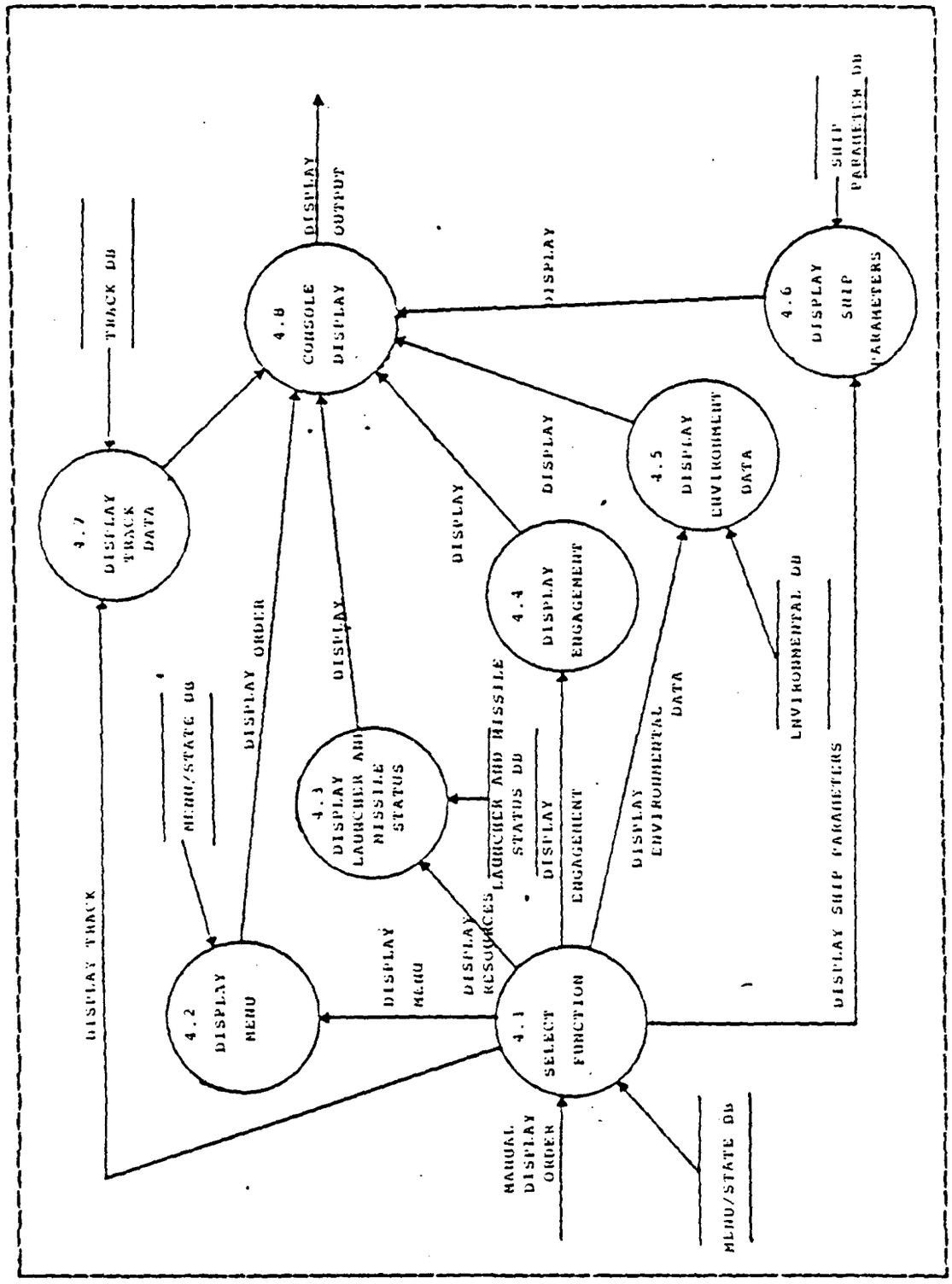


Figure 4.6 HSCLCS Decode Output DFD.

After your paper is filled with lines and bubbles, you should label the data flows. Make sure the names of the data flows are honest, concise and descriptive. Be careful not to group disparate items together into one data flow when they have no business being treated as a whole. If the name is not very obvious, it is possible that you need to repartition or break down the flow into levels. The naming process is designed to help you catch errors in your data analysis so be prepared to back up and reconsider at this point.

After the data flows are appropriately labeled, it is time to label and number the process bubbles. Use similar guidelines for naming the bubbles as you did for the data flows. Additionally,try to construct names with a singular action verb and singular object. If you find yourself caught using two verbs for one bubble, it may be time to repartition.

After one iteration of the DFD process, a good practice is to set it aside for awhile before beginning the refinement process. The refinement process consists of examining each bubble and data flow line to determine if further decomposition is required. Information continuity is required on all refinements in that all incoming and outgoing data flows in a refinement must have appeared on the previous version. Figures 4.7 and 4.8 show a initial decomposition of a process bubble and a subsequent refinement. The iterative process continues until the analyst feels that all bubbles and data flows have been completely developed or until further decomposition would not be of any practical use in his opinion. Clearly, experience will best teach the analyst when the bottom level is reached. Furthermore, final versions of DFD's from this stage of the design methodology may be required to be modified during the next phases of the methodology.

Examples of DFD development for the HSCLCS are contained in Appendix A.

c. Using the DFD

The initial use of the DFD is to convert this product into a Function Hierarchy via the transform/transaction analysis technique described in the next section. The Program Manager will use the DFD's to familiarize himself with the basic data flow of the design of the system graphically without having to trace the flow of data through a lengthy algorithm, the Broad Specifications, or the final design. This initial graphic understanding of the system to be managed will also allow the Program Manager to more easily understand the final design itself and to be able to quickly conceptualize the flow of information refered to in the design decisions documentation.

The data flow analysis, completed in the form of data flow diagrams, will lay the corner stone for the development of the design. This process must be done carefully to insure that the foundations for modularity and implicit information hiding are established from the beginning of the system development process.

2. Transform/Transaction Analysis

a. Definitions

Transform/transaction analysis is an algorithmic technique for developing a Hierarchy of Functions which is dependent only on the structure of the input product, the Data Flow Diagrams. As the method name implies, there are only two high-level structural forms indigenous to data flow diagrams: transform flow and transaction flow. The method supposes that certain fundamental characteristics exist in all software systems: data must be input, manipulated, and

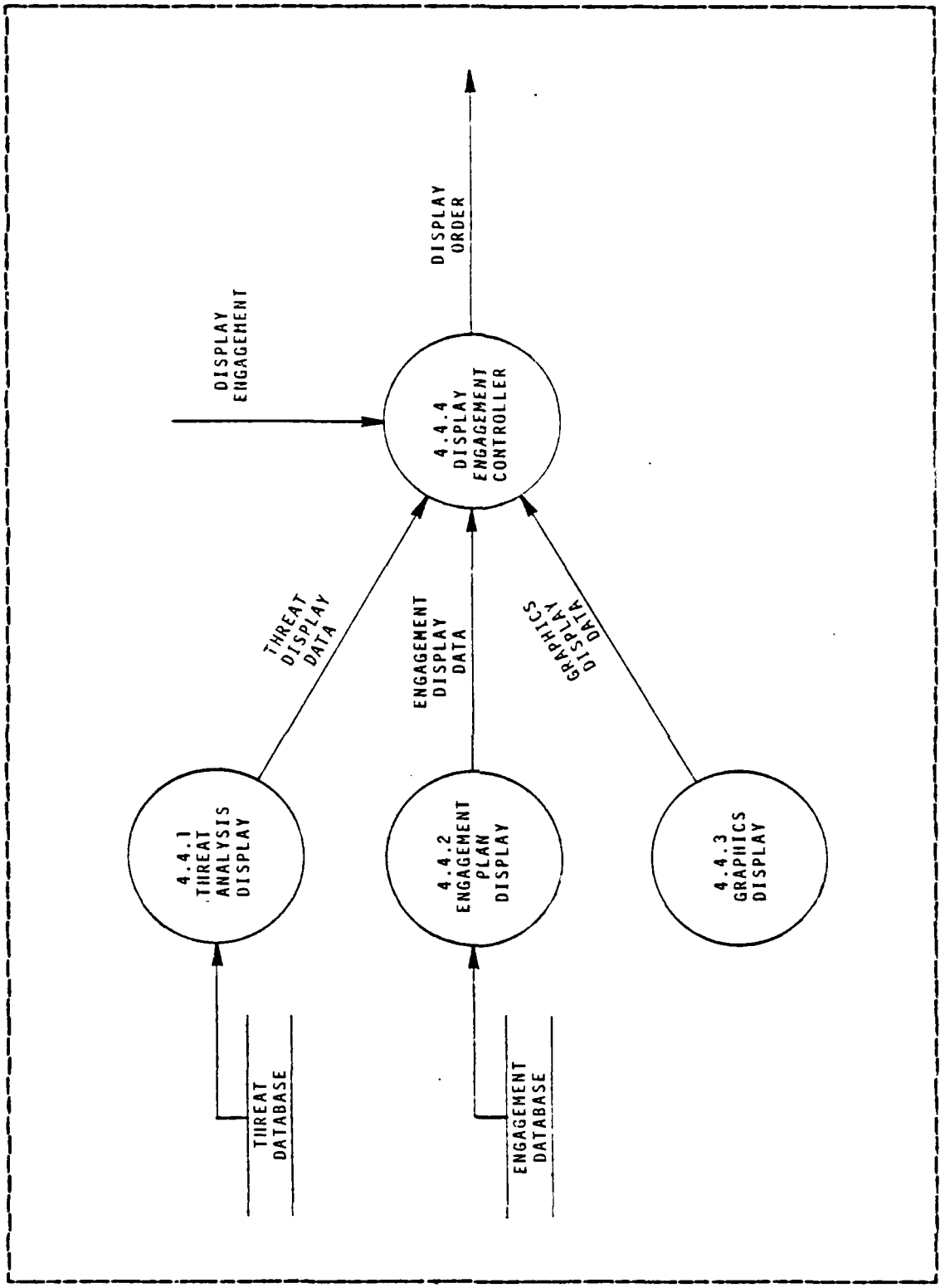


Figure 4.7 HSCLCS Display Engagement DFD.

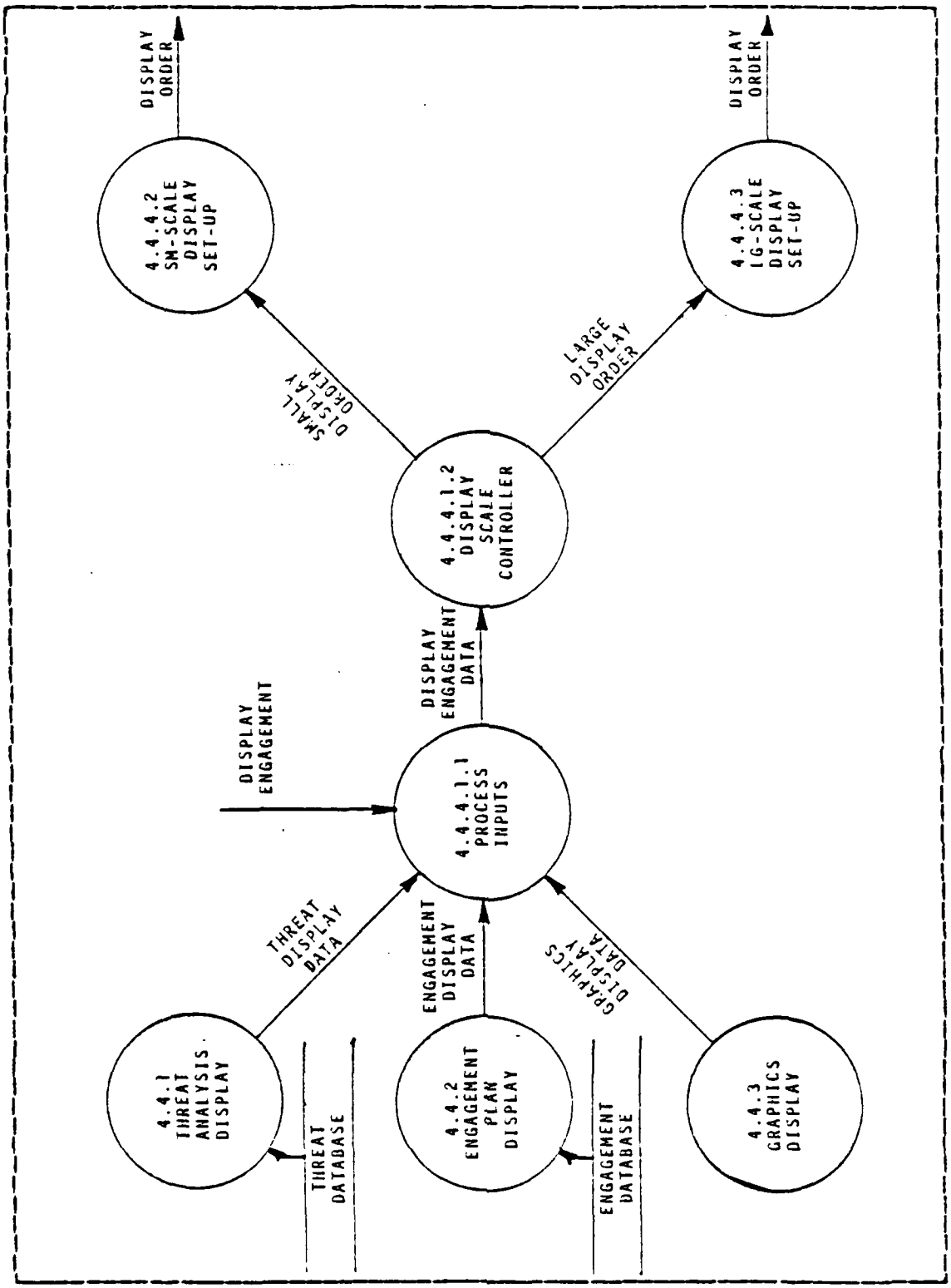


Figure 4.8 HSCLCS Display Engagement DFD Refinement One.

output. These characteristics are broad enough in nature to make the technique widely applicable to many types of software development. Specifically, the method is highly compatable with the development of real-time systems making it ideal for our purposes. The reader desiring further discussion of the technique should consult Pressman [Ref. 5].

Transform flow, our fundamental system model for all data flow, envisions the system as inputting and outputting data in an "external world" form and processing (transforming) of information in an internal form. Transform flow is necessary to accommodate both the user who must input and interpret data in the external form and the computer which must process data in the internal form. Simply stated, if the flow of information can be viewed over time as: (1) an afferent flow from the external representation of the inputs to the internal representation; (2) a process flow where the internally represented data is manipulated to produce the desired results; and (3) an efferent flow from the internal representation to some appropriate external display for the information then transform flow is present. Figure 4.9 illustrates the transform flow of information. Transform flow, as a basic model for all software development, characterizes systems very simply. They input data, change it to an internal form, process it, change it to a suitable output structure, and output it.

To solidify the above discussion, we must define afferent and efferent flow which is the key to the characterization of transform flow. Afferent flow is information flow along paths which cause the gradual transformation of data from an external format to an internal format. The transformation can be viewed as a funneling of the information through external/internal interface translators toward a central processing point, a transform center. Efferent flow is the flow of information outward from the transform

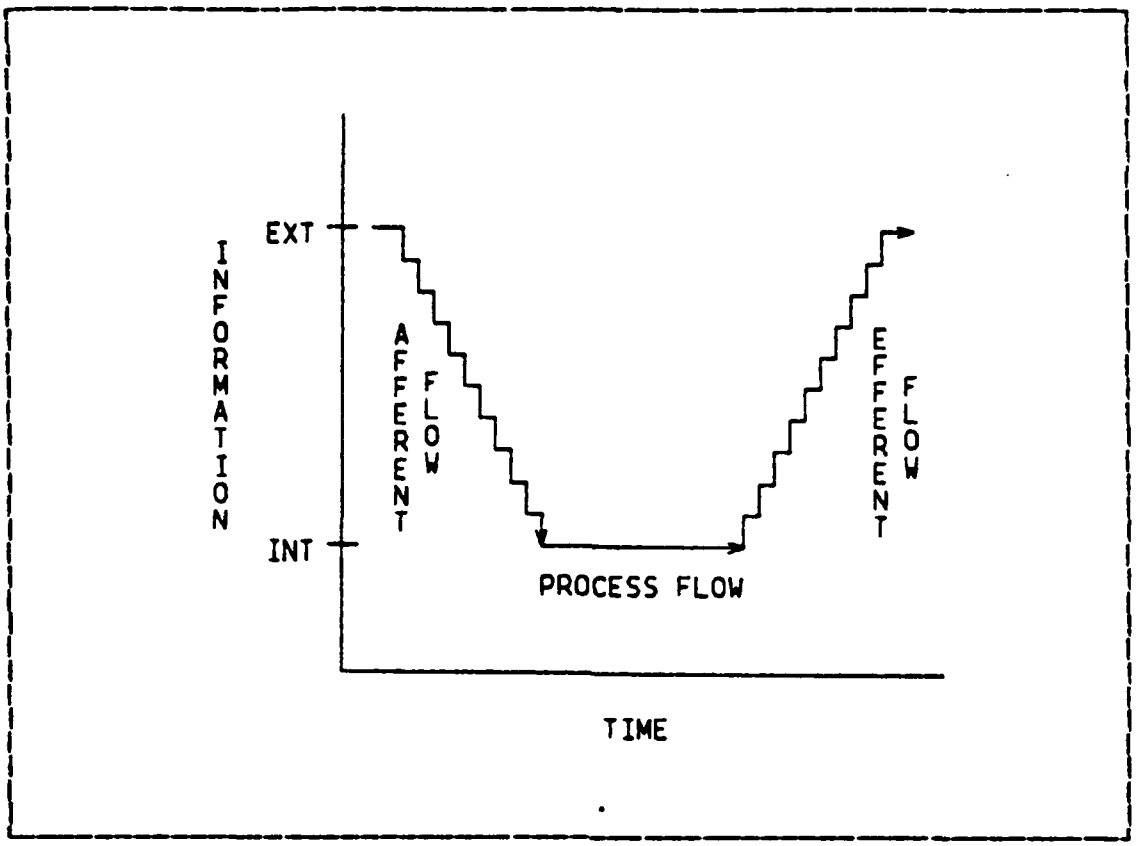


Figure 4.9 Transform Flow.

center through internal/external interface translators to the devices which will display the results of the processing to the system user.

Transaction flow is characterized by a process, called a transaction center, which takes an external impetus and causes the data flow to be directed down one of several paths emanating from the transaction center. The path taken is determined by the value of the input. Figure 4.10 shows the generic form of a transaction flow. An easy visualization of transaction flow is to compare it with the standard case statement. The case statement structure corresponds to the transaction center, the case variable value is equivalent to the external impetus (input), and the subprocess

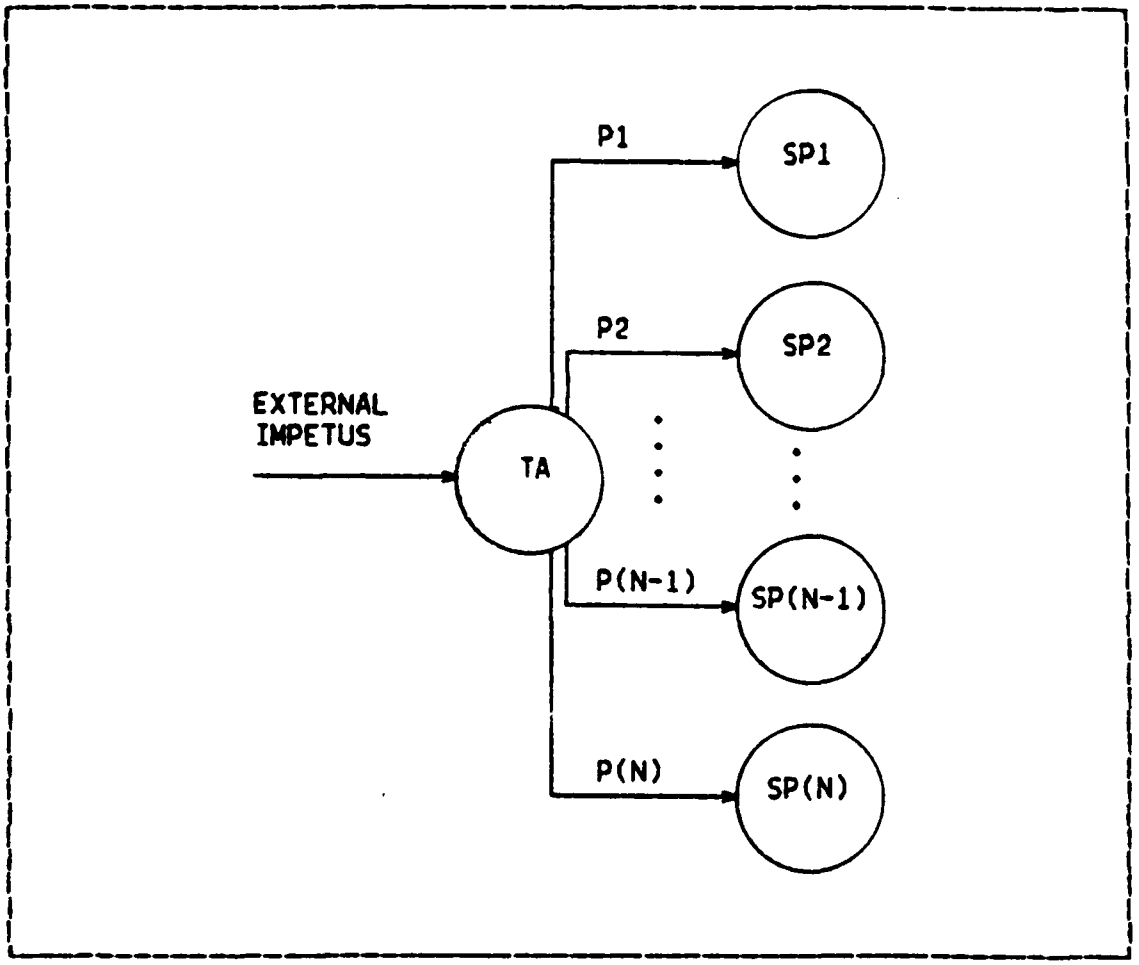


Figure 4.10 Transaction Flow.

called when executing the case statement corresponds to the action path taken in the Data Flow Diagram.

b. Procedures: A Case Study

To present the transform/transaction analysis technique, a case study of the HSCLCS Display Engagement Module will be used. The Data Flow Diagram pertinent to the case study is shown in Figure 4.8. A general rule applicable to this analysis is that the entire refinement process of the Data Flow Diagrams must be completed before commencing

the procedure. Otherwise, the proper structure of the Function Hierarchy cannot be assured. The procedure detailed below provides a template for a generic system. In some relatively simple developments, all of these steps may not be needed, e.g. secondary flow analysis, and can therefore be omitted.

(1) Flow Characteristics. The first step of the procedure is to determine the characteristic flow of the data. It is possible for both types of flow to exist on a single diagram; this is the case for our example. Under these circumstances the dominant flow pattern must be determined. In the case study, the transaction flow about the bubble labeled "Display Engagement Controller" appears to be dominant.

(2) Marking the Diagram. Next the Data Flow Diagram is annotated to show the various flow boundaries. Because the transaction flow is dominant, we will apply the rules for marking the transaction flow first and look for the afferent/efferent boundaries to mark the transform flow second. The rules for transaction analysis begin with finding and isolating the transaction center. As the definition states, the transaction center is that procedural bubble which contains multiple radially emanating data paths. Figure 4.11 shows the isolation of the transaction center for the case study. This identification of the major flow will ultimately develop the upper level modules on the Function Hierarchy. To provide details for a good Hierarchy Chart, further refinement of the flow characteristics must be performed. Since all of the secondary flow in the case study is transform in nature, the next step is to locate the transform centers. They are easily found by observing the afferent flow into selected procedural bubbles and the efferent flow out of others. In the case study, the secondary flow on the left side of the transaction center is detailed

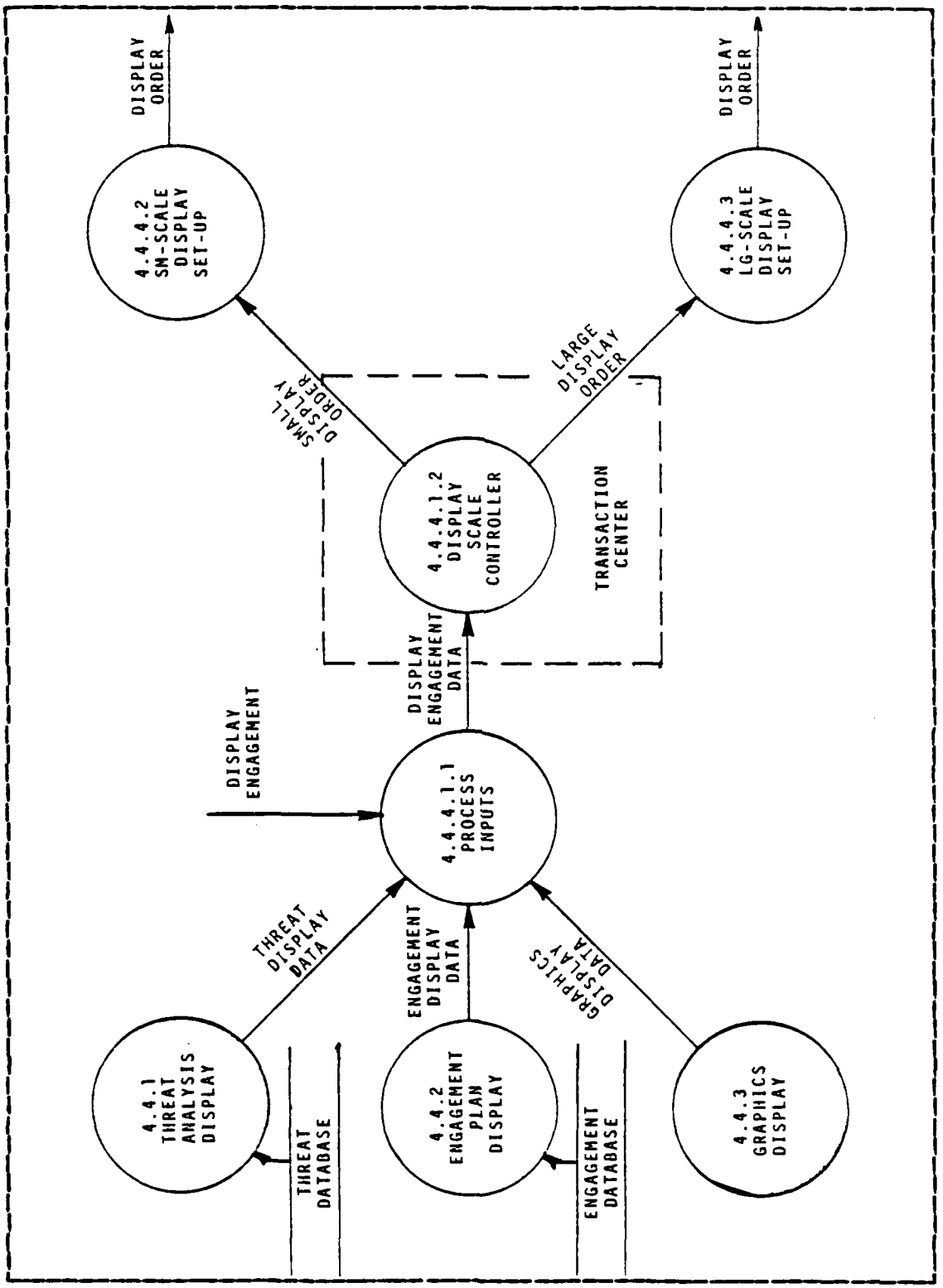


Figure 4.11 Isolation of the Transaction Center.

while on the right side it is trivial (see Figure 4.12). The right side is trivial because the flow boundaries are in lowest terms: a single datum afferent flow; a single processing bubble; and a single datum efferent flow. Thus, one would expect the modular breakdown on the input side of the hierarchy to be somewhat more detailed than that on the controller side. Later this will be shown to be the case.

(3) Hierarchy of the Dominant Flow. Once the Data Flow Diagram is appropriately marked, the first cut hierarchy for the dominant flow is generated. The fact that flow is the key to generating the hierarchy supports the supposition that the structure built during the data flow analysis will be maintained. Both types of flow have strictly mechanical means to arrive at the first cut hierarchy. This is because of the way data flow diagrams are partitioned when marked.

When the dominant flow is transform in nature then the first-level factoring produces a two level hierarchy. The upper level is a control module with a generic name chosen to illustrate the global function of the procedure. The second level contains three generic control modules with the following functions: one coordinates the afferent information, the second controls the processing, and the third coordinates the efferent information. The process bubbles controlled by these three modules are captured by the afferent/processing/efferent boundaries marked on the Data Flow Diagrams.

Should the dominant flow be transaction in nature, the first-level factoring produces a three level hierarchy. The upper level performs the same function as its counterpart used in transform flow. The middle level consists of two controllers: one for controlling modules which handle the input flow to the transaction center and one for controlling modules which handle the individual

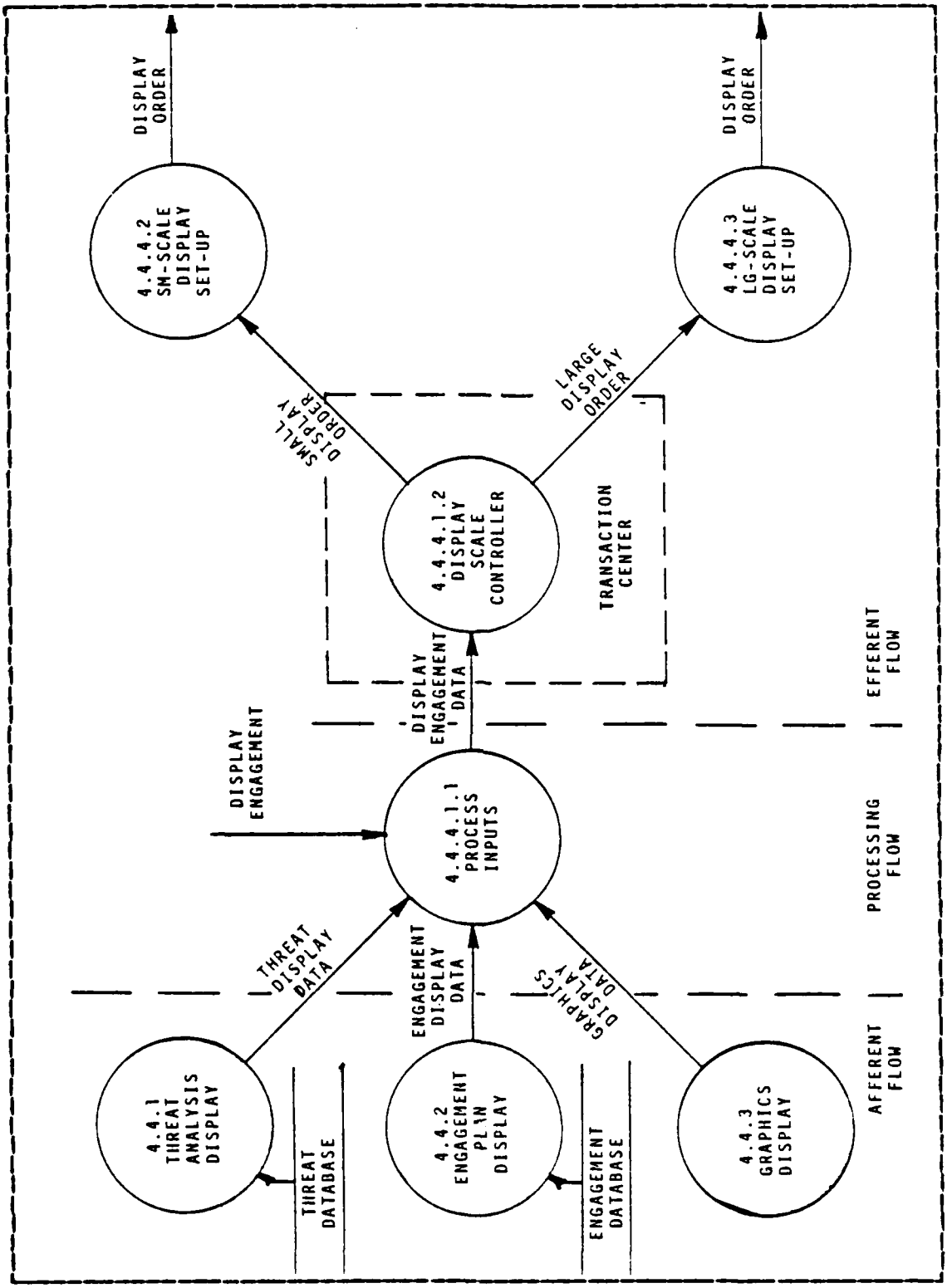


Figure 4.12 Marking the Secondary Flow.

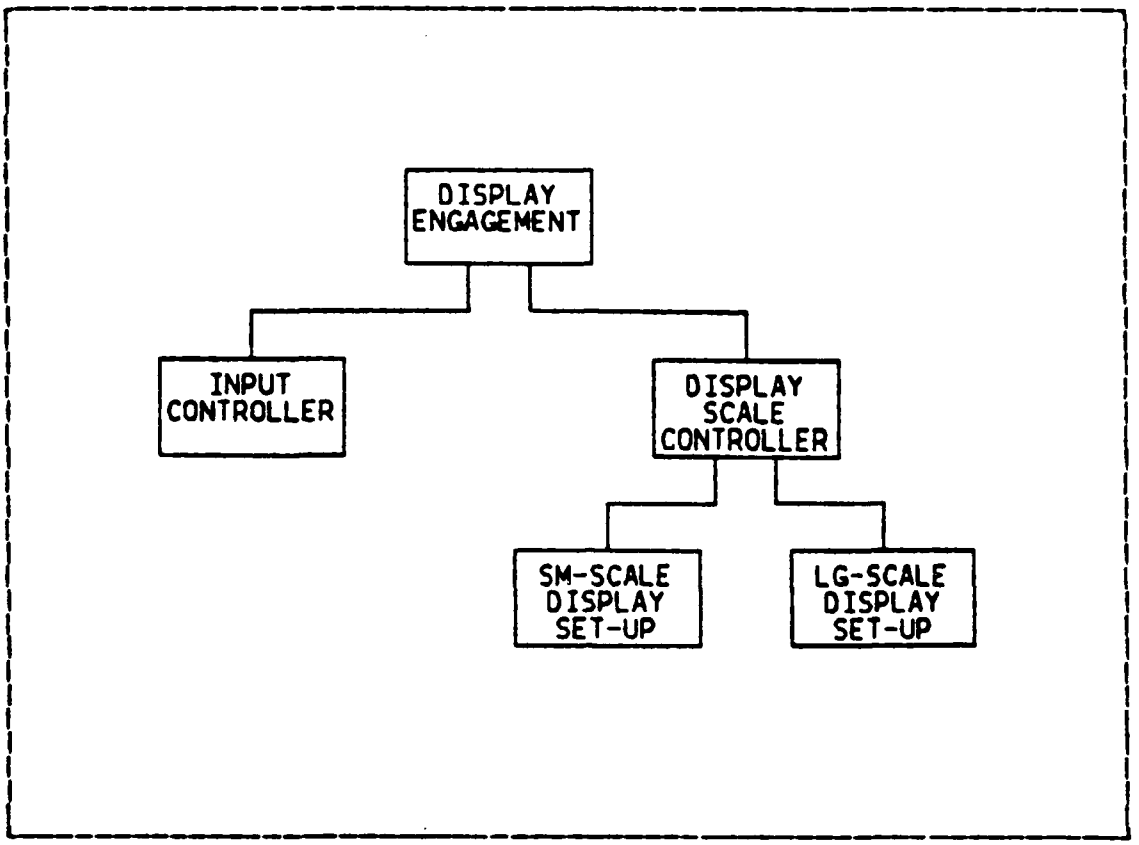


Figure 4.13 Dominant Flow First Cut Hierarchy.

paths emanating from the transaction center. The bottom level consists of a group of modules each corresponding to a single data path out from the transaction center. Figure 4.13 shows the first cut hierarchy for the dominant transaction flow of the case study.

(4) Hierarchy of Secondary Flows. The first cuts of the secondary flows, which are handled next, are performed in exactly the same manner as the dominant flow. The only difference is that the top level module, the controller, for secondary flow must be identified as some module on the first cut dominant flow hierarchy. Because the secondary flows are marked in relation to the markings of the dominant flow and cannot cross already existing flow

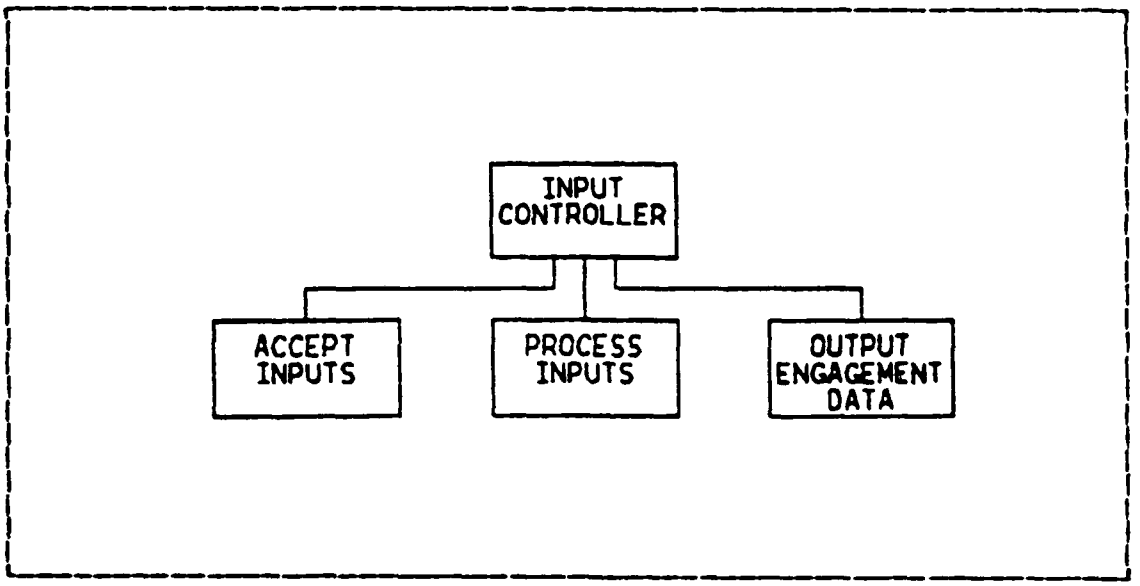


Figure 4.14 Secondary Flow First Cut Hierarchy.

boundaries, secondary flows are always encapsulated within either the dominant or another secondary flow. Therefore the top level controller of a secondary flow must map into some lower level module of the dominant flow's (or a controlling secondary flow's) first cut hierarchy. Finding this lower level module is easy; it is the one which performs the labeled function on the Data Flow Diagram. Figure 4.14 shows the first cut hierarchy for the secondary flow.

(5) Second-Level Factoring. Secondary factoring is concerned with developing the lower level modules in the hierarchy. It is basically a mechanical process of locating modules which perform the same functions as their data flow diagram bubble counterparts. The bubbles contained within the flow boundaries created by marking the diagrams are required to be mapped into modules subservient to the controller for that particular subflow (i.e. the afferent, processing or efferent transform flows or the input or dispatcher transaction flows).

It is not mandatory to have a one-to-one mapping between bubbles and modules although the degree of mechanicalness of the process is dependent upon this. This step should be performed as mechanically as practical to preclude loss of information due to premature refinement. However, mapping strictly by mechanics without regard for obvious simplifications fails to decrease the complexity of further factoring. Practical considerations dictate the outcome of the second-level factoring. Figure 4.15 shows the second-level factoring for the case study. It was done in a mechanical fashion so that the refinement techniques discussed below could be more adequately shown.

(6) Refinement Heuristics. The first cut structure of the hierarchy diagram has many rough edges. The smoothing process is not well defined; it applies a series of heuristics to the Function Hierarchy to refine the system structure. These refinements are necessary to promote the software principles discussed throughout the thesis. The following heuristics, offered by Pressman [Ref. 5], meet our needs:

1. "Evaluate the preliminary software design to reduce coupling and improve cohesion." If a module encompasses multiple functions, the software structure will suffer a loss of cohesion. Explosion of the module into a set of single-function modules regains the cohesion. If a module has an unreasonably complex interface, coupling will increase. Implosion of the function into the parent module will simplify the interface. Note that implosion and explosion have opposite effects on coupling and cohesion. The optimal balance between coupling and cohesion is the goal and drives the module refinements.
2. "Attempt to minimize structures with high fan-out; strive for fan-in as depth increases." An example of

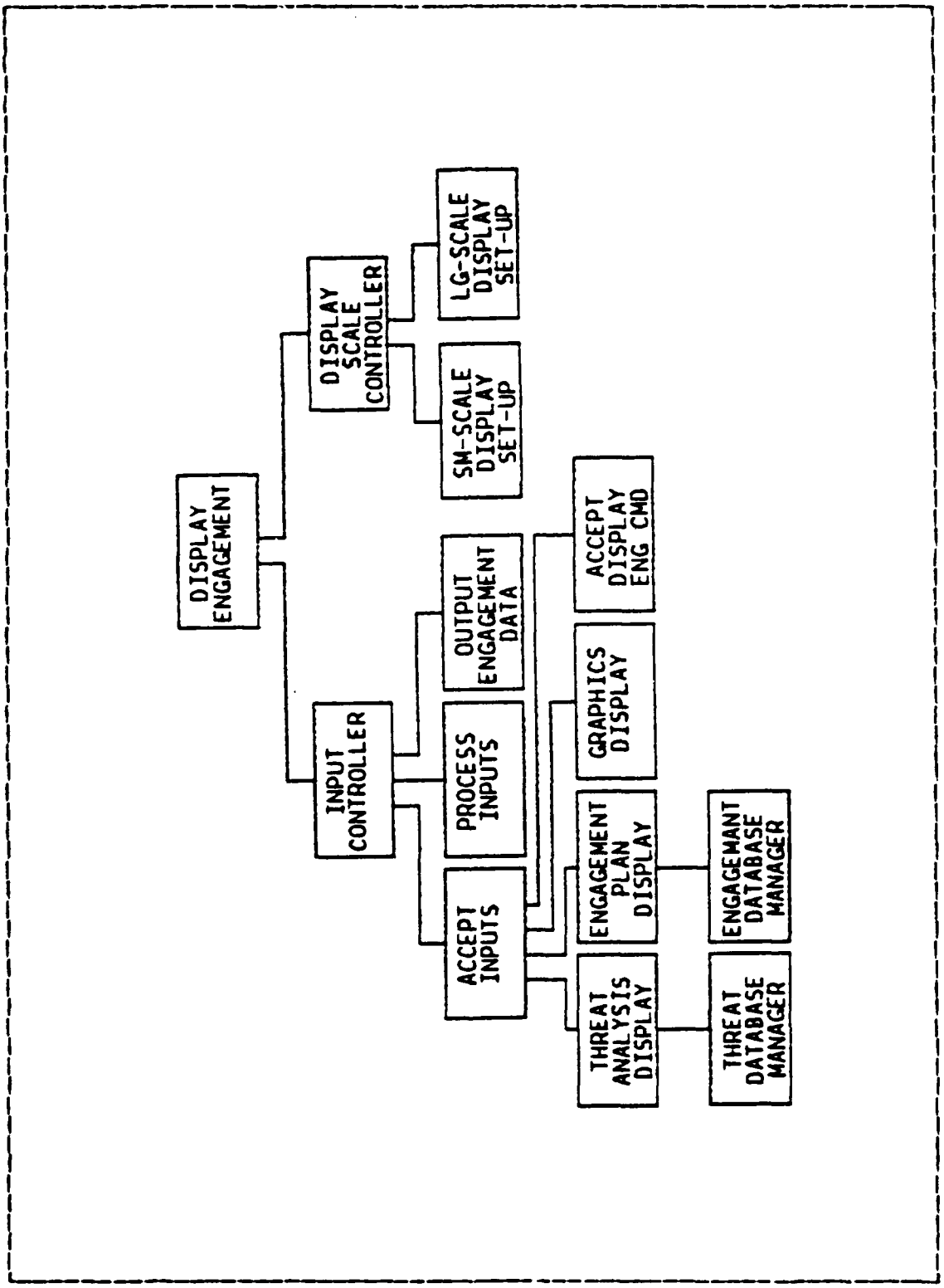


Figure 4.15 Complete Second-Level Factoring Hierarchy.

a high fan-out structure is a tree. This type of structure does not attempt to abstract similar parts from modules and make them subprocedures to a multitude of higher level modules. It is therefore a wasteful structure. Fan-in at a low level generally indicates a well abstracted structure with singular purpose modules.

3. "Evaluate module interfaces to reduce complexity and to improve consistency." The parameters passed to a module must be simple and consistent with the function of the module. Otherwise low cohesion and confusion on the part of the module user will result. If a complex interface is necessary to reasonably perform the desired task then all the modules in the immediate area should be reevaluated.
4. "Strive for single-entry, single exit modules, avoiding pathological connections." This simply warns us to develop modules which are entered at the top and exitted at the bottom. Pathological connections are branches into or out from the middle of a module. They must be religiously avoided.

(7) Refinement Process. The refinement heuristics listed above fall under the general category of module independence promoters. Seeking high cohesion and low coupling by the implosion/explosion routine is necessary in varying degrees to gain this independence. The degree of necessity is dependent upon the level of refinement of the Data Flow Diagrams. As the DFD's capture more detail, the number of correct, efficient Function Hierarchies decreases because detail limits design options. Thus, as the set of DFD's approach maximum refinement the transform/transaction analysis process approaches a fully mechanical algorithm. But because the level of refinement of the Data Flow

Diagrams is realistically (and desirably for complexity reduction reasons) rough, heuristics are needed to refine the Hierarchy Chart. As indicated, these heuristic procedures are not mechanical. They rely on common sense decisions by the user to transform the current structure to a form which simplifies the design. The final arbiter is human judgement.

In the case study, several refinements can be made. Refer to Figures 4.15 and 4.16 throughout the narrative. First, to aid abstraction and control coupling all references to data in databases will be through a generalized data interface, an abstract database management system (DBMS). Thus, the two database manager modules have been replaced on the final hierarchy by a generic controller for all calls to databases. It is beyond the scope of this discussion to refine the DBMS module. Next, the "Accept Display Engagement Command" module, which performs no processing, was for simplicity reasons imploded into the "Accept Inputs" module. Third, the processing of the "Process Inputs" module, which is done by the "Accept Inputs" module, and the processing of the "Output Engagement Data" module, which consists of only a parameter pass, are not necessary to control cohesion. This type of redundancy is common to secondary flow analysis. Consequently, they were imploded into the "Input Controller" module. Next, because the function of the "Input Controller" and the "Accept Inputs" modules are identical, the structure can be simplified to a single module to reduce coupling with no loss of cohesion. Note that the final name chosen for this second level module was "Process Inputs" rather than "Input Controller" or "Accept Inputs". This is because the name "Process Inputs" is the most descriptive of these three candidate names for the module. The final modification to the design, that of abstracting similar data from the two

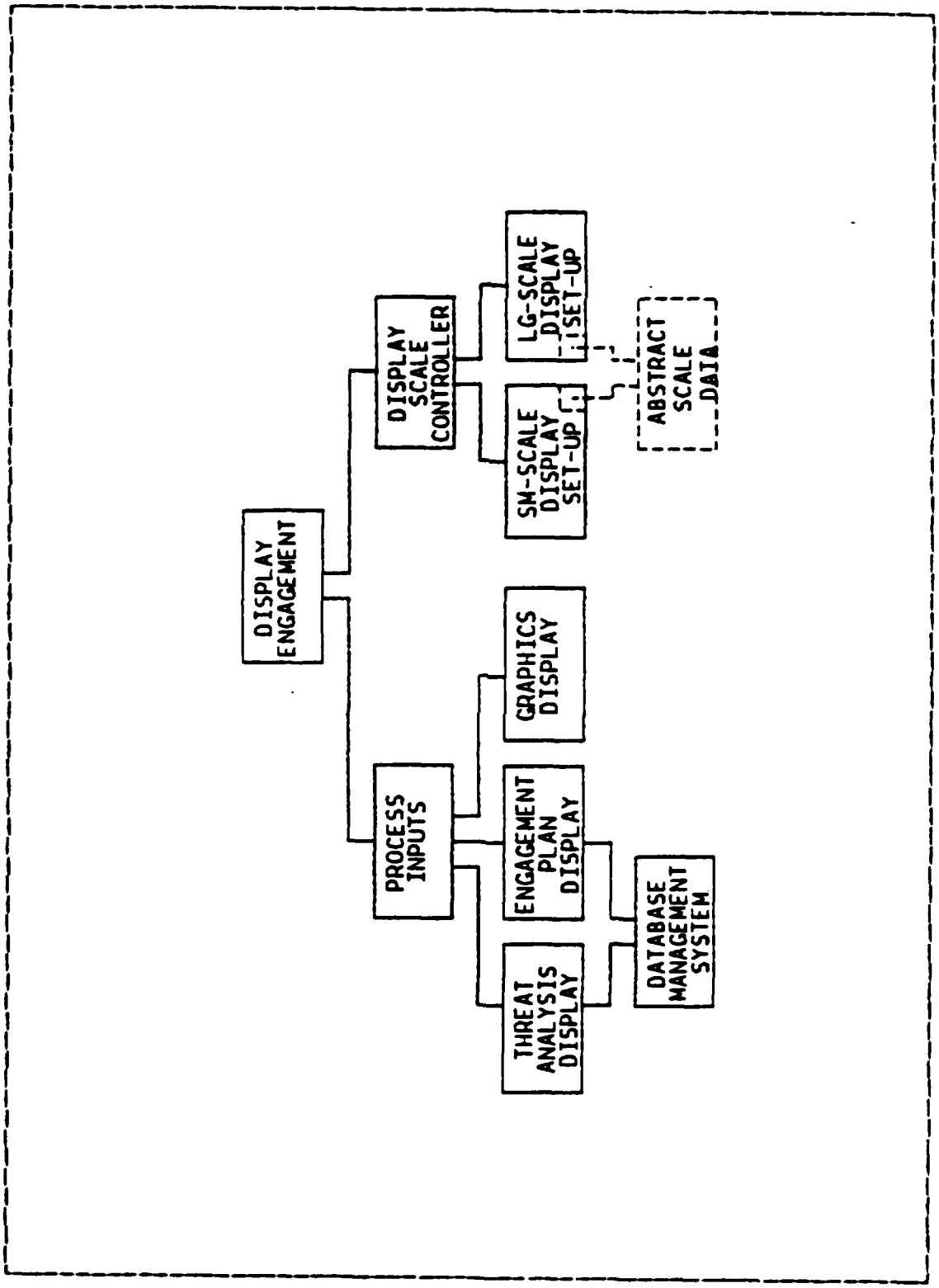


Figure 4.16 Hierarchy of Functions: Final Refinement.

scaling modules, attempts to "fan-in" the structure. It is shown as a dotted procedure on Figure 4.16 to show that factoring is possible but not assured.

3. Modular Development

Modular development as a formalized procedure is a technique for transforming the properties of the Hierarchy Chart structure into Module Descriptions. All of the data required to define the modules in very general terms is contained in the structure of the Function Hierarchy. The transformation, while seemingly subtle in nature, is a very important step in the methodology. It provides an elegant way of changing the system specifications from their totally graphic form into a short narrative form. This transition is a necessary first step toward using an SDL, an ADA SDL in our case, to further design the system.

The components of a Module Description capture the necessary details of modules commensurate with their high-level position in the design refinement process. While the actual format of the Module Descriptions is not critical, the contents contained within them is. The components provide a complete description of the module for this stage of the design. The definitions of each of the Module Description parts are listed below:

1. Module Name. This name must be the same as the one on the corresponding module of the Function Hierarchy.
2. Module Function. This narrative provides the purpose of the module in broad terms. It should reveal a singular purpose in order to meet the criteria of a module.
3. Supervisory Modules. These are the modules that call this module. It is the interface with the supervisory modules which is explained below.

4. Module Interface (Parameters). Here the ADA SDI-styled parameters are listed and further explained in a short narrative. The narrative reveals the basic structure for the data type of the parameters. The interface is a bridge between this module and all of its supervisory modules.
5. Subordinate Modules. These are the modules called within the body of this module. Their interface definitions are handled by the subordinate module's interface.
6. Design Decision Encapsulation. This is the singular decision which the module hides within its body. It must be a singular decision to meet Parnas's criteria for a module. As the module is further developed, additional lower level decisions will usually be necessary. To maintain the Parnas singularity requirement, these decisions must also be encapsualted in their own procedural structure. Thus, the Hierarchy Chart is a dynamic product being continually refined as design questions arise and decisions are made to accommodate them.

Figure 4.17 shows a recommended style for module descriptions. The example is one of the modules developed in the case study, THREAT_ANALYSIS_DISPLAY. Viewing Module Descriptions as the first step of the design in the SDL and therefore the first products of a step-wise refinement process for the system design, we present the descriptions in ADA SDL comment form. Because the Module Descriptions contain the necessary details to fully define the modules, they can be used as the user interface in the ADA SDL design.

The modules should be developed independently by first producing the Module Descriptions in separate files

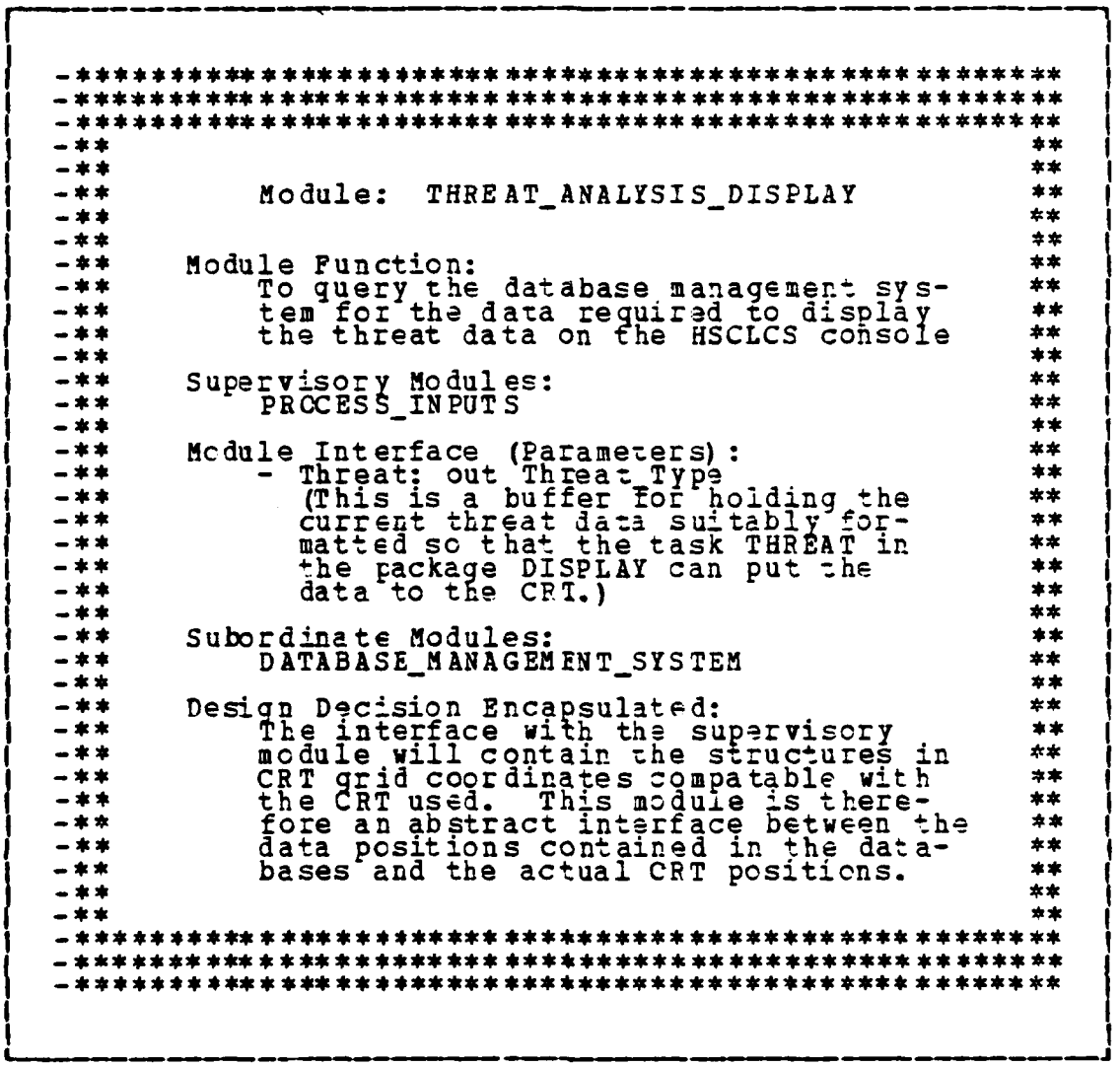


Figure 4.17 Sample Module Description.

and then writing the SDL "code" appended to the Module Descriptions. This accomplishes two goals. First, it preserves module documentation in its most logical place and most desirable form. Second, it provides physical encapsulation of the module encouraging its independent use in some other system. This is an initial step toward programming-in-the-large.

4. Transition to ADA Design

The transition to ADA design function completes the system design. Using the method for segregating and documenting the Module Descriptions explained in the last section, the transition builds the narrative structure into an ADA-compilable System Design Language "program". It incorporates the stepwise refinement approach of detail abstraction throughout the process. The steps involved in this function are simple to comprehend and follow for those moderately versed in the stepwise refinement technique.

Stepwise refinement is a well-known concept in the software engineering community. It is not universally accepted, however, as the all-encompassing detail abstraction methodology. Because it is very useful at this point in our methodology, we have endorsed it. In a nutshell, stepwise refinement is a decomposition procedure which refines a previous, higher-level view of a module. It is different from a similar technique, top-down design, because unlike the top-down method, stepwise refinement is limited to developing only structured constructs within modules.

Before proceding with the refinement process, the Data Flow Diagrams, Module Hierarchy, and Module Descriptions must be reviewed to identify potential modules for packaging and to look at the concurrency profile (best shown by the DFD's) of the system. The packaging criteria consists of four general categories of applications for packages each with multiple subcategories. The broad applications are: named collections of declarations; groups of related program units; abstract data types; and abstract state machines. Bocch [Ref. 8] discusses these criteria in detail. Applying these criteria to the case study, notice that the three display modules along with the "PROCESS_INPUTS" module in Figure 4.16 would be candidates

for ADA packaging. This is because by packaging these modules the required inputs can be hidden from the "DISPLAY_ENGAGEMENT" module thus realizing both a grouping of related program units and an abstract state machine. The criteria for ADA tasks also serve four applications areas: concurrent actions; routing messages; managing shared resources; and interrupt handling. Again Booch [Ref. 8] explains these applications in detail. Returning to the case study, these three display modules perform functions independently of each other as shown on Figure 4.8, the DFD, (i.e. they do not operate on the same input parameters) and consequently are candidates for concurrency control using the ADA task mechanism.

The procedures of the stepwise refinement are not particularly rigid. As previously stated, the main idea is to decompose modules into structured constructs. The following steps form our methodology for completing the design.

1. Write the procedure, package, function, task, etc. specification including all of its parameters.
2. Write the body name of the procedure, package, function, task, etc. with its parameters, a short narrative of the basic flow, and the appropriate end statement.
3. Replace the narrative of the body by the high level constructs of the algorithm.
4. Continue refining the algorithm by adding detail until the desired purpose is clear. Give no more detail than necessary to meet the above criteria.
5. If during this process the need for design decisions arises, defer the decision by creating a subordinate module specifying only its interface.
6. Recheck the interfaces for clarity, simplicity, and completeness.

7. Determine the design decision encapsulated in the module and check that it is entered in the appropriate Module Description.

Figure 4.18 shows the "THREAT_ANALYSIS_DISPLAY"

```
task THREAT_ANALYSIS_DISPLAY;
with DATABASE_MANAGEMENT_SYSTEM;
task body THREAT_ANALYSIS_DISPLAY is
  type THR_DBMS is
     record
        SHIP_NAME:              - these types are not
        SHIP_CLASS:             - pertinent to the case
        WEAPONS:                - study and therefore
        ECM_EQUIPMENT:          - not developed
        ENGAGEMENT_PLAN:
     end record;
  THREAT : THREAT_TYPE;
  RECORD_FROM_THREAT_DB : THR_DBMS;
  END_FILE : BOOLEAN;
begin
  END_FILE := false;
  while (not END_FILE) loop
     THREAT_DB_MGR (RECORD_FROM_THREAT_DB, END_FILE);
     - develop the CRT coordinates for this display
     - consistent with the known coordinates of the
     - actual threat.  put the names and coordinates
     - in the buffer, THREAT.
  end loop;
end THREAT_ANALYSIS_DISPLAY;
```

Figure 4.18 Sample Module Design in ADA SDL.

design which completes the case study for this module. Stepwise refinement was used both to develop the specifications of the task and the flow in the task body. Because all of the specifications were accurate and the interface well-defined, this step in the methodology was easily performed.

5. Specification Refinement

One of the primary goals of the methodology is to produce better specifications and at this point in the

process the existing specifications are updated by incorporating the documented design decisions. Also if certain decisions were deferred until now such as exception handling, it is the time to include them in the specifications. On the first iteration of the methodology the input for the update process would be the Broad Specifications.

We are assuming that the Broad Specifications are well structured and in accordance with current directives. However, at each review point of the specifications they should be screened for ambiguity, confusing description, overspecification, orthogonality, and completeness. The preceding processes in the methodology will iteratively ensure completeness, but the other undesireable attributes must be found editorially. We will not belabor the reader with definitions of the attributes but they can be found in [Ref. 7].

A sample set of software specifications for the HSCLCS is given in Appendix F. These specifications were the product of a first iteration of the methodology for the system and, if approved by a Program Manager, would be the final refined software specifications.

Although this section of the methodology appears short in comparison to the long algorithms of the other methodology components, the review of the specifications is extremely important and should be given an equal if not greater segment of the methodology time. These specifications will reflect the principles incorporated into the other methodology products only if this updating process is done with care.

C. METHODOLOGY EVALUATION

In the first section of this chapter we listed and discussed the principles and goals that the methodology was

designed to produce. In this section we will show how the products produced conform to the guidelines specified.

To begin with, all of the products were created using concrete algorithms which were presented in a manner that a Program Manager could easily understand. Although the processes did include some heuristics, the thought process behind the heuristics was explained thoroughly.

Each of the products exhibit logical design flow at all levels of development. The stepwise refinement methods in each phase of the methodology insured that in no way did a cart ever get in front of a horse. From data flow diagrams to design to specifications, system design decisions were deferred until the last possible moment in order to maintain the maximum degree of flexiblility and to enforce abstraction. This logical design coupled with the algorithmic methods along with emphasis on simplicity and structure in each of the products gives the methodology one of the primary goals: understandability.

The four basic principles (i.e. modularity, abstraction, independence, and hiding) that were enforced during the phases of system development all basically contribute to the modifiability and maintainability goals of the methodology. Modularity is the first of these principles and the entire system development process steered the analyst toward abstracting common characteristics of the system into modules. The abstraction process kept details of design at the lowest possible level. By hiding design decisions within modules, a design decision change can easily be incorporated into each of the products by simply changing one module (ideally). Furthermore since there exists a fairly simple mapping between products, a revision could be implemented easily across the board. Since independence was also one of the principles, strong cohesion and weak coupling of modules also enhances relatively effortless

system software modification. Clearly, the software developed is easily modified and consequently highly maintainable.

The iterative nature of the methodology ensures that the products will be reliable. The inherent design error checking of the process allows the designer to be confident that the design will meet all previously required specifications and that the refined specifications produced by the methodology effectively encompass all design parameters.

To simply state that the major principles used to develop the individual products insures that the desired characteristics are passed from phase to phase may not be enough. Abstraction and completeness are natural byproducts of the data flow analysis methods of DeMarco, but do these traits proliferate through the Pressman and Booch module and design development processes? Does the modularity and independence gained from Pressman carry over to augment the hiding principle emphasized by Booch? The answers are emphatic affirmatives based on the iterative nature of the principle inclusion process of the methodology.

Upon close examination it is easy to see that the principle inclusion process is additive in nature. Abstraction and completeness are qualities enforced in the derivation of the Data Flow Diagrams. Since these characteristics are imbedded in the DFD's, which form the basis for the Pressman transaction analysis phase, they are necessarily a characteristic of that phase also. Additionally, modularity and independence are emphasized on top of the abstracted, complete foundation. Similarly, the characteristics brought forward from Pressman to the design development of Booch are added to information hiding which is stressed in that phase. In this way we are guaranteed that the ultimate products possess the desired traits. Iterative refinement of the processes then solidifies the placement of the principles.

An evaluation of the methodology would not be complete without some comment on ADA as the system design language, however, no in-depth analysis of ADA will be given because it is outside the scope of this thesis. Besides being the DOD language of the future, ADA, with its package and task mechanisms, is especially suited to enforce the information hiding principle that is the major emphasis of the Booch phase of the methodology. Without such a language, the implementation of this principle would be tedious if not practically impossible. In short, ADA is not just used by the methodology; without it the methodology would not be complete.

Even though we have shown that this methodology will be an effective tool for the Program Manager, it must be stressed that if all of the guidelines and principles are not adhered to rigidly thoughout each phase of the process then the products may not reflect the qualities specified by the goals. To use simply modularity without hiding in mind could result in software that is not easily modifiable while not applying all of the rules of abstraction could yield a very inefficient design. The major distinguishing trait of this methodology from others such as PSL/PSA and SADT is that the various software engineering techniques of Booch, De Marco, Pressman, and Parnas have been blended to form a process that generates products that can begin to cut the cost of software maintenance and development. To employ the process you must be well schooled in the basic principles.

It is apparent that the goals of the methodology have been achieved from the previous discussion, but only through implementation of the process can it be evaluated. The only readily obvious improvement upon the methodology would be to automate the process which is well within the realm of possibility due to the algorithmic nature of the methodology. The methodology was used to produce the design

products for the HSCLCS were are included in the appendices. For this size project the methodology appeared excellent. However, the implementors found that increased experience with the phases produced results which more closely adhered to the goals and principles. Further proof will be in the pudding.

V. CONCLUSIONS

The goal of this thesis was to develop a system to make the Program Manager's task of monitoring software development of embedded weapons systems less complex by providing him with comprehensible, easy to use management tools. In the previous chapter we have outlined and evaluated a methodology which produces these management tools, and in the appendices are samples of the products of the methodology. The methodology was simple to implement and produced good, complete system software specifications that were understandable and well documented. An explanation of the Program Manager's procedure in utilizing these software development tools remains.

The Program Manager will receive broad software specifications on which he will conduct a first cut evaluation prior to giving them to a Contract Support Service (CSS) for generation of refined specifications using the methodology of this thesis. After a specification refinement, the Program Manager reviews the methodology products and feeds the Refined Specifications back to the CSS if necessary for another iteration of the process of refinement. This process continues until optimal specifications are achieved in the opinion of the Program Manager and his staff. A close working relationship between CSS and Program Manager can accellerate the process considerably. Specifications of high quality is the most visible product of the process external to the Program Manager's office, but the other products are equally important to the management of a software system.

The Data Flow Diagrams, Module Descriptions, and design with documentation are of high value. The Data Flow

Diagrams provide an easy-to-read graphic representation of the system. The Module Descriptions and the final ADA Design produced give further simple, understandable documents that a Program Manager and especially his successor (relief) can use to grasp the details of the software. The Documented Design Decisions are even more important to the pass-down evolution that so often interrupts the continuity of a system's development. A new Program Manager can not only see the design with relative ease, but he now has a history of how and why decisions were made that led to the design. Corporate knowledge which has historically been the most frequent casualty of the turnover process can therefore be a survivor. The increased insight into the design of the proposed system also puts the Program Manager into a better position to evaluate the ultimate system contractor's design proposals.

Another byproduct of better specifications is the potential for overall system development costs to be lowered. By refining the specifications "up front", there is a reduced probability that the contractor will discover omitted items in the specifications that require costly change orders to integrate the items into the system. Since change orders allow contractors to adjust the cost of a system above the original bid, reducing the number of changes minimizes overall costs. Further in the costs lifecycle area is the capital spent on the maintenance of a system once it is implemented. Modification of software generated by this methodology is relatively inexpensive as discussed in the previous chapter. In most cases changes in requirements would not require a complete system redesign but only redesign of the module affected. Maintainance costs would then be obviously lowered.

The most intangible benefit gained from the methodology is the economy of effort gained within the Program Manager's

cffice. The algorithmic style of the methodology and the uniformity of the products provide guidelines for the software development. The lack of an adequate institutionalized procedure for software development organization in the past has caused effort tc be wasted performing non-productive tasks. Therefore, the increased efficiency due to standard development procedures can be substantial.

Software development, as mentioned before, is but a mere fraction of the total effort needed to realize an embedded system in the fleet. However, the escalating cost of software makes it contribute an inordinate amount to the total system costs including system maintenance. It is therefore imperative that the best software engineering techniques be used to reduce the exponential growth of development and maintenance expense and to ensure the Program Manager's task has minimum complexity. The methodology promulgated by this thesis is a major step in developing standardized management procedures for software development that will reduce costs and provide more maintainable weapons systems to the fleet.

APPENDIX A

HSCLCS DATA FLOW DIAGRAMS

This appendix contains the HSCLCS Data Flow Diagrams from [Ref. 1]. This set of diagrams is by no means complete and is provided as a sample methodology product. The same caveat applies to each of the appendices.

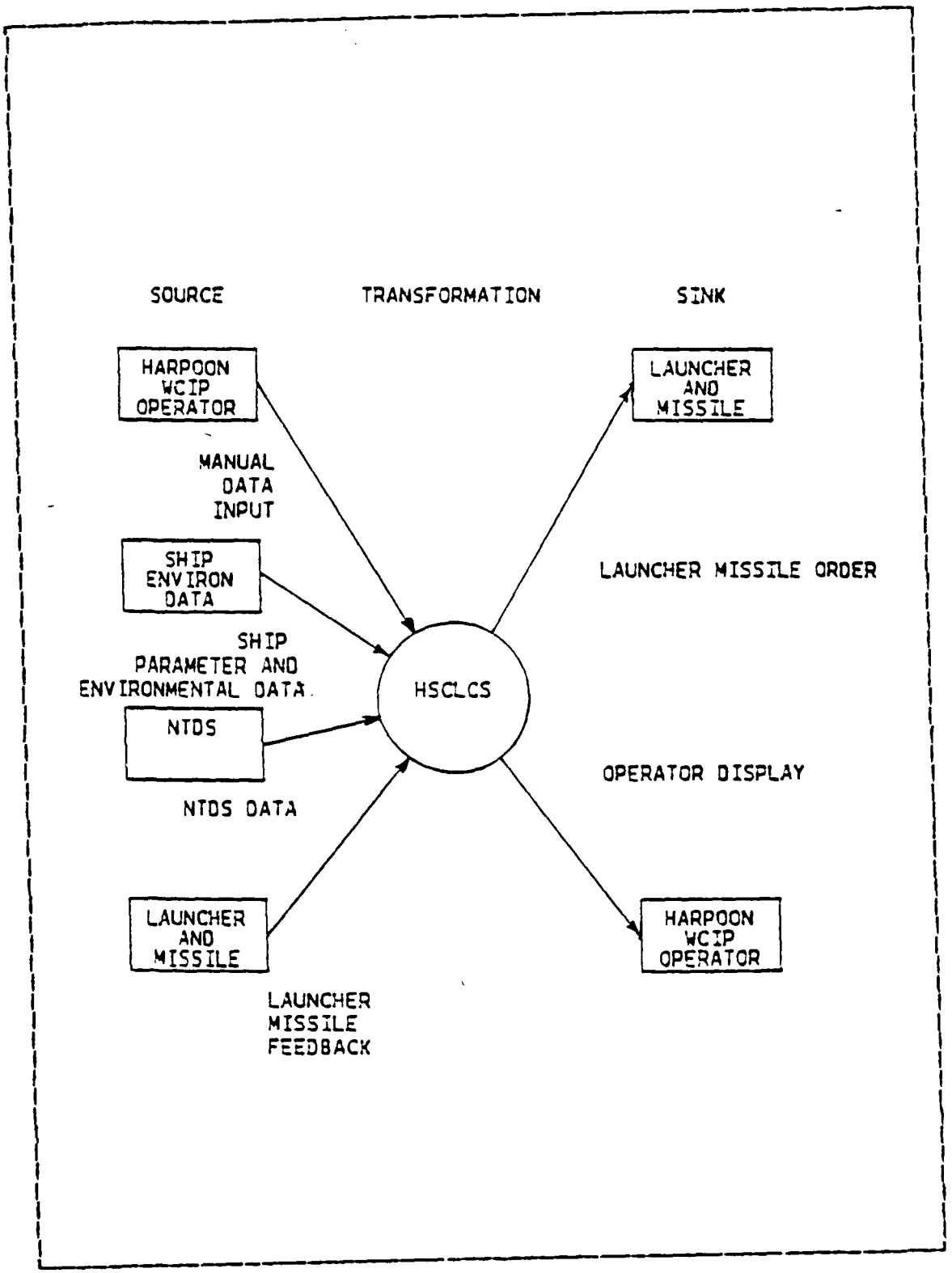


Figure A.1 Source/Sink Diagram.

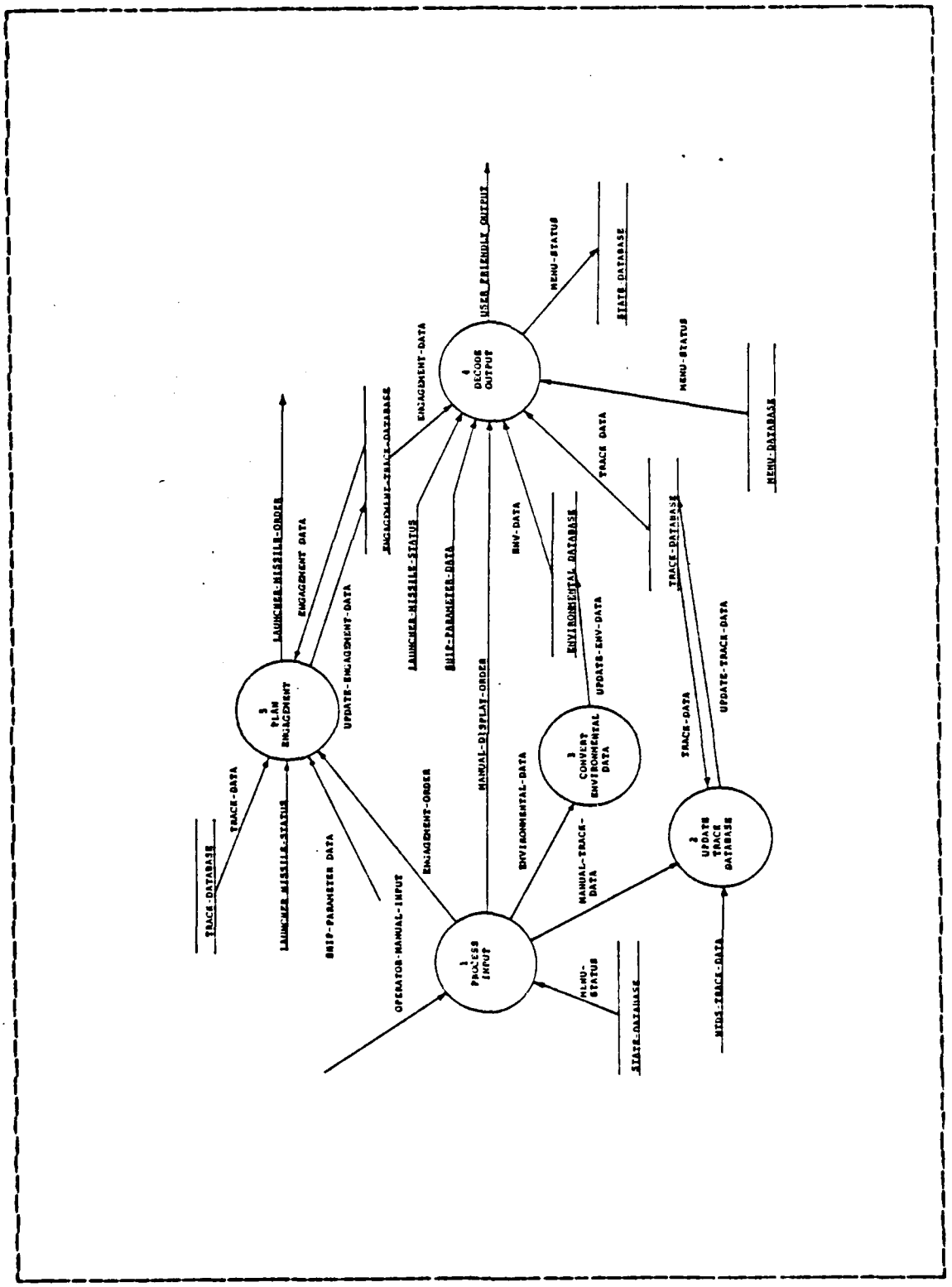


Figure A.2 System Overview DFD.

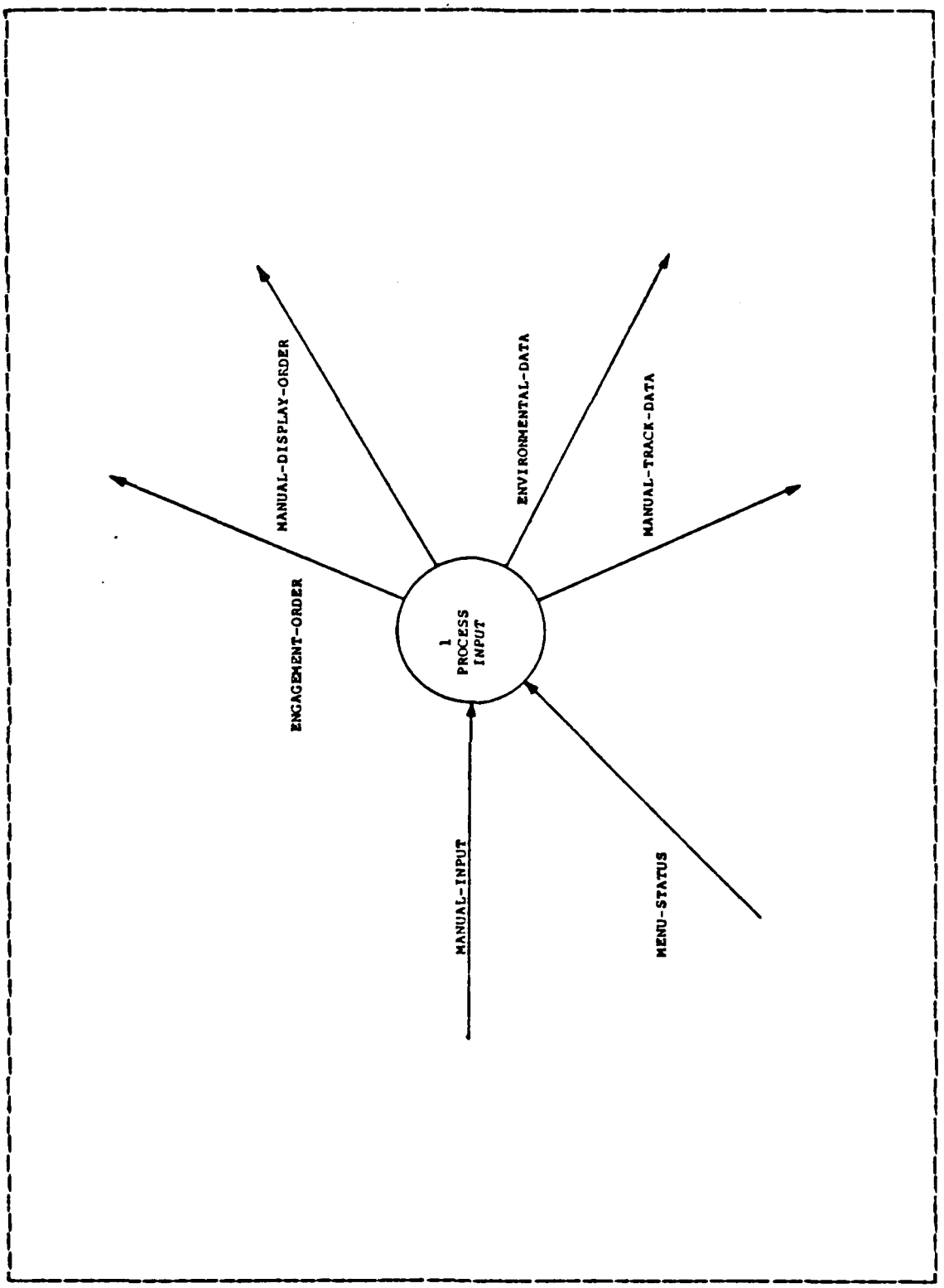


Figure A.3 Complete Manual Process Data Flow Diagram.

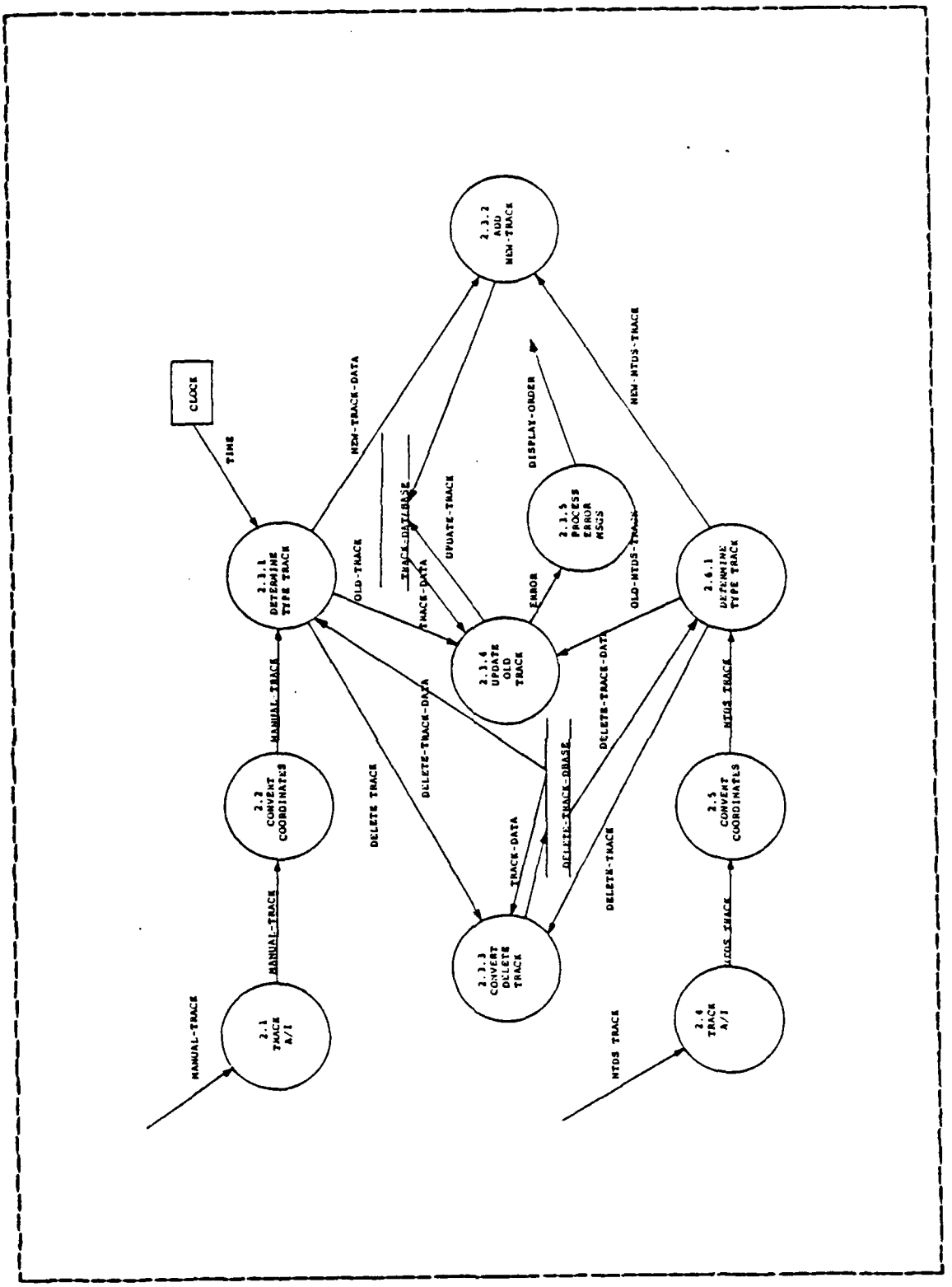


Figure A.4 Update Track Data Base DFD.

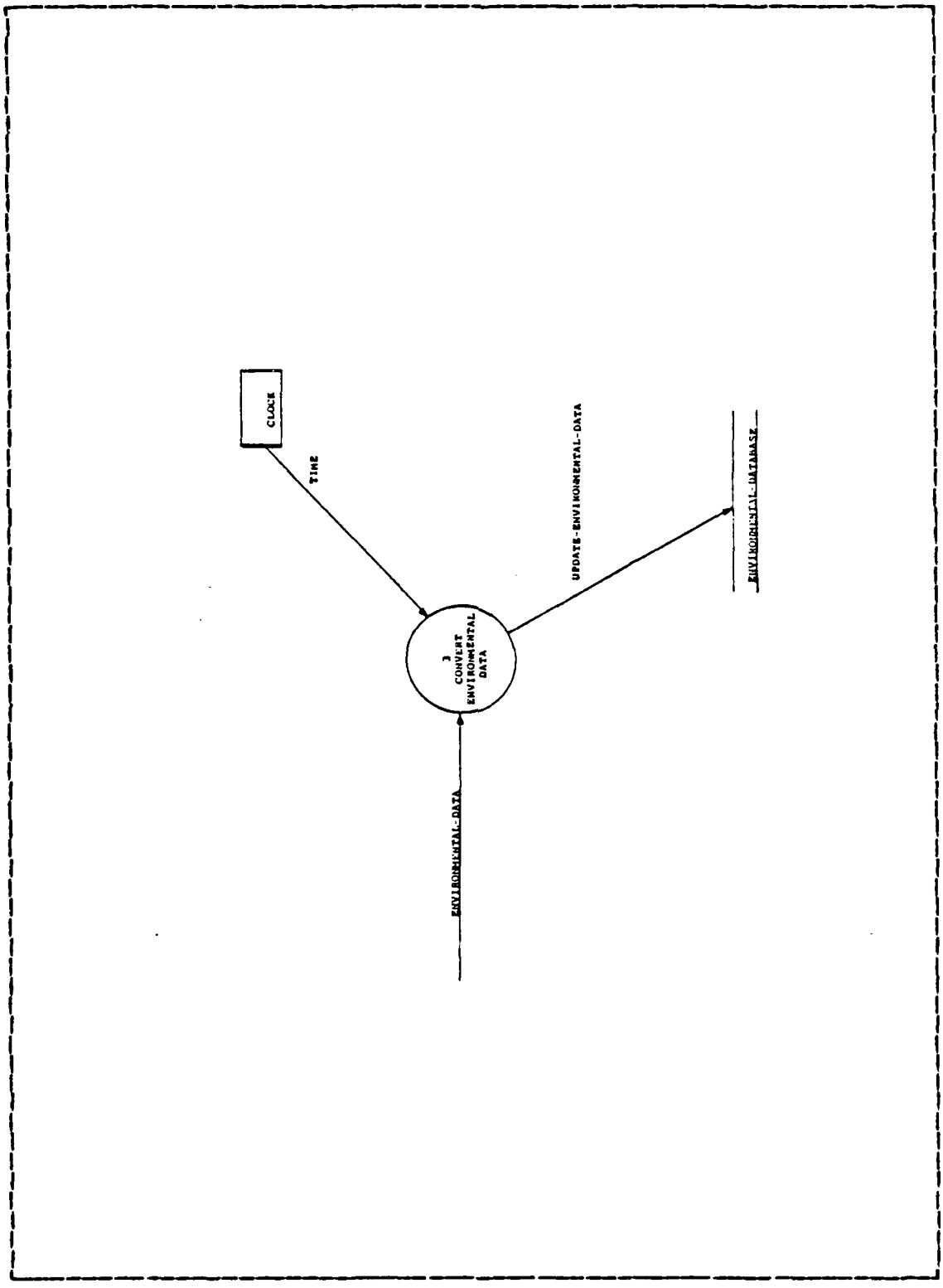


Figure A.5 Complete Convert Environmantal Data DFD.

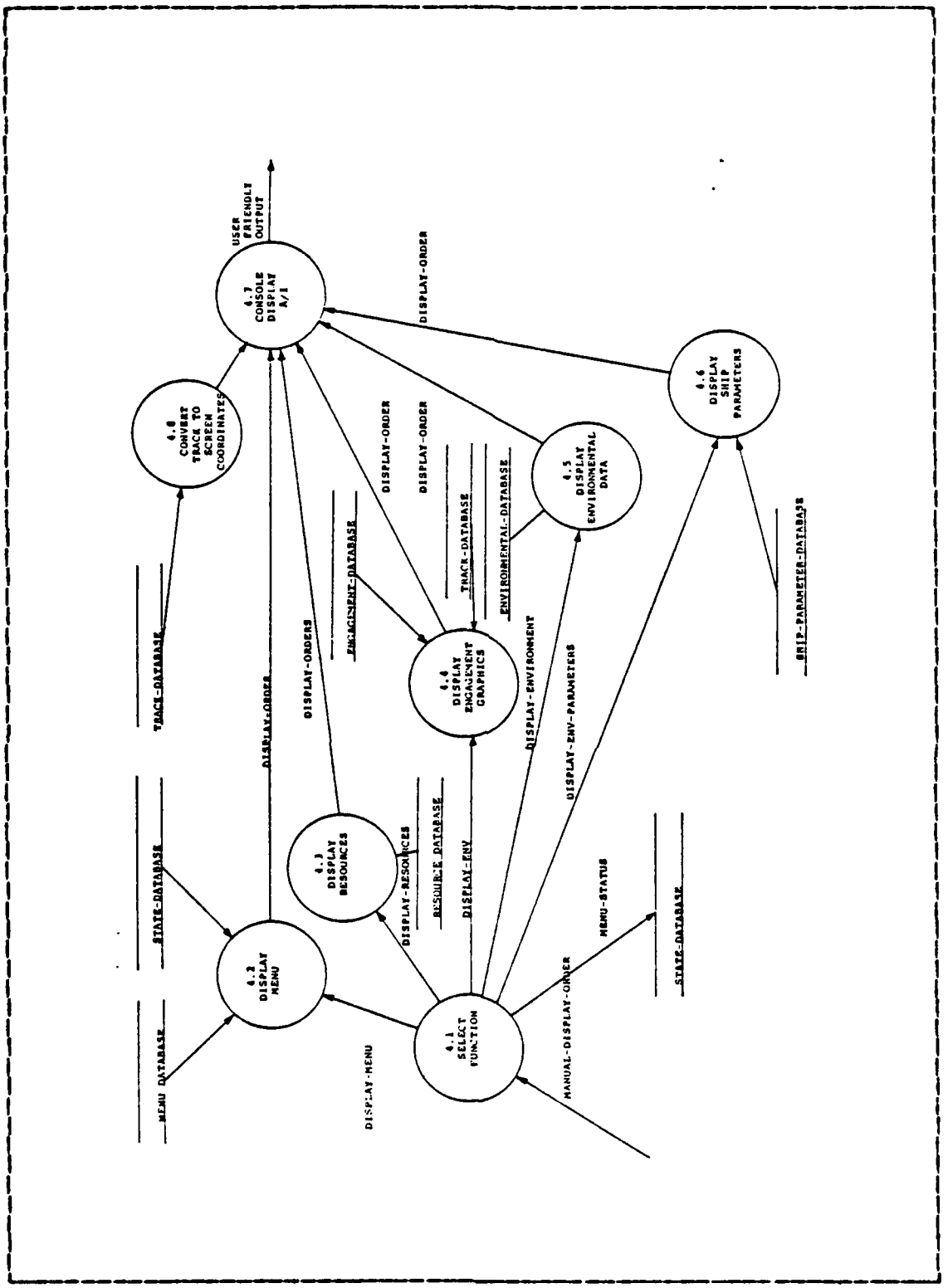


Figure A.6 Decode Output DFD.

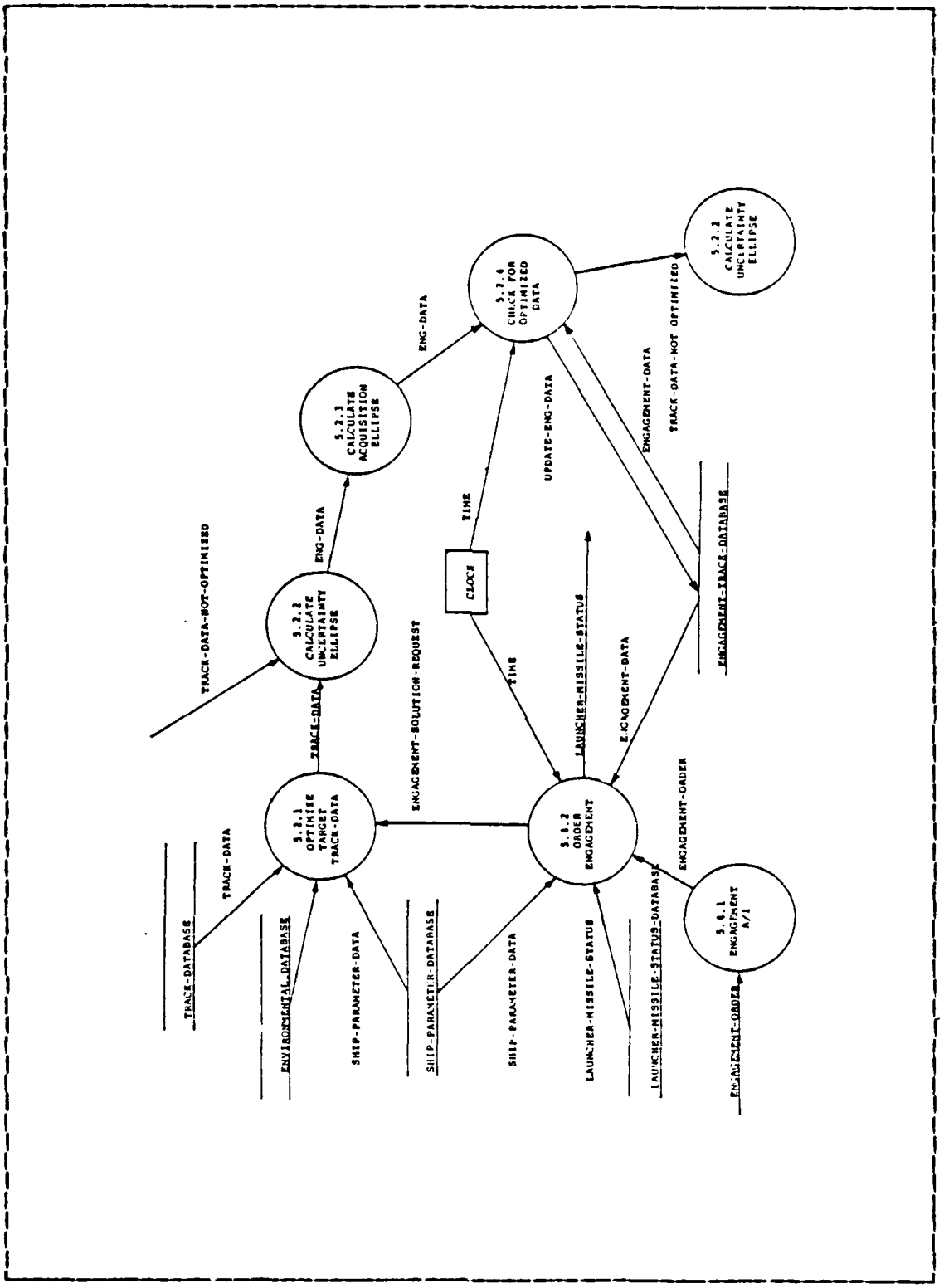


Figure A.7 Plan Engagement DFD.

APPENDIX B
HSCLCS HIERARCHY CHARTS

These are the HSCICS Hierarchy charts from [Ref. 2].

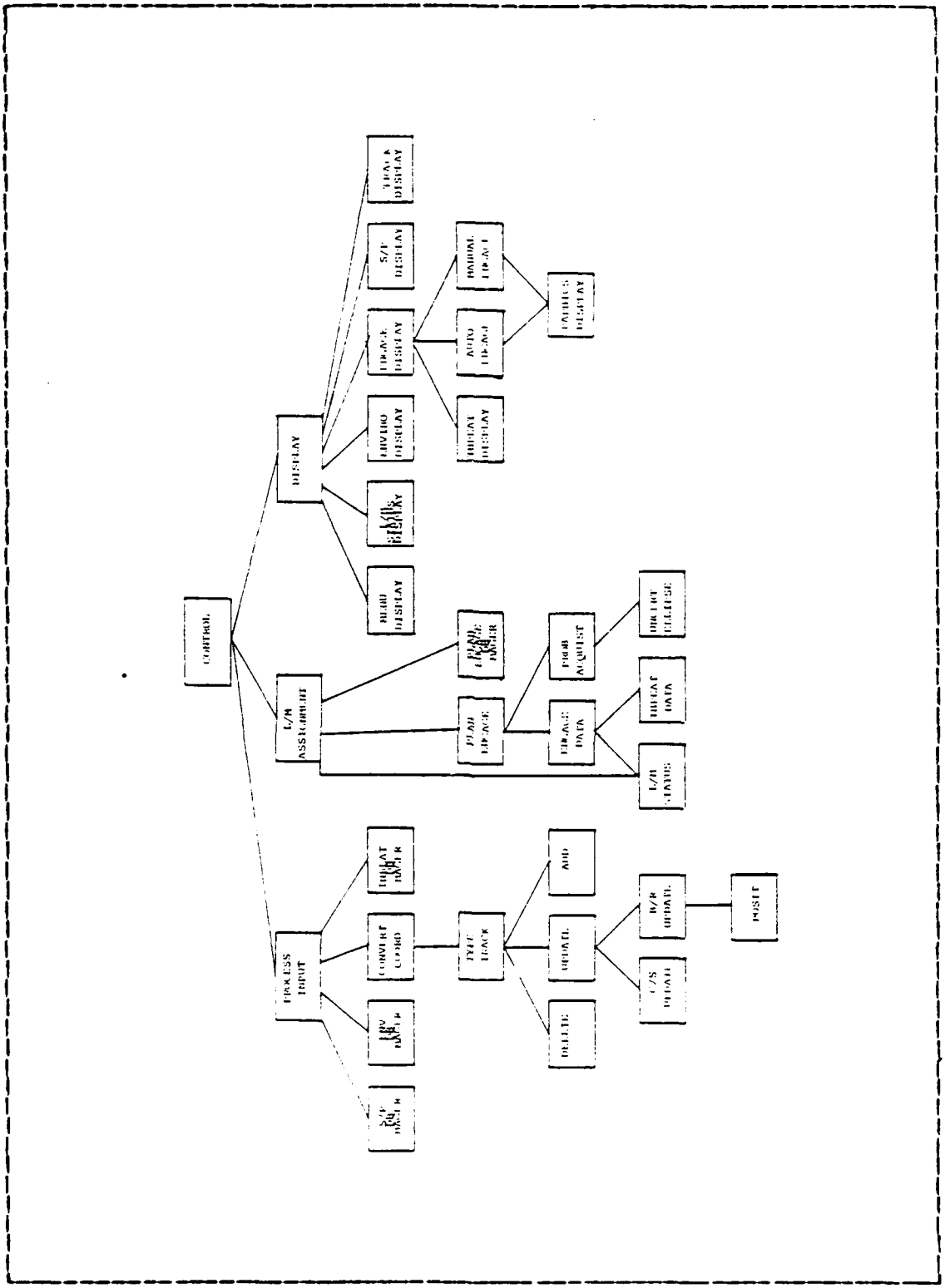


Figure B.1 First Cut Transform Analysis.

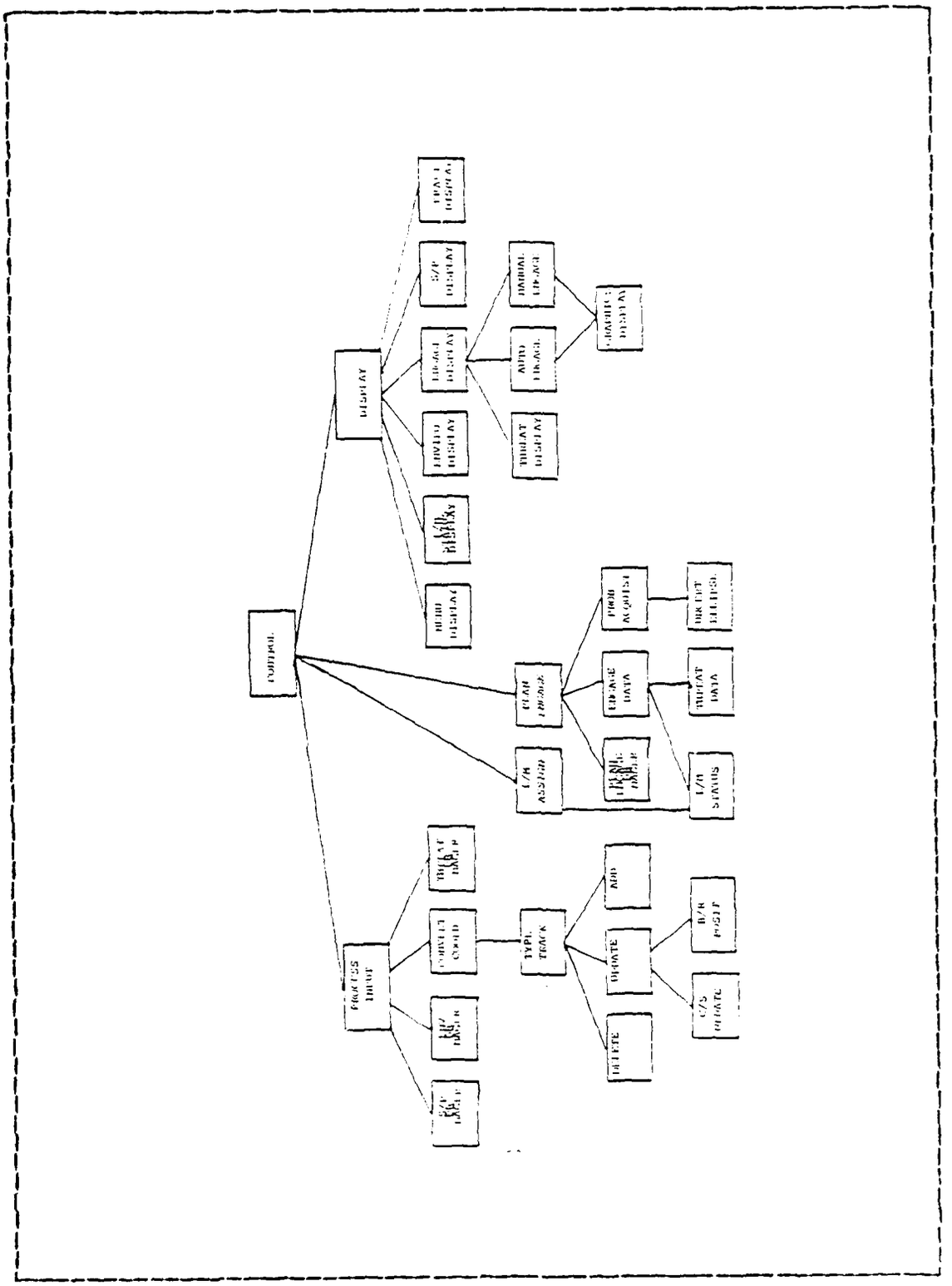
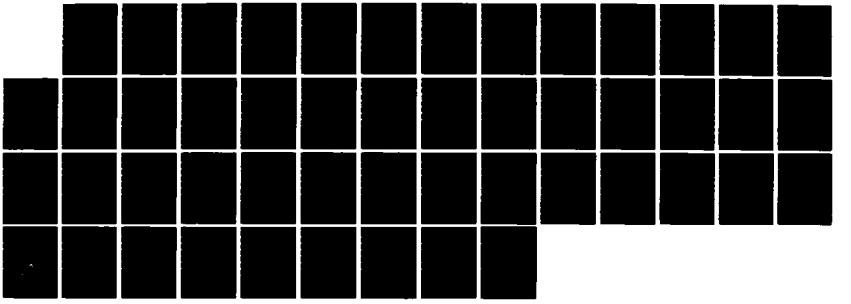
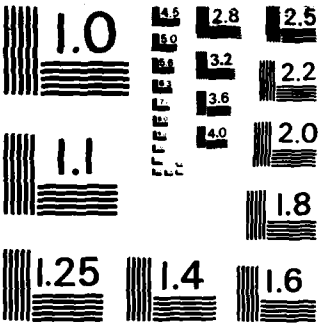


Figure B.2 Refinement of Transform Analysis.

AD-A140 079 A PROGRAM MANAGER'S METHODOLOGY FOR DEVELOPING STRUCTURED DESIGN IN EMBEDDED WEAPONS SYSTEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA J I RANSBOTHAM ET AL. 2/2

UNCLASSIFIED DEC 83 F/G 9/2 NL





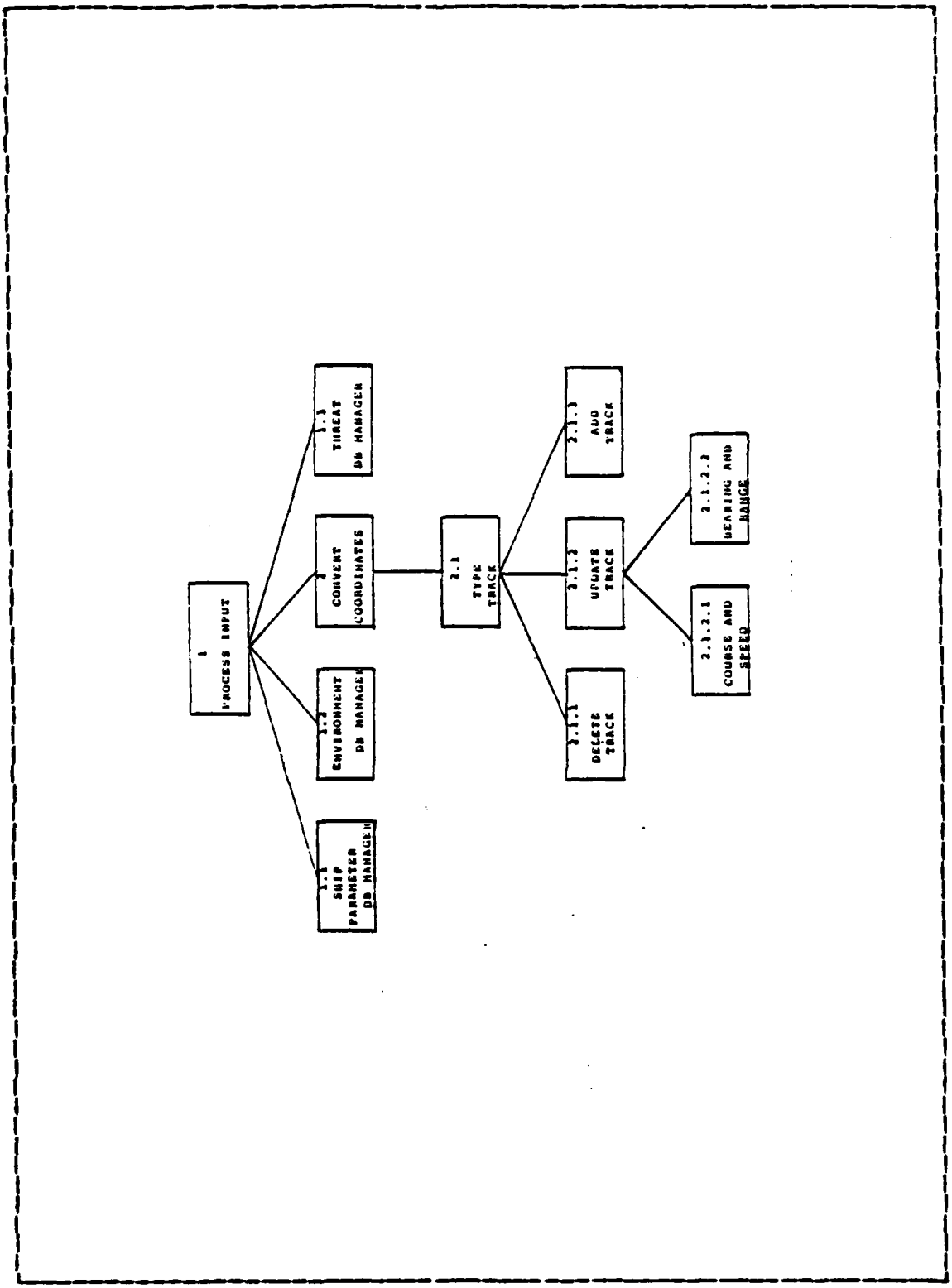


Figure B.3 Process Input.

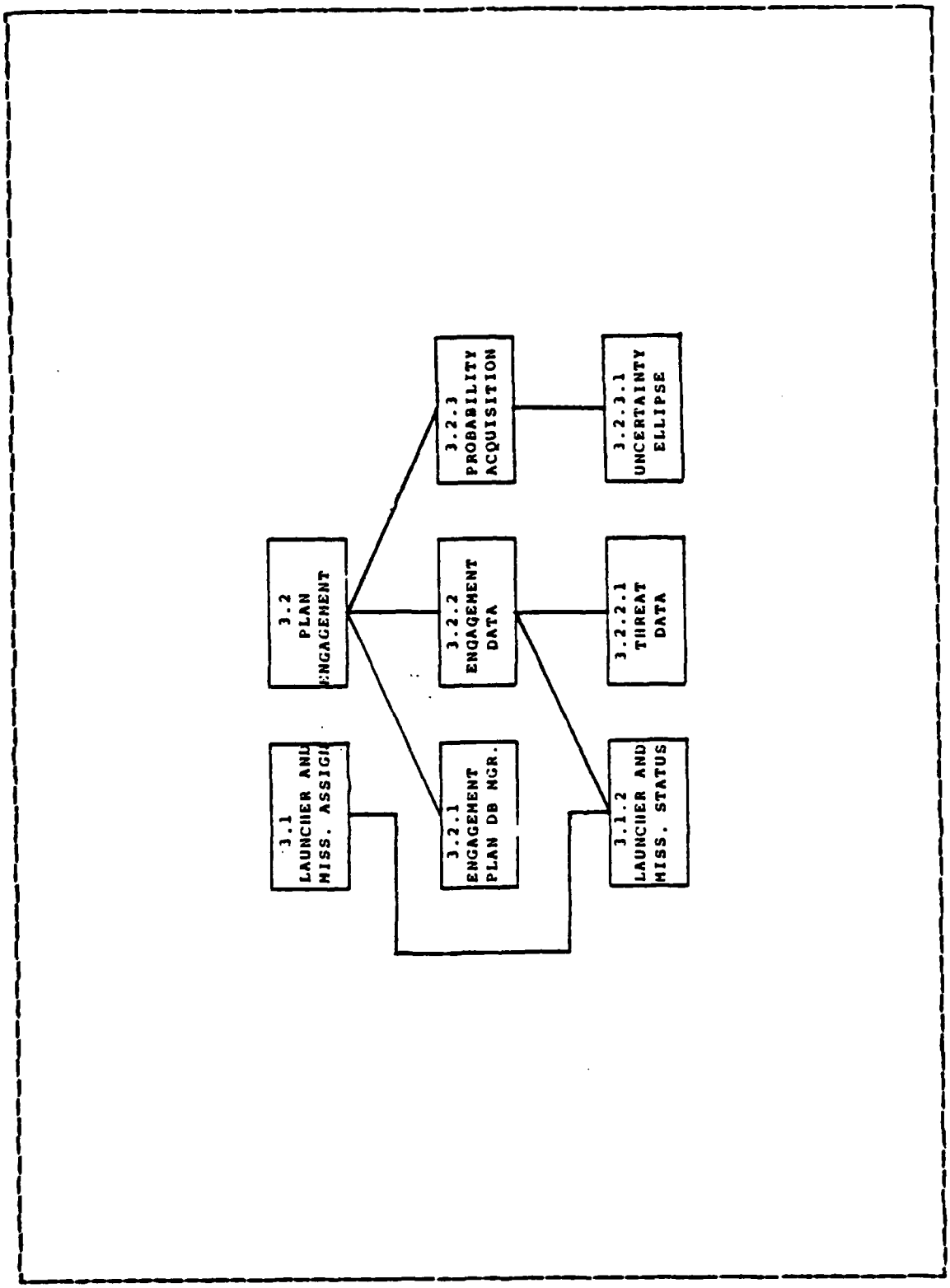


Figure B.4 Process Engagement.

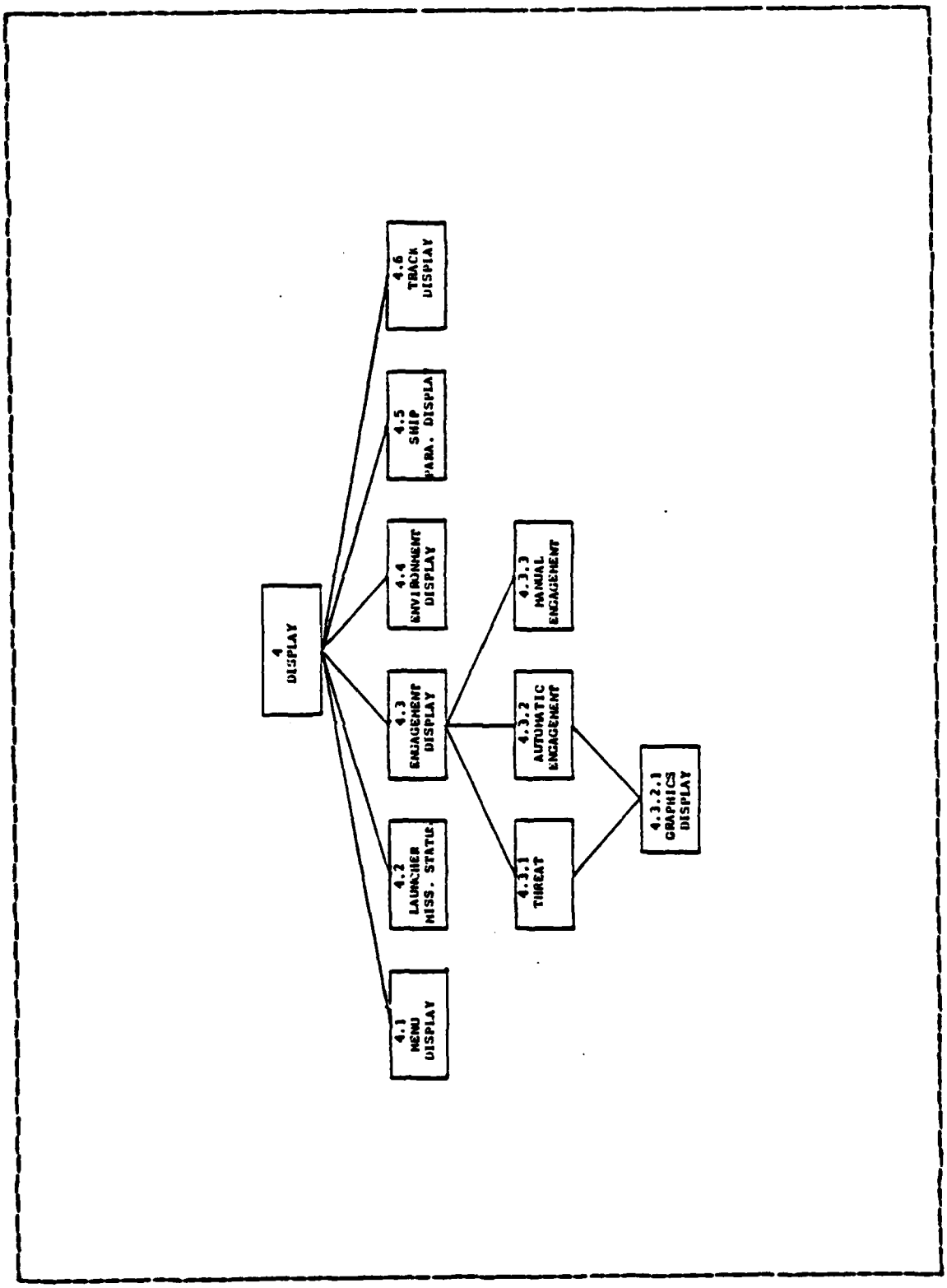


Figure B.5 Process Display.

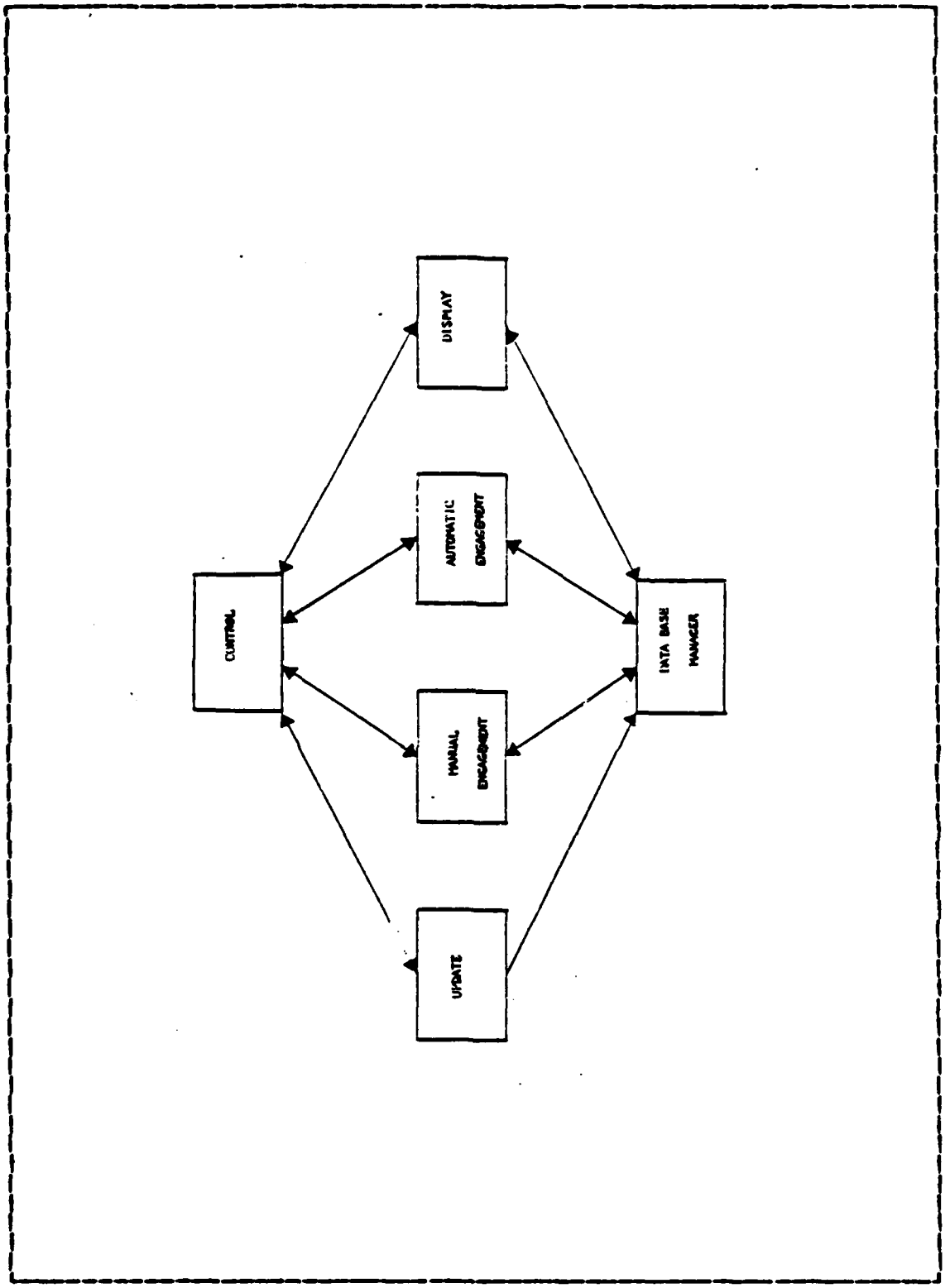


Figure B.6 Program Design Structure.

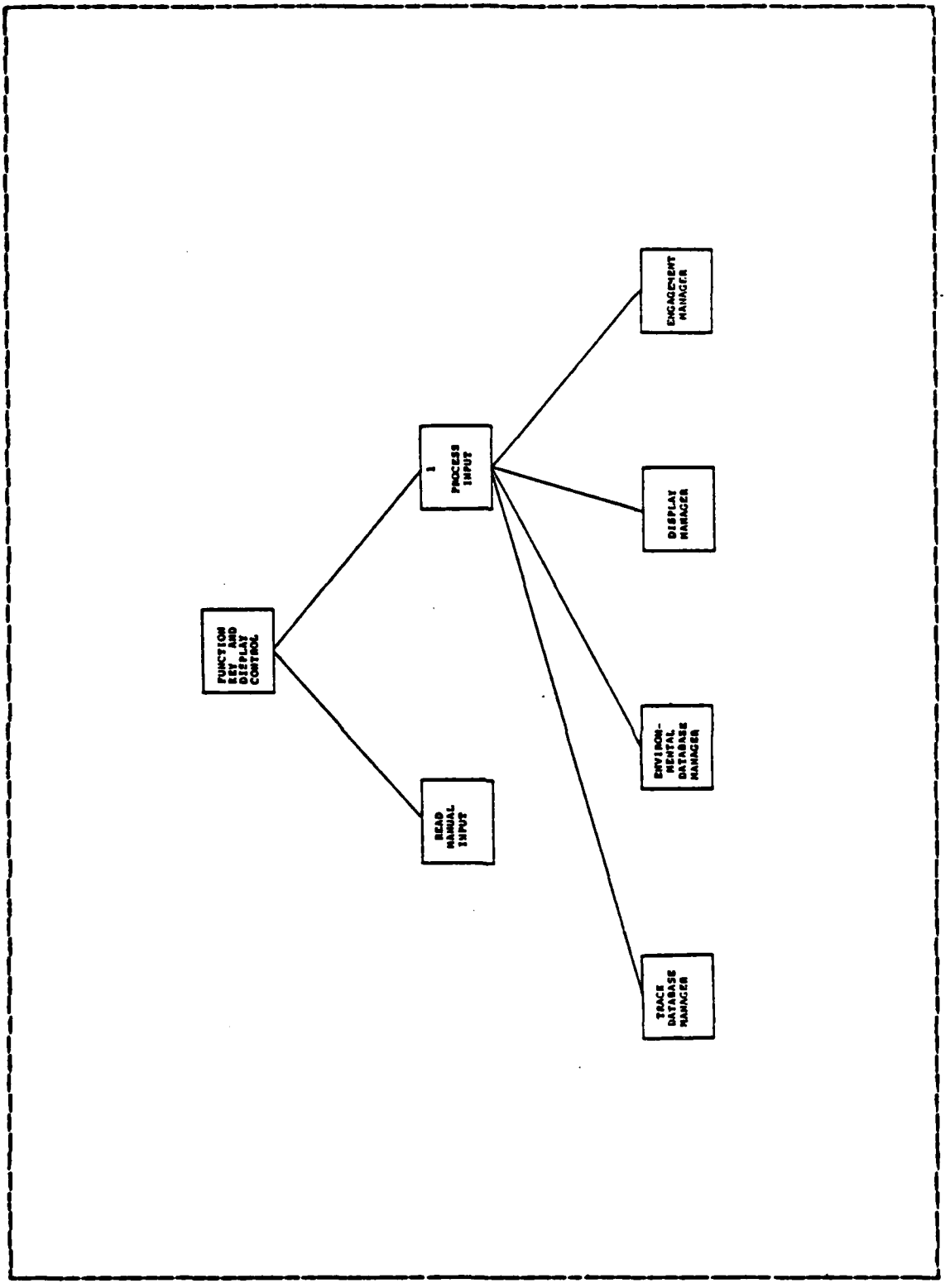


Figure B.7 Transition Structure of Figure B.6.

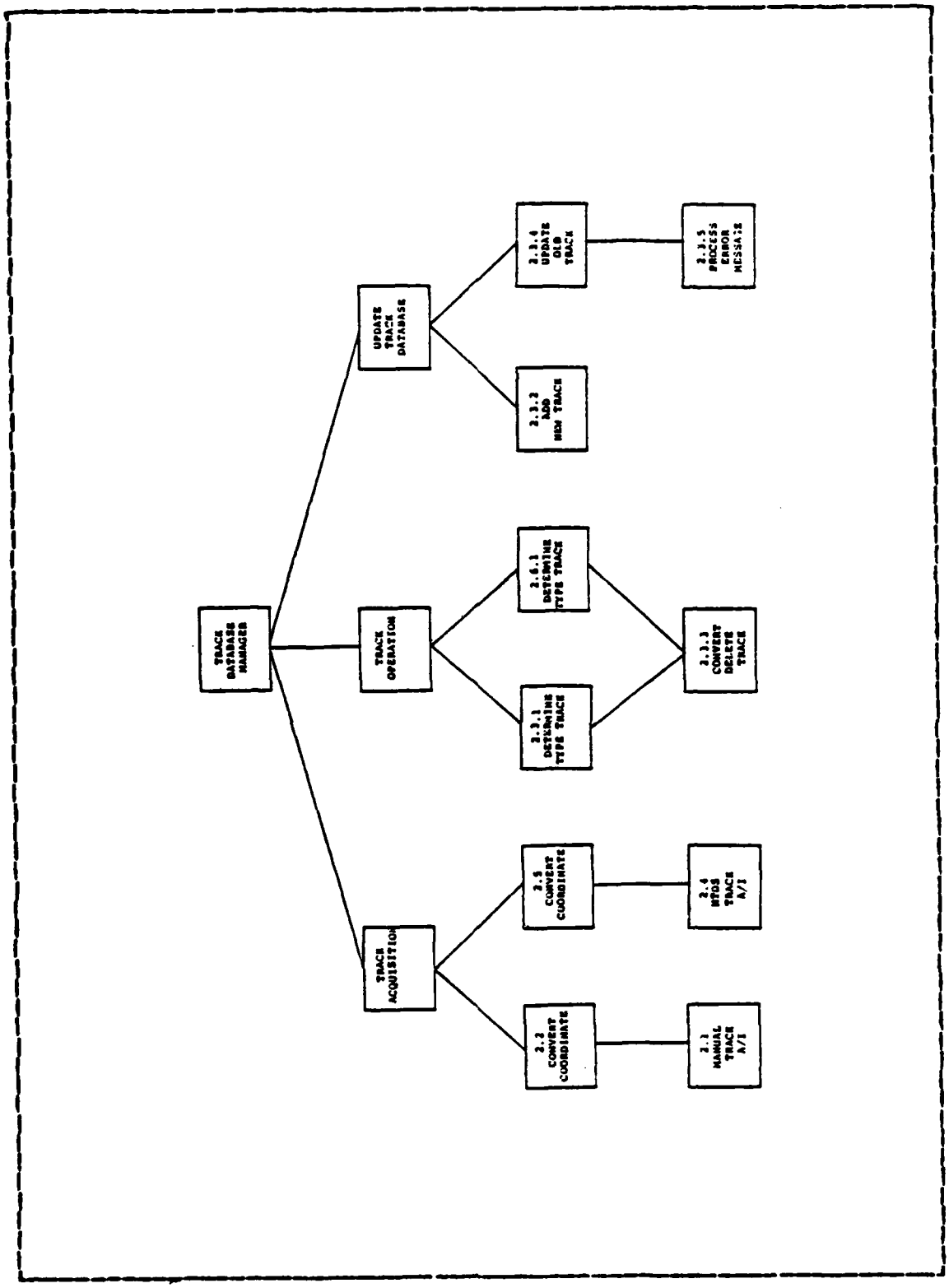


Figure B.8 Action Path Structure of Track Data Base Manager.

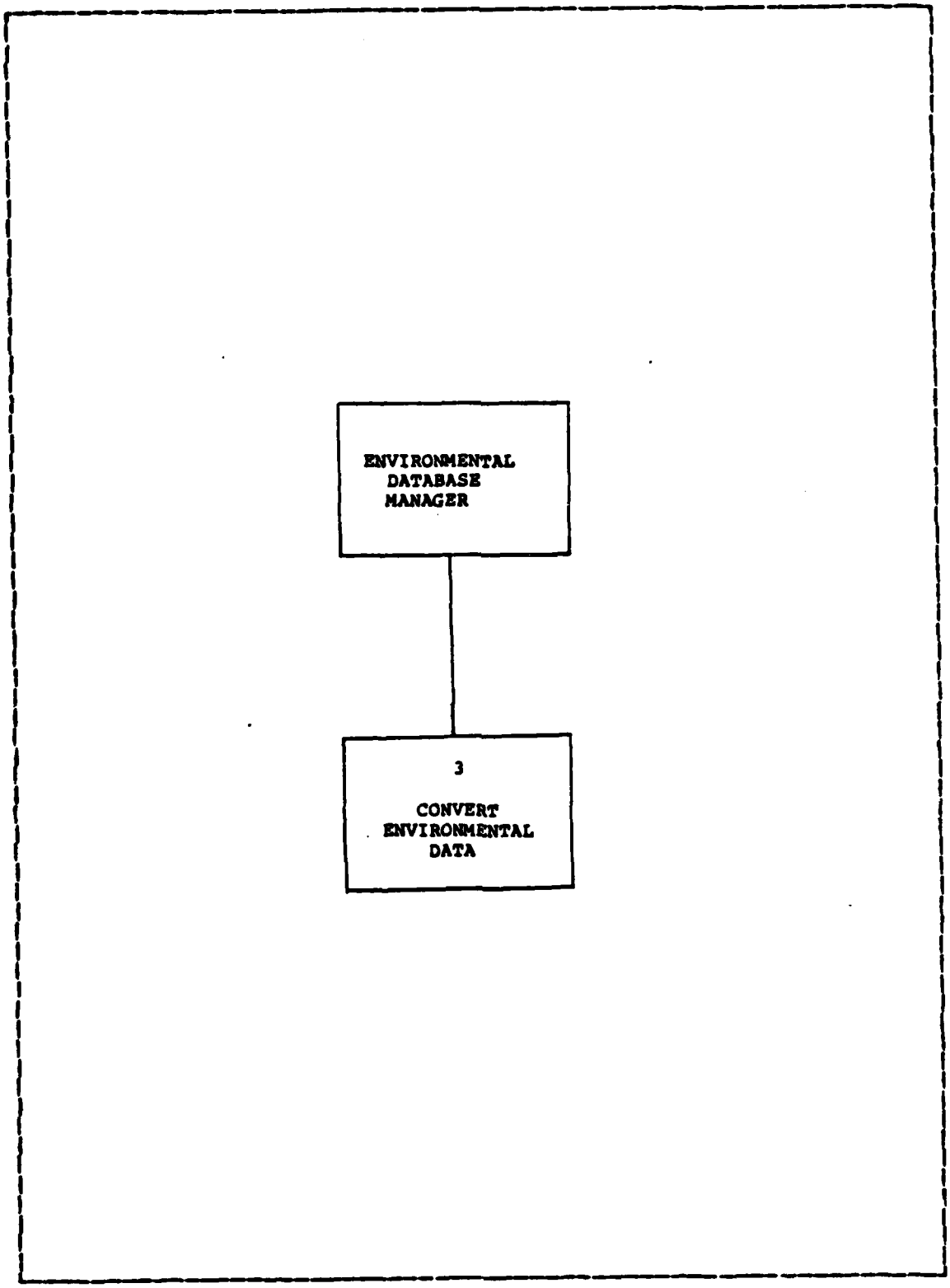


Figure B.9 Action Path of Environmental Data Base Manager.

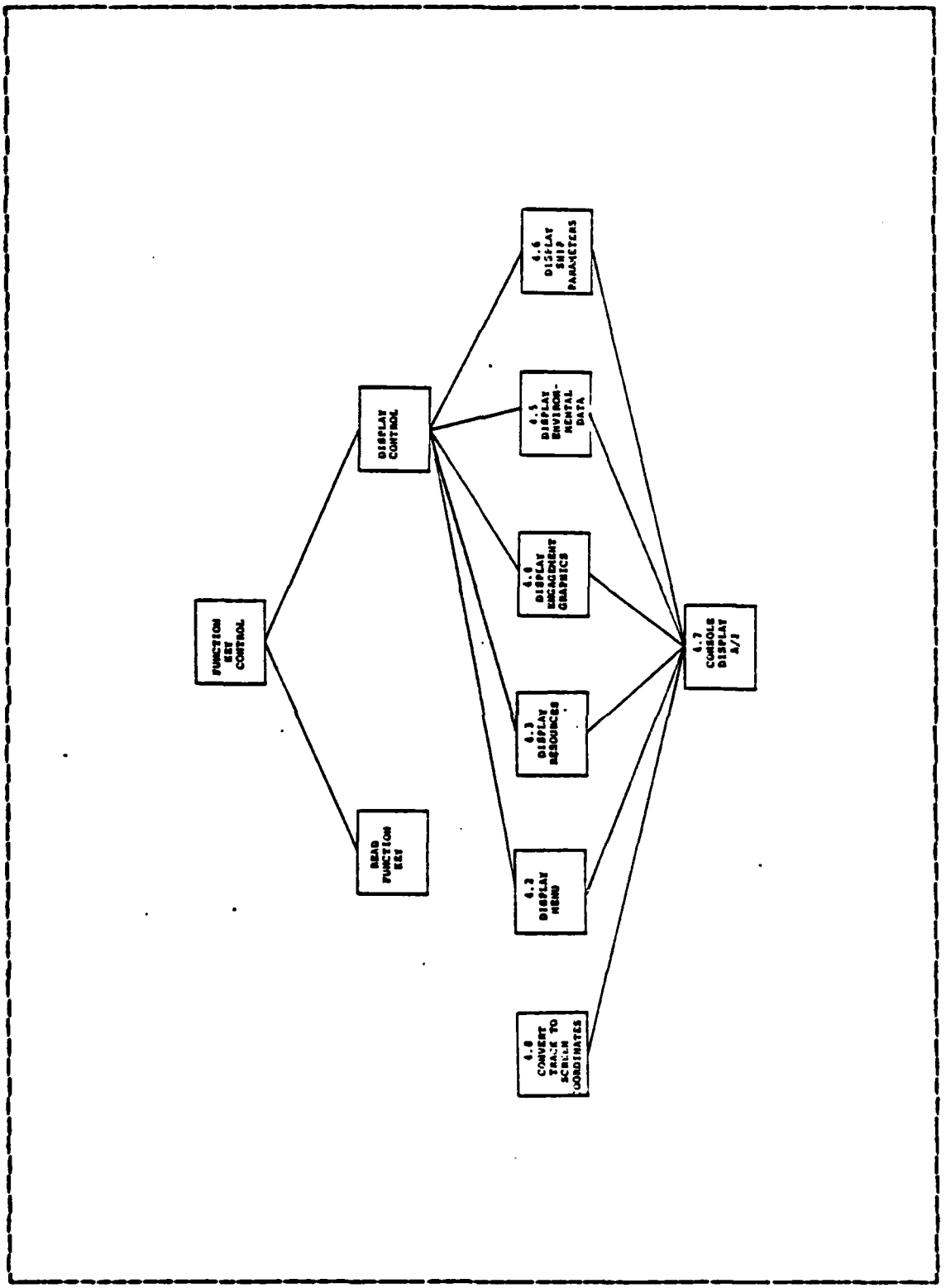


Figure B.10 Action Path of Display Manager.

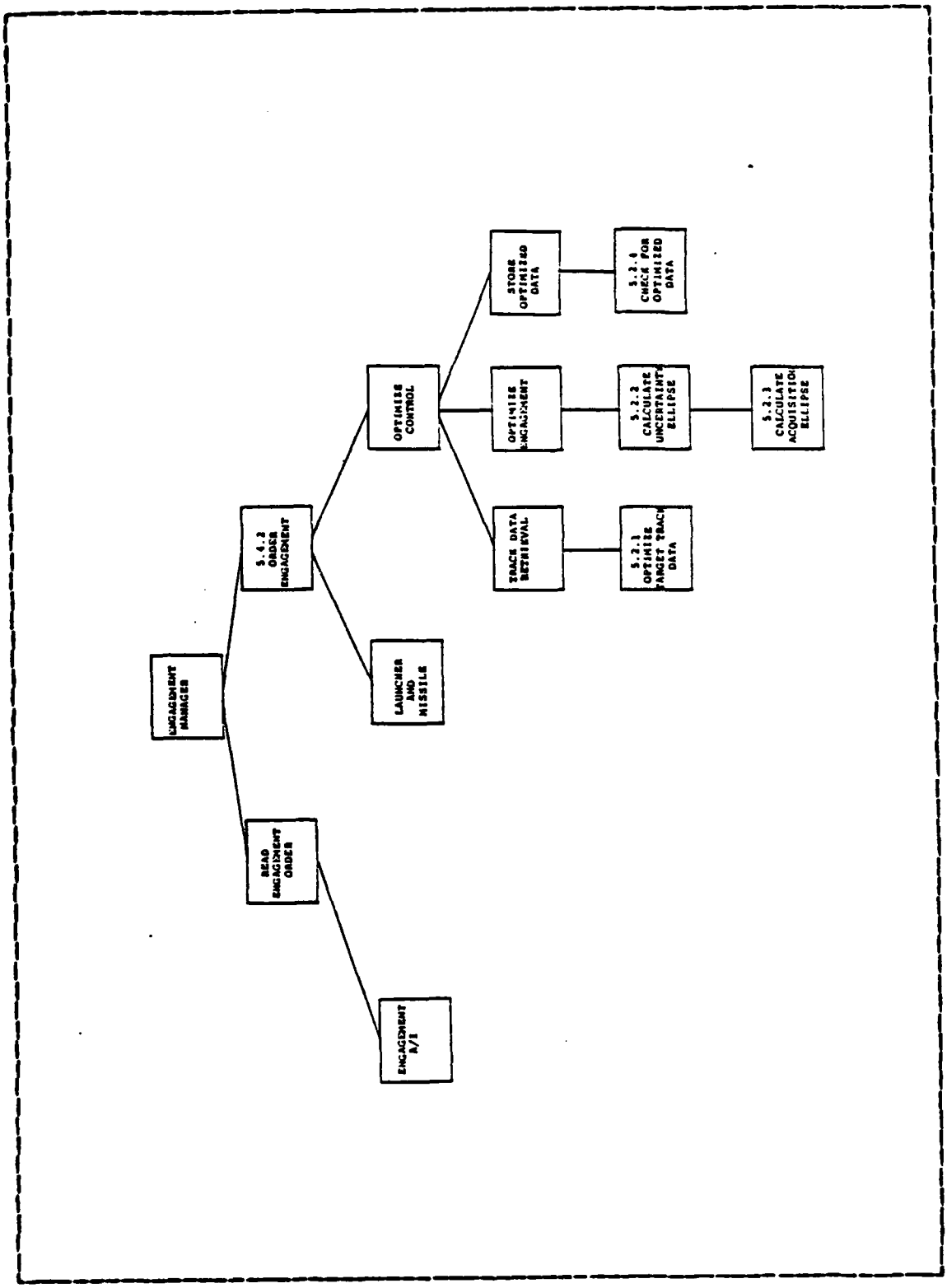


Figure B.11 Action Path of Engagement Manager.

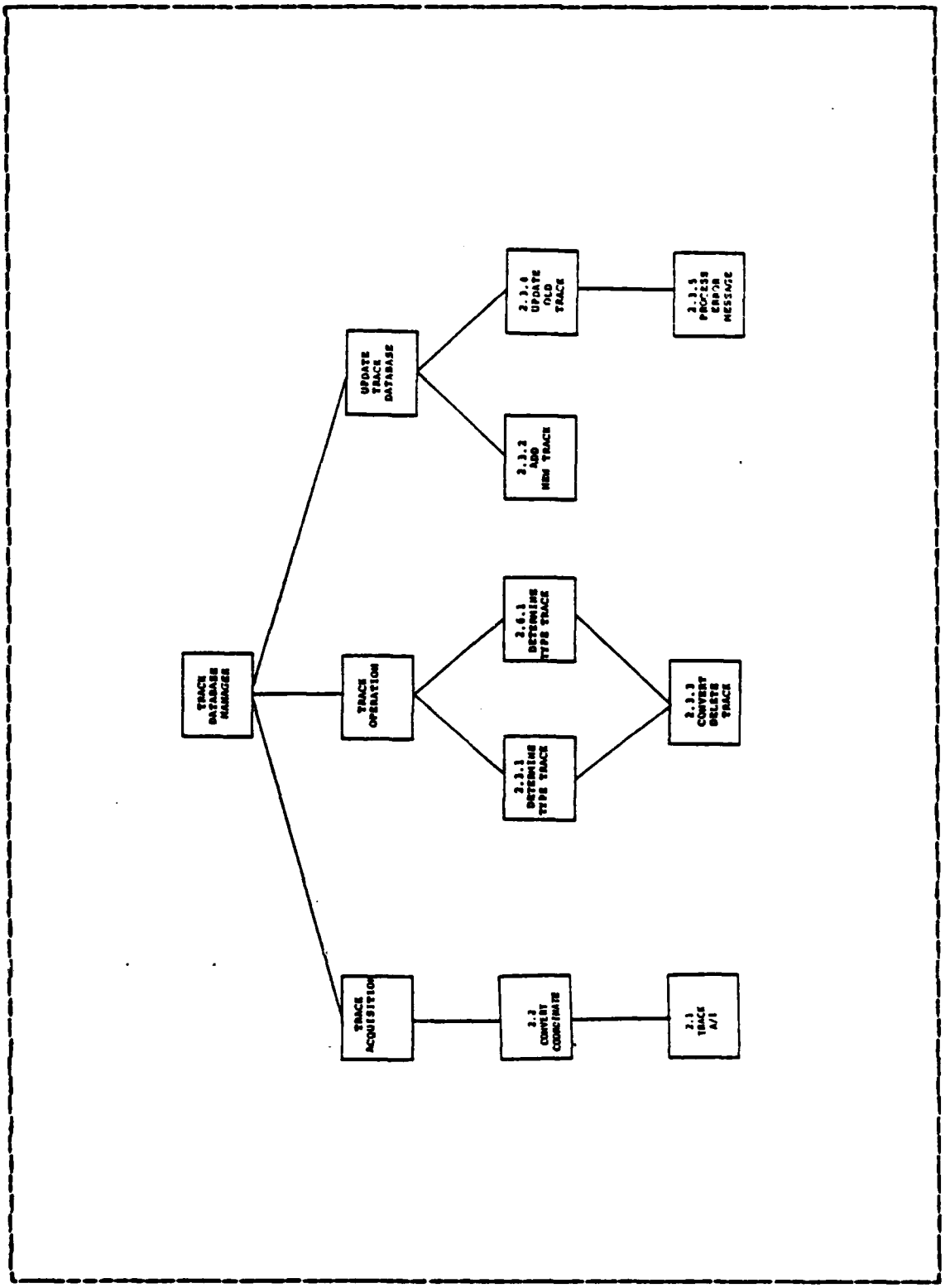


Figure B.12 Action Path of Track Data Base Mgr, with Heuristic.

APPENDIX C

HSCLCS MODULE DESCRIPTIONS

This appendix contain descriptions of the thirty one modules of the HSCLCS from [Ref. 1].

1 -	0	Control
2 -	1	Process Input
3 -	1.1	Ship Parameter Data Base Manager
4 -	1.2	Environmantal Data Base Manager
5 -	1.3	Threat Data Base Manager
6 -	2	Convert Coordinates
7 -	2.1	Type Track
8 -	2.1.1	Delete Track
9 -	2.1.2	Update Track
10-	2.1.2.1	Course and Speed Update
11-	2.1.2.2	Bearing, Range, and Position Update
12-	2.1.3	Add Track
13-	3.1	Launcher and Missile Assignment
14-	3.1.2	Launcher and Missile Status
15-	3.2	Plan Engagement
16-	3.2.1	Plan Engagement Data Base Manager
17-	3.2.2	Engagement Data
18-	3.2.2.1	Threat Data
19-	3.2.3	Probability of Acquisition
20-	3.2.3.1	Uncertainty Ellipse
21-	4	Display
22-	4.1	Menu Display
23-	4.2	Launcher and Missile Status Display
24-	4.3	Environmental Display
25-	4.4	Engagement Display
26-	4.4.1	Threat Display
27-	4.4.2	Automatic Engagement Display
28-	4.4.2.1	Graphics Display
29-	4.4.3	Manual Engagement Display
30-	4.5	Ship Parameter Display
31-	4.6	Track Display

****

1. Module Name: CONTROL, NUMBER 0
2. Module function: This module calls all other modules and determines the program flow.
3. Supervisory modules: None
4. Module interface(parameters):
5. Subordinate Modules: Process Input
   Launcher-Missile Assignment
   Plan Engagement
   Display
6. Design decision encapsulated:

****

1. Module Name: PROCESS INPUT, NUMBER 1
2. Module function: Selects subordinate module to update corresponding data bases.
3. Supervisory modules: Control
4. Module interface(parameters):
5. Subordinate Modules: Ship parameter Data Base Manager
   Environmental Data Base Manager
   Convert Coordinates
   Threat Data base Manager
6. Design decision encapsulated:

****

1. Module Name: SHIP PARAMETER DATA BASE MANAGER, NUMBER 1.1
2. Module function: Update the Ship Parameter Data Base by either manual or automated means.
3. Supervisory modules: Process Input
4. Module interface(parameters):
5. Subordinate Modules: None
6. Design decision encapsulated:

****

1. Module Name: ENVIRONMENTAL DATA BASE MANAGER, NUMBER 1.2
2. Module function: Update the Environmental Data Base by either manual or automated means.
3. Supervisory modules: Process Input
4. Module interface(parameters):
5. Subordinate Modules: None
6. Design decision encapsulated:

****

1. Module Name: THREAT DATA BASE MANAGER, NUMBER 1.3
2. Module function: Update the Threat Data Base by either manual means or through use of a standard chip that can be periodically updated and sent to all ships with HARPOON capability.
3. Supervisory modules: Process Input
4. Module interface(parameters):
5. Subordinate Modules: None
6. Design decision encapsulated:

****

1. Module Name: CONVERT COORDINATES, NUMBER 2
2. Module function: To convert all the inputs to update track data to common coordinates. The inputs can be manual, from own ship's tracking equipment, or from an NTDS link from other platforms.
3. Supervisory modules: Process Input
4. Module interface(parameters):
5. Subordinate Modules: Type Track
6. Design decision encapsulated:

****

1. Module Name: TYPE TRACK, NUMBER 2.1
2. Module function: Type Track determines if the track is to be deleted from the data base, added to the data base, or some parameters of an existing track are to be altered. These actions are performed by selecting the appropriate subordinate module.
3. Supervisory modules: Convert Coordinates
4. Module interface(parameters):
5. Subordinate Modules: Delete Track
   Update Track
   Add Track
6. Design decision encapsulated:

****

1. Module Name: DELETE TRACK, NUMBER 2.1.1
2. Module function: To eliminate tracks from the data base that the operator determines are no longer useful.
3. Supervisory modules: Type Track
4. Module interface(parameters):
5. Subordinate Modules: Track Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: UPDATE TRACK, NUMBER 2.1.2
2. Module function: To update the information contained in the Track Data Base.
3. Supervisory modules: Type Track
4. Module interface(parameters):
5. Subordinate Modules: Course and Speed
   Bearing and Range
6. Design decision encapsulated:

****

1. Module Name: COURSE AND SPEED UPDATE, NUMBER 2.1.2.1
2. Module function: To update the course and speed information on each track contained in the Track Data Base.
3. Supervisory modules: Update Track
4. Module interface(parameters):
5. Subordinate Modules: Track Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: BEARING,RANGE AND POSITION UPDATE, NUMBER 2.1.2.2
2. Module function: To update the bearing/range and position (Lattitude/Longitude) information on each track in the Track Data Base.
3. Supervisory modules: Update Track
4. Module interface(parameters):
5. Subordinate Modules: Track Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: ADD TRACK, NUMBER 2.1.3
2. Module function: To allow new tracks to be put into the Track Data Base.
3. Supervisory modules: Type Track
4. Module interface(parameters):
5. Subordinate Modules: Track Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: LAUNCHER AND MISSILE ASSIGNMENT, 3.1
2. Module function: Allow the operator to bypass the engagement planning automatic selection of missile cell and simply select and launch the missile manually.
3. Supervisory modules: Control
4. Module interface(parameters):
5. Subordinate Modules: Launcher and Missile Status
6. Design decision encapsulated:

****

1. Module Name: LAUNCHER AND MISSILE STATUS, NUMBER 3.1.2
2. Module function: To provide current information on what launchers (port - starboard) are ready to fire and which and what types missiles are ready to fire.
3. Supervisory modules: Launcher Missile Assignment Engagement Data
4. Module interface(parameters):
5. Subordinate Modules: None
6. Design decision encapsulated:

****

1. Module Name: PLAN ENGAGEMENT, NUMBER 3.2
2. Module function: To determine the optimum engagment plan for a given target.
3. Supervisory modules: Control
4. Module interface(parameters):
5. Subordinate Modules: Plan Engagement Data Base Manager
   Engagement Data
   Probability of Acquisition
6. Design decision encapsulated:

****

1. Module Name: PLAN ENGAGEMENT DATA BASE MANAGER, NUMBER 3.2.1
2. Module function: To update the Engagement Plan Data Base.
3. Supervisory modules: Plan Engagement
4. Module interface(parameters):
5. Subordinate Modules: None
6. Design decision encapsulated:

****

1. Module Name: ENGAGEMENT DATA, NUMBER 3.2.2
2. Module function: This module supplies the data needed by the Plan Engagement module to generate the engagement plan.
3. Supervisory modules: Plan Engagement
4. Module interface(parameters):
5. Subordinate Modules: Launcher and Missile Status
   Threat Data
6. Design decision encapsulated:

****

1. Module Name: THREAT DATA, NUMBER 3.2.2.1
2. Module function: To provide the information contained in the the module to the Engagement Data module when requested.
3. Supervisory modules: Engagement Data
4. Module interface(parameters):
5. Subordinate Modules: None
6. Design decision encapsulated:

****

1. Module Name: PROBABILITY OF ACQUISITION, NUMBER 3.2.3
2. Module function: To determine what the probability is that if a missile is fired at a given target that the missile can acquire and hit the target
3. Supervisory modules: Plan Engagement
4. Module interface(parameters):
5. Subordinate Modules: Uncertainty Ellipse
6. Design decision encapsulated:

****

1. Module Name: UNCERTAINTY ELLIPSE, NUMBER 3.2.3.1
2. Module function: To compute the parameters for an ellipse of uncertainty around a contact's position.
3. Supervisory modules: Probability of Acquisition
4. Module interface(parameters):
5. Subordinate Modules: Track Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: DISPLAY, NUMBER 4
2. Module function: To call subordinate modules as necessary to generate required displays.
3. Supervisory modules: Control
4. Module interface(parameters):
5. Subordinate Modules: Menu Display
   Launcher and Missile Status Display
   Environmental Display
   Engagement Display
   Ship Parameter Display
   Track Display
6. Design decision encapsulated:

****

1. Module Name: MENU DISPLAY, NUMBER 4.1
2. Module function: To access the Menu/State Data Base and display the required menu when called and keep track of the state of the program.
3. Supervisory modules: Display
4. Module interface(parameters):
5. Subordinate Modules: Menu/State Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: LAUNCHER AND MISSILE STATUS DISPLAY, NUMBER 4.2
2. Module function: To access the Launcher and Missile Status Data Base and provide a display of the information contained in that data base.
3. Supervisory modules: Display
4. Module interface(parameters):
5. Subordinate Modules: Launcher Missile Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: ENVIRONMENTAL DISPLAY, NUMBER 4.3
2. Module function: To access the Environmental Data Base and provide a display of the information contained in that data base.
3. Supervisory modules: Display
4. Module interface(parameters):
5. Subordinate Modules: Environmental Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: ENGAGEMENT DISPLAY, NUMBER 4.4
2. Module function: To graphically display the flight path of missiles that are to be flown against a set target. Threat data on the target will also be displayed. The engagement plan will have the capability to be superimposed over the general track display.
3. Supervisory modules: Display
4. Module interface(parameters):
5. Subordinate Modules: Threat Display
   Automatic Engagement
   Manual Engagement
6. Design decision encapsulated:

****

1. Module Name: THREAT DISPLAY, NUMBER 4.4.1
2. Module function: To access the Threat Data Base and provide a display of the information contained in that data base.
3. Supervisory modules: Display
4. Module interface(parameters):
5. Subordinate Modules: Threat Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: AUTOMATIC ENGAGEMENT DISPLAY, NUMBER 4.4.2
2. Module function: To graphically display the engagement plan that was generated by the Plan Engagement module and stored in the Engagement Plan Data Base.
3. Supervisory modules: Engagement Display
4. Module interface(parameters):
5. Subordinate Modules: Engagement Plan Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: GRAPHICS DISPLAY, NUMBER 4.4.2.1
2. Module function: To provide the operator with the capability to manually input an engagement plan for attacking a given target.
3. Supervisory modules: Automatic Engagement Display
   Manual Engagement Display
4. Module interface(parameters):
5. Subordinate Modules: None
6. Design decision encapsulated:

****

1. Module Name: MANUAL ENGAGEMENT DISPLAY, NUMBER 4.4.3
2. Module function: To provide the operator with the capability to manually input an engagement plan for attacking a given target.
3. Supervisory modules: Engagement Display
4. Module interface(parameters):
5. Subordinate Modules: Graphics Display
6. Design decision encapsulated:

****

1. Module Name: SHIP PARAMETER DISPLAY, NUMBER 4.5
2. Module function: To access the Ship Parameter Data Base and provide a display of the information contained in that data base.
3. Supervisory modules: Display
4. Module interface(parameters):
5. Subordinate Modules: Ship Parameter Data Base Manager
6. Design decision encapsulated:

****

1. Module Name: TRACK DISPLAY, NUMBER 4.6
2. Module function: To access the Track Data Base and provide a continuous display of all tracks being maintained in that data base.
3. Supervisory modules: Display
4. Module interface(parameters):
5. Subordinate Modules: Track Data Base Manager
6. Design decision encapsulated:

APPENDIX D
HSCLCS ADA DESIGN

ADA HSCLCS design from [Ref. 2].

```
Package Update is
     Task Launcher-Missile_Status is
          entry  Update            (Launcher-Missile  Status:
            in Status Type)
     End  Launcher-Missile_Status
     Task Ship-Parameter is
          entry       Update       (Ship-Parameter:        in
            Ship-Parameter-Type)
     End  Ship-Parameter
     Task Environment is
          entry Update (Environment: in Environment-Type)
     End  Environment
     Task Threat is
          entry Update (Threat: in Threat-Type)
     End  Threat
     Task Update-Track is
          entry Add (Track: in Track-Type)
          entry Delete (Track: in Track-Type)
          entry Modify (Track: in Track-Type)
     End  Update Track
End Update
```

```
Package Auto-Engagement is
     Procedure A-Engagement  (launcher-Missile-Status:    in
                Status type,  Threat:  in Threat Type,
                Engagement-Plan: out Engagement-Plan-Type);
     Procedure Prob-of-Acquisition (Engagement-Plan:  in out
                Engagement-Plan-Type);
     Procedure Uncertainty-Ellipse (Engagement-Plan:  in out
                Engagement-Plan_type);
End Auto-Engagement


Package Manual-Engagement is
     Procedure M-Engagement  (Launcher-Missile Status:   in
                Status-Type,      Engagement-Plan:      out
                Engagement-Plan-Type);
End Manual-Engagement
```

```
Package Display is
     Task Menu-Display is
          entry Access (Menu: out Menu-Type)
     End  Menu-Display
     Task Launcher-Missile-Status is
          entry  Access  (Launcher-Missile-Status:  out
            Status-Type)
     End  Launcher-Missile-Status
     Task Environment is
          entry Access (Environment: out Environment-Type)
     End  Environment
     Task Ship-Parameter is
          entry      Access      (Ship-Parameter:      out
            Ship-Parameter-Type)
     End  Ship-Parameter
     Task Track is
          entry Access (Track: out Track-Type)
     End  Track
     Task Threat is
          entry Access (Threat: out Threat-Type)
     End  Threat
     Task Engagement-Plan is
          entry      Access      (Engagement-Plan:      out
            Engagement-Plan-Type)
     End  Engagement-Plan
```

```
    Package Engagement-Display is
        Procedure Manual-Engage-Display (Engagement-Plan:
                  in out Engagement-Plan-Type, Threat:
                  in out Threat-Type);
        Procedure Auto-Engage-Display (Engagement-Plan:
                  in out Engagement-Plan-Type, Threat:
                  in out Threat-Type);
        Procedure Graphics (Engagement-Plan: in out
                  Engagement-Plan-Type)
    End Engagement Display
End Display
```

```
Package Data Base Managers is
     Package Launcher Missile Status Manager is
          Type Status Type is
               Record
                    Empty     :  Boolean
                    Miss Type:  String range A .. C;
               End record
          Task Launcher Missile Status is
                    entry Update (Launcher Missile Status in
                      Status Type)
                    entry  Access (Launcher  Missile  Status
                      out Status Type)
          End Launcher Missile Status
     End Launcher Missile Status Manager
     Package Ship Parameter Manager is
          Type Ship Parameter Type is
               Record
                    Course         :  Integer range 0..359;
                    Speed          :  Integer range 0..50;
                    Position Lat   :  Latitude;
                    Position Long:  Longitude;
               End record
          Task Ship Parameter is
                    entry Update  (Ship Parameter:   in Ship
                      Parameter Type)
                    entry Access (Ship  Parameter:  out Ship
                      Parameter Type)
          End Ship Parameter
     End Ship Parameter Manager
```

```
Package Environment Manager is
     Type Environment Type is
          Record
               Visibility  :  Real Range 0..30;
               Sea-State   :  Integer range 0..5;
               Wind Dir    :  Integer range 0..359;
               Wind Spd    :  Integer range 0..100;
               Temperature :  Integer range -100..150
               Barometric  :  Integer range 900..1200
               .
               .
               .
     Task Environment is
          entry  Update (Environment:   in  Environment
            Type)
          entry Access  (Environment:  out  Environment
            Type)
     End Environment
End Environment Manager
Package Threat Manager is
     Type Threat Type is
          Record
               Ship Name   :  String;
               Ship Class  :  String;
               Weapons     :  String;
               ECM Equip   :  String;
               Attack Plan :  String;
               .
               .
               .
          End record
     Task Threat is
          entry Update (Threat: in Threat Type)
          entry Access (Threat: out Threat Type)
     End Threat
End Threat Manager
```

```
Package Track Manager is
     Type Track is
          Record
               Type Track     :  Boolean;
               Class Vessel   :  String;
               Bearing        :  Integer range 0.359;
               Range          :  Integer range 0..500;
               Position Lat   :  Latitude;
               Position Long  :  Longitude;
               Course         :  Integer range 0..359;
               Speed          :  Integer range 0..50;
          End record
     Task Track is
          entry Add (Track: in Track Type)
          entry Delete (Track: in Track Type)
          entry Modify (Track: in Track Type)
          entry Access (Track: out Track Type)
     End Track
End Track Manager
Package Menu Manager is
     Type Menu is
          Record
               Undetermined at this time
          End record
     Task Menu Display is
          entry Access (Menu: out Menu Type)
     End Menu Display
End Menu Manager
```

```
Package Engagement Plan Manager is
     Type Engagement Plan is
          Record
               Track Desig  :  String;
               Type Plan    :  Boolean;
               Num Missiles :  Integer range 0..24;
               Sequence     :  Array;
               Miss Type    :  String range A..C;
               .
               .
               .
          End record
     Task Engagement Plan is
          entry Access (Engagement Plan: out Engagement
            Plan Type)
     End Engagement Plan
End Engagement Plan Manager
End Data Base Manager
```

APPENDIX E

HSCLCS SAMPLE SOFTWARE SPECIFICATIONS

Sample software specifications for HSCLCS from [Ref. 13].

Operational Data/Information	Requirement	
	Baseline	Design Goal
1a. Surface Contact Position (range/bearing). The use of bearing line in addition to the 1b requirement reduces the number of displayed surface contacts by two per bearing line.	10	20 (min)
-Designated Target Target Category and Classification Displayed.	X	
-Unintended Target(s) Target Category and Classification Displayed.	X	
1b. Surface Contact/Bearing Line	1	3 (min)
2. Own Ship Position	X	
3. Air Contact Position	1	3 (min)
4. 3rd Party Targeting Data Source Designation WCIP shall resolve target position based on range and bearing input from 3rd party or bearing lines from 3rd parties or own ship.	2	3 (min)

-Manual Entry of Bearing Lines X

-Manual Entry of Range and Bearing X

5. Target Classification

-Large (default) X

Larger than a patrol boat.

-Small X

Patrol boat or smaller.

6. Contact/Track Course Direction Indicator

Program automatically compensates for own ship's motion.

-Direction Indicator X

-Dead Reckoning (Own Ship Only) X

7. Contact/Track Targeting Data Source

-Manual Input X

With appropriate data source error; includes 3rd party.

-Automatic Input

-NTDS X

-RADAR X

-SONAR X

-EW/ESM X

-Target Designation System X

8. Wind Parameters (relative)

-Speed

-Actual X

Manual input.

-Default value X

-Direction

-Actual X

Manual input.

-Default value X

9. Temperature
   - Actual X
     Manual input.
   - Default value X

10. Precipitation
   - Yes X
     Manual input.
   - No (default value) X

11. Operator Cues/Lockouts
   - Launch Inhibited (lockouts/cue) X
     All launch inhibits except roll/pitch cutout.
   - Missile Ready (cue) X
   - Data Age (cue) X
     Target and environmental data.
   - Missile Launch Status (cue)
     - Cell/Rail Empty (missile away) X
     - Missile Dud Declaration X
   - New Contact/Track to be Input (cue) X
   - Illegal Action (lockout/cue) X

12. Time/Clock
   - ZULU Time X
     Start clock: Automatic entry via NTDS Interface and/or manual entry.
   - Time on Target X
     Manual entry.
   - Time of Launch X
     Computation.
   - Countdown X
     Includes Time-to-Fire and Time-to-Impact.

13. Loadout Status/Missile Variant Identification

-Baseline/Block I Tactical Missile (RGM-84A)	X	
-Royal Navy Submarine HARPOON (RGM-84B) When reconfigured for surface launch.	X	
-Block IB Tactical Missile (RGM-84C)	X	
-Block IC Tactical Missile (RGM-84D)	X	
-Supplemental Identification (manual entry: info from loadout logbocks of hybrid/nonstandard seeker-MGU combinations).	X	
-Training All-Up Round (RTM-84A/C/D and RNSH)	X	
14. Missile In-Flight Tracks	X	
15. Up to 180 degree Off-Axis Launch	X	

Operational Selections

1. Reference System		
-True Target Bearing/Relative Target Range Top of display is north.	X	
-NTDS Grid		X
-Geographic (latitude & longitude)		X
2. Planned Missile Flight Path	3	
Software to ensure that no flight path may be selected which could result in the acquisition of own ship.	WPS	
3. Search Mode Selection		

-On Line Sizing (default) w/Manual Override X
On Line Sizing shall be automatically selected if RBL or BOL are not selected.

-Range and Bearing Launch (RBL) X
RBL pattern size shall be a function of total flight path (range traveled to target).

-Bearing Only Launch (BOL) X

4. Selectable Search Pattern Expansion (0 - 360 degrees) X
For RGM-84D missile only, applies to RBL mode and On-Line-Sizing (OLS) which results in an RBL search pattern.

-Normal Center Expansion X
For RGM-84A/BGM-84B/RGM-84C missiles; default for RGM-84D missile.

5. Enable and Destruct Ranges BOL X
Default values or manual entry (ranges not supplied over NTDS interface).

6. High Altitude Hold
RGM-84D only.

-No Entry; Default X
The High Altitude Hold default range not to interfere with search initiation and not to exceed 10nm; i.e., High Altitude Hold range is set to the minimum of 10nm or range to search initiation.

	-Manual Entry The selected High Altitude Hold range must be less than the range tc search initiation.	X	
7.	Presearch Fly-Out		
	-Sea Skim (RGM-84D only) Default mode - Presearch Fly-Out is set to sea skim altitude following the High Altitude Hold.	X	
	-Manual Entry Presearch Fly-Out at normal HARPOON run-in altitude as used in current HSCLCS.	X	
8.	Terminal Attack Mcde (RGM-84D only)		
	-Sea Skim (default)	X	
	-Pcp-up Default override by manual selection of pop-up, "SMALL TARGET" designation by NTDS, or when "SMAII TARGET" is entered manually.	X	
9.	Missile Assignment for Engagement Planning Manual entry.	X	
10.	Multi-Missile Engagement of Designated Target. Baseline: Up to 4 missiles from a single launcher. (Note: Single launcher includes TARTAR and ASROC). Design Goal: Up to 8 missiles from 2 CML's.	4	8
	-Salvo Missiles Against One Target For Simultaneous Arrival (STOT Salvc).	X	

Operator-planned engagement.

-Salvo Against Up to Four Targets (single aimpoint) From One Launcher For Simultaneous Arrival (STOT Salvo). Same aimpoint and a different RBL search expansion for each RGM-84D missile in order to distribute salvoed missiles among the targets in a formation. X

-Ripple Salvo as per current HSCLCS CML Configuration. X

-Quick Reaction/Preprogrammed STOT Salvo. Modified HSCLCS automatically will calculate and enter a different waypoint for each RGM-84D missile in a quick reaction salvo for simultaneous time-on-target (STOT). X

11. Background data and sector data request. Usable with NTDS interface only. X

ENGAGEMENT DISPLAYS

1. Contact/Track Uncertainity Ellipse
   -Designated Surface Target X
   -Unintended Targets X
   If selected by operator.

2. Predicted Time-on-Target X

3. Probability of Acquisition
   Numerical value.
   -Designated Targets X
   -Unintended Targets X

If selected by operator.

4.	Seeker Search Pattern Outline For selected search mode.	X
5.	Missile Flight Path For all selected missiles.	X
6.	Booster Drop Zone	X
7.	Missile Power Application Warning	X

OTHER

1.	Test/Maintenance	
	-Missile BIT Results	
	-Go/No-Go Indication	X
	-Failure Status Code	X
	-HSCLCS BIT Results	
	-Go/No-Go Indication	X
	-Failure Status Code	X
2.	Training Mode Inherent capability provided by system design. Design to utilize data from NTDS and/or external training support devices via an RS 232 serial interface.	
	-Contact/Track Location (actual or simulated).	
	-Off Board Source/NTDS	X
	-Own Ship Sensors/NTDS	X
	-Manual Input	X
	-Own Ship Position (actual or simulated).	X
	-Training Scenario Parameters	
	-Environmental Conditions	X
	-Operational Planning Selections	X

3. Data Extract

Design to be compatible with an RS 232 serial interface to provide for data storage/display in off-line devices (e.g., tape cassette recorder).

-Target/Targeting Data	X
-Missile Initialization Data	X
-BIT Results	X

4. Major Display Features

-Variable Range Scale	X

16K-, 32K-, 64K-, 128K-, 192K-, or 256K-yard radius. The 256K-yard is the default scale.

-Offset	X
-Zoom	X

8K-, 16K- or 32K-yard radius.

-Special Symbols	X
-Cursor, with Bearing/Range readout	X

Manually controlled.

APPENDIX F
HSCLCS SYSTEM DIAGRAMS

These four diagrams illustrate the current configuration of the HSCLCS and the new proposed one.

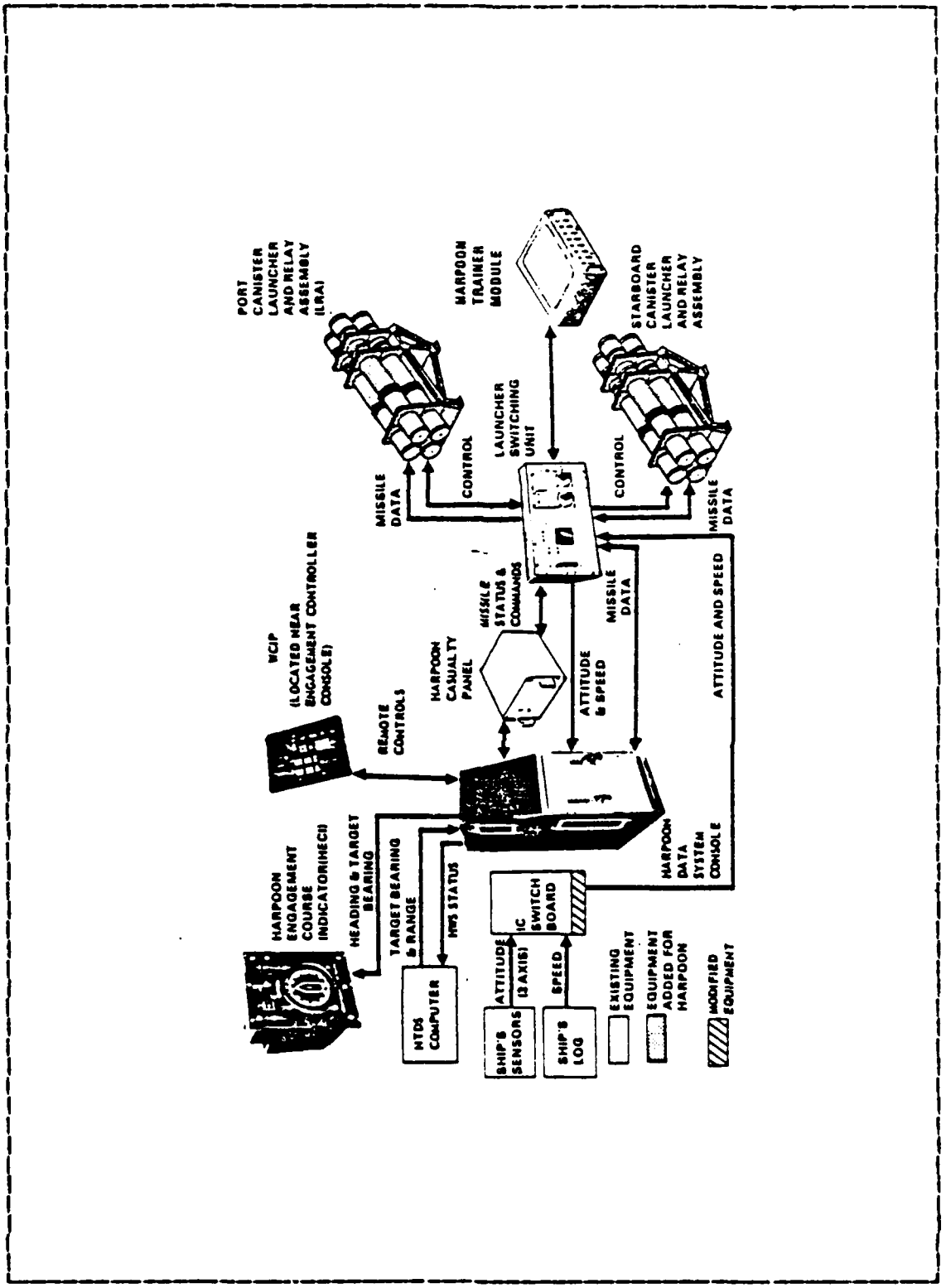


Figure F.1 Hardware Component Overview of HARPOON Weapon System.

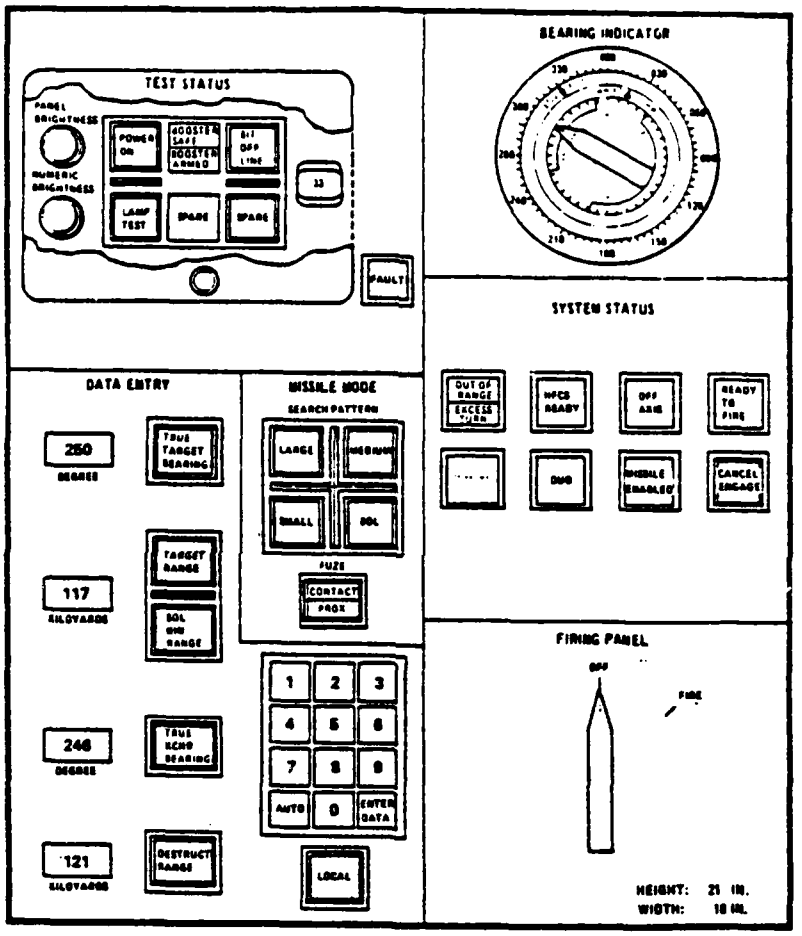


Figure F.2 Existing Cannister Launch HSCLCS HCIP.

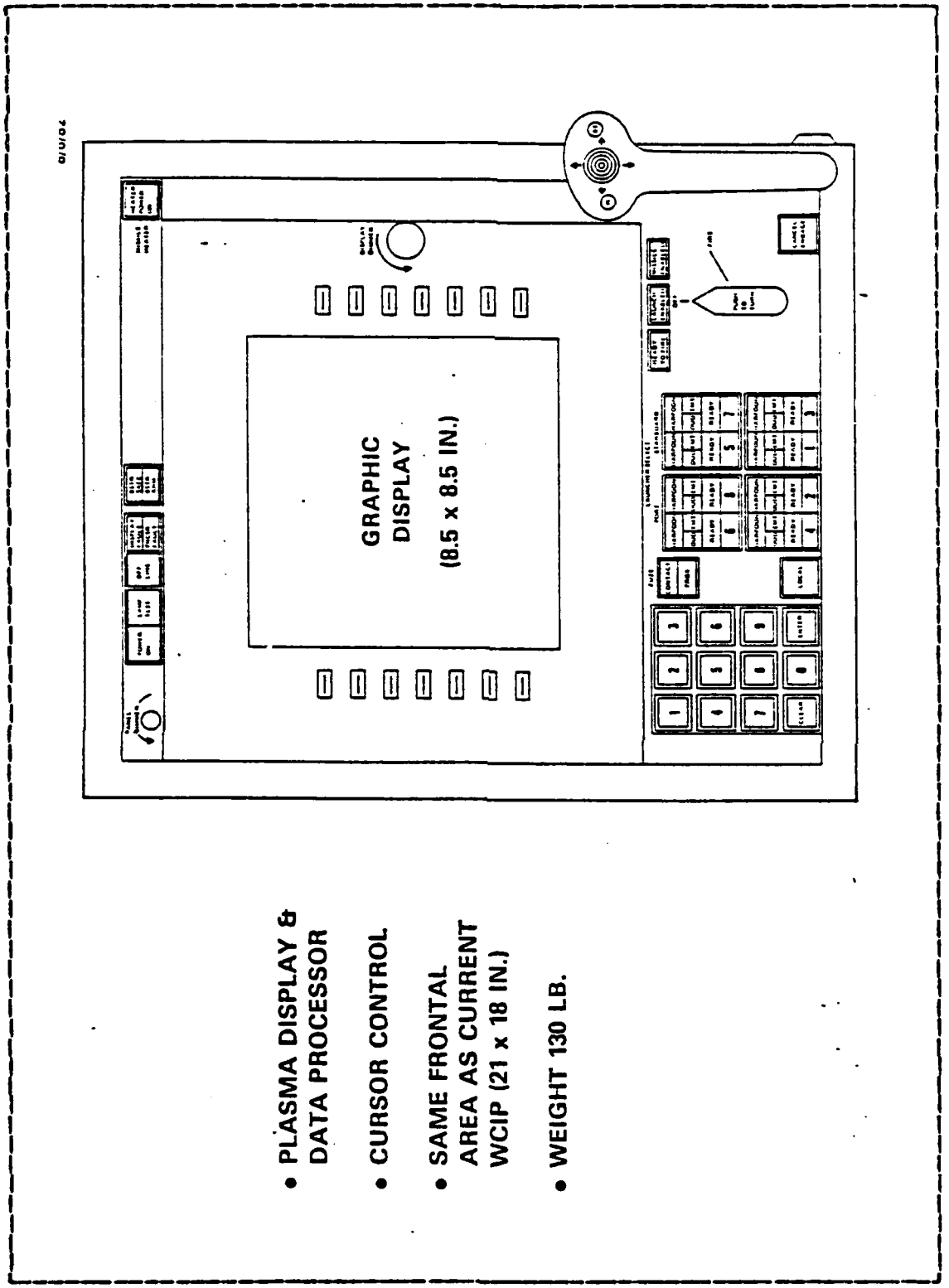


Figure F.3 Propcsed Cannister Launch HSCLCS WCIP.

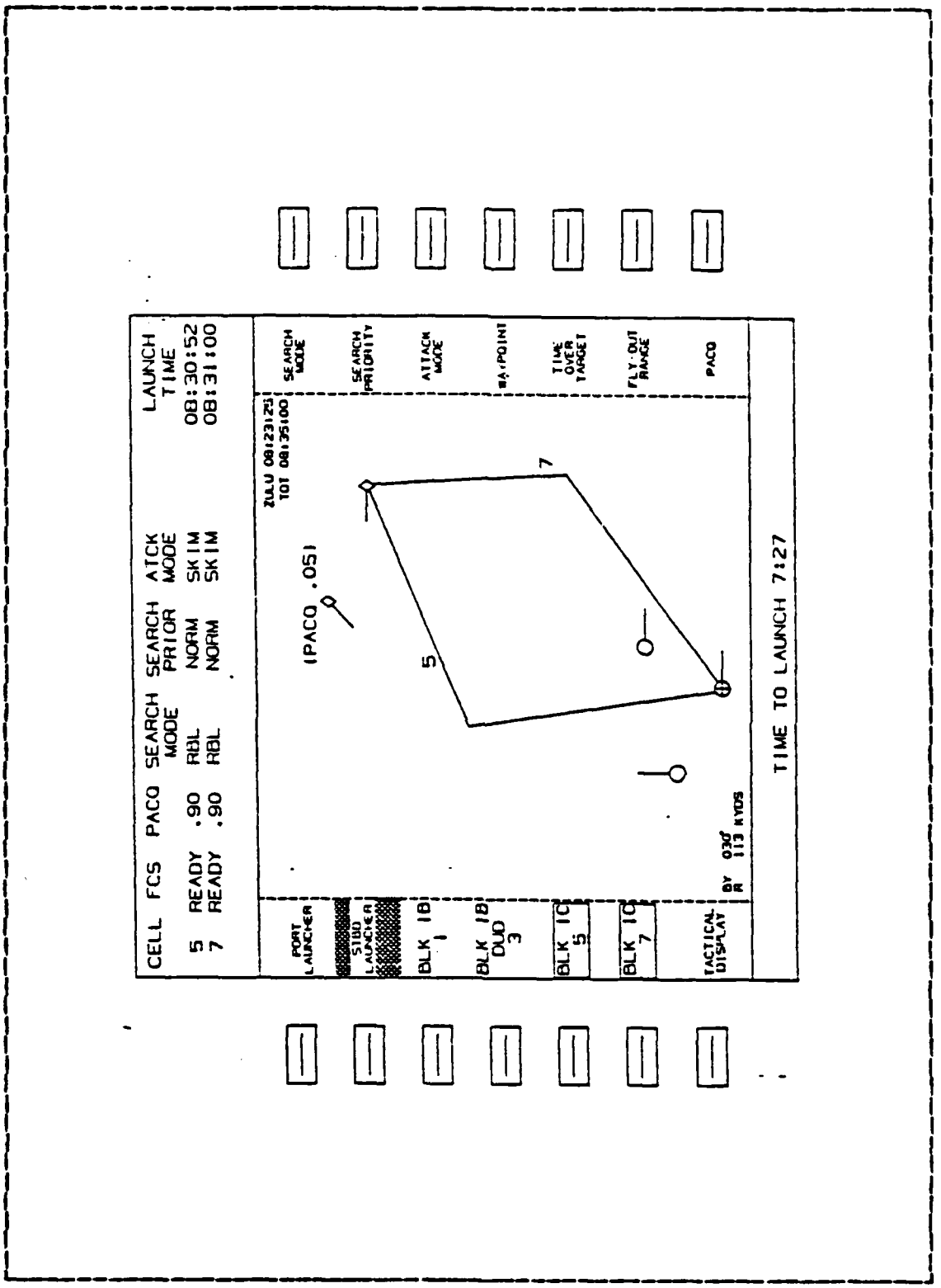


Figure F.4 Sample Display from Proposed WCIP.

LIST OF REFERENCES

1. Maroney, Randall and Sentman, Lawrence, Integrated Design Specifications for the Harpoon Shipboard Command-Launch Control Set, Master's Thesis, Naval Postgraduate School, Monterey, California, December 1982

2. Olivier, Daniel and Olsen, Kevin, A Design Methodology for Embedded Weapons Systems using the Harpoon Shipboard Command-Launch Control Set (HSCLCS), AN/SWG-1A, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1983

3. Ross, Douglas T., Goodenough, John B., and Irvine, C. A., "Software Engineering: Processes, Principles, and Goals", Computer Magazine, (May 1975), pp. 54-55

4. Boehm, Barry W., "Software and Its Impact: A Quantitative Assessment", Datamation Magazine, Vol. 19, 5(May 1973)

5. Pressman, Roger S., Software Engineering: A Practicner's Approach, Mcgraw-Hill Book Company, 1982

6. Yourdan, E. and Constantine, L. L., Structured Design, Yourdan Press, 1978

7. De Marco, Tom, Structured Analysis and System Specification, Printice-Hall Book Company, 1978

8. Booch, Grady, Software Engineering with ADA, The Benjamin/Cummings Publishing Company, Inc., 1983

9. Shooman, Martin R., Software Engineering, McGraw-Hill Bock Company, 1983

10. Belady, L. A. and Lehman, M. M., The Characteristics of Large Systems, MIT Press, 1979, pp. 106-138

11. Parnas, David L., "On the Criteria To Be Used in Decomposing Systems into Modules", Communications of the ACM, Vol. 15, 12(December 1972)

12. Rentsch, Tim, "Object Oriented Programming", SIGPLAN Notices, 17, 9(September 1982), pp. 51-57

13. Eschliman, Judd, Preliminary Design for HSCLCS Simulation, Master's Thesis, Naval Postgraduate School, Monterey, Califcrnia, December 1983

BIBLIOGRAPHY

Artzer, S.P. and Neidrauer, R.A., Software Engineering: A Primer for the Project Manager, Master's Thesis, Naval Postgraduate School, Monterey, California, June 1982

Bauer, F. L., Software Engineering, An Advanced Course, Springer-Verlag, 1977

Boehm, Barry W., Software Engineering Economics, Prentiss-Hall, Inc, 1981

Gore, Marvin and Stubbe John, Elements of Systems Analysis, WM. C. Brown Company Publishers, 1983

Katzan, Harry, Jr., Systems Design and Documentation, Van Nostrand Reinhold Company, 1976

Laden, H. N. and Gildersleeve, T. R., System Design for Computer Applications, John Wiley and Sons, 1965

Machol, Robert E., System Engineering Handbook, McGraw-Hill Book Company, 1965

Martin, James, Design of Real Time Computer Systems, Prentiss-Hall, Inc, 1967

Martin, James and McClure, Carma, Software Maintenance, Prentiss-Hall, Inc., 1983

Myers, G., Composite Structured Design, Van Nostrand, 1978

Sommerville, I., Software Engineering, Addison-Wesley Publishing Company, 1982

Stevens W., Myers, G. and Constantine, L., "Structured Design", IBM System Journal, Vol. 13, 2, 1974, pp. 115-139

Tou, Julius T., Software Engineering, Academic Press, 1971

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943	2
2.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
3.	Department Chairman, Code 54 Department of Administrative Sciences Naval Postgraduate School Montery, California 93943	1
4.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
5.	LCDR James I. Ransbotham, Jr., USN 597 D Michelson Rd. Monterey, California 93940	1
6.	LCDR Donald F. Mocrehead, Jr., USN 4716 Dermott St. Chesapeake, Virginia 23320	1
7.	Charles Arnold NUSC New London, CT 06320	1
8.	CDR James Williamson, USN Joint Cruise Missile Project Office, JCM-52 Bldg. NC1 2511 Jefferson Davis Highway Washington, D.C. 20363	1
9.	Naval Postgraduate School Computer Technologies Curricular Office Code 37 Monterey, California 93943	1
10.	LCDR Ronald Modes, USN VAQ 129 NAS Whidbey Island Oak Harbor, Washington 98278	4
11.	LCDR Ronald Kurth, USN Department of Computer Science Naval Postgraduate School Monterey, California 93943	1
12.	NSWSES Code 4613 (Mr. Paul Gognon) Port Hueneme, California 93043	1

13. Mr. Fred S. Gais 1
Director - HARPOON Ship Integration
McDonnell Douglas Astronautics Company
St. Louis Division
P.O. Box 516
St. Louis, Missouri 63166

14. Naval Sea Systems Command 1
Department of the Navy
ATTN: Mr. Lee Minin
NAVSEA Code SEA-62WB
Washington, D.C. 20362