UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER RADC-TR-79-168	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (*and Subtitle*) SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY (DEVELOPMENT)		5. TYPE OF REPORT & PERIOD COVERED Final Technical Report Aug 78 - Mar 79
		6. PERFORMING ORG. REPORT NUMBER N/A
7. AUTHOR(*s*) M. W. Alford J. T. Lawson		8. CONTRACT OR GRANT NUMBER(*s*) F30602-78-C-0026
9. PERFORMING ORGANIZATION NAME AND ADDRESS TRW Defense and Space Systems Group 7702 Governors Drive West Huntsville AL 35805		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55811805
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIE) Griffiss AFB NY 13441		12. REPORT DATE June 1979
		13. NUMBER OF PAGES 204
14. MONITORING AGENCY NAME & ADDRESS(*if different from Controlling Office*) Same		15. SECURITY CLASS. (*of this report*) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A

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*)

Same

18. SUPPLEMENTARY NOTES

RADC Project Engineer: Roger W. Weber (ISIE)

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

Software Requirements Methodology
Integrated Tools
Data Processing System Engineering
Requirements Definition
Requirements Validation
Requirements Traceability
Requirements Decomposition
Requirements Allocation
Embedded Computer System Software Process Design

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

This report documents the results of a one-year study of the front-end problems involved in the development of complex weapon systems and their embedded real-time software. Means to alleviate those problems through an integrated requirements engineering system supported by automated tools are proposed. The body of this report is organized into three parts. Part I is an Executive Summary which briefly outlines the purposes and accomplishments of the study. Part II explores the nature of weapon systems, requirements,

DD FORM 1473 1 JAN 73

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (*When Data Entered*)

front-end problems, characteristic activities and problems of front-end development phases, as well as candidate tools for addressing those problems. Part III presents formal mathematical foundations for front-end requirements engineering and design, and outlines a methodology that can be supported by a fully integrated set of tools. More than fifteen existing automated systems of tools and techniques were evaluated for application to the front-end problems. Of these, nine were selected for further consideration, because of unique properties, or global concepts that could be applied across an integrated system.

Mr. Rosell

From Current Awarenes

RADC-TR-79-168
Final Technical Report
June 1979

SOFTWARE REQUIREMENTS ENGINEERING METHODOLOGY (DEVELOPMENT)

TRW Defense and Space Systems Group

M. W. Alford
J. T. Lawson

ROME AIR DEVELOPMENT CENTER
Air Force Systems Command
Griffiss Air Force Base, New York 13441

RADC-TR-79-168 has been reviewed and is approved for publication.

APPROVED: Roger W. Weber

ROGER W. WEBER
Project Engineer

APPROVED: Wendall C Bauman

WENDALL C. BAUMAN
Colonel, USAF
Chief, Information Sciences Division

FOR THE COMMANDER: John P. Huss

JOHN P. HUSS
Acting Chief, Plans Office

PREFACE

The body of this report is organized into three parts. Part I is an Executive Summary which briefly outlines the purposes and accomplishments of the study. Part II explores the nature of weapon systems, requirements, front-end problems, characteristic activities and problems of front-end development phases, and candidate tools for addressing those problems. Part III presents formal mathematical foundations for front-end requirements engineering and design, and outlines a methodology that can be supported by a fully integrated set of tools.

A structured evolutionary development plan that leads to a fully integrated set of tools in six years, with usable interim increments, is reported in a separate interim report, TRW Document No. 32697-6921-001, which is CDRL Item A002 of this contract.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

TABLE OF CONTENTS (Continued)

LIST OF ILLUSTRATIONS

LIST OF ILLUSTRATIONS (Continued)

LIST OF TABLES

EVALUATION

By the early 1970's, the high cost and poor quality of software development was recognized as a critical issue on large high-technology DoD programs. Techniques for software development were not keeping pace with the increase in system complexity. Software Engineering, as an emerging discipline, was focusing on the more visible activities of software construction and test; however, the major cause of inadequate software, poor requirements definition and design, had been relatively neglected by the R&D community. The few efforts which did address the more tangible pre-coding and pre-design activities, yielded prototype developments for specific and limited applications. These were products of isolated research teams. Lacking was a documented and useful description of the system and software development process. A broad and comprehensive view of the initial user-developer interactions was needed; one which portrayed goals and alternative solutions, in spite of complexity, being successively defined and refined within a framework of effective common understanding.

This contractual effort, part of RADC TPO 5, Software Cost Reduction, addresses three principal technical needs: (1) Definition of front-end processes (concept definition, requirements validation, and preliminary design); (2) Identification of capabilities and limitations of existing automated support tools and methods; and (3) a comprehensive R&D plan to evolve and demonstrate an integrated requirements engineering support system.

The comprehensive study draws heavily upon pioneering research of the U.S. Army Ballistic Missile Advanced Technology Center (BMDATC) which has been concentrating on disciplined system and software engineering methods. A promising methodology approach, based upon formal mathematical foundations, has been identified. A common tool approach has been suggested wherein all development phases would be supported by a single nucleus of software utilities employing a single meta-language, data base analysis, and simulation generation concept.

The recommended approach takes into account the DOD emphasis on a high level programming language (DOD-1/Ada) and on critical technical issues involved in the design of distributed processing systems. The methodology underlying the Proposed Development addresses known problems elsewhere unaddressed; hence it represents a significant advance, possibly a breakthrough, toward early identification and resolution of critical data processing issues in the system design front-end.

Roger W. Weber
ROGER W. WEBER
Project Engineer

PART I

1.0 EXECUTIVE SUMMARY

1.1 INTRODUCTION AND BACKGROUND

The development of large, high-performance weapon systems has always been one of the most technology-stressing activities undertaken by man. In modern times, these systems consume large amounts of technical and economic resources, but all too often do not work as intended, or do not work at all. Because the need for these systems is driven by the potential capabilities of hostile adversaries, development takes place in an atmosphere of constant schedule pressure. To meet schedules, large groups of people, often involving many agencies and organizations must work in close coordination at the breaking point of practical manageability. In this environment, solutions to problems cannot await the natural evolution of powerful technical and management techniques to comfortably deal with the issues.

Computer software plays a vital role in the modern weapon system -- either as a controller of weapon system operations and resources, or as a critical link in the organization and presentation of information to human tactical commanders. Software, because of its abstract nature and relatively short history as an engineering discipline, has been costly and difficult to develop for large, complex weapon systems.

By the early 1970's the high cost of software development (and the fact that software often did not meet operational needs) was becoming a critical and visible programmatic issue on ultra-high technology defense programs. The techniques for software development were simply not keeping pace with the increasing complexity of weapon systems. Several major studies identified poor software requirements definition as a major cause of costly, inadequate software. In the past few years the software problems, first perceived in the high-technology defense community, have become increasingly visible in commercial and industrial systems. The degree of concern is indicated by the number of conferences and workshops devoted to the topic. In the month of April 1979, three such events in the United States and Europe will focus on requirements and related problems.

Pioneering research in requirements engineering by the U.S. Army Ballistic Missile Advanced Technology Center (BMDATC), the ISDOS project at the University of Michigan, and others, produced a number of tools and techniques (e.g., CARA, SREM, PDS) to address requirements-related problems. However, these tools were developed for specific applications and specific phases of front-end development, and were developed by groups working in isolation from each other. A broad, comprehensive view of the entire front-end system development process and its impact on software requirements has been needed to provide a basis for an integrated attack on the total requirements problem.

1.2 PROJECT PURPOSE AND SCOPE

In FY 1978, RADC sponsored the Software Requirements Engineering Methodology (Development) study. The purpose of the study was to define a unified methodology approach and recommend an evolutionary development plan for construction of an integrated requirements engineering system supported by automated tools to address Air Force requirements problems.

The statement of work consisted of five tasks:

- Identify current state-of-the-art tools and techniques applicable to software requirements and preliminary design.
- Investigate the front-end problems of data processing system development.
- Investigate how the identified tools and techniques can be applied to the front-end problems, identify gaps, and recommend improvements and additional tools.
- Identify approaches for a methodology to effectively use the tools.
- Prepare an evolutionary development plan for constructing an integrated requirements engineering system.

Because software problems often originate from earlier system level decisions, the scope of the study was to include all development effort from first perception of the need for a weapon system to preliminary software design.

1.3 PROJECT ACCOMPLISHMENTS AND CONCLUSIONS

The project has performed all of its tasks and met its objectives. Specifically, the project made the following accomplishments.

- The characteristics of weapon systems, of the development of weapon systems, and of requirements were identified and studied.
- The phases of weapon system and software front-end development, and their problems, were analyzed. Although each phase has its own manifestation of problems, the various phases were found to have a common set of problems associated with human thought processes, information organization, decision-making and communication.
- More than fifteen existing automated systems of tools and techniques were evaluated for application to the front-end problems. Of these, nine were selected for further consideration, because of unique properties, or global concepts that could be applied across an integrated system.
- Three approaches for integration of the tools were evaluated. Of these, a common tool approach was selected, wherein all development phases would be supported by a single nucleus of software utilities employing a single meta-language, data base analysis, and simulation generation concept. The single meta-language provides the foundation for an extensible language capability to express the specialized vocabulary and concepts appropriate to each development phase.
- A promising methodology approach based upon formal mathematical foundations was identified and evaluated. This approach is based upon break-throughs made on two TRW programs for BMDATC (Axiomatic Requirements Engineering, and Advanced Data Processing Concepts) in 1978.

These basic research results have been evaluated as they emerged and are found to be applicable to Air Force weapon system problems. In particular, they provide formal foundations and insights for the proper placement of tools within an integrated system.

- An evolutionary development plan for the construction of an integrated requirements development system and its transfer to the Air Force was prepared. Aggressive implementation of this plan could lead to a complete capability in six years. Forty-nine R&D tasks in the areas of technology consolidation, technology extension, and technology transfer were identified and evaluated. These tasks were then grouped into twenty-nine packages for time-phased procurement with consideration for incremental capability delivery. (This plan is separately reported in TRW Document 32697-6921-001, which is CDRL Item A002 of this contract.)

This project has found a common body of requirements-related problems existing across all phases of front-end development. An integrated approach to solving these problems using a common nucleus of automated tools appears to be feasible, practical, and beneficial.

1.4 RECOMMENDATIONS

Timely and aggressive research in this field is needed because the advent of distributed processing systems and startling advances in hardware technology foretell an explosive increase in the complexity of technically feasible systems. We have barely mastered fairly good software engineering approaches for conventional single-processor systems, yet we are about to be engulfed by a tidal wave of hardware capabilities that offer the potential of spectacular software successes or failures. We now have to run when we barely know how to walk.

It is recommended that the Air Force give critical consideration to early sponsorship of critical-path research increments identified in the evolutionary development plan. Initial introduction of the powerful DoD-I programming language into operational use is expected in 1982-83. If substantial progress is not made in reducing front-end development problems by the early 1980's, the downstream software engineering problems will be compounded far beyond the levels that arouse alarm today.

It is further recommended that on-going research in the ballistic missile defense community be continuously monitored for application and adaptation to Air Force use. Earlier research sponsored by BMDATC has been of great potential benefit for real-time systems outside the BMD focus of interest. Continuing research is expected to refine and clarify the gross methodology themes presented in Part III of this report, and is expected to introduce new tools of potential interest to the Air Force.

PART II

1.0 INTRODUCTION

This report documents the results of a one-year study of the front-end problems involved in the development of complex weapon systems and their embedded real-time software, and means to alleviate those problems through an integrated requirements engineering system supported by automated tools.

1.1 BACKGROUND

The development of large, high-performance weapon systems has always been one of the most technology-stressing activities undertaken by man. In modern times, these systems consume large amounts of technical and economic resources, but all too often do not work as intended, or do not work at all. Because the need for these systems is driven by the potential capabilities of hostile adversaries, development takes place in an atmosphere of constant schedule pressure. To meet schedules, large groups of people, often involving many agencies and organizations, must work in close coordination at the breaking point of practical manageability. In this environment, solutions to problems cannot await the natural evolution of powerful technical and management techniques to comfortably deal with the issues.

Nowhere are the problems of complexity more apparent than in the area of software development. Software, by nature, is an abstraction. It is an entity that produces actions and behavior completely unrelated to its physical form. Correctly defining the requirements for software (i.e., the actions it will cause based on specified information) is a major logical and conceptual effort, even when schedule is not a consideration. Yet the correct definition of software requirements is critical to the development of successful weapon system software and vital to the success of the weapon system mission. This is because software exercises partial or nearly total control over the operation of the modern weapon system and its resources. Even where its control functions are minimal, software plays a vital role in processing and displaying the information that is the basis for tactical judgements by humans.

By the early 1970's the high cost of software development was becoming a critical and visible programmatic issue on ultra-high technology programs such as ballistic missile defense. Several studies at that time revealed the staggering cost penalties of late detection of requirements and design errors. Figure 1-1 illustrates the relationship.

Realizing the high cost leverage of error-free requirements, in 1973 the U. S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC) initiated pioneering research to address the issues of requirements engineering and process design for real-time weapon system software. This effort culminated in 1977 with the delivery of the TRW Software Requirements Engineering Methodology (SREM) and the Texas Instruments Process Design System (PDS). Current BMDATC research is focused on bringing the same rigor to system engineering disciplines related to data processing, and to problems in distributed processing.

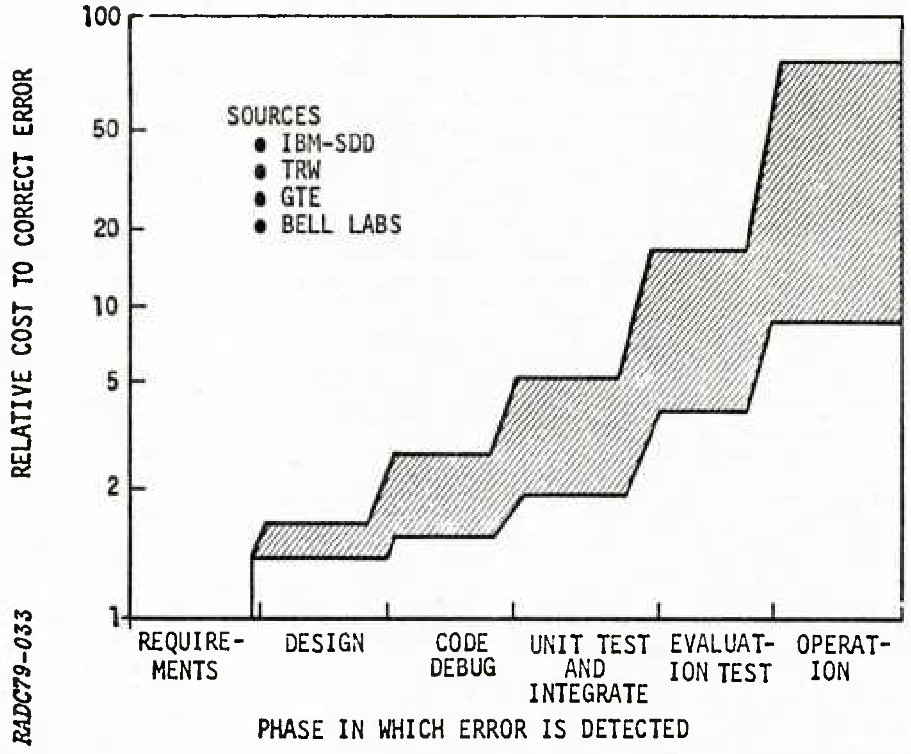


Figure 1-1 The Penalty of Requirements Errors

Sufficient research experience has been gained to postulate that an integrated set of requirements engineering tools to address the requirements definition problems from initial weapon system concept to preliminary software design is feasible and practical. RADC has funded the present Software Requirements Engineering Methodology (Development) study to confirm that belief and to define an evolutionary development plan for a system adapted to Air Force requirements engineering problems.

Timely and aggressive research in this field is needed because the advent of distributed processing systems and startling advances in hardware technology foretell an explosive increase in the complexity of technically feasible systems. We have barely mastered fairly good software engineering approaches for conventional single-processor systems, yet we are about to be engulfed by a tidal wave of hardware capabilities that offer the potential of spectacular software successes or failures. We now have to run when we barely know how to walk.

1.2 OVERVIEW OF PART II

Section 2.0 presents a context for investigation of the problems and issues surrounding requirements engineering for weapon systems and their embedded software. Section 2.1 describes some of the properties of weapon systems, emphasizing fundamental concepts common to all weapon systems, regardless of their detailed design or implementation technology. In Sections 2.1.1, 2.1.2, and 2.1.3 we discuss generic characteristics, components, and component interactions. In Section 2.1.4 we discuss the one-on-one encounter between a weapon system unit and a threat. The encounter level of consideration is readily amenable to standard engineering analyses and is the usual first step toward modeling the system's operation. In Section 2.1.5 we discuss weapon system engagements -- concurrent encounters between a weapon system and multiple threats. Engagements present complex control and resource management problems. Operations research disciplines have found these problems to be difficult, sometimes impossible, to analyze and model with any fidelity. We conclude Section 2.0 by recapitulating the current primary specification types applicable to weapon systems, subsystems, and software as defined in MIL-STD-490.

Section 3.0 discusses requirements. In Section 3.1 we present a hierarchy of requirements types (processing, non-processing, development) affecting software. In Section 3.2 we discuss various other characteristics of requirements, and the relationship between requirements and design. In Section 3.3 desirable attributes of software requirements specifications are summarized.

Sections 4.0 and 5.0 examine front-end development problems and their manifestations in the various development phases. Section 4.0 discusses five specific problems: complexity, communication, validation, traceability, and change response. These problems appear to be at the root of many observable symptoms of poor requirements. In Section 5.0 we characterize six phases of the front-end system and software development cycle, and for each phase, discuss the scope, content, and problems of the phase. In Section 5.7 we conclude that, despite superficial differences, the phases have a common set of problems.

In Section 6.0 we identify and summarize our evaluation of candidate tools and integration approaches for producing an integrated requirements engineering system. References cited in Part II are listed in Section 7.0.

2.0 PROBLEM CONTEXT

Our investigation deals with the front-end development problems for a particular class of software: that developed to support the operation of weapon systems. Before we examine the accompanying requirements definition problems, let us examine the features of weapon systems to determine why they present exceptional software development problems.

In Section 2.1, we begin by stating the salient characteristics of weapon systems. Next we identify the fundamental generic components common to all weapon systems and examine the types of interactions between these components, and between the components and the threat and environment. Finally, we examine the characteristics of single weapon system/threat encounters and then discuss aggregates of encounters called engagements. In Section 2.2, we summarize the current specification standards for weapon system software and for systems and subsystems in which the software is embedded.

2.1 THE WEAPON SYSTEM CONTEXT

The problems of software development for weapon systems differ from those of civilian applications in many respects. This is due to the unique nature of weapon systems. In this section we will discuss the characteristics, components, interactions and encounter sequences of weapon systems that form the context of the requirements development problem.

2.1.1 Characteristics of Weapon Systems

In the modern world, six characteristics are implicit in the concept of a "weapon system":

1) A weapon system is an organization of men and equipment designed for use against specific classes of enemy targets under certain presumed operating conditions and rules of engagement. The input to the system is an enemy target or group of targets. The output of the system is the destruction of the enemy targets, usually required to be accomplished before the targets can contribute to significant damage upon friendly forces or facilities.

2) A weapon system is a real-time system. The effectiveness of the system is dependent upon the ability to respond to an input within a specified time. The required performance of a defensive weapon system is defined by the characteristics of the input enemy offensive threat and the desired reduction of that threat. The required performance of an offensive weapon system is defined by the characteristics of the target, the characteristics of enemy systems defending the target, and the minimum acceptable damage to be inflicted on the target.

3) Without modifications to the system, the effectiveness of any weapon system is reduced with the passage of time. The enemy will upgrade his offensive threat in numbers and sophistication to maximally stress

and hopefully break the defense. He will also upgrade his defense to minimize damage to his own targets. Because each side strives to minimize its resource expenditures, the requirements for a weapon system must change and evolve over time.

4) A weapon system is embedded in a chain-of-command hierarchy reaching to the highest levels of government. Although local commanders may be given discretionary authority to use ordinary tactical weapons in response to given situations, authority to use a weapon system is always conditional and granted from above. The weapon system designer must meet requirements for interfaces with one or more command and control systems, and must incorporate features to preclude unauthorized use of critical weapon systems.

5) A large weapon system is difficult or impossible to test under realistic combat loads and conditions prior to operational deployment and use. Simulation methods must be used to represent the threat, environment, and certain system components and actions. The question then becomes "Do the simulators accurately model real physical phenomena, event timing, threat characteristics, and enemy tactics?"

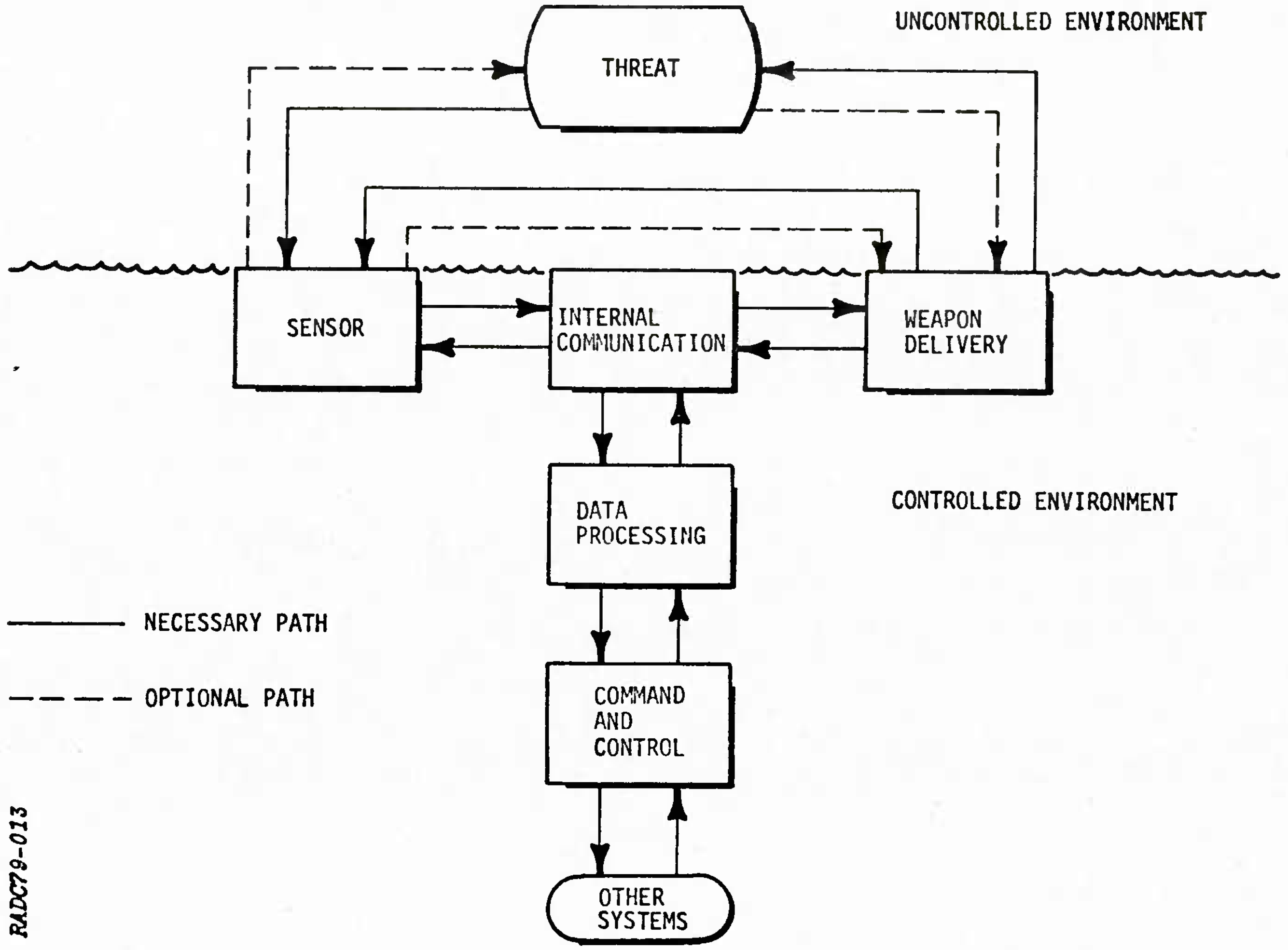
6) A weapon system is actually engaged in a mission for a miniscule fraction of its deployed lifetime. For critical strategic systems, there is no opportunity to resolve erroneous assumptions by trial-and-error means. Such systems are designed to be used once, and failure would be catastrophic.

These characteristics separate weapon systems from most systems used in the commercial and industrial world. We should also expect that the methods of system development would be different. We find this to be true. Because the penalty for error is so high, all requirements that the system is to satisfy must be more carefully developed and validated. Because the system is pitted against an intelligent opponent, the requirements will be in a state of flux and will tax the limits of state-of-the-art technology.

Before we consider better means of developing requirements for weapon system software, we will examine the components, interactions and operating behavior of a typical weapon system.

2.1.2 Weapon System Components

While weapon systems exhibit many different forms and levels of complexity, all weapon systems are variants of a common underlying structure. A weapon system can be categorized as a responsive system, also called a second-order feedback system. A responsive system has a defined goal or mission, and has the capability to choose, from alternative tactics, the tactic most effective in the current operational situation. The basic weapon system model, presented in Figure 2-1, can be used to grossly describe responsive weapon systems ranging from a man with a rock to a sophisticated air defense system.



RADC79-013

Figure 2-1 Basic Weapon System

To effectively direct a weapon against a target, one must be able to: 1) detect the presence of a potential target, 2) establish that the detection is a desired target, and 3) predict the probable location and motion of the target at the time of intercept. A sensor of some type (eye, ear, radar, optical device) is required to permit these actions. Sensors may be active (e.g., radar) or passive (e.g., eye, electro-optical telescope). In either case, controls are generally needed to shift the field-of-view and adapt to environmental variations. A given system may employ a single sensor, a number of sensors of the same type, or a mix of sensors. Multiple sensors may operate independently (e.g., monostatic radars) or cooperatively (e.g., multistatic radars).

A weapon delivery mechanism is also needed to bring the weapon to the target if the target is outside the weapon's lethal radius. The oldest weapon delivery mechanism is the arm and hand used to throw the rock. Modern weapon delivery systems are often multi-stage (e.g., manned aircraft + air-air missile). All weapon delivery means have a limited action radius, velocity envelope, and correction capability. Effective weapon delivery therefore requires careful timing, aiming, preplanning and, where practical, real-time compensation and control. These activities in turn require knowledge of weapon delivery capability and limitations, estimation of current weapon delivery system state, prediction of future target state, a sense of time, and computational capability. Such a system must have memory.

Facilities for memory, computation, and timing coordination are provided by the data processing subsystem (DPSS). Until recently, the human brain served as the data processing subsystem for most weapon systems. As threat performance has increased, sensors and weapon delivery systems have become more complex. The unaided human operator can no longer keep pace with the data throughput encountered in modern systems. Hence, the bulk of data processing activities have been off-loaded onto computers or networks of computers.

Operation of a sophisticated weapon system requires situation assessment, timely selection of strategies and tactics, allocation of resources to accomplish chosen goals, and positive control over the system. Thus, every weapon system has a command and control (C&C) subsystem -- a decision making element. Ultimately, all command and control components of the system are human. Machines are used to structure information displays for humans and to assist in executing decisions made by humans, but they do not choose goals or make independent decisions except as preprogrammed. Although great progress has been made in computer systems, the human operator will not be totally replaced, because only he can respond to novel and unanticipated situations which call for original responses and value judgements.

The fifth necessary component of a weapon system is the internal communications subsystem. The other subsystems are ineffective unless they operate as an orchestrated whole. Coordinated action requires the capability to move information from one part of the system to another when needed. The internal communications subsystem provides this capability.

The weapon system does not exist in isolation. It is surrounded by the system environment, which is simply everything in the outside universe that is affected by, or has appreciable effect on the system. It is useful to separate the system environment into the controlled environment and the uncontrolled environment. The controlled environment is simply that which the system designer or operator can modify, or influence in some degree. Local air temperature, local electromagnetic radiation, interfacing system message formats, and engagement rules for external friendly systems are examples of controlled environment elements.

The uncontrolled environment is that which the designer or operator cannot modify or must accept as fixed. Final decisions by higher authority, tables of military organization and responsibility, the weather, the laws of physics, and the initial threat scenario are not controlled by the designer or operator. Even if these factors cannot be controlled, their range can be anticipated within limits and the system design can compensate for them to an acceptable degree. The hardest factor to anticipate is the threat, since it is the only component purposefully trying to defeat the weapon system.

The definitions of system, subsystem, and system environment are relative. Certainly what we define as a weapon system is merely a subsystem in the context of the total U.S. defense posture. In the other direction, the subsystems of a weapon system may be "systems" in their own right, with an environment consisting of the original system environment plus all other subsystems in the weapon system. For instance, we can focus on the sensors which search for and detect threats and call these a surveillance system or early warning system. Similarly, we can detach the sensors and weapon delivery subsystems and call the remainder a command, control, and communication (C^3) system. An important task of the weapon system analyst is to develop an understanding of the relationships between a system and its superordinate, subordinate, and coordinate systems.

2.1.3 Weapon System Component Interactions

The basic weapon system exhibits a characteristic pattern of interactions between its components and with the environment. These are shown as paths in Figure 2-1. Necessary paths are those found in all weapon systems. Optional paths are those characteristic of the classes of components used in the specific weapon system. These interactions define the nature of the information flow through the system.

The C&C subsystem interfaces with external systems and the chain of higher command. At some time, the system is activated by external authorization, perhaps accompanied by specific mission tasking orders, intelligence inputs and forward acquisition data from other systems. Supplemental directives and a termination order will enter the system at subsequent times during an engagement. At appropriate intervals the C&C subsystem will release situation reports, kill reports, and casualty reports to higher command levels, and will transmit processed mission data to other systems if needed. The C&C subsystem supervises transfer of bulk data between the data processing subsystem and external systems.

The C&C subsystem interacts with the rest of the weapon system through the data processing subsystem, and voice or teletype communication with various subsystem operators (not shown in the diagram). The C&C subsystem establishes the initial mission configuration of the DPSS which then implements lower level activities to bring the weapon system to readiness. During the mission, the C&C subsystem may provide resource allocation directives, tactical decisions, and requests for information to the DPSS. The DPSS provides data for summary information displays and responses for requested information to the C&C subsystem. The allocation of decision-making responsibility between the C&C subsystem and the DPSS is a function of the required system response time, the load on the system, and the variability or novelty of engagement situations. However, the C&C subsystem always controls termination of the mission and deactivation of the system.

The DPSS communicates with sensor and weapon delivery subsystem elements via the internal communication subsystem. The DPSS issues control commands to the sensor and the sensor returns partially processed observation data to the DPSS. The presence of a potential threat or target is indicated by sensor detection of electromagnetic or acoustic energy reflected or radiated by the object. Thus, there always exists a directed interaction path from the threat to the sensor. With active sensors (e.g., radar, sonar) the sensor transmits the energy reflected by the object. Hence, there is an interaction path from sensor to threat, not used with passive sensors. The interactions between threat and sensor occur in the uncontrolled environment, which attenuates the sought-after signals and corrupts them with noise. Further noise is introduced within the sensor itself (thermal noise). Complex analog or digital processing is required to recover the desired signal.

Before weapon delivery elements are committed, the weapon delivery subsystem provides health and status reports to the DPSS. When a target is identified and designated, a specific weapon delivery unit is selected by the DPSS (or by the C&C subsystem via the DPSS) to engage the target. The DPSS then transfers intercept planning information to the selected unit. This may consist of a completed intercept plan, or only target state vector data if the unit is to form its own plan. The DPSS may provide state vector updates to the weapon delivery unit at intervals.

Once the unit is launched, interactions with the remainder of the weapon system depend upon the type of weapon. Manned aircraft may be vectored to the target via communications from ground elements. Missiles may be guided through the sensor or separate guidance transmitters. Or, the weapon delivery unit may function autonomously, using on-board sensors to acquire and home in on the target.

There is usually an interaction path from the weapon delivery unit to the sensor, because the unit will appear in the sensor's field-of-view as another object. The sensor capacity must allow for both threats and weapon units. Observations of the unit as it closes on the target may be used for active guidance or for passive kill assessment.

Finally, there is an interaction between the weapon delivery unit and the threat (a hit or a miss) which decides the outcome of the encounter. A detected miss would lead to commitment of another weapon delivery unit, if feasible.

2.1.4 Weapon System Encounter Sequence

From the general description of a basic weapon system we can see potential for major variations of the basic system interactions, determined by the particular choices of weapon delivery subsystem, guidance mode, sensor/DPSS processing allocation, and command and control philosophy. However, there is an underlying commonality expressed in the encounter sequence of events shown in Figure 2-2. Within this sequence, there are dominant information flows and types of data processing activity. In this section we will describe an encounter typical of a defensive weapon system. With minor changes, the sequence could apply to an offensive weapon system.

An encounter can be divided into three basic phases -- observation, decision, action. The observation phase begins when the sensor commences search and ends when enough information has been gathered to determine if a threat exists. The decision phase begins with a decision that a threat exists and ends with a decision that launches a specific interceptor. The action phase begins with interceptor launch and ends with a positive kill assessment.

The observation phase can be further subdivided into search, detection, track, and discrimination subphases. Throughout the observation phase, the dominant information flow and data processing in the weapon system is between the sensor and the DPSS. The flow from sensor to DPSS is characterized by high throughput, repetitive signal processing, thresholding, peak detection, correlation, association, and state estimation. The information arriving in the DPSS is used to adjust sensor control parameters and, for active sensors, define and schedule sensor transmission and reception. The tight coupling between the sensor and DP, the large bandwidths of modern sensors, and the precise synchronization to be maintained lead to stringent real-time performance requirements on the DPSS. In the past, these requirements could only be met by expensive special purpose hardware or high throughput "super-computers". The advent of low cost LSI components and microprocessors offers the potential for specialized high performance architectures at reasonable cost. However, this potential cannot be realized until data processing considerations are given more weight in system definition activities.

- Is the object a threat?
- Should it be engaged?
- Can it be engaged?
- What interceptor shall be assigned?
- Is a back-up feasible and necessary?
- When should the interceptor be launched?
- What are the side effects?

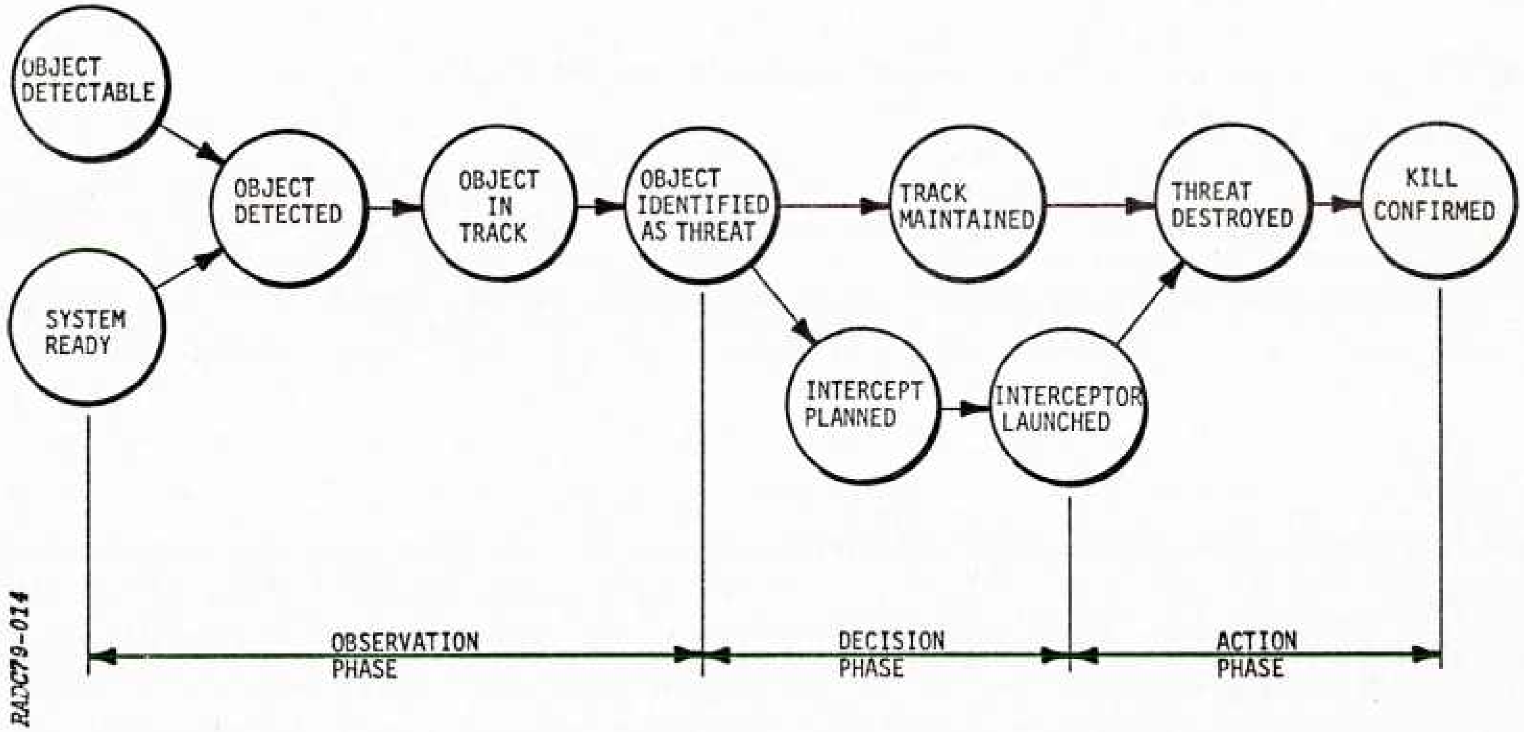


Figure 2-2 Nominal Encounter Event Model

The measurements made during the observation phase are designed to reveal the observable characteristics of the object and determine its possible or probable destination. Comparison of the observations against a data base of friendly and hostile force observables results in a probabilistic identification of the object. Projected friendly and hostile force movement data may be used to refine the identification. In the event of a high system load, objects not likely to be threats may be dropped from the system in favor of more likely threats.

The decision to engage a threat is a function of the reliability of identification, the defended target threatened, the available system resources, and the overall battle situation. The identification and engagement decisions may be performed entirely in the DPSS using prespecified decision algorithms. Or, the decisions may be made by human operators in the C&C subsystem, on the basis of supporting computations from the DPSS and other information.

The remaining decisions leading to interceptor launch are performed within the DPSS based upon interceptor performance envelopes and current status data, or are performed by some combination of DP and C&C subsystem resources. Processing loads are not severe, unless a large number of alternatives must be examined. However, the processing can be logically complex and potentially requires access to data on any element in the system.

During the decision phase, the sensor and DP subsystems are holding the target in maintenance track. When an interceptor has been selected, target state information is routed to that interceptor and updated as necessary until interceptor launch.

The two major problems of the action phase are: 1) vectoring the interceptor to the target, and 2) determining if the intercept was successful. The data processing rate required for interceptor control is proportional to the acceleration characteristics of the interceptor and the maneuverability of the target. Ground-based guidance offers the potential to apply large-scale data processing power and centralized battle management, but can place a high load on the DPSS and sensors and can create serious resource scheduling conflicts and response lags. On-board guidance eliminates many of the problems of a tightly coupled system, but data processing capacity is severely constrained by size, weight, and power restrictions.

Kill assessment requires that the intercept be observed and that a kill can be distinguished from a non-kill. This activity is academic if there is no opportunity for a second shot at the target.

2.1.5 Weapon System Engagements

A weapon system engagement can be defined as a discrete set of encounters followed by a period of inactivity. Typically, an engagement must be fought with the resources on hand at the start of the engagement because repair and replenishment of resources is not feasible.

The visualization and design of a system to handle a single encounter is relatively easy compared to the task of visualizing and designing a system capable of successfully fighting the majority of possible engagements. The difficulties at the system level are essentially the same as those faced by the designer of a responsive data processing facility with uncertain demand, but compounded by more stringent response times, a hostile "user", and the fact that engagements vary in the space domain as well as the time domain.

Engagements are presumed to be fought under conditions of limited resources. To ensure that an attack is successful, the attacker must bring sufficient force to bear such that the defense is eventually overloaded or depleted. The defender is anxious to avoid development and maintenance costs for defensive capacity that is unlikely to be needed. Hence, he sizes his system according to the maximum force attack believed within the practical capability of the attacker, with some allowance for stronger attacks believed to be improbable. While the key element in winning a single encounter is performance, the key elements in winning an engagement are: adequate resources, effective management of scarce resources, and a system design such that overall performance degrades "gracefully" under overload (i.e., does not suddenly collapse under a small increase in attack strength).

There are numerous tradeoffs to be made between system performance and required resources in the design of an effective weapon system. To complicate matters, the effect of a performance change in one part of a system may show up as a significant change in resources needed in a completely separate part of the system.

To illustrate this point, we will consider a simple system responding to an attack scenario, as shown in Figure 2-3. For simplicity we will ignore the spatial geometry of the attackers and consider only their time sequencing. We shall first consider a system where the reaction time is just adequate to destroy a single attacker before he can inflict damage on the defended target. As described in the previous section, we will consider an encounter to be divided into observation, decision, and action phases, each phase demanding different resources. We will also consider that n concurrent encounters in a given phase demand n units of resource for that phase. We will further assume that one-half unit of observation resources will be committed to each attacker in the decision and action phases for purposes of maintenance track. All encounters and each phase within encounters will consume the same time for all attackers. Figure 2-3 shows the relative amounts of each resource needed versus time (i.e., the system load profile) to successfully fight the engagement.

Now let us postulate a performance improvement in the observation phase (e.g., improved track filter convergence, improved identification algorithm) such that the time required from first detection to threat identification is reduced by twenty percent. This reduces the system reaction time by 10 percent for a single encounter and introduces a slack time interval between first possible detection and latest permissable intercept. The revised engagement timeline and system load are shown in Figure 2-4. The heavy dots at the left of the engagement timeline represent the earliest detection point for an attacker and the X's at the right indicate the latest permissible intercept.

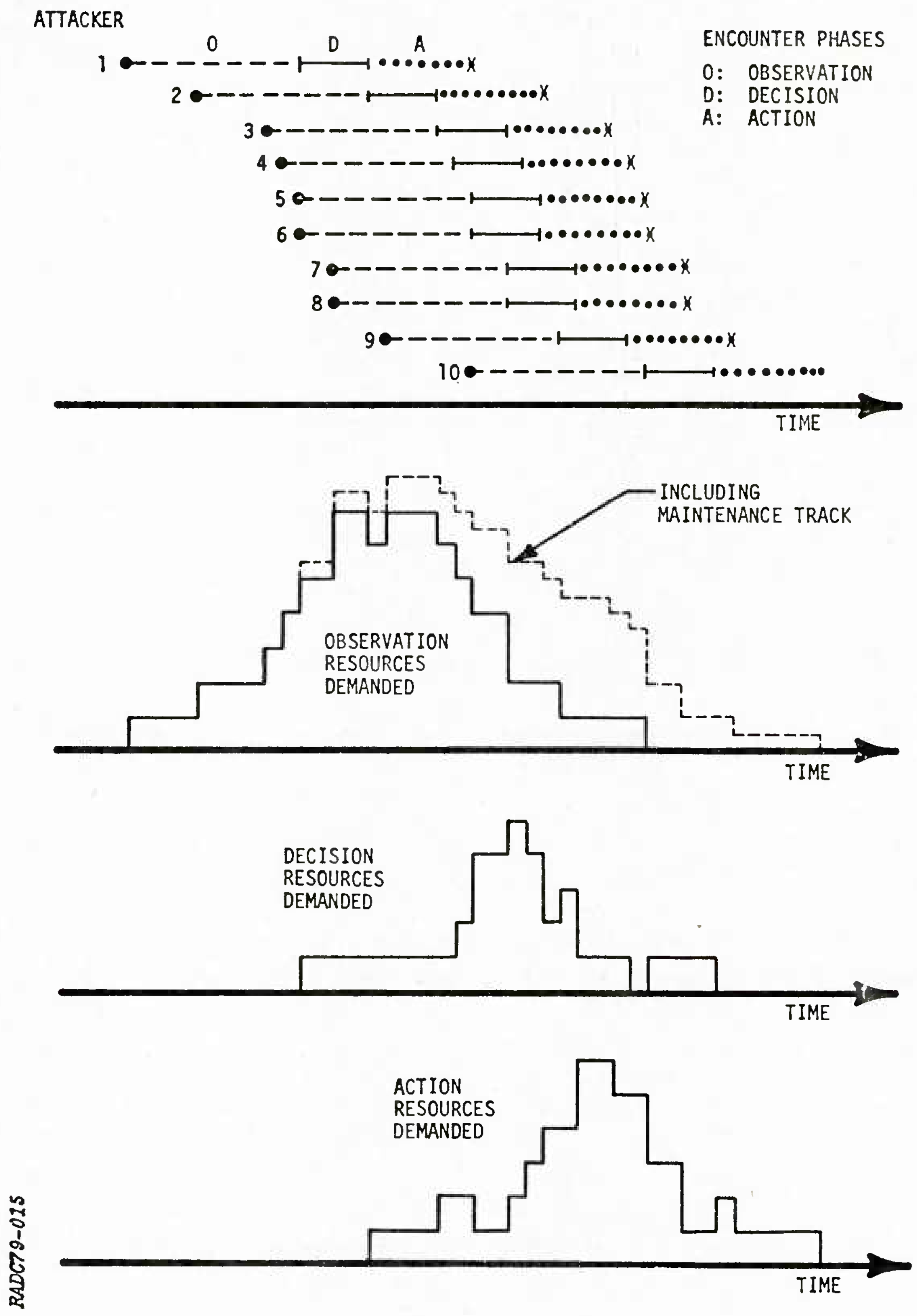
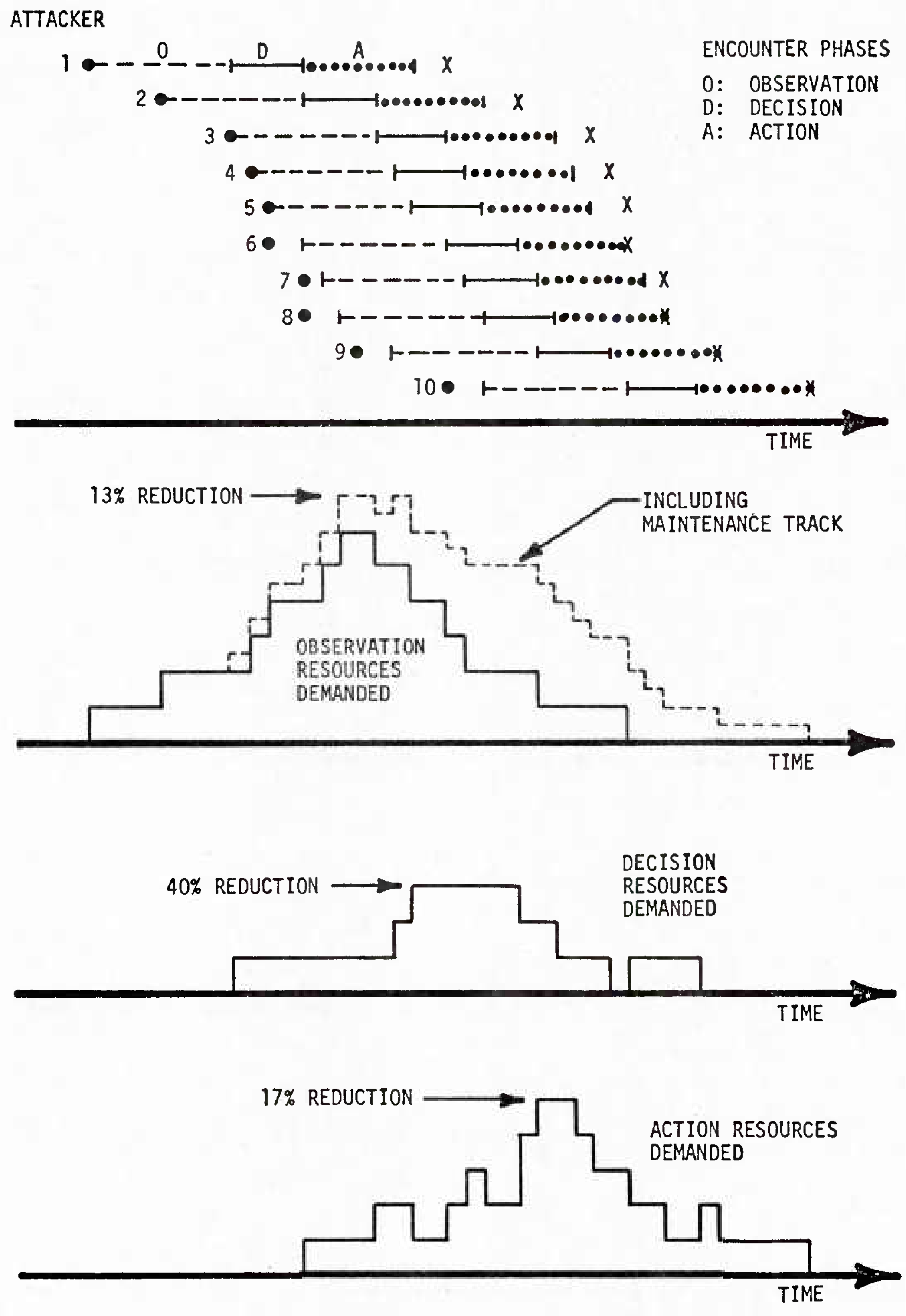


Figure 2-3 System Load During Example Engagement



RADC79-016

Figure 2-4 Revised System Load

Because of the slack time in the encounter, handling of certain encounters can be delayed somewhat until busy resources are freed. Hence, the maximum needed amount of resources can be reduced. Strangely enough, the performance improvement in the observation phase permits 40 percent reduction of decision resources and 17 percent reduction of action resources, but only 13 percent reduction in observation resources. Even stranger phenomena occur in scheduling theory where it can be shown that sometimes improved performance of individual tasks (i.e., greater speed) can actually lengthen the minimum schedule to perform a set of tasks [Ref. 1].

The principle problems of engagement planning and analysis can be characterized as widely-studied resource allocation and scheduling problems addressed by operations research. One type of problem is the 1 x n assignment problem (i.e., given one interceptor and n targets, which target should be intercepted?). Another is the m x 1 assignment problem (i.e., given m interceptors and one target, which interceptor should be tasked to perform the intercept?). The solutions to these problems are highly context-dependent. The general problem is the m x n assignment problem which is practically solvable under very restricted conditions and simplifying assumptions. The 1 x n scheduling problem can be stated as: given one resource and n tasks that utilize the resource (with specified arrival times, execution times and possibly predecessor-successor constraints) what is the sequence of task execution that results in the minimum completion time for all tasks. The general m x n scheduling problem permits variable allocation of resources to minimize the schedule.

Despite all the research devoted to these problems by operations researchers over three decades, practical techniques for finding optimal solutions without extensive computation have not been found except for very limited cases. Some heuristic techniques have been invented that yield near-optimal solutions with certain assumptions. The absence of powerful analytical techniques has led to reliance on simulation as the primary tool for verifying the adequacy of proposed weapon system designs.

Engagement management has traditionally been performed by human tactical commanders, and the role of automation has been to collect, consolidate, and display relevant data for input to human decisions. This permits the commander, trained in military science, to make a variety of situation assessments and introduce novel tactics in response to the particular real-time situation. When engagement management is automated, the designer must anticipate all possible contingencies and develop algorithms to yield effective system response. If the appropriate tactical responses are not delineated in the system requirements, the military user is effectively surrendering command of the system to technologists who may be completely ignorant about military science.

2.2 SPECIFICATION STANDARDS

The current specification standards for requirements statements in the weapon system development process are stated in MIL-STD-490 and amplified in MIL-STD-483. These standards apply to all services. The pertinent Type A, B, and C specifications for systems, subsystems, and software are described below. The following text is excerpted directly from MIL-STD-490.

2.2.1 Type A - System Specification

This type of specification states the technical and mission requirements for a system as an entity, allocates requirements to functional areas, and defines the interfaces between or among the functional areas. Normally, the initial version of a system specification is based on parameters developed during the concept formulation period or an exploratory preliminary design period of feasibility studies and analyses. This specification (initial version) is used to establish the general nature of the system that is to be further defined during a contract definition, development, or contract design period. The system specification is maintained current during the contract definition, development, or equivalent period, culminating in a revision that forms the future performance base for the development and production of the prime items and subsystems (configuration items), the performance of such items being allocated from the system performance requirements.

2.2.2 Type B - Development Specifications

Development specifications state the requirements for the design or engineering development of a product during the development period. Each development specification shall be in sufficient detail to describe effectively the performance characteristics that each configuration item is to achieve when a developed item is to evolve into a detail design for production. The development specification should be maintained during production when it is desired to retain a complete statement of performance requirements. Since the breakdown of a system into its elements involves items of various degrees of complexity which are subject to different engineering disciplines or specification content, it is desirable to classify development specifications by sub-types. The characteristics and some general statements regarding each sub-type are given in the following paragraphs.

2.2.2.1 Type B1 - Prime Item Development Specification

A prime item development specification is applicable to a complex item such as an aircraft, missile, launcher equipment, fire control equipment, radar set, training equipment, etc. A prime item development specification may be used as the functional baseline for a single item development program or as part of the allocated baseline where the item covered is part of a larger system development program. Normally items requiring a Type B1 specification meet the following criteria:

a) The item will be received or formally accepted by the procuring activity on a DD Form 250, sometimes subject to limitations prescribed thereon.

b) Provisioning action will be required.

c) Technical manuals or other instructional material covering operation and maintenance of the item will be required.

d) Quality conformance inspection of each item, as opposed to sampling, will be required.

2.2.2.2 Type B2 - Critical Item Development Specification

A Type B2 specification is applicable to an item which is below the level of complexity of a prime item but which is engineering critical or logistics critical.

a) An item is engineering critical where one or more of the following applies:

   1) The technical complexity warrants an individual specification.

   2) Reliability of the item significantly affects the ability of the system or prime item to perform its overall function, or safety is a consideration.

   3) The prime item cannot be adequately evaluated without separate evaluation and application suitability testing of the critical item.

2.2.2.3 Type B5 - Computer Program Development Specification

This type of specification is applicable to the development of computer programs, and shall describe in operational, functional, and mathematical language all of the requirements necessary to design and verify the required computer program in terms of performance criteria. The specification shall provide the logical, detailed descriptions of performance requirements of a computer program and the tests required to assure development of a computer program satisfactory for the intended use.

2.2.3 Type C - Product Specifications

Product specifications are applicable to any item below the system level, and may be oriented toward procurement of a product through specification of primarily function (performance) requirements or primarily fabrication (detailed design) requirements.

a) A product function specification states: 1) the complete performance requirements of the product for the intended use, and 2) necessary interface and interchangeability characteristics. It covers form, fit, and function. Complete performance requirements include all essential functional requirements under service environmental conditions or under conditions simulating the service environment. Quality assurance provisions include one or more of the following inspections: qualification evaluation, pre-production, periodic production, and quality conformance.

b) A product fabrication specification will normally be prepared when both development and production of the item are procured. In those cases where a development specification (Type B) has been prepared, specific reference to the document containing the performance requirements for the item shall be made in the product fabrication specification. These specifications shall state: 1) a detailed description of the parts and assemblies of the product, usually by prescribing compliance with a set of drawings, and 2) those performance requirements and corresponding tests and inspections necessary to assure proper fabrication, adjustment, and assembly techniques. Tests normally are limited to acceptance tests in the shop environment. Selected performance requirements in the normal shop or test area environment and verifying tests therefore may be included. Preproduction or periodic tests to be performed on a sampling basis and requiring service, or other, environment may be prepared as Part II of a two-part specification when the procuring activity desires close relationship between the performance and fabrication requirements.

2.2.3.1 Type C1 - Prime Item Product Specifications

Prime item product specifications are applicable to items meeting the criteria for prime item development specifications (Type B1). They may be prepared as function or fabrication specifications as determined by the procurement conditions.

2.2.3.1.1 Type C1a - Prime Item Product Function Specification

A Type C1a specification is applicable to the procurement of prime items when a "form, fit and function" description is acceptable. Normally, this type of specification would be prepared only when a single procurement is anticipated, and training and logistic considerations are unimportant.

2.2.3.1.2 Type C1b - Prime Item Product Fabrication Specification

Type C1b specifications are normally prepared for procurement of prime items when: a detailed design disclosure package needs to be made available; it is desired to control the interchangeability of lower level components and parts; and service maintenance and training are significant factors.

2.2.3.2 Type C2 - Critical Item Product Specifications

Type C2 specifications are applicable to engineering or logistic critical items and may be prepared as function or fabrication specifications.

2.2.3.2.1 Type C2a - Critical Item Product Function Specification

Type C2a specification is applicable to a critical item where the item performance characteristics are of greater concern that part interchangeability or control over the details of design, and a "form, fit and function" description is adequate.

2.2.3.2.2 Type C2b - Critical Item Product Fabrication Specification

A Type C2b specification is applicable to a critical item when a detailed design disclosure needs to be made available or where it is considered that adequate performance can be achieved by adherence to a set of detail drawings and required processes.

2.2.3.3 Type C5 - Computer Program Product Specification

A Type C5 specification is applicable to the production of computer programs and specifies their implementing media, i.e., punch tape, magnetic tape, disc, drum, etc. It does not cover the detailed requirements for material or manufacture of the implementing medium. When two-part specifications are used, Type B5 shall form Part I and Type C5 shall form Part II. Specifications of this type shall provide a translation of the performance requirements into programming terminology and quality assurance procedures necessary to assure production of a satisfactory program.

3.0 WHAT IS A REQUIREMENT?

The totality of interactions between any real deployed system and the rest of the universe is unknown, in large part unmeasurable, and, thus, unknowable. What we conceive as the "system" is, in all cases, an abstraction from reality that retains a limited set of measurable parameters meaningful in fulfilling the system objectives.

Within the range of these system parameters, and others with measurable effects on the system or its environment, there are "desirable" and "undesirable" values. The purpose of stating requirements is to define the boundary between desirable and undesirable, and especially that between acceptable and unacceptable. A requirement is simply a statement of something needed to ensure that the system meets an operational objective at the proper time.

This does not mean that real operational needs will always be within current technological capabilities at acceptable cost. Practicality demands that only those needs that are technically and economically feasible be addressed at a given time. Thus, a requirements engineering discipline must provide mechanisms to avoid infeasible combinations of requirements early, before ill-fated developments are undertaken.

Beyond the concept of a requirement as "something needed", there are different types of requirements, different notions of what separates requirements from design, and certain properties of good requirements that make things easier for the development team. The following sections explore some of these issues.

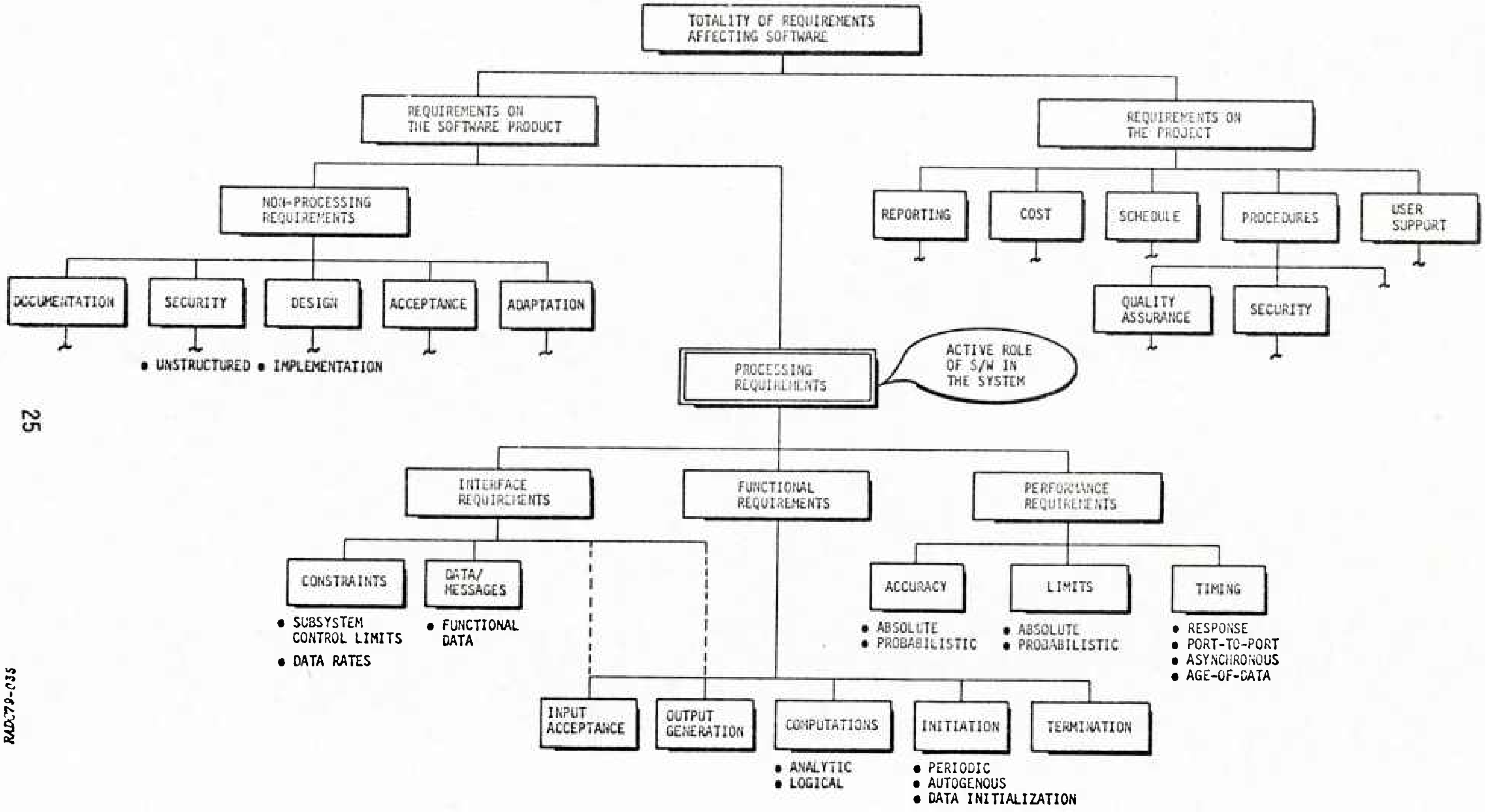
3.1 A HIERARCHY OF SOFTWARE REQUIREMENTS

Figure 3-1 decomposes the totality of requirements affecting software into a hierarchy of categories of requirement types. Each of the three major categories is discussed in the following paragraphs.

3.1.1 Processing Requirements

Processing requirements are those that define the active role of the software in the weapon system and those features of the software that affect the proper operation of other subsystems. Processing requirements can be further decomposed into three categories:

- Functional Requirements -- define the conditions for initiation and termination of software elements and define "what the software is to do" during its period of operation.
- Performance Requirements -- define "how well" the software is to perform its functions, principally in terms of computational accuracy and response times to given stimuli.
- Interface Requirements -- define the agreed-upon assumptions that the developers of one subsystem can make about the operation of other subsystems, and the physical or information links between subsystems.



RADC79-035

Figure 3-1 Types of Software Requirements

At early stages in the development of a system it is preferable to state the requirements in a computer-independent form (i.e., not presuming a particular processor or operating system). This provides a baseline to accommodate later changes in host processors and encourages attention to portable software. As system design proceeds, specific machine-dependent requirements may be levied, but they should be identified as such for traceability purposes.

In this report, we are primarily concerned with the problems of processing requirements. These are concerned predominantly with technical considerations. The non-processing requirements and project requirements are driven primarily by management considerations.

3.1.2 Non-Processing Requirements

Non-processing requirements are those that deal with the software as a manufactured component rather than an action-producing entity. Included in this category are requirements on the form and content of supporting documentation, constraints on programming languages, structural design restrictions (e.g., structured programming), requirements on the physical medium for software delivery (e.g., punched cards, tapes), and restrictions on routine length.

The non-processing requirements deal with things that can usually be verified by inspection of physical items, including program listings and support documentation. They generally affect the methods of production only when the consequences of those methods are visible directly in the deliverable software, or its representation. Most non-processing requirements evolve from practices that are proven or believed to produce higher quality software.

3.1.3 Requirements on the Project

Requirements on the project are those that constrain cost and schedule, and promote management visibility and orderly progress. Examples are requirements for design reviews, progress reporting, implementation plans, quality assurance plans, and configuration management plans. These developmental requirements affect the software indirectly by promoting an orderly and manageable development environment.

Generally, requirements on the project are established through contractual provisions independent of the specifications on the product. Although this study is not concerned with generating these types of requirements, it should be pointed out that a disciplined requirements engineering methodology for product requirements makes it easier to comply with project requirements and can provide auxiliary information to demonstrate compliance.

3.2 REQUIREMENTS ISSUES

In this section, we will discuss the relationship between requirements and design and explore some other characterizations of requirement types.

3.2.1 Requirements and Design Freedom

At any level of system development, the requirements at that level should state the needs of the system without inappropriate assumptions or constraints on the solution. In this way, the designer is left with the maximum latitude to find an effective solution.

Design freedom is not an exercise in technical democracy; it must be justified from the overall systems development point-of-view. It cannot be assumed to be obviously good just because it sounds good (i.e., who can be against freedom?). Its real justification must stem from the concept that some design decisions are more appropriately made at a lower level upon consideration of:

- Information available
- Technical skills required to properly make the decisions
- Cost associated with delaying decisions
- Cost associated with making wrong decisions
- Interdependence of decisions with other decisions at the same level
- Lead time, resources, schedule impact of implementing the decisions.

In particular, a decision made as soon as possible has many benefits <u>if</u> it is made correctly. Furthermore, in many cases there are many workable ("correct") approaches and the quest for an optimum is not cost-effective. Therefore, feasibility must be considered and design decisions made at all levels, else the process may proceed down costly, impractical paths. The requirements development process should, therefore, provide data to support a growing confidence that the system is feasible, and consider potential feasibility problems when making design decisions.

For instance, the following are considered to be examples of process design decisions:

- Algorithm approach (e.g., decoupled vs. fully coupled Kalman filter for tracking)
- Software packaging (e.g., data base organization, algorithm boundaries)
- Computer scheduling approach (e.g., specific interrupt priority scheme).

The following are not process design decisions, and are to be specified in the software requirements:

- Paths of processing steps to be applied to DP stimulus data
- Data to be saved and output (functional description)
- Accuracies and time responses.

Certain requirements on lower levels of development will, of course, automatically follow from design decisions at a higher level. However, constraints that are not implied by these decisions are to be avoided. For instance, if the operation of the weapon system demands that certain information be available within a given response time, it is appropriate for the system engineer to specify the information items and the response time. It is not appropriate for him to specify the structure of the data base. As long as the information is available at the right time, he can be indifferent to the organization of the data base.

3.2.2 Requirements by Choice and Inescapable Requirements

"Requirements by choice" are, in effect, design decisions already made. Inescapable requirements are those that automatically follow from design decisions or from uncontrollable threat and environment characteristics.

All system developments evolve from a single "requirement by choice" -- that choice being to build a system. Immediately, a large set of inescapable requirements are imposed by that choice. For instance, it is known that an opponent has just developed, and intends to deploy, a tactical fighter-bomber, X, with maximum attack speed, V knots, and weaponry including air-to-ground missiles with 50 NM range. Aircraft X can be used in attacks against a class of point targets, Y, (and presumably can attack from any azimuth). The choice here is whether or not to defend Y against X. If the answer is yes, the first requirement on the system is "defend Y against X".

From this "requirement by choice", a set of inescapable requirements immediately follows, defined by the properties of X. One of these is that no X can be allowed to penetrate within 50 NM from a defended target (the range of X's weaponry). A second requirement is, given a defense system reaction time, t_R, between first detection and intercept, the range of the attacker from the target, R_D, at which detection is assumed must obey the relationship

$$R_D \geq Vt_R + 50 \qquad \text{(in nautical miles).}$$

We still have freedom to vary t_R and R_D, but the relationship that must be maintained between them is an inescapable requirement.

The same interaction between design choices and requirements holds at each level of system development. Let the requirements for a Level N system component be defined from functional analysis and design decisions at Level (N-1) as shown in Figure 3-2. The designers at Level N receive these requirements and evaluate them. Through decomposition of the functional and performance requirements on the component, alternative designs are proposed to satisfy the requirements. Each alternative design is described in terms of subcomponents that perform subfunctions of the functions allocated to the component. After evaluation of the alternatives, a "best" design is selected for development. Inherent in this design are several design decisions, and a definition of subcomponents. For each subcomponent, a set of requirements dictated by the design at Level N is prepared. These requirements are input to the subcomponent designers at Level (N+1).

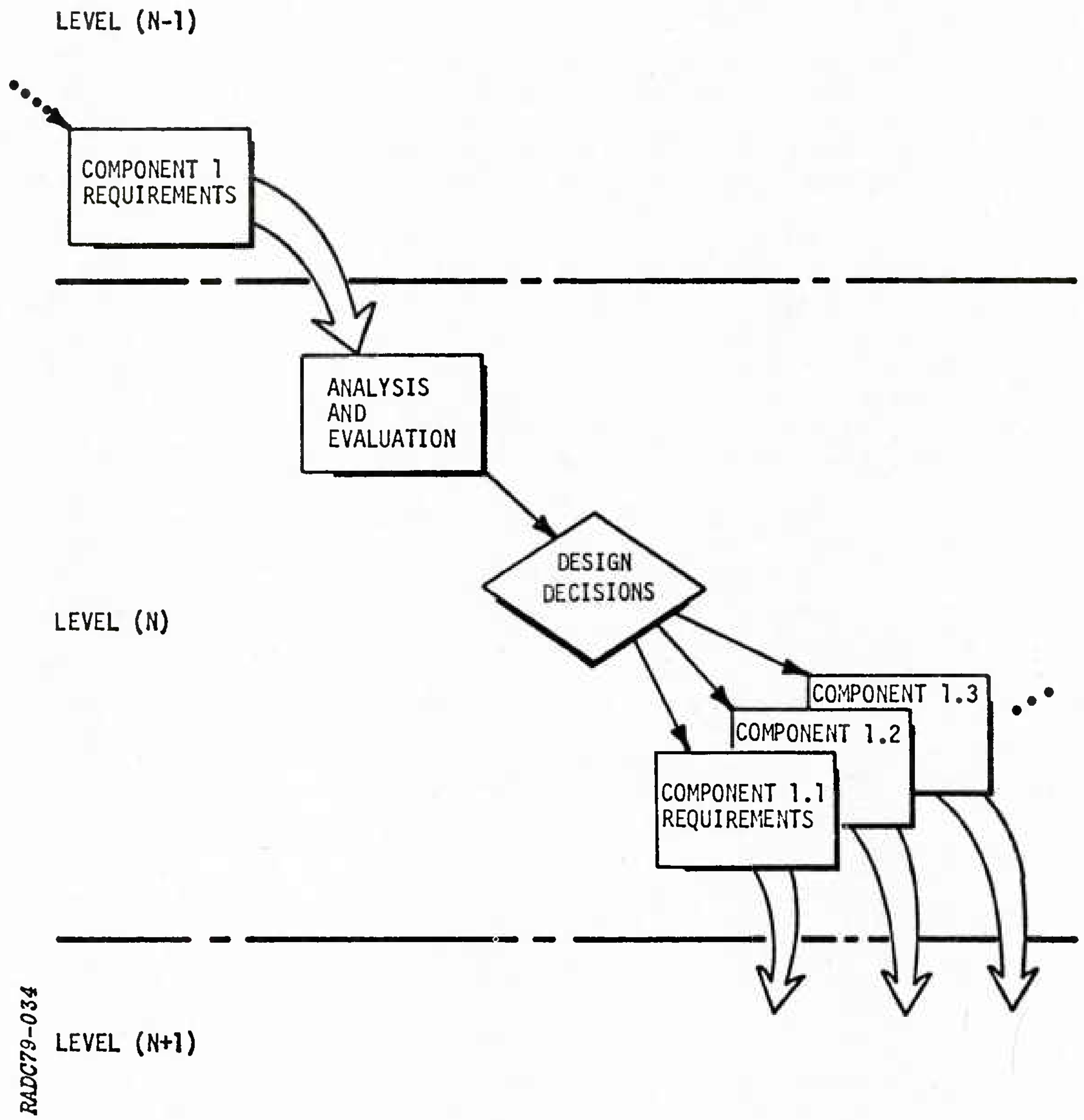


Figure 3-2 Interaction of Requirements and Design

The legitimate requirements to be passed to the Level (N+1) designers are the necessary "requirements by choice" at Level N, the inescapable requirements that follow, and the inescapable requirements from higher levels. If many alternative design choices are possible at Level (N+1) and they are all satisfactory in terms of the design at Level N, then the Level (N+1) designer should make the choice, not the Level N designer, unless there are explicit, defensible reasons for doing otherwise.

3.2.3 Problem-Oriented Versus Solution-Oriented Requirements

A requirement is problem-oriented (i.e., "top-down") if it states a need in terms of a higher level context or mission, and levies that requirement upon the entire unit that is tasked with satisfying that requirement. If we are identifying requirements for a data processing subsystem (DPSS), the requirement should be stated as, "The DPSS shall ...", and not, "Routine X of Program Y shall ...".

A requirement is solution-oriented (i.e., "bottom-up") if the need is stated indirectly, in terms of specific components of the unit that is tasked with the requirement, or in terms of how the need is to be fulfilled. The abstract statement of the linear filtering problem with all attendant assumptions explicitly stated is a problem-oriented requirement for a "tracking filter". The description of a seven state-variable Kalman filter that "shall be implemented" is a solution-oriented requirement for a tracking filter.

The danger of solution-oriented requirements is that they obscure the real problem at a given level. The presumed solution may not be the best or most practical one, and in some cases may be completely inappropriate. At best, the solution-oriented requirement complicates the traceability of the solution to the real problem, and at worst delays the detection of erroneous assumptions. It may also complicate the satisfaction of other requirements where a strong relationship between the requirements is not obvious.

3.2.4 Soft and Firm Boundary Requirements

The designation of the hardness of constraints, restrictions, and boundaries of performance are of two classes:

(i) Statistical or probabilitic (soft), and

(ii) Absolute or deterministic (firm).

Example:

3.2 The estimate of range shall not differ from the actual value (computed from precision trajectory generation as discussed in ...) by:

Type (i) - A normally distributed distance error with zero mean and standard deviation of 1500 feet.

Type (ii) - 1000 feet in absolute value.

Statistical or probabilistic requirements, when well-defined, are fundamentally complex. They are usually couched in terms of stochastic processes with associated probability distributions seldom defined. However, the performance boundaries erected by these requirements provide an excellent environment for design freedom. Due to the boundary elasticity, so-called off-nominal cases are easily covered. This type of requirement is easily levied when ideas and details are fuzzy. Even when not well-defined, those of type (i) still communicate important gross information.

Type (ii) requirements are fundamentally simple because they are in terms of elementary inequalities. They are easily stated with high precision. Mathematical analysis and proofs of type (ii) propositions are more easily accomplished. It is clear that improper allocation of these requirements can yield overly-restrictive design constraints. Type (ii) requirements are not usually levied when ideas are fuzzy, but only when details and underlying structures are readily seen.

Type (i) requirements are more numerous in early stages of system development, whereas type (ii) requirements occur more frequently in the later stages of development (see Figure 3-3). It should be noted that some types of requirements (e.g., reliability, availability) will always be type (i) because of the probabilistic definition of the parameters.

3.2.5 Long and Short Time-Span Requirements

Two categories of requirements applying to periods of system action are:

(a) End item or whole process, and

(b) Intermediate item or segmented process.

These are illustrated in Figure 3-4.

Example: Two type (a) requirements are:

3.2.a The system shall obtain at least a 40 percent defended target survivability.

3.2.b The leakage due to the DP shall be no more than 10 percent of the system leakage.

The type (b) requirements are:

3.2.c After three valid track returns have been received, the estimate of object state shall be known sufficiently well so that ....

3.2.d Radar power usage shall satisfy the short term restrictions ....

The type (a) requirements are more strategic in nature. They are levied in the early stages of the development process because they deal more directly with the system goals and objectives. As a result, these requirements affect

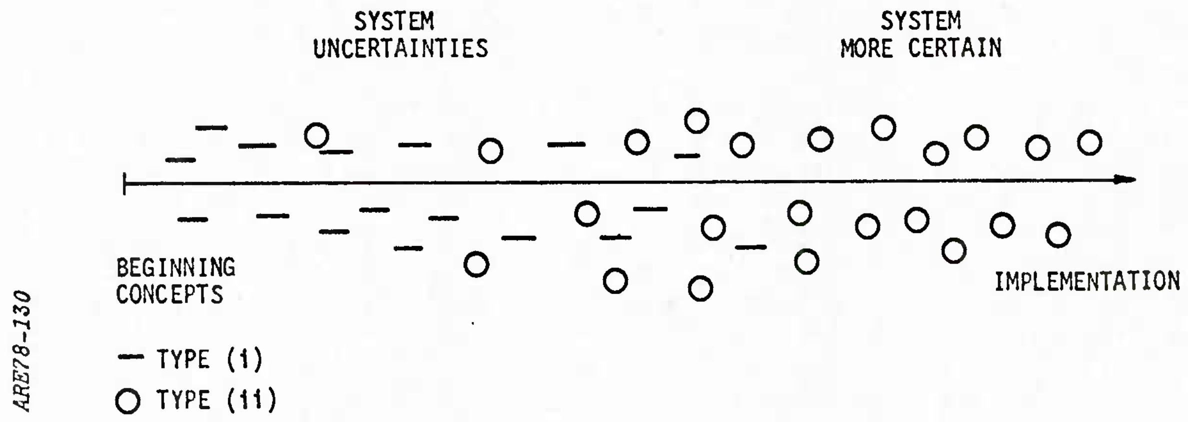


Figure 3-3 Requirements Densities in Systems Development

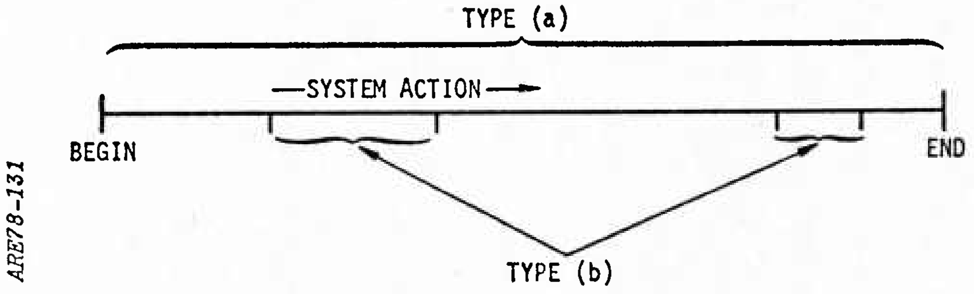


Figure 3-4 Relative Time-Spans of Type (a) and Type (b) Requirements

many parts of the system and apply to the time interval elapsed by the complete system engagement. Generally, the terminology in the type (a) requirements is simple and easy to understand. The impact of the type (a) requirements on design freedom is at two extremes. On one hand, if the state-of-the-art is not pushed, design choices are quite numerous. On the other hand, if type (a) requirements are improperly levied or the circumstances dictate very difficult goals and objectives, then most alternative designs are eliminated from consideration. It is clear that requirements of type (a) require entire system dynamics for testing.

The type (b) requirements have tactical implications. These are more often levied in the later stages of system development and apply to fewer parts of the system. The type (b) requirements apply to specified segments of the system engagement period. The terminology usually is not simple and is filled with minute system details. The segmenting of the system process interval usually admits many design decisions so that freedom is diminished by introducing (b) requirements. Testing of type (b) requirements is much more like "Unit Testing" and requires only a limited interval of system dynamics.

3.2.6 Open System Versus Closed System Requirements

An example of an open system requirement is that proposed to be levied upon the guidance software for a strategic missile:

> The software shall ensure that the warhead target miss distance in operational use upon enemy targets shall be no more than X (3σ) from the aim point in a horizontal plane.

This simple statement transfers to the guidance software all responsibility for: faulty inertial measurement units, control misalignments, propellant faults, geodetic measurement errors, reentry vehicle aerodynamic variations, and weather variations in the target area. If taken literally, a full-scale war (carefully instrumented and monitored) would be necessary to test the software for compliance with the requirement, and the test results would be inconclusive. In any case, no contractor can rationally be responsible for unspecified environmental conditions beyond his knowledge or control. The fact that "open-ended" requirements are accepted implies that no one takes them seriously or literally. In many cases this eventually results in turbulent misunderstandings between customer and contractor.

A more reasonable and objectively testable statement of the requirement would read something like this:

> Using identical stimuli and responses provided by Missile and Environment Model A, the software shall ensure that the computed target miss distance in a horizontal plane (at simulated burnout) varies by no more than X (3σ) from that computed by Model B.

The combination of Model A and Model B in this case forms a closed reference system. The combination of Model A and the software under test forms another closed system. The differences in results between Model B and the subject software are amenable to analysis. A word of warning is appropriate here.

If one wishes to levy such requirements, he must ensure that Model B is the most accurate model of the desired behavior obtainable, and be ready to prove it.

In real-time system development, the high fidelity Model B generally cannot meet the real-time performance requirements. Otherwise, it probably would have been the basis for the actual software. Although not perfect, the concept of testing in the context of a closed system model (and stating requirements in those terms) is better than stating open-ended requirements that are not subject to test, hence, meaningless.

3.3 ATTRIBUTES OF A GOOD REQUIREMENTS SPECIFICATION

Analysis of the aforementioned problems and issues leads to a better understanding of the necessary attributes of a good requirements specification. Although the requirements development process must be capable of producing specifications which address a long list of "abilities", eight attributes seem to be dominant, and are summarized below:

1) Correctness -- A specification is said to be correct if, when all of the requirements in the specification are satisfied, the product will satisfy the originating specification.

2) Modularity -- Requirements should be modular for the same reasons that the software should: a) Change is to be expected as a way of life -- if the requirements are modular, then changes are easier to analyze, invoke, and control. b) As the details get filled in and the total volume of material and work grows, division of labor becomes a must. Modularity allows a rational division of labor. Useful modularity means that each "module" of requirements be internally complete and that it fits into the entire system through very well-defined (controlled and traceable) interfaces with other "modules".

3) Completeness -- "What you see is what you get" is the rule. If a capability, feature, or performance parameter is not specified as a requirement, there is absolutely no reason to believe that it will appear in the final product. Engineers and programmers are typically honest and professional, but they are subject to schedule and budget constraints. Implementing things that are not specified is poor management on their part. Therefore, it is imperative that the requirements specification must contain everything expected of the system.

4) Explicitness -- This attribute is a corollary to completeness and testability. The requirements must be stated explicitly. The specification should not require the reader to "read between the lines", correlate two statements to obtain an implied requirement, or to otherwise apply analysis to discover what is required. Additionally, all terms used must be unambiguous. The law of perversity guarantees that if two meanings can be applied to a statement in a specification, the wrong one will be followed.

5) Testability -- It is obvious that the system should be tested for conformance to the specification to which it was built. However, it is surprisingly simple to write requirements which look very good only to discover later that there is absolutely no way to test the end product for compliance. Every requirement specified must be examined for testability. If it is found to be untestable or unverifiable, it should be changed.

6) Traceability -- In a large system where several levels of requirements and design specifications exist, modularity enhances traceability. Both upward and downward traceability must exist. Downward traceability allows one to verify that every requirement in a specification has been considered in lower level documents and allows identification of where a change in requirements affects design. It allows verification of performance against the parent requirements and allows an impact analysis to be made in the event that a detailed performance requirement cannot be met. If the requirements specification is intended to serve more than one user, special constraints are imposed in presenting the information content. If lateral traceability is imposed on the specification, and a change occurs in one part of the specification, its impact can be traced throughout the specification to maintain consistency of requirements.

7) Feasibility -- A specification is said to be feasible if there is at least one design for the product which will meet the specification. We distinguish between analytical feasibility (given the input data, there exists a sequence of algorithms which will achieve the specified performance), and real-time feasibility (there exists algorithms, a data processor, and a software design which will satisfy both the analytic and timing requirements). Obviously, real-time feasibility cannot be insured without performing a real-time design for at least one data processor.

8) Design Freedom -- The software designer must be told what degrees of freedom are available to meet the constraints. This includes:

- Design Independence -- A requirements specification should state "what" is to be done, when, and how well, but not "how" it is to be accomplished for real-time software. A good requirements specification should allow a maximum of freedom in the subsequent design and implementation phases. This does not imply that design decisions are not made in the development of the requirements -- they are. But, no design decision should be arbitrarily made which unnecessarily restricts the design freedom of the next phase of the development cycle. This means that the techniques, formats, and means of presenting the requirements must not inadvertently introduce unintentional design choices.

- Sufficiency -- A requirements specification must not only state everything which is required of the system, but must also supply information needed by the designer to do his job. Information known to the requirements engineer should not be left for the designer to reinvent or rediscover. Information which would be useful to the designer, and does not logically fit into the specification itself, can be included in a for-information-only appendix or in separate documents.

4.0 FRONT-END PROBLEMS

Several recent studies have examined the problems of the requirements generation process. In nearly every software project which fails to meet performance and cost goals, requirements inadequacies play a major and expensive role in project failure. In too many projects, requirements have been late, incomplete, inconsistent, ambiguous, overconstraining, or incorrect. Analysis of problem reports from various projects indicates that incorrectness is the dominant requirements problem. A consistent one-third of such reports deals with incorrect or infeasible requirements. Incompleteness is the second most serious problem, resulting in 21 to 29 percent of the problems reported. Ambiguity causes 25 to 30 percent of early problem reports, but as a project matures the percentage decreases to less than 10 percent. Inconsistency, however, causes a stable 9 to 10 percent of reports at all stages of a project [2].

It would be convenient, but superficial, to say that these errors originated with the persons who actually wrote the software requirements. While many errors do emerge from this source, there are many earlier errors in system analysis and system engineering not found until late in a project. These errors are critical, sometimes resulting in cancellation of projects.

The fact that requirements errors occur is merely symptomatic of underlying factors and issues encountered in modern weapons system development. We believe that the most significant factors are:

- Complexity
- Communication
- Validation
- Traceability
- Change response.

4.1 COMPLEXITY

Modern military systems are complex technology products involving many scientific disciplines and specialized engineering expertise. This will always be so because military systems operate in an environment where the enemy is constantly trying to complicate the problem and the mission of the system.

The inherent complexity of any system arises from several sources:

- Total number of components
- Intricacy of interconnection
- Number of different types of components
- Strongly coupled interactions between components

- Variety of system responses
- Number of operational mission objectives.

The complexity of a given type of system tends to increase faster than the capacity of the system, indicating a general "diseconomy of scale" with respect to the number of system components. For instance, in a telephone switching network connecting N parties to N trunks without blocking, the number of switches increases, at best, proportionally to N ln N [3].

The inherent complexity of the system is a primary factor of the system operational cost and its components, such as training and maintenance. It is also reflected in increased development complexity. The development complexity of a system is indicated by several factors:

- Number of contractors
- Total number of people on the project
- Number of product versions
- Number of requirements
- Number of interfaces with other systems
- Number of alternative solutions
- Number of distinct design decisions
- Degree of abstraction of the product
- Number of distinct technical disciplines.

To some extent the inherent complexity of the delivered system may be reduced at the expense of increased development complexity. This can be done by increased design effort to find a better solution and rigorous planning and control of the design process. In military systems, the increased development complexity is often needed to predict the performance of elements that cannot be tested under operational loads in the true environment.

Approaches to reducing both the inherent and development complexity of large systems are well advanced in the area of hardware engineering. Less progress has been made in the area of software engineering because the product is an abstract entity, not subject to physical measurement and inspection.

Several means for reducing complexity have been used in system development. These include:

- Abstraction
- Decomposition/Allocation
- Refinement/Partitioning
- Analogy/Simulation
- Hierarchical Organization
- Specialization.

The first three concepts are means for reducing multi-dimensional problems (and multi-dimensional solutions) to simpler, more easily comprehended parts. While these methods describe wholes in terms of parts and relationships between parts, they are static representations that give little insight into the dynamic interactions of the system. By use of analogy and simulation, we make decisions about the dynamic behavior of a system by evaluating the dynamic behavior of system models.

Hierarchical organization is widely used, both to define management responsibility and to conceptually structure a system. This is because the human brain is able to perceive and manipulate only about five to nine distinct things at a given moment. Hierarchical structure is a device that allows the human brain to span a larger set of distinct things in a systemmatic manner. Similarly, specialization is a means to allow groups of human beings to perform more tasks or consider more distinct ideas than otherwise possible.

The complexity of a system has an immediate impact on the problems of requirements definition, analysis, and maintenance. We can expect that, the more complex the system, the greater the number of requirements. Further, because relationships exist between requirements, the complexity of requirements analysis and subsequent system design grows faster than the number of requirements.

Let n be the number of distinct requirements and let p be the probability that any two requirements are inconsistent or otherwise conflict with each other. The expected number of inconsistencies is given by

$$E = \frac{n(n-1)}{2} \cdot p$$

The average number of inconsistencies per requirement is E/n. We can tabulate E and E/n as a function of p as follows:

n	E	E/n
10	45p	4.5p
100	4950p	49.5p
1000	499500p	499.5p
10000	49995000p	4999.5p

This example is conservative because conflicts between sets of individually consistent requirements are not considered, and because more requirements implies more people generating requirements, which increases the probability of inconsistency. However, the example indicates that the amount of work necessary to remove requirements inconsistencies in large systems can be substantial, even if the probability of inconsistency is very small.

4.2 COMMUNICATION

The specialized division of knowledge and labor forced by system complexity leads to the communication problem. The complete documentation of all requirements for a system and its components requires a multi-level hierarchy of specifications with many separate specifications and interface documents at each level. Many people with different backgrounds and specialties contribute to this effort. Hence, identical words in different parts of a specification may have different intended meanings, and may be interpreted in yet another way by the reader. The effects of interpretation and transformation, propagated through a specification hierarchy, lead to erroneous mutation of requirements which is later detected as ambiguity, inconsistency, and incorrectness.

The communication problem can be separated into three subdivisions:

- Horizontal communication
- Vertical communication
- Self-communication.

Horizontal communication is between parties operating at the same level of system development, either within a technical discipline or across technical disciplines. Communications between process designers, or between data processing subsystem engineers and radar subsystem engineers are examples.

Vertical communication is between parties operating at different levels of system development. Communications between system engineers and process designers or between software designers and programmers are examples. The party at the higher level generally has a broader but more shallow view of the system than the party at the lower level. The system engineer will know what effect a particular tracking algorithm has on the outcome of an engagement, but may be unaware of and unconcerned about how that algorithm is implemented on a particular computer. The programmer, on the other hand, will know the most efficient coding of the algorithm in assembly language for a particular machine, but may have no idea of the role of the algorithm on the total system.

Self-communication is between a party and himself at a later time. This is the process of memory and recall, perhaps augmented by external recording of information.

In each of these types of communication, both parties must share a common definition of terms, relationships, and concepts. This is very difficult in advanced technology work where implied relationships are multi-dimensional and abstract concepts are poorly understood, and is made even more difficult by the need for specialization which emphasizes the difference in knowledge and viewpoint between individuals.

A second difficulty in communication is the limitations inherent in the communication medium. Natural language, in addition to its semantic ambiguity, is presented in a one-dimensional sequence (i.e., relationships between n parts are described one at a time). Diagrams and pictures capture two-dimensional relationships and three-dimensional relationships by projection. Three-dimensional models and holographic projections capture three-dimensional relationships (and four-dimensional projections), but are generally impractical to reproduce and distribute in large quantity. Fifth-dimensional and higher relationships can be represented only abstractly in tables, mathematical equations, and lately, in computer data bases.

Effective communication of data processing requirements is particularly difficult because we are dealing with abstract entities: information and actions on information. Our abilities to visualize the dynamic behavior of software are severely limited. The limited success in specifying software requirements to date is probably due more to the assumptions associated with single sequential data processors (one program in execution at any instant) than to advances in requirements technology. As multiprocessors and distributed processing are exploited, we are becoming aware that we have great difficulty in representing and describing concurrent behavior, and that even basic dynamic concepts such as "process" are ill-defined and poorly understood.

Within small project groups, interpersonal communication can be reasonably effective without impairing productivity. As a project grows in size, communication becomes indirect, the reliability of information exchange decreases, and a significant fraction of the project staff is involved solely with documentation and liaison functions. This has a direct impact on the cost of a project and the feasible minimum-time schedule.

Project productivity is the amount of useful work produced by a project, divided by the elapsed time required to produce it (task output/time). Individual productivity is the average rate of output per individual (project productivity/manpower). The general trends of project productivity and project cost as functions of the manpower applied to a project are shown in Figure 4-1.

As manpower is added to a project, productivity improves rapidly at first. As the group becomes larger, pressure to produce grows and a synergism of effort develops. At some point, however, the project becomes so large that coordination of effort starts to become a problem. A peak in individual productivity is reached. This corresponds to the minimum of project cost.

If more manpower is added, an increasing fraction of it will be devoted to coordinating the activities of others. Individual productivity declines, slowly at first. The additional work output of the added manpower stays ahead of the loss in individual productivity. Finally, however, we reach a point where individual productivity losses start to exceed the work produced by additional manpower. This is at the peak project productivity, which corresponds to the minimum time in which the project can be done.

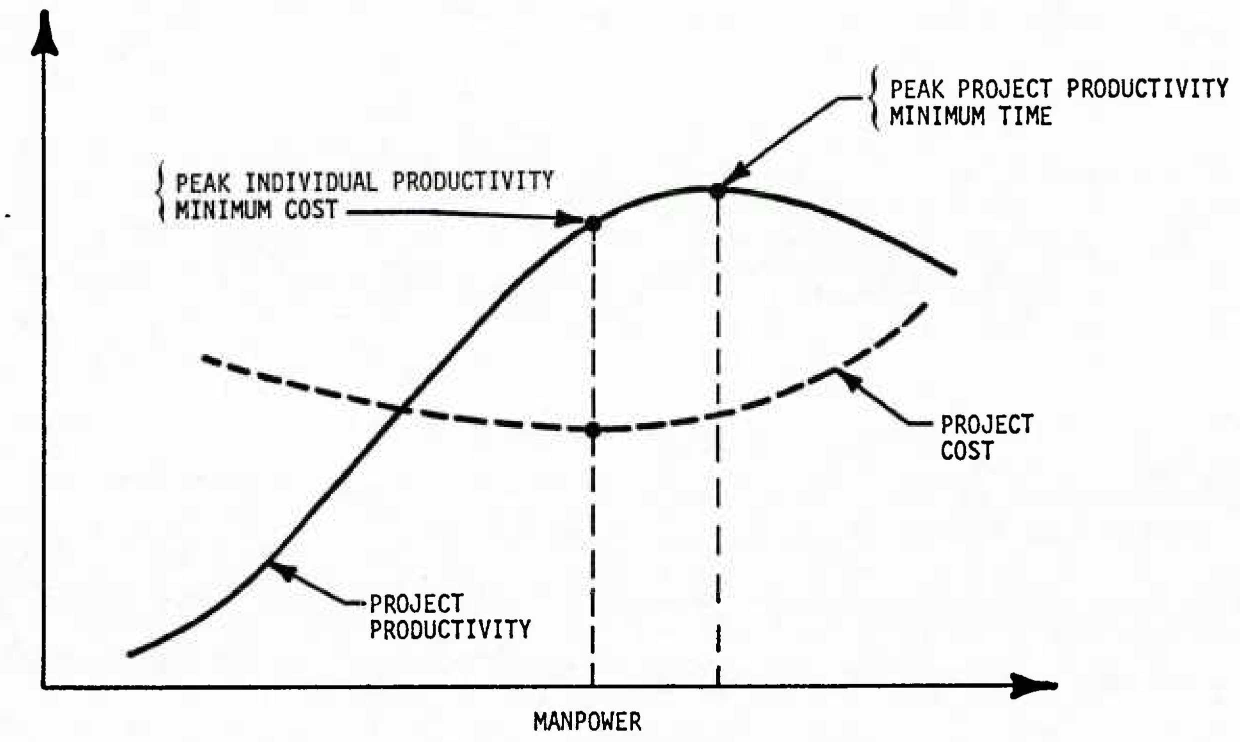


Figure 4-1 Productivity and Cost Versus Manpower

After this point, addition of manpower will lead to a breakdown of coordination and saturation of supervision. Individual productivity drops off rapidly and the project actually takes longer to complete. Since more men are working for a longer period, cost rises rapidly. Fewer men could have done the project in the same time, at less cost.

Two approaches can be pursued to reduce the communication problem and increase project productivity: 1) reduce the need for communication, and 2) improve the effectiveness of communication.

In the software development field the so-called "Parnas Principle" [4,5] is a design rule that exemplifies the first approach. Parnas defines "modules" as things that have to be designed and developed together -- in effect, a natural work assignment. Parnas identifies the connections between modules as the assumptions modules make about each other. The criterion for modularity proposed by Parnas is that each module should implement a design decision and isolate and hide that decision from other modules (i.e., every module hides a secret). In this way, the inter-module interfaces must remain constant even if the internal design decisions change. Under this principle, different work groups need only agree on interface assumptions and do not need to exchange information on details of internal design decisions. Effective use of the principle demands that the "problem structure" (i.e., the requirements) be defined in a structured, analyzable form, and that work units be assigned according to that structure.

The second approach to reducing communications problems is typified by requirements statement languages (e.g., RSL, URL) and program design languages (e.g., PDL). Each of these languages provides an English-like, yet structurally constrained, form of expression that is computer-analyzable to some degree. The intent of these languages is to reduce ambiguity, ensure consistency, and minimize the chances of incompleteness.

4.3 VALIDATION

Many requirements problems would be detected before they caused significant harm if requirements were effectively validated at each stage of system development. However, requirements expressed in free-form English text are difficult to validate with any objective degree of confidence. Two approaches have been widely tried in the past: independent review and simulation. Independent review has been partially effective because the reviewers are consciously questioning and critical. But, there is no objective evidence that a review has been thorough and many discrepancies slip by unnoticed because of incorrect assumptions and communication problems.

Simulation is useful in uncovering faulty assumptions about dynamic phenomena resulting from static visualizations, but is plagued with all of the difficulties of requirements interpretation in main-line software development. There is no assurance that the simulation faithfully models the characteristics intended in the specification because it is not even subjected to the degree of testing and scrutiny demanded for deliverable software.

The question to be asked in requirements validation is "do the stated requirements conform to the problem?". The question to be asked in product validation is "does the software product conform to the requirements?". Obviously, objective product validation is not possible if the stated requirements are not testable. Yet, requirements are often stated without thought to their specific testability. Much later it is realized that a particular requirement is not testable, hence, meaningless, or has several possible meanings, each subject to a different test.

In developing the Software Requirements Engineering Methodology (SREM), TRW found a means to guarantee that stated performance requirements are testable. The techniques also provide the means to remove ambiguity about the intent of the test through identification of precise "validation points" on stimulus-response paths through the software.

In current software development practice, about 10 to 15 percent of the budget is allocated to requirements definition, while 40 to 50 percent of the budget is spent on testing. Anyone with extensive experience in software integration and test is familiar with the inordinate amount of time and effort needed to interpret requirements and the relationships between them in order to generate efficient and effective test plans. It is our hypothesis that a significant percentage of testing costs are the result of inadequate requirements definition practices, and that the investment of time and money in the requirements definition effort will be more than recovered by avoidance of the "hidden costs" of bad requirements in the testing phase. Unfortunately, proof of this hypothesis on a conclusive and scientific basis would require costly and impractical experiments on large-scale projects, with provision for independent and parallel "control experiments".

4.4 TRACEABILITY

In any large system, the original requirements can be expected to change, after operational deployment and, in today's environment, during system development. These changes result from changing missions, changing threats and technical difficulties, either at a lower level or in a different subsystem. To completely incorporate the effects of changes at any level, detailed traceability between all related elements of the system must be ensured. In the past, the effects of change were laboriously traced from document to document, manually and subjectively. The process was expensive and inefficient. Effects of changes were accounted for in one passage of a specification, but related items in other sections were often overlooked. This resulted in inconsistencies to be detected at a later date. Recent data management practices have improved the situation from one level to the next. But, comprehensive traceability, backward and forward, from initial problem assumptions to preliminary design, has not yet become common practice, even though it is technically feasible.

One of the advantages of using automated data base systems to retain and maintain requirements is that traceability relationships can be established as an integral and disciplined part of the requirements generation process, rather than as an afterthought. Once established, the structured relationships between requirements can be displayed at will (or suppressed if answers to different queries are being sought). Unlike manually generated documents, the automated data base implicitly keeps track of all references to a given element at a given development level. Between development levels (e.g., system engineering to DP engineering, or system engineering to process design) human intervention to establish traceability relationships is still required. This is because the elements and representations are different. For instance, the relationship that (Task X) implements (Requirement Y: The radar shall be commanded to track a given object at no more than 10 HZ) is a matter for humans to decide. To those who object that this mapping process is unnecessary, we reply that it must be done at some point to ensure design responsiveness, management visibility, adequate testing, and adaptability to change. Enforced traceability from the beginning reduces the risk of unresponsiveness and inflexibility at a later date when time may be critical.

4.5 CHANGE RESPONSE

Manual change control procedures result in significant delay between the initiation of a change proposal and the propagation of necessary changes to other affected parts of the system. Designers must either continue work on elements made obsolete by change, or must halt work until the change is approved, resulting in non-productive work or lost schedule time. Often, the customer wants to know the detailed impact of specific changes before he decides to formally request them. With manual procedures, impact assessment is costly and slow. Automated requirements systems, such as SREM, with designed-in traceability features, have significantly reduced change delay times and have made impact assessment into a fast, practical procedure. Extension of these techniques to the entire front-end development process will improve productivity and reduce development cost.

5.0 DEFINITION OF THE FRONT-END OF DP DEVELOPMENT

The "front-end" of a data processing development encompasses all of the analysis and engineering activity from the time that the need for a system is perceived until a preliminary design for the system is specified. Six broad generic steps are necessary to systematically proceed through the preliminary design stage. They are illustrated in Figure 5-1, and briefly outlined below.

- System Analysis -- When first perceived, most operational problems are ill-defined and not quantified. The job of the systems analyst is to precisely formulate and structure the problem, and to analyze alternative solution concepts so that decision makers can choose necessary actions. The analyses at this level are intended to identify the threat, define the system mission, estimate the performance and cost of alternative system constructs, examine the sensitivity and risk inherent in the alternatives, and compare the alternatives on a common metric.

- System Engineering -- One or more of the most promising alternatives are selected for intensive system engineering study. Threat models are quantified and the system concept is refined to include functional subsystem models, subsystem interactions and system operating logic. Major tradeoff studies are conducted, cost and performance are quantified, and a preferred system is selected. System performance is allocated among the subsystems and subsystem interfaces are established.

- Data Processing Subsystem (DPSS) Engineering -- The early DP subsystem work is in concert with and supports the system engineering tradeoff studies. When subsystems have been established for the preferred construct, the DPSS definition is expanded by subsystem engineers. The functional capabilities of the DPSS are defined and traced to system level requirements. The performance allocated to the subsystem is decomposed and allocated to the subsystem functions. These elaborated requirements are expressed in terms of system level parameters, such as "threat leakage". The subsystem interfaces are refined and the system operating rules are interpreted from the standpoint of the DPSS in relation to other subsystems. A major portion of the DPSS engineering work is concerned with hardware/software, tradeoffs, identification of suitable DP architectures, evaluation of candidate processors, and allocation of requirements to hardware, software, and firmware. The DPSS engineer is concerned with DP availability, reliability, maintainability, and cost, in addition to performance. Although he may defer hardware selection until after process design in some cases, he is responsible for the selection. In current practice, the hardware is usually selected before detailed software requirements engineering and process activities are done.

- Software Requirements Engineering -- The software requirements engineering step transforms the DPSS functional definition and performance requirements, based on system parameters, into a more detailed definition of requirements, expressed in data processing terms. SRE is data

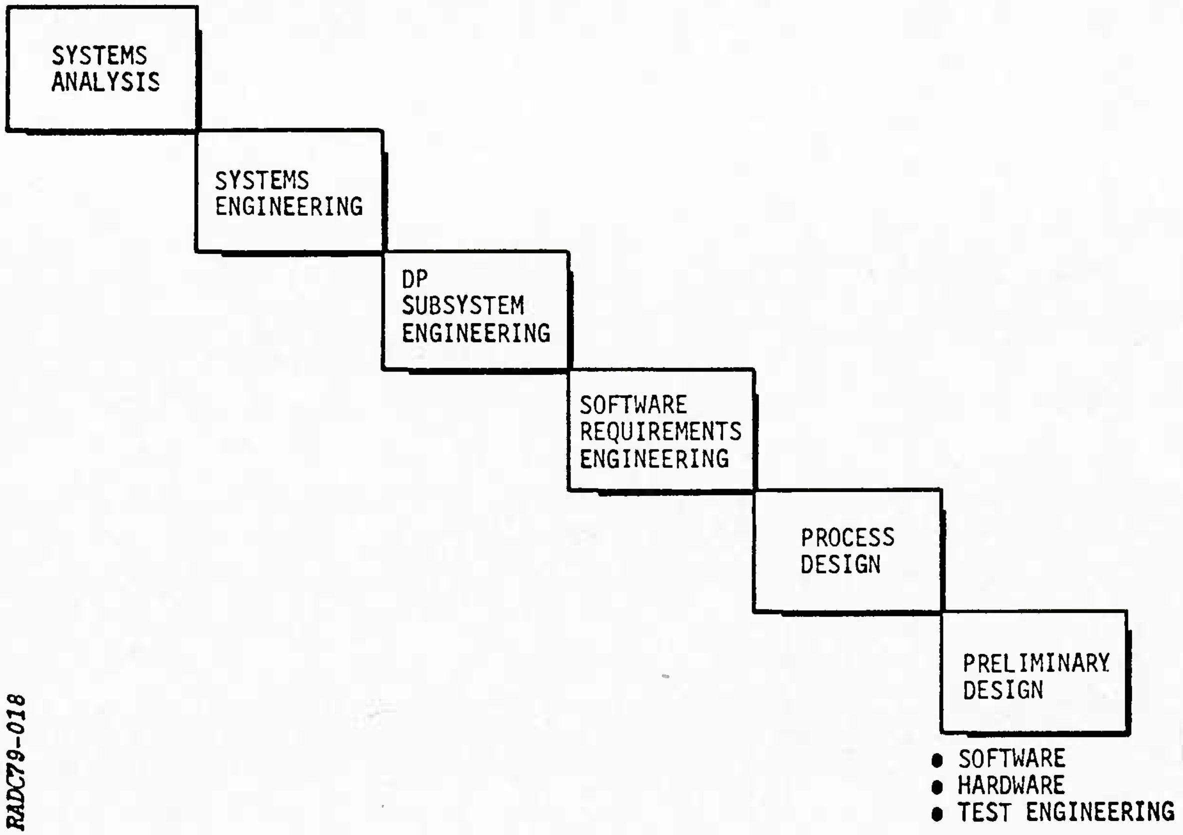


Figure 5-1 Front-End Development Phases

oriented. The contents of messages passing through input and output interfaces are defined to the data item level. Data hierarchies about entities which must be maintained by the DPSS are defined. The logical structure of the problem and system operating rules are analyzed to determine how the data are to be processed within the system. The end of this phase is reached when a logical structure defining the problem in DP terms has been validated, so that only DP software and hardware knowledge is required for the design activity. This structure includes definition of all data paths through the DPSS, precise location of measurement points for response time requirements, and models of tests which verify that the performance requirements are testable.

- <u>Process Design</u> -- The primary function of process design is to derive and develop the properties of a software/firmware/hardware combination which simultaneously satisfies all functional and performance requirements. The process designer must decompose the DPSS into a set of software tasks which are the lowest unit scheduled by the operating system. He is responsible for defining the application system, operating system, and hardware, and for ensuring that they work as a unified system. His responsibilities also encompass algorithm development and evaluation, global data base definition and maintenance, and timing/sizing budgets to the task level. Ultimately, he is responsible for integrating tasks and construction of the real-time process.

  If the project demands selection of commercially available computers, the process designer may be responsible for benchmark testing and evaluation of alternative condidates. The advent of problem-oriented distributed data processing systems expands the process designer's job. He will be called upon to devise system and component architectures for specialized problems, and to define interconnection networks and protocols. Distributed systems will require additional levels of software specifications for multiple computers.

- <u>Preliminary Design (Software Design, Hardware Design, and Test Engineering)</u> -- The expanded design activities leading to the Preliminary Design Review (PDR) vary in scope and complexity, dependent on the problem and process design. The purpose of a PDR is to verify that the design developed to that point is feasible, and is consistent with the stated requirements. Documents available for review at PDR include the Preliminary Software Design Specification, Preliminary Hardware Design Specification, Acceptance Test Plan, and Preliminary User's Manual. The preliminary design effort expands the process design to a greater level of detail, primarily, definition of task structure and timing/sizing budgets for routines. The integrity, testability and feasibility of the process design is confirmed by analysis. Algorithm selection is validated, and design approaches for critical issues are defined in detail.

Figure 5-1 implies a strict sequential ordering of the phases (i.e., completion of one phase before commencing the next). This occurs rarely in practice, and considerable overlap in time is the usual case. Another useful view of the process, the organizational hierarchy shown in Figure 5-2, clarifies the relationships between the phases.

The sponsor has responsibility for the entire system development, initiates the system analysis work to justify engineering development, and uses those results to decide whether or not to proceed with system engineering. The system engineering organization is responsible for the definition, coordination, and integration of the various subsystem engineering efforts. The DP subsystem engineering organization is responsible for the definition, coordination, and integration of the software requirements engineering, process design, and hardware engineering activities. The process designer is responsible for the overall software system architecture, and defines, coordinates, and integrates the various software preliminary design efforts. The only strict sequence is between system analysis and system engineering. The remaining phases are initiated earlier than their successors, but because of their coordination and control functions, proceed interactively with the phases at the next lower level.

In our definition of the phases we have strived to isolate the most significant activities that characterize that phase. In truth, in any given phase, many of the activities of other phases are pursued to some extent. We are seeking here to identify the principal emphasis, and show the similarities of problems between phases.

In very large projects the work of the various phases is performed by separate organizations and may involve a community of government agencies, civilian contractors, and subcontractors. For smaller projects, all of the phases may be done within one organization and may be abbreviated or prolonged according to the nature of the development (e.g., new system, upgrade, minor modification). For instance, system analysis, system engineering, and subsystem engineering are often lumped together as system engineering. Process design and preliminary design are often combined. We feel that it is important to separate the phases as much as possible for this report because future distributed systems will demand increased engineering specialization and, possibly, additional phases in the development process.

The front-end development phases defined herein differ somewhat from those typically described. Figure 5-3 correlates these phases with the usual MIL-STD-490 specification cycle and DoD Life Cycle milestones as found in most projects.

5.1 SYSTEM ANALYSIS PHASE

5.1.1 Scope

A simple, but elegant, definition of "system analysis" has been provided by J. D. Couger [6]:

"System analysis consists of collecting, organizing and evaluating facts about a system and the environment in which it operates. The

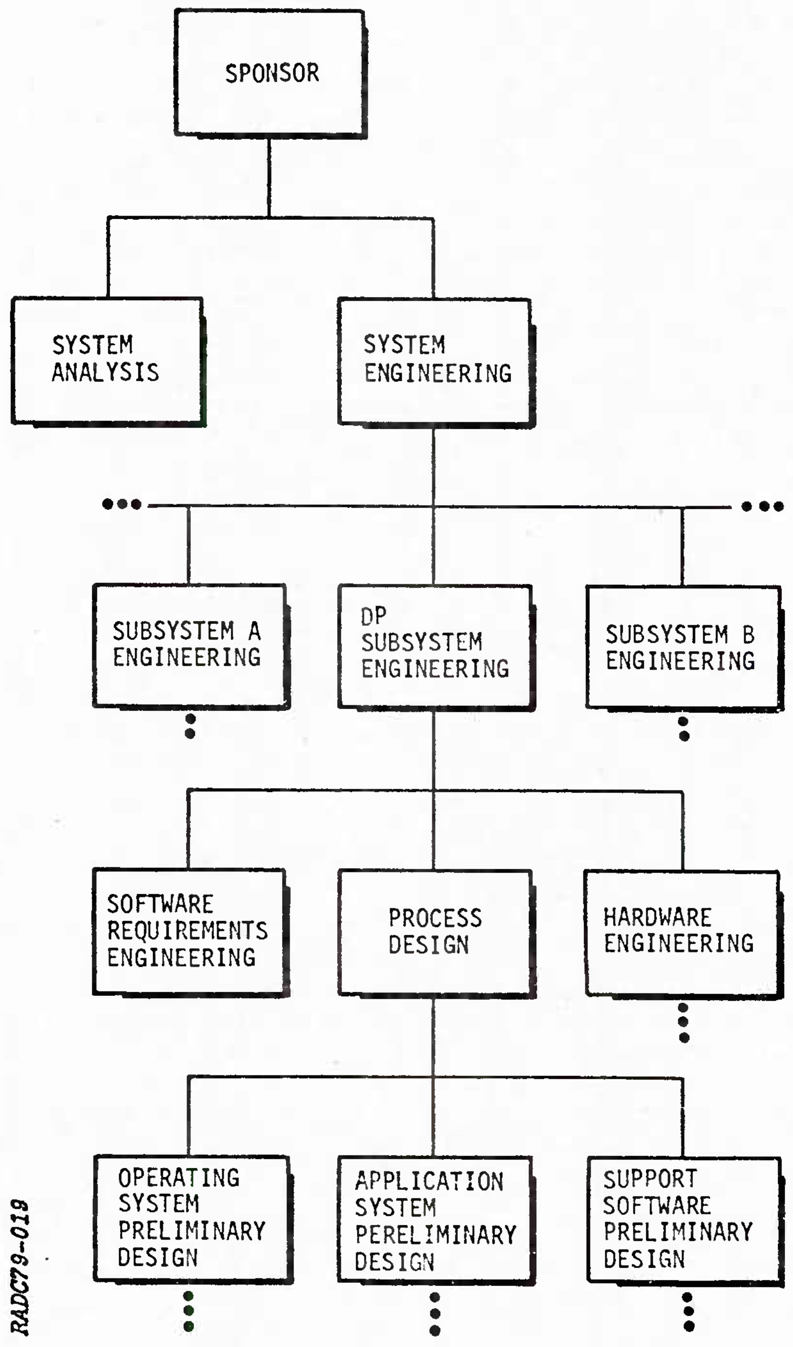
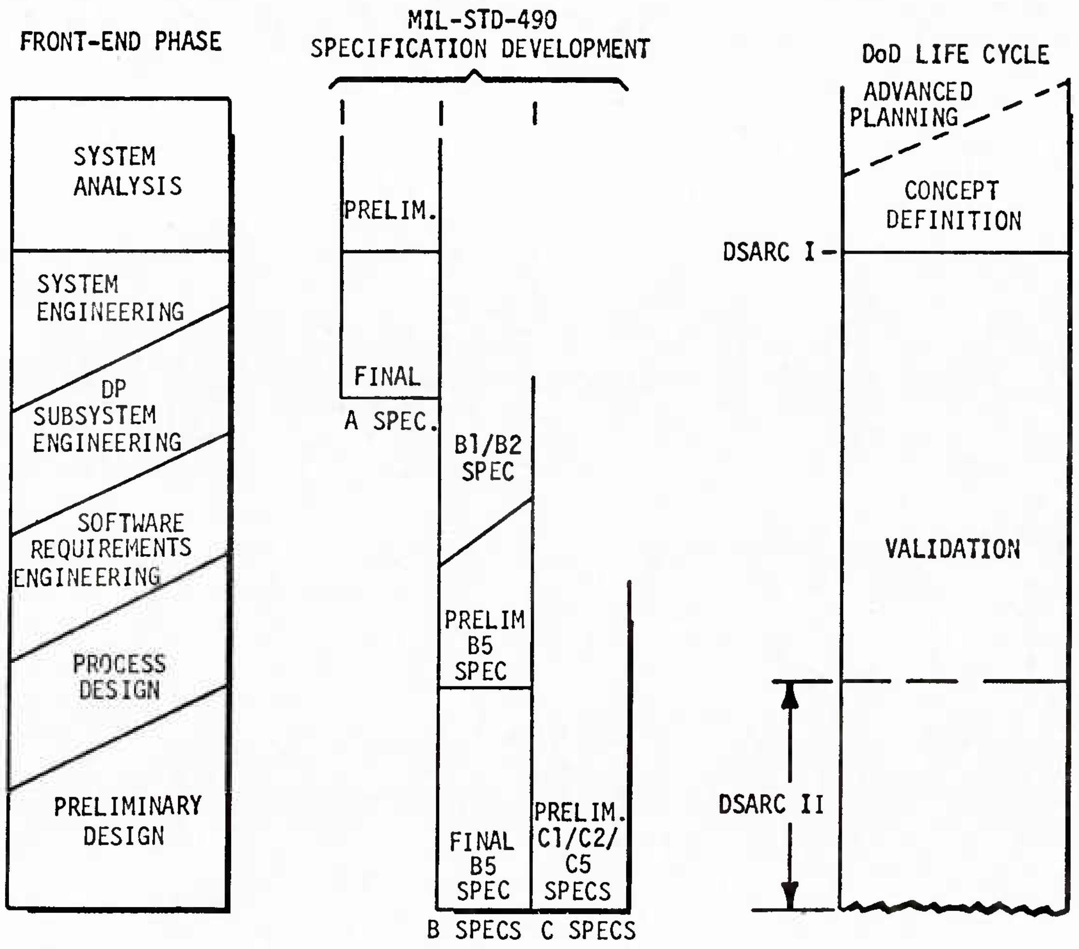


Figure 5-2 Hierarchical Organization of Phases



RADC79-023

Figure 5-3 Relationship of Front-End Phases to Other Cycles

objective of system analysis is to examine all aspects of the system -- equipment, personnel, operating conditions, and its internal and external demands -- to establish a basis for designing and implementing a better system."

In the context of the modern incremental and measured approach to weapon system procurement, system analysis can be characterized as the initial investigations to determine whether or not further expenditures toward solution of a perceived problem will be productive and with predictable results. System analysis is an on-going activity at various levels within the defense establishment. The scope and perspective of analysis varies widely, from consideration of the entire U.S. defense posture and major force mix strategies to detailed consideration of alternatives for limited-mission tactical systems. In all cases, however, major activities are:

- Verification that the problem-as-given exists
- Mission identification and definition
- Threat and environment definition
- Formulation and evaluation of alternative approaches
- Identification of feasible and superior approaches
- Assessment of sensitivities, uncertainties, and risks.

Evaluation activities must consider all aspects of the system (e.g., performance, life-cycle cost, growth, reliability, schedule, resource needs).

While system analysis activities occur all through the development process, we will characterize the "system analysis phase" for our purposes as those activities which aid a decision-maker in choosing a course of action relative to a weapon system problem, and in defining a mission package to implement that course of action. Accordingly, the objective of the system analysis phase is to define a mission package in sufficient detail so that a decision maker 1) can be satisfied that the program is feasible and cost effective, and 2) can compare the package against other programs contending for budgeted funds. For small programs the system analysis phase may be brief. On major programs it may be a multi-level effort involving both government analysts and contractors, with an extensive concept definition period prior to DSARC I.

5.1.2 Content

Rudwick [7] describes three related problems that are useful in characterizing the initial steps of the system analysis phase. These are: the "problem as given" (PAG), the "problem as understood" (PAU), and the "problem to be solved" (PTBS).

The PAG is an initial statement of the perceived problem that initiates the system analysis effort. Typically, it may consist of a vague notion that no current system is adequate to deal with a certain class of enemy threat, or that the remaining operating cost of a deployed system is too high with respect to system worth or technological alternatives. The PAG generally is not quantified, and often is incomplete and/or inconsistent. It tends to be symptomatic rather than diagnostic.

The PAU is a structured and quantified elaboration of the PAG, developed by the system analyst. It consists of those factors and relationships identified by the analyst as relevant to the original PAG. Moreover, it may be expanded beyond the PAG to include a broader class of related problems and/or solutions of which the PAG is a subset.

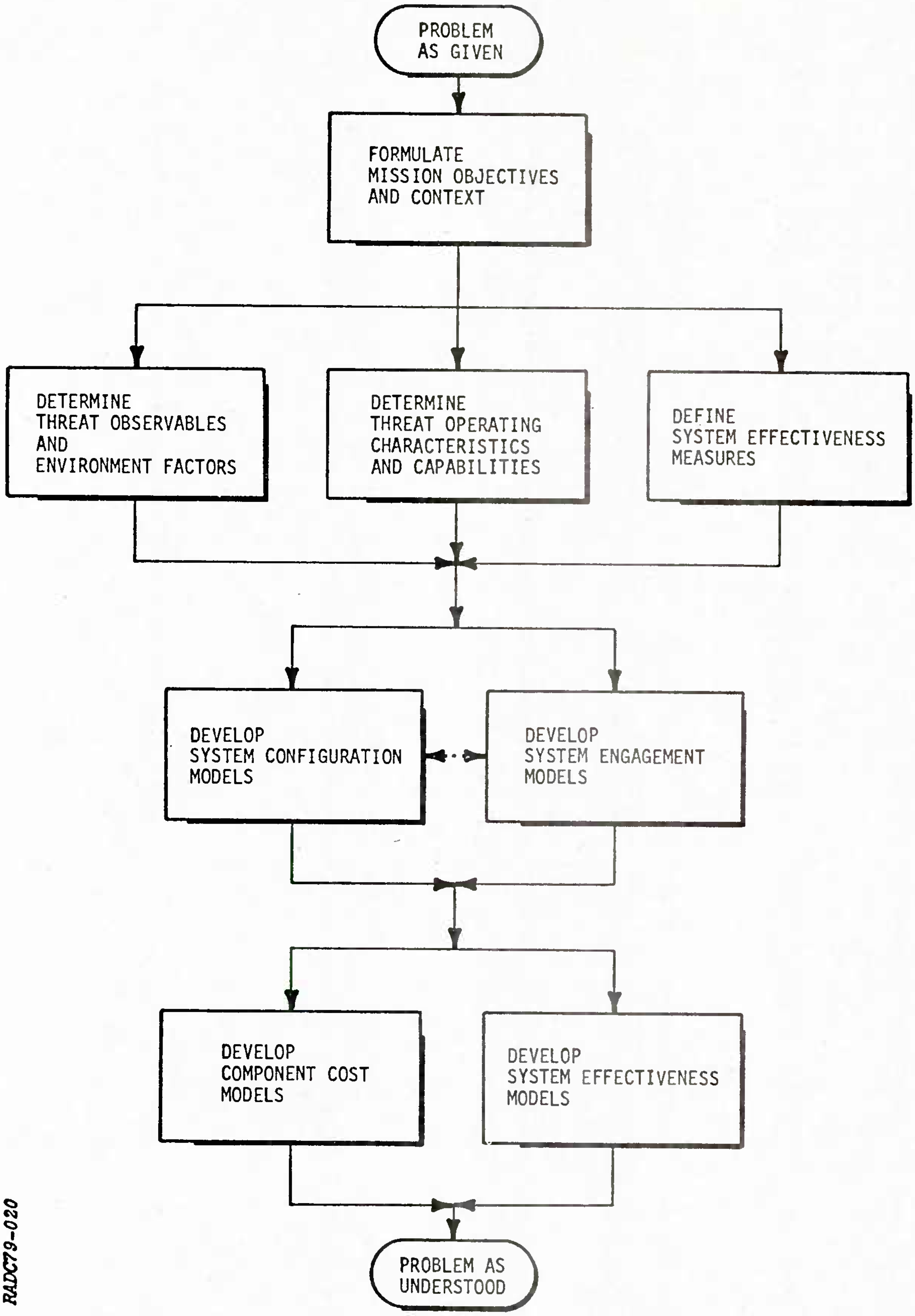
In the course of evaluation of the PAU, the decision-maker or the analyst may decide that the problem is too broad with respect to proposed solutions, that certain factors have insignificant impact, or that certain postulated threat scenarios are unlikely. The agreed-upon PTBS is a subset of the PAU which forms the basis for the system requirements. While the PAU is essentially an implementation-independent statement of the problem, and a set of candidate system alternatives, the PTBS is constrained by state-of-the-art technology projections for the system development period, and by solution cost and worth considerations.

The analyst develops the PAU from the PAG in a series of steps typified in Figure 5-4. The first step is to formulate mission objectives and the surrounding context from the information in the problem as given. This step surfaces many key questions and undefined aspects of the problem, and may lead to larger issues not previously considered.

The mission definition identifies a threat or classes of threats to be addressed by the system. Before candidate systems can be defined, we must characterize the observables and performance envelope of the threat; the weapons, sensors, and penetration aids used by the threat, and an attack sequence of events. In addition, the properties of the environment, as they affect the threat observables and performance, must be defined. Much of the information may be tentative or unknown. Many of the threat and environment characteristics can be estimated from known physical relationships and similarities to other threat systems. Another facet of the environment to be identified is composed of other systems with which the proposed system may or must interface.

The definition of effectiveness measures that capture the essence of the problem is critical to both the further elaboration of the problem and the identification and evaluation of candidate solutions. Generally, several pertinent effectiveness measures could be defined for a system, each one emphasizing certain aspects of the problem at the expense of others. The chosen measures should reflect the capability, availability, and dependability components of system effectiveness, and should also consider the utilization of system resources in the engagement environment.

When the foregoing information has been assembled and organized, the analyst has a basis from which possible system alternatives can be considered. The actual synthesis of alternatives is a highly individualistic and creative process that probably cannot be mechanized. However, the ability of the analyst to visualize alternatives can be substantially augmented by automatable methods of organizing and structuring the relevant data for his consideration.


PROBLEM AS GIVEN
FORMULATE MISSION OBJECTIVES AND CONTEXT
DETERMINE THREAT OBSERVABLES AND ENVIRONMENT FACTORS
DETERMINE THREAT OPERATING CHARACTERISTICS AND CAPABILITIES
DEFINE SYSTEM EFFECTIVENESS MEASURES
DEVELOP SYSTEM CONFIGURATION MODELS
DEVELOP SYSTEM ENGAGEMENT MODELS
DEVELOP COMPONENT COST MODELS
DEVELOP SYSTEM EFFECTIVENESS MODELS
PROBLEM AS UNDERSTOOD

RADC79-020

Figure 5-4 Problem Analysis Steps

The first alternative to be explored in any problem is the "null alternative" (i.e., what happens if no action is taken). Evaluation of this alternative often reveals that the perceived problem does not exist, or is not as bad as perceived. Sometimes it will be shown that no alternative solution is significantly better than the current system or no system at all. In any case, the null alternative is the yardstick for comparison of other alternatives.

For each suggested alternative two models are developed, usually in an iterative manner. The system configuration model is a static description of the system in terms of its components and the relationships between components. This model essentially describes "what the system is" and how it is deployed. The system engagement model is a dynamic description of how the system operates and interacts with the threat. This model describes "what the system does". While the system configuration model is described in terms of physical components and interconnections, the system engagement model is described in terms of distinct system functions and events.

Ideally, one would like to have a single system engagement model, applicable to all alternative systems. This can be done, but only at a high level of abstraction. As the system functions are progressively decomposed into sub-functions, the definition of the sub-functions becomes more dependent upon the characteristics of physical devices. Since the system analysis phase is oriented toward high level assessment of technological feasibility within given cost and schedule constraints, the analyst can often use a single engagement model for several alternatives. If he cannot, then the performance of the system must at least be described by effectiveness measures common to all models.

The system effectiveness model establishes the relationship between the system description parameters and the effectiveness measures. Typically, it is a procedure for collecting engagement simulation outputs and computing values for the effectiveness measures. This model may also determine effectiveness as a function of system resources employed.

The component cost models are parametric cost estimating relationships based upon current technology and projections into the future. Typical parameters for radars would be power, frequency, waveform types and number of units produced. For data processors, typical parameters are instruction execution rate (MIPS), word size, and memory capacity. Life-cycle cost estimates for each system alternative are generated by applying the models to the set of parameter values for each alternative.

The combination of mission definition, threat and environment definition effectiveness measures, and the set of models described above comprise the description of the problem as understood. Evaluation of PAU will isolate the "best" alternative and provide the information needed to determine the problem to be solved.

The process of evaluating each candidate system is represented in Figure 5-5. Generally, the process is one of iterative optimization because the initial estimates of system parameters and operating rules are usually sub-optimal. Representative threat and environment characteristics are combined into engagement scenarios. Fixed and variable system parameters from the

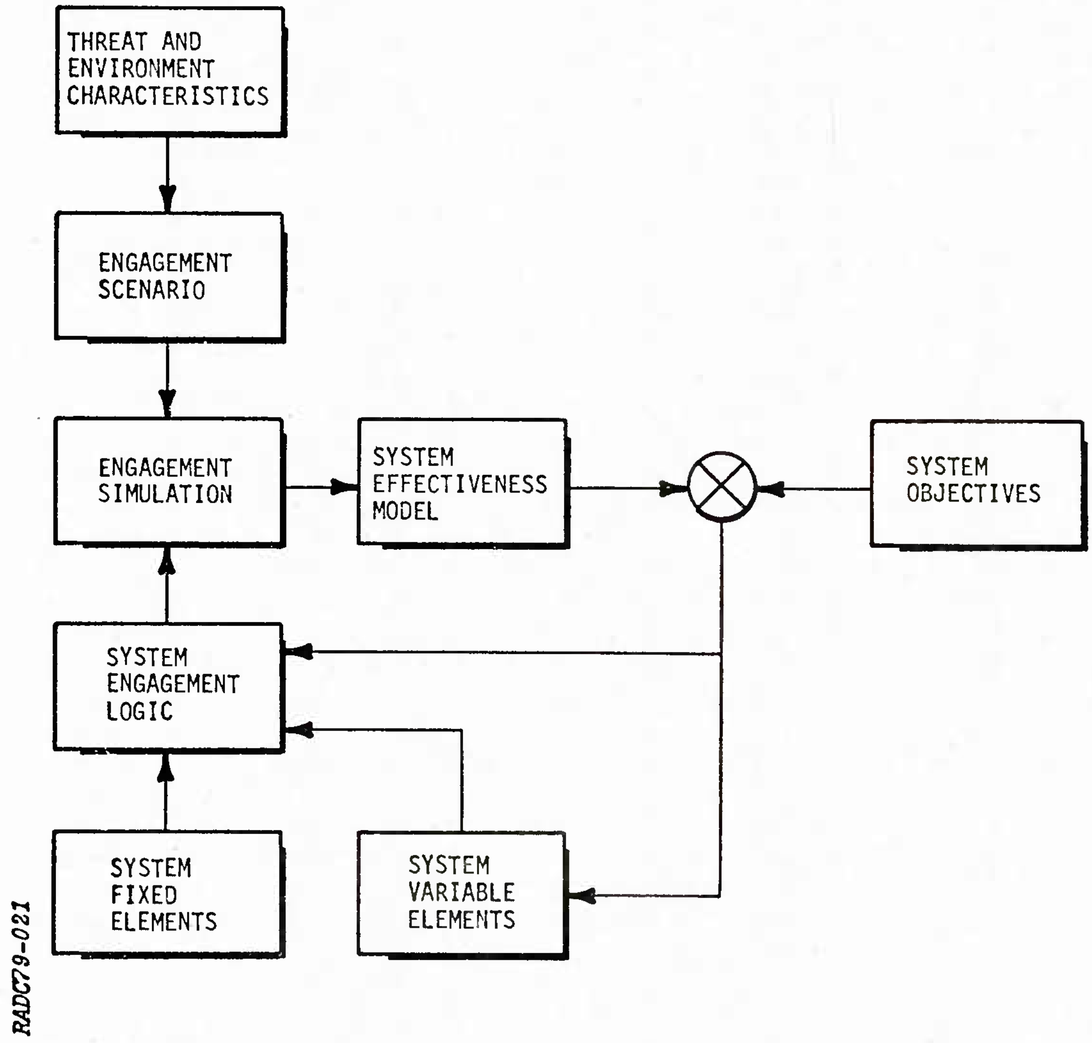


Figure 5-5 Candidate Evaluation Process

system configuration model are combined with the system engagement model to form a simulation of the candidate system. The system simulation is exercised against each engagement scenario in an engagement simulation. Monte Carlo replications are generally desirable because weapon system engagements are highly stochastic. The system effectiveness model is then applied to evaluate the results of the engagement.

The measured effectiveness values are then compared against the system objectives. If the system fails to meet the objectives, the variable system elements and the engagement model are modified to improve the performance. Because the system is ill-defined, this "tuning" is usually a trial-and-error process, supported by trade-off analysis to the extent possible. The output of this exploratory evaluation is a set of response surfaces defining the system effectiveness over a range of system, threat, and environment parameters.

The above analysis is conducted based on nominal assumptions about the mission, system threat, and environment. The next step is to question the nominal assumptions and examine the system performance under different conditions. This step is called sensitivity and risk analysis. The effects of system cost and system resource constraints should be examined as part of this analysis.

There are two general approaches for selecting the preferred candidate system:

- Fixed Effectiveness Approach -- the system that meets the required effectiveness level at the lowest cost is selected.
- Fixed Cost Approach -- the system that has the highest effectiveness for a given cost is selected.

Occasionally, but rarely, one candidate system dominates the others (i.e., has the highest effectiveness at all levels of cost). In this case the selection is obvious, provided the candidate is acceptable on the basis of sensitivity, risk, and development schedule. As a rule, however, none of the candidates dominate and selection calls for expert judgement considering all factors of effectiveness, cost, uncertainty, and schedule. An excellent discussion of the selection problem can be found in Quade and Boucher [8].

Eventually, one preferred candidate system or a pair of closely ranked contenders must be selected for further development, provided that at least one of the candidates is acceptable. At the same time, the scope of the mission may be narrowed and certain threat scenarios might be discarded as unlikely.

The problem to be solved (PTBS) is formally documented in a preliminary system specification (Type A). The system is described in terms of its operational functions (i.e., system engagement model) and operating rules. The specification should explicitly contain the following:

- Mission definition
- Background assumptions
- Threat definition (present and extrapolated)

- Physical environment definition
- System interfaces with other systems
- Operating modes and concepts
- Performance requirements
- Availability, reliability requirements
- Survivability, graceful degradation requirements
- Other constraints (size, weight, power)
- Growth requirements
- Logistics requirements
- Human factors.

Schedule constraints are defined in the RFP. Cost constraints may be contained in the RFP (design-to-cost systems), or may be withheld from prospective bidders.

At this point, the system analysis effort has established the technological, performance, cost and schedule credibility of the system. Decision makers have evaluated the analyses, the uncertainties and the risks, and have found them acceptable. The next step is to proceed with the system engineering phase.

5.1.3 Problems

Quade and Boucher [8] contains a detailed discussion of the pitfalls and limitations of system analysis. Fisher [9] summarizes the more common pitfalls as follows:

- Failing to allocate and spend enough of the total time available for a study deciding what the problem really is.
- Examining an unduly restricted range of alternatives.
- Trying to do too big a job.
- Determining objectives and criteria carelessly.
- Using improper costing concepts.
- Becoming more interested in the details of the model than in the real world.
- Forcing a complex problem into an analytically tractable framework by over-emphasizing ease of computation.
- Failing to take proper account of uncertainty.
- Treating the enemy threat too narrowly.

These pitfalls are symptomatic of a single underlying problem -- complexity.

The system analyst is faced with starting from an ill-defined and, perhaps, wrongly perceived problem, and developing a comprehensive analysis of that problem with limited time and resources to do the job. His apparent productivity is low because he must spend a large amount of his time gathering information and gaining an understanding of the relevant factors involved in the problem. To gain this understanding he must do experimental modeling and simulation. Much of this work will be discarded. To handle the breadth of his task, the analyst must usually sacrifice depth. Yet the decision maker, who uses the analysts' work, expects to see an analysis supported by quantitative information and mathematical relationships, even if approximate.

If the analyst considers a large number of alternative system concepts, he runs the risk of addressing each one superficially without time for adequate sensitivity analysis. If he restricts himself to a limited set of alternatives, he runs the risk of omitting an unrecognized superior candidate.

While the analyst can generally represent the performance of system components (such as radars and weapons) by relatively simple mathematical equations, he has great difficulty representing data processing needs in a meaningful way. Consequently, data processing issues tend to be deferred until late in the system engineering phase. At that time, however, significant and irreversible decisions about system structure have been made. These often place such burdens on the data processing system that the entire system concept becomes infeasible or far more expensive than originally estimated.

In addition to the problems of complexity, the system analyst is faced with communication and validation problems. The communication problems fall into four categories:

- Organization and retention of data for the analyst's own use.
- Representation of concepts and information for review by operational users of the system.
- Representation of analysis results for decision-maker consideration.
- Representation of system requirements in system specifications.

The efficiency and effectiveness of the analyst is largely determined by his ability to organize, retain, and structure a large body of data relevant to the mission, threat, environment, and potential system components. In the past, much of the needed information has been widely scattered in reports, textbooks, notes, and undocumented experience. Modern data base technology offers a powerful means of organizing and retaining often used data, particularly within agencies dedicated to specific-mission system areas.

The analyst rarely has the military combat experience to view the system from the eyes of the operational user. The operational user often lacks the specialized technical expertise needed to evaluate system details. Hence, a communication gap exists between the user and designer, in addition to the fact that direct consultation between them is rare. Unless this gap can be filled by representations of concepts and information that are easily understood by both parties, the designed system may be a complete mismatch to operational needs.

The system analyst does not select the system option to be pursued. That choice is in the hands of a decision maker who makes a judgement from a broader perspective than that of the analyst. The objective of system analysis is to assist decision makers by providing a better basis for judgement. The decision maker is not served if the critical system issues are lost in obscure representations and mazes of details, or if the form of the presentation is not comparable to data for other systems competing for funding.

Finally, the system requirements formalized in the system specification must reflect the true needs of the operational user, and must be presented in such a way that the intent cannot be misunderstood. While it is important to allow the system designer maximum design freedom, it is not wise to let him decide how the system is to be operated. Yet, the weakest parts of most specifications are those dealing with operating concepts and rules of engagement.

Despite the fact that the results of the system analysis are used to make far-reaching program decisions, little effort is applied to validation of the analysis. The tight cost and schedule constraints of the system analysis phase, the communication gap between analyst and operational user, and the faith that later system engineering work will uncover any faults, are the factors responsible for this curious lack of validation. However, once a project is headed in the wrong direction from the start, it is difficult and expensive to undo the effects of a faulty system analysis.

Effective system analysis, particularly for very large systems, is increasingly dependent on large-scale simulation models that require significant software development. This software is usually defined for the problem of the moment. Little effort is devoted to construction and maintenance of standardized software libraries that can be used for a variety of projects. Because the software is not a formal deliverable, rigorous testing and validation is not pursued. Yet, because errors in these models have significant impact, some means must be found to subject them to formal software engineering practices without stressing the limited resources available.

To address these problems and permit increase in the system analyst's productivity, the following capabilities are needed:

- Better tools for structuring and retaining information about the problem.
- Representation techniques that present information understandable to analysts, operational users, and decision makers.
- Better tools for constructing and validating simulation models which can be exercised at low cost and built with limited resources.
- Better tools for post-processing, summarization, and presentation of simulation-generated data.

- Techniques for defining data processing needs and issues earlier in the system definition effort.
- A methodology for defining and describing system operating concepts and engagement rules.

5.2 SYSTEM ENGINEERING PHASE

5.2.1 Scope

The system engineering phase confirms the results of the system analysis phase and extends them into a detailed plan for system composition, operation, development, and support. The focus of the system analysis phase is on definition and selection of feasible goals and objectives. The focus of the system engineering phase is on implementing those goals and objectives within the technological state-of-the-art. The principle task is the direction and coordination of subsystem design activities which produce a set of sub-optimal subsystems that function together as a well-balanced system.

Although system engineering activities continue throughout system development, the primary requirement definition activities are completed with the finalization of the system specification and the generation of subsystem specifications and interface control documents (ICDs). Thus, what we call the system engineering phase generally corresponds with the "validation phase" of the system life-cycle. The validation phase typically begins after DSARC I and a System Requirements Review (SRR) and concludes with DSARC II and a System Design Review (SDR). The lower level activities of Data Processing Subsystem Engineering, Software Requirements Engineering, and Process Design also fall within this timespan.

The system analysis phase is usually carried out within DoD agencies, supported by study contracts to advisory groups or industry. The system engineering phase is carried out by industry contractors, either System Engineering and Technical Direction (SETD) contractors retained by the sponsoring agency or the prime contractor for the specific system development.

The system analysis phase tries to examine the entire spectrum of practical solutions to the particular weapon system problem. During this phase one or two of the most promising solutions are selected for possible development and the rest are discarded. The system engineering phase then elaborates and refines the definition of the chosen constructs, performs any remaining analysis necessary to select a single system design, and guides the lower level design of component subsystems.

System engineering is concerned with life-cycle cost, development cost, schedule, risk, logistic support, training and maintenance, as well as all aspects of nominal and off-nominal system performance. Hence, system engineering requires multi-disciplinary expertise involving all engineering specialties, physics, mathematics, computer science, psychology, economics, and management sciences.

5.2.2 Content

The preliminary system specification developed in the system analysis phase provides the basic information about "what the system does". The final system specification completed in the system engineering phase, additionally completes the definition of "what the system is" and "how the system is made". The definition of "what the system is" requires identification of lower level parts or components of the system (e.g., system segments, subsystems, prime configuration items) and definition of how these components are interconnected and interact to form the system. The definition of how the system is made involves planning for assembly, integration, and testing of system components; and specification of standards constraining design and development practices.

The requirements on the system as a whole consist explicitly of "what the system does" (functional requirements) and "how well the system does those things" (performance requirements), and explicitly or implicitly of the system operational need date and lifespan, and what it costs (i.e., the system's capabilities are valuable only within a certain time interval and have only a finite value). The "acceptable solution" defined by the system engineers identifies "what the system is" (parts and interconnections), how it is made (development and production plan) and "how it is used and maintained" (logistics, operations, and maintenance plans). The solution is not necessarily unique, but aims to be nearly optimal over the ranges of important parameters considered.

The additional constraints on the system imposed by the system engineer's design decisions then set the context for and become part of the requirements on development of the identified subsystems. One of the system engineer's objectives is to achieve a well-balanced system. This also means that the degree of engineering difficulty and complexity should be fairly distributed across the subsystems.

It would be impossible to capture in a short paragraph all of the diverse disciplines, views, and techniques in the system engineering process. Indeed, the textbooks written to date can only state high level rules, and highlights of important disciplines. However, the essence of system engineering is the decomposition of complex problems into simpler, more tractable sub-problems that can be attacked by specialist groups in a manageable manner, followed by balanced synthesis of sub-problem solutions into a system solution. In general, the approaches used by the system analyst are applicable. However, the system engineer generally has more resources committed to his support and is considering a specific system concept, rather than a broader set of alternatives.

5.2.3 Problems

With respect to data processing requirements, one problem in the system engineering phase has been perceived for some time, and has resulted in the promulgation of DoD Directive 5000.29 requiring that DP be treated as a subsystem on an equal basis with other subsystems. The symptoms of the problem have surfaced as systems that would not work because the software requirements could not be satisfied.

The system engineer is of necessity a generalist, working with a limited set of descriptive subsystem parameters to define a balanced system. As discussed below, a lack of appropriate DP parameters that can be traded-off against other subsystems has forced DP issues into the background at the system level. This postponement of concern, coupled with a blind faith that software can resolve any hardware interface problems has resulted in irrevocable system decisions that preclude a satisfactory DP solution. Unfortunately, the problem is getting worse.

The exploding technological capabilities emerging in computing hardware today (semiconductor logic and memories, optical processing, holographic devices, and distributed systems) may revolutionize the design and construction of large systems. Increased functional performance and decreased component cost is leading to new tradeoff opportunities in system engineering that must be considered in the early requirements phase to exploit their benefits.

Previously, monolithic centralized data processing was tacit in weapon systems development, and DP requirements were generally characterized by specifying:

- DP interfaces with other subsystems
- The computation required at a centralized location
- The limitations and performance indices.

Now, however, with distributed systems and newer supporting technology, several practical problems arise which accentuate the widening gap between system engineering and DP requirements needs (Table 5.1). All of these DP issues have moved higher on the list of critical areas, due to the necessity to distribute system elements in the advanced constructs being considered and the cost/reliability economies of modern microcomputer technology.

Effective interaction between DP and system engineers has been hindered by the lack of appropriate descriptive performance parameters. As indicated in Figure 5-6, the other subsystems can be represented at the system level by specifying the values of a small set of defining parameters which also serve as top-level requirements for those subsystems (e.g., maximum range for a radar, fly-out time for a missile). No such parameters exist for data processing. The lack of such parameters for DP has been a major interest in establishing an effective interface between DP and system engineers. If appropriate quantitative parameters could be found which characterize DP performance at the system level and represent system requirements levied on the DP subsystem, then the DP subsystem could be considered directly in system level trade-offs at a very early stage of system design.

The difficulties of specifying DP requirements in the absence of a well-defined set of characteristic top-level parameters, and the complexities associated with specifying the requirements on a set of distributed DP elements, have led to the practice of defining system constructs and their resulting DP requirements in terms of a "preferred" construct design. This practice not only limits the design freedom available at subsequent stages,

Table 5.1 Current Practical Problems in Specifying DP Requirements

- HOW MUCH DISTRIBUTION OF PROCESSING IS APPROPRIATE WITHIN OTHER SUBSYSTEMS?
- WHAT IS THE EFFECT OF DP TECHNOLOGY LIMITATIONS ON SUBSYSTEM DEFINITION AND OPERATING RULES?
- HOW MUST RESOURCES BE MANAGED, INCLUDING THOSE OF DP?
- HOW AND WHEN ARE THE MULTITUDE OF INTERFACES TO BE SPECIFIED?
- TO WHAT EXTENT DOES COMMUNICATION PLAY A ROLE IN THESE CONSIDERATIONS?
- HOW DO YOU SHOW EVIDENCE OF DP FEASIBILITY?
- HOW CAN DP REQUIREMENTS BE CONFIGURED SO AS TO BE ADAPTABLE TO CHANGE?
- AT WHAT POINT IN THE SYSTEM LIFE-CYCLE SHOULD DP REQUIREMENTS DISTINGUISH BETWEEN SOFTWARE, FIRMWARE, AND HARDWARE?
- HOW ARE THE CRITICAL ISSUES IDENTIFIED?
- HOW DO YOU OBTAIN EARLY RELIABLE ESTIMATES OF DP COST, SCHEDULES, SIZE, WEIGHT, AND POWER?
- HOW ARE DP/C, DP/SENSOR, DP/WEAPON TRADES IDENTIFIED?

TRW78-031.1

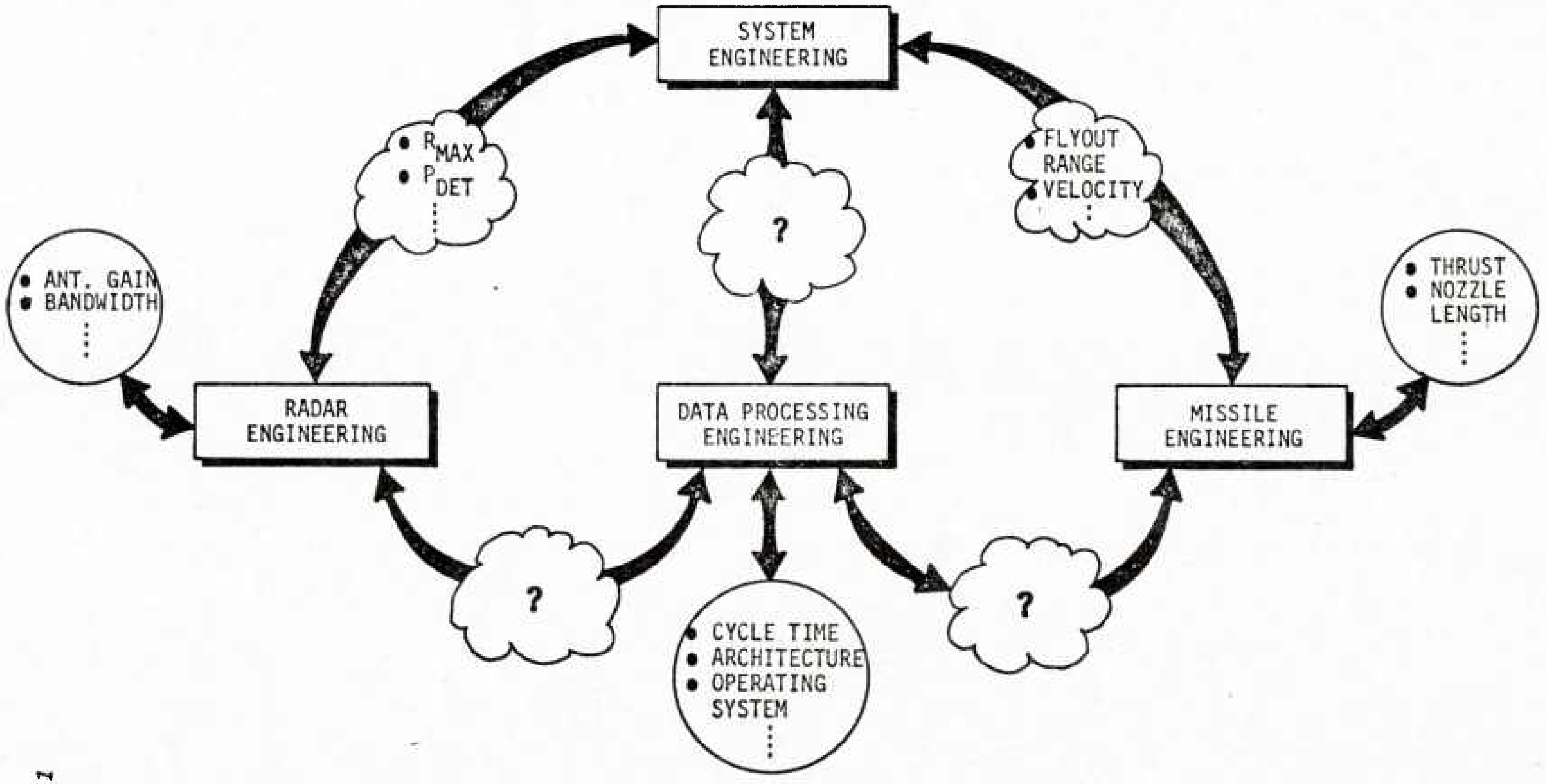


Figure 5-6 The Poor Fit of DP Parameters in the System Context

but it also firmly obscures the real requirements. If this practice is to be eliminated, we must develop a true "requirements first" point of view which concentrates on the definition of the real requirements without resorting to describing them in the context of a specific design.

It is clear that the "distribution", "DP parameter" and "requirements first" issues have a multitude of facets. Our objective is to limit our attention to a practical methodology which:

- Makes DP integral to the system definition process.
- Identifies critical issues early.
- Relates subsystem requirements to the tradeoffs among alternate system configurations.
- Develops complete and consistent system operating rules and function descriptions through simulation.
- Documents system models, performance requirements and design variables.
- Emphasizes early formation of system level performance requirements and exhibits alternatives in subsystem structure to meet them.
- Treats DP as a finite resource.

Another problem, one of complexity for the system engineer, but one of communication for those involved downstream, concerns the allocation of data processing functions to subsystems. Consider a weapon system with identified subsystems including a radar subsystem and a DP subsystem. The DP subsystem can be conceived as providing all DP support of other subsystems, and must act in an integrated manner to support the system mission. Many of the DP functions, however, are relatively autonomous in that they are bound to a specific subsystem (e.g., radar) and are transparent to other subsystems. Moreover, the testing of the radar subsystem as an entity depends on the presence of these DP elements. Question: Should the system engineer allocate these DP functions to the radar subsystem or the DP subsystem for development purposes? A good practical case could be made for either choice. Yet, either choice complicates the human interface and coordination problems downstream.

5.3 DATA PROCESSING SUBSYSTEM (DPSS) ENGINEERING PHASE

5.3.1 Scope

DP subsystem engineering activities are a subset of the on-going system engineering effort. The DPSS engineer provides specialized knowledge during the definition and tradeoff studies that identify the required characteristics of the various subsystems. Working with other subsystem engineers (e.g., sensor, command and control, weapon delivery) the DPSS engineer helps to identify workable system operating rules, DP limitations imposed by physical laws and the technology state-of-the-art, and the characteristics of interfaces between subsystems.

During the system engineering phase, the system requirements are decomposed until subrequirements are sufficiently detailed that they can be uniquely allocated to subsystems. In parallel, a functional system model is synthesized and exercised against simulated threat scenarios. The system model includes submodels defining the behavior of each subsystem, and the interactions between subsystems and between subsystems and the threat/environment model. By systematic variation of model parameters and operating rules, satisfactory conditions of system behavior are identified, and decisions are made about preferred system parameters and values. The DPSS engineer is responsible for DPSS functional modeling to support these activities.

When the system requirements have been decomposed and allocated to the subsystems, the DPSS engineer is responsible for consolidating the requirements allocated to the DPSS and refining them to form a coherent subsystem requirement package of functional, performance, interface, and development requirements. The DPSS requirements may be documented in various ways (e.g., a B1 specification, a B2 specification, or an informal technical report). There is much variation in the documentation of DPSS requirements because MIL-STD-490 is ambiguous about the proper specification (e.g., B1, B2) for a DP subsystem as an entity.

At this point, the functional and performance requirements on the DPSS are stated in terms of weapon system parameters and in the context of the system mission. The remaining DPSS engineering activities are concerned with the selection and development of a hardware/software/firmware combination that meets these requirements and has acceptable availability, reliability, maintainability and cost properties. The DPSS engineer is ultimately responsible for both the hardware and software architecture of the DPSS and the selection and procurement of appropriate hardware. As part of this responsibility, he monitors and coordinates the efforts of the software requirements engineering, process design and hardware engineering phases, and reports upward to the system engineering organization.

An important part of the DPSS engineer's job is maintaining upward traceability between the DPSS requirements and the system requirements, and downward traceability from the DPSS requirements to the separate packages of software, hardware, and firmware requirements. It is generally more difficult to maintain downward traceability because the lower level requirements are stated in DP-oriented terminology rather than system terminology.

5.3.2 Content

The discussion in 5.2.2 is applicable to DPSS engineering as well as system engineering. The primary difference is that the DPSS engineer is focused in a more limited area. His "system" is the DPSS. His "environment" is all other components of the weapon system plus the environment of the weapon system. The components that he must select to form a problem solution are processors, memory, communications links, peripheral devices, and various classes of software. While the primary emphasis in the past has been on digital devices and discrete phenomena, new technological advances (e.g., optical processing, holographic processing) are demanding that DPSS engineers become involved with essentially analog devices and continuous phenomena.

The DPSS engineer must move from a consideration of "what the DPSS does" to "what the DPSS is" in a series of steps. What the DPSS does is stated in terms of information and actions on information. The information is about things in the outside world (e.g., aircraft; position, speed and heading of aircraft) of interest to the system. The actions on information (e.g., provide launch signal to interceptor missile when target is within ten mile range and ...) are related to the functions and operating rules of the total weapon system. The functional requirements on the DPSS are, thus, concerned with "information processing".

"What the DPSS is" is described in terms of "data processing" and the physical components necessary to do the data processing. "Data processing" is concerned with the representations of information and the logical or arithmetic manipulation of those representations. Thus, topics such as word size, formats, addressing mechanisms, paging, and queue management are data processing concerns, not information processing concerns.

Thus, before the data processing needs and physical components can be accurately assessed, a well-defined statement of the information processing requirements must be available. This statement can be defined in terms of in-coming and out-going discrete packets of information called "messages", the individual information items contained in the messages, the logical information structure (i.e., the logical data base model) to be maintained, the information processing responses to given stimuli, and the relative priority or importance between information processing actions. For each information item the appropriate range of values should be identified (e.g., slant range varies between 10,000 feet and 600,000 feet). The highlights of the major information groupings and high level processing steps can be summarized in diagrams such as Figure 5-7.

The next step is to identify the information processing performance requirements and the load (both average and peak) on the DPSS. The response time requirements are defined by identifying each stimulus-response path through the DPSS and determining an acceptable response time for each path, based on system simulation model behavior. Each path can be further broken down by assigning time budgets to each action along the path. The peak and average load estimates can then be developed by functional simulation techniques.

At this point the DPSS engineer can begin to identify candidate processing algorithms for each subsystem action and estimate instruction counts and memory needs for each action. He can then examine various functional processing architectures that seem appropriate to the problem and identify required instruction execution speeds for candidate hardware. Certain critical paths and algorithms will be identified and engineering effort can be applied to find acceptable solutions.

The emphasis of the early efforts in DPSS engineering is not to develop comprehensive designs for the DPSS, but to develop reasonable estimates of feasible DP performance, identify critical issues, assess the ability to meet system requirements, develop a satisfactory allocation of DPSS requirements between hardware, software, and firmware, and chart the direction of further engineering work in those areas.

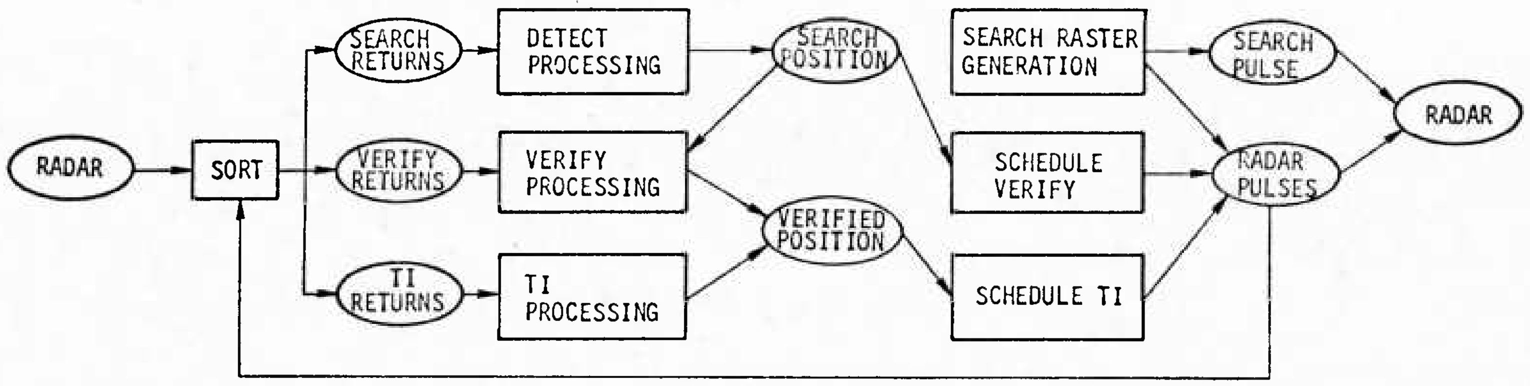


Figure 5-7 Data Flow Diagram

5.3.3 Problems

The lack of an effective set of DP performance parameters for system engineering tradeoff studies has been discussed in 5.2.3. The impact on the DPSS engineer has been that the resultant allocation of performance requirements to the DPSS has been unduly difficult or impossible to satisfy.

Hardware selection is usually one of the critical issues in DP Subsystem Engineering. In the past, hardware has been selected early in a project based on gross estimates of software memory and execution time needs. This approach has been successful when the properties of the application are relatively well known from past experience. In applications where prior experience is absent, the software sizing and timing estimates tend to be much lower than true values. Consequently, the selected hardware must operate at near-saturation levels. Boehm [10] has illustrated the effect on software development cost for real-time systems, as shown in Figure 5-8.

In the future there will be a wider choice of computer architectures, and general purpose architectures will be "modifiable" by microprogramming. To extract maximum performance from these configurations, the architecture and the software algorithms must be carefully matched. At present, there is no effective, widely available, set of tools to rapidly identify high performance architecture/algorithm combinations and evaluate hardware/software/firmware tradeoffs in a systematic manner. The problems of geographically distributed systems add another dimension of complexity because communications factors (bandwidth, delay) must be considered to determine where data processing nodes should be located. In the face of these complications, how can anyone be sure that preliminary DP estimates are credible?

Little has been done in the past to characterize the total information processing needs of a system before the boundaries between subsystems have been chosen. The traditional radar has included analog devices and custom-made signal processors (analog or digital). The traditional DP subsystem has usually consisted of one or more general purpose digital processors. Today, digital techniques are becoming competitive with traditional analog solutions, and networks of general purpose microprocessors provide a means to build customized special purpose processors from standard components. As a result, several tradeoffs may be needed to define the best sensor/DP boundary. Similar problems exist in defining DP/communications boundaries. Many communications functions are now supported by digital devices and the communication subsystem itself is increasingly dedicated to data transfer in support of the DP subsystem.

It is clear that the growing complexity of DPSS engineering must eventually force a more structured approach to this phase -- grossly, a three-stage approach: 1) information processing requirements, 2) data processing requirements, and 3) processing hardware requirements. For this to be possible (in a visible and controllable way) the MIL-STD-490 specification hierarchy needs to be reconsidered and altered to address the unique problems of DP subsystems. The present abrupt transition from system level A spec to computer program level B5 spec leaves most of the critical DP requirements and design decisions invisible and undocumented.

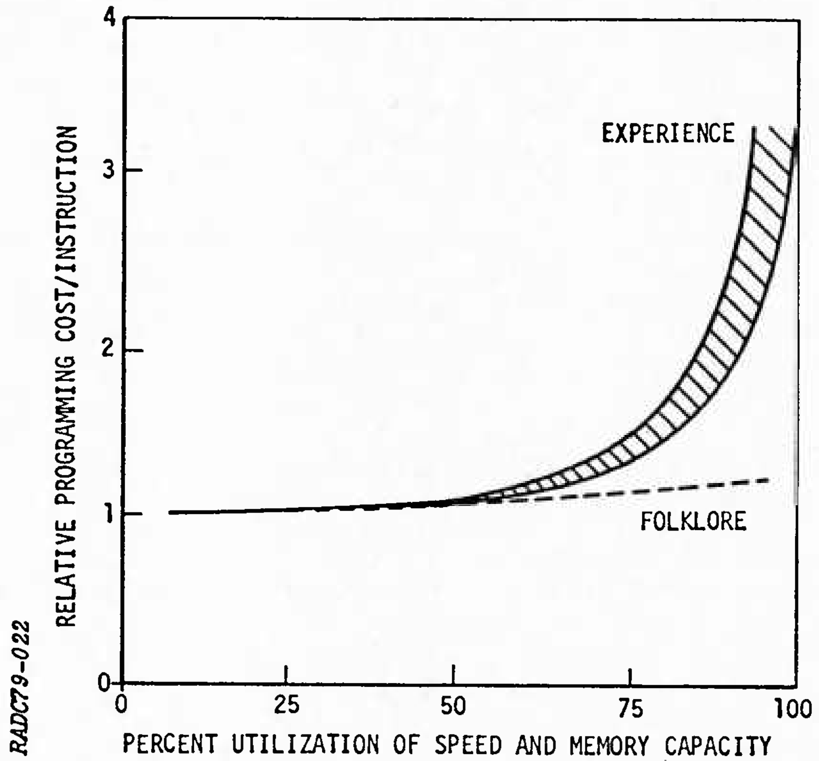


Figure 5-8 How Does Hardware Selection Affect Software?

This "documentation gap" creates a problem in maintaining traceability between system requirements and software requirements. As a result, the impact of system requirements changes on software cannot readily be assessed and serious problems may not be discovered until the system is deployed.

5.4 SOFTWARE REQUIREMENTS ENGINEERING PHASE

5.4.1 Scope

The purpose of the software requirements engineering phase is to transform the DPSS functional and performance requirements, expressed in system terminology and parameters, into a more detailed definition of requirements expressed in data processing terms. The high-level information processing description of the DPSS, derived in the DPSS engineering phase for estimation and assessment purposes, is extended and refined to a level of detail sufficient to state precise software requirements. In a sense the term "software requirements engineering" is a misnomer because the same techniques can be used to specify information processing requirements for hardware and firmware.

Traditionally, software requirements engineering has been one of the activities of the DPSS engineering phase. The concept of a separate software requirements engineering phase was first introduced in the development of the BMDATC Software Development System (SDS) [11]. The TRW Software Requirements Engineering Methodology (SREM) was explicitly designed to meet the needs of this phase as conceived by BMDATC.

The software requirements engineering phase provides a necessary bridge between the system engineer's view of the DPSS and the software designer's view of the DPSS. The software designer should not require expertise in weapon systems in order to do his job. He should be able to understand the DP problem in terms of messages arriving and departing through interfaces, information flow and information maintenance within the DPSS, and processing actions upon that information. The software requirements engineering phase is completed when: 1) the system functional and performance requirements allocated to the DPSS have been stated in these terms, 2) the software requirements are traced to the DPSS requirements, 3) the software requirements have been validated for completeness, consistency and other desirable properties, 4) the requirements have been evaluated for feasibility, and 5) the requirements and supporting information have been documented for input to the process design phase.

5.4.2 Content

The traditional approach to software requirements generation has revolved around writing a specification document, with little thought given to an orderly methodology for determining actual requirements. Typically, a group of people knowledgeable about the system problem would be convened to describe, in English, what the software should do. The job was considered finished when the allocated time and money ran out, and a decently readable document in some approved format was published. Major difficulties were that management had no reliable interim visibility into how the job was progressing, and at completion there were no objective criteria for determining the quality of the requirements document.

Although many requirements generation aids and description tools (e.g., CARA) have evolved in recent years, specific methodologies for using them have been avoided. The major exception was the TRW SREM, where an explicit methodology for the SRE phase was a prime objective of BMDATC research. We will describe the content of the software requirements engineering phase in terms of the SREM viewpoint since that research was oriented specifically toward real-time software requirements for weapon systems.

Figure 5-9 presents an overview of the software requirements engineering activities for real-time weapon system software. It is presumed that work during the DPSS engineering phase has identified the logical input and output interfaces (i.e., the sources and sinks of information) to the DPSS, and the information content of messages passing those interfaces. Further, it is presumed that the types of appropriate responses to various stimuli and conditions have been defined in terms of the system operating rules and the gross nature of the processing to generate these responses has been identified. It is also presumed that the various subsystems that interact with the DPSS have been functionally defined to a level of detail sufficient to support dynamic system modeling and that performance allocations to the subsystems have been made in terms of system performance parameters.

Although desirable, it is not necessary to have all of this information in final form before beginning the software requirements engineering activities. SRE will usually proceed iteratively with the system and subsystem engineering tradeoffs and will expose new issues for resolution as analysis proceeds. As the level of technical awareness of software issues increases, decisions can be made about combination of interfaces (i.e., communication multiplexing) and higher level system control issues.

The first activity in the computer-aided SREM methodology deals with the development and analysis of a functional requirements data base. This begins with a structured definition of input and output interfaces, messages that are passed through each interface, data elements and files that make up each message, and data available at system initialization. Next, attention is turned to the paths of processing steps that are involved in reacting to the various input message stimuli to cause state changes in the DPSS and/or output messages to other subsystems. In SREM the sequences of functional processing steps are described as requirements networks (R-Nets) that must conform to certain structure rules. Next, attention is focused on the logical structure of data maintained internal to the DPSS and the input-output and creation-destruction operations of each processing step. During this process, data base analysis procedures are employed to ensure the completeness and consistency of the defined data base. As information becomes available or necessary, detailed attributes and descriptions of each element in the data base are defined, and traceability relations are established between elements of the requirements data base and the original documentation provided by DPSS engineering.

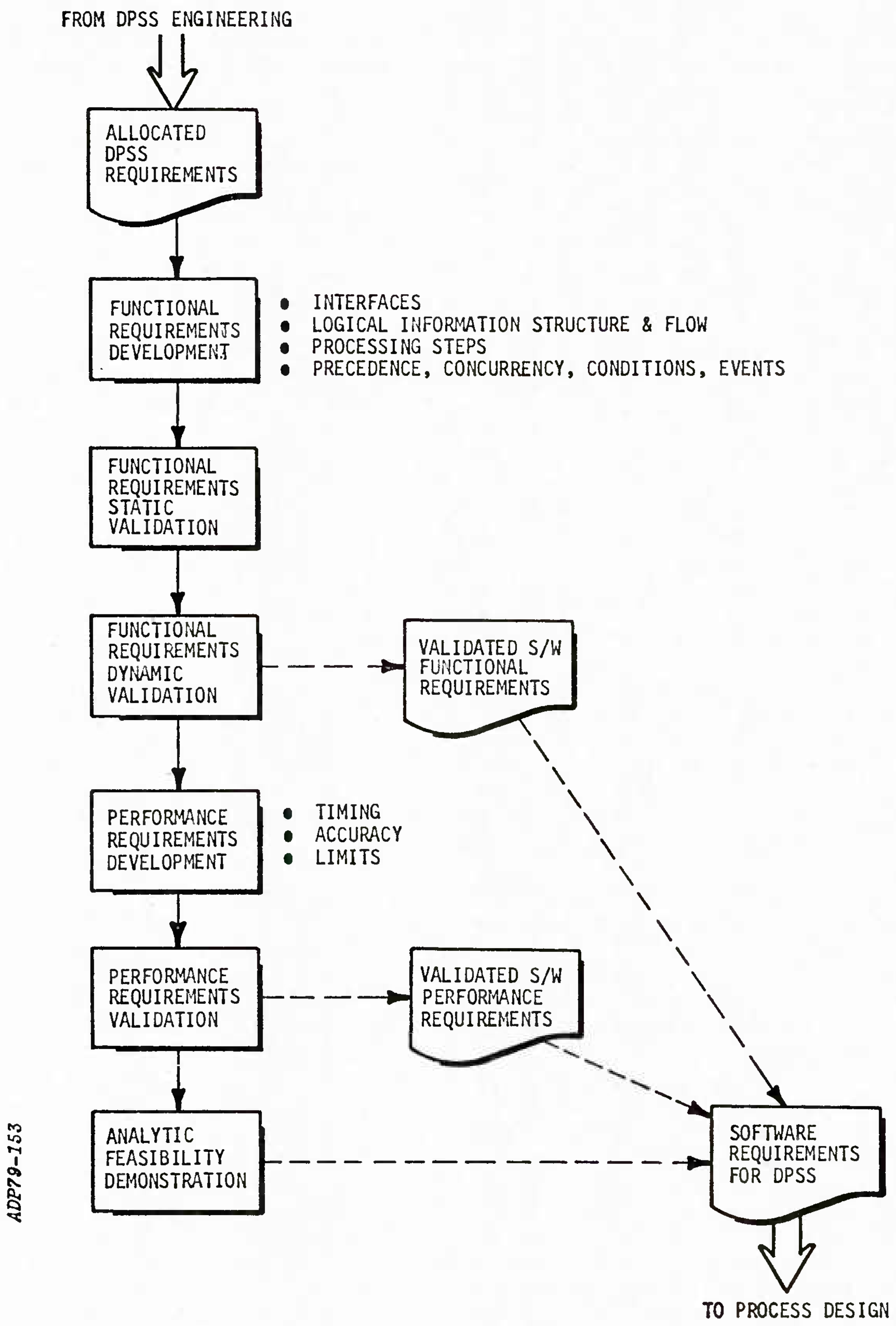

FROM DPSS ENGINEERING
ALLOCATED DPSS REQUIREMENTS
FUNCTIONAL REQUIREMENTS DEVELOPMENT
● INTERFACES
● LOGICAL INFORMATION STRUCTURE & FLOW
● PROCESSING STEPS
● PRECEDENCE, CONCURRENCY, CONDITIONS, EVENTS
FUNCTIONAL REQUIREMENTS STATIC VALIDATION
FUNCTIONAL REQUIREMENTS DYNAMIC VALIDATION
VALIDATED S/W FUNCTIONAL REQUIREMENTS
PERFORMANCE REQUIREMENTS DEVELOPMENT
● TIMING
● ACCURACY
● LIMITS
PERFORMANCE REQUIREMENTS VALIDATION
VALIDATED S/W PERFORMANCE REQUIREMENTS
ANALYTIC FEASIBILITY DEMONSTRATION
SOFTWARE REQUIREMENTS FOR DPSS
TO PROCESS DESIGN
ADP79-153

Figure 5-9 Software Requirements Engineering Activities

As the definition of the functional requirements passes specific milestones, appropriate parts of the evolving data base are subjected to formal (in the programmatic sense) static validation by automated procedures. When all appropriate validation tests have been passed, documentation of those requirements can be started.

While documentation proceeds, the completed data base is subjected to dynamic validation through construction and execution of functional simulations directly linked to the requirements data base elements, attributes, relationships, and structures. This process uncovers deficiencies not detectable by static analysis alone, and provides increased confidence in the validity of the requirements.

At this point, attention is turned to the precise definition of performance requirements (e.g., timing, accuracy) that specify "how well" the DPSS is to perform the functional processing. The performance requirements allocated to the DPSS expressed in system terminology are decomposed, analyzed, and related to specific points on the processing paths through the DPSS. As part of this effort the SREM approach demands definition of the particular test data to be recorded, and procedures (i.e., pass/fail tests) necessary to analyze that data to ascertain that the requirements are met. Thus, "testability" of the stated performance requirements is ensured.

The validation of the performance requirements employs the static analysis aids and functional simulations previously used in validating the functional requirements. Frequently the functional models are insufficient to express the relationships needed for performance requirement validation and more detailed analytic models are constructed, perhaps including prototype algorithms.

After the performance requirements have been validated and documented, the DPSS software requirements are complete and detailed process design activities can be started. There is one major engineering issue that has not been resolved, however. That is, can a software design be developed that will meet the stated requirements? The last phase of SREM, analytic feasibility demonstration, addresses this issue concurrently with early process design activities. This demonstration involves the development of a candidate design (analytic models and algorithms) which meets all of the stated requirements except those associated with timing. The objective is to demonstrate that at least one computational solution to the problem exists, whether or not this solution is adequate for real-time performance. This step provides assurance that the functional requirements are computationally feasible and provides "calibration" models for assessment of the eventual real-time software. The real-time feasibility of the requirements is a major process design issue.

It should be pointed out that modifications to and reappraisals of earlier activities are often required by discoveries in later analysis activities. The methodical approach of SREM ensures that these iterations are confined as much as possible within SRE steps and that major requirements problems have surfaced before substantial design effort is expended.

5.4.3 Problems

The problems of traditional software requirements definition approaches have long been acknowledged and have been the topic of major DoD investigations [12, 13, 14] as well as research by major software contractors (e.g., [2]). These investigations have indicated the major problems identified in Section 4, and various symptoms (e.g., incompleteness, inconsistency, ambiguity, overconstraint, incorrectness, volatility). Indeed, the research leading to the TRW SREM was motivated by these deficiencies and the fact that they had become critical in the ballistic missile defense (BMD) technology area [11].

The SREM approach has received favorable comment during evaluation studies and initial production usage [15, 16, 17]. However, current experience indicates problems that hinder the full effectiveness of SREM, and many other advanced development techniques. These can be considered "growing pains" in the transfer of new technology.

The first problem concerns the time and effort allocated to software requirements engineering. Traditionally, 3 to 6 months and 7 to 12 percent of the development budget have been allocated to the primary requirements definition effort with an additional 5 percent perhaps devoted to requirements maintenance activities throughout the project schedule. Despite the inadequacies of the generated requirements, program managers, customer agencies, and procurement agencies have come to expect and demand the traditional allotments to requirements activities. Modern requirements engineering approaches and research have indicated that the traditional allotments result in superficial definition of requirements and that thorough requirements engineering, although more productive, may demand more resources. The resulting benefits, in terms of time and cost saved and risks avoided, cannot be substantiated without a long "track-record" of successful projects. However, management inertia with regard to resource allotments inhibits the use of new technologies and makes the demonstration of the full capabilities of the technology difficult.

A second problem is that the full efficiency of a methodology for a specific phase (e.g., SRE) is not realized as long as the products from previous phases are lacking in information or quality. Trial applications of SREM to existing system or subsystem specifications often indicates that critical information is either absent or superficially treated. The most frequent deficiency is that concepts of operation and system operating rules are either completely absent or obscured in lower level details. Further development of software requirements in a systematic manner would require protracted coordination and data-gathering sessions with system and subsystem engineering personnel, thus stretching the SRE schedule.

The dependency of an effective software requirements engineering discipline upon equally rigorous system and subsystem engineering disciplines that recognize data processing needs has created growing awareness that system engineering methodologies are non-existent, beyond general high-level approaches to problems. The Axiomatic Requirements Engineering (ARE) program sponsored by BMDATC is a current research thrust to fill this gap.

A third problem is that of profitably using the products of a new technology. Until a new tool or technique has been installed and in use for some time, and until downstream users have been shown how to use the outputs, business will go on as usual in the downstream areas and the output products will be largely ignored. In the case of SREM, it is so recent that few process designers have had the opportunity (or the slack time) to experiment with the new possibilities for analysis that the SREM outputs provide. Other potential uses of the SREM output, particularly in test planning, have been identified, but needed research has not yet been funded.

These problems indicate that mere introduction of new technology into isolated phases of the front-end development process is not sufficient to solve even the problems of that phase, let alone global problems. What is needed is an orchestrated and balanced attack on the problems of all phases in an integrated manner, with consideration of the proper downstream use of the products, and strenuous attention to the training and education of users, project managers, customer agencies and procurement agencies.

5.5 PROCESS DESIGN PHASE

5.5.1 Scope

The dictionary definition of a "process" is "a continuing development involving many changes". The term is commonly used in engineering to refer to the complex evolution of a system from an initial state to a terminal state (e.g., a chemical process, a manufacturing process). Complex phenomena in the physical environment (e.g., rainstorms) are often described as processes. In weapon system engineering the interacting offense-defense encounters and engagements can profitably be viewed as processes.

In the world of computer science the term "process" has a similar connotation, and is a higher-level abstraction than the term "program". The concept is necessary to describe the activities and status of real-time event-driven systems, interactive time-sharing systems and distributed systems.

Consider an interactive time-sharing system where multiple users may have concurrent access to the same set of software programs. To describe the state of the system at any time, it is necessary to account for the allocation of resources among the users and the nature and status of each user's activities. A useful way to accomplish this is to view each user's interactive session as a process which may be active or inactive, and when active may be ready, running, or blocked. The overall system activity is a process which includes all of the user processes plus the actions of higher-level system control functions such as resource allocation, scheduling, and data management.

A geographically distributed system such as ARPANET, can be described as a single process, composed of the network control and processes representing each site's activity. The process at each site is composed of the site control and the user processes which have access to that site. Note that user processes may "migrate" from site to site (i.e., may be active at one site and inactive at all others) or may span several sites (i.e., be concurrently active at more than one site). These concepts are also useful in dedicated real-time weapon systems where the "users" of the system are the encounters and engagements needing service.

The concept of "process design" in weapon system data processing engineering has arisen from the notion that to properly service the encounter and engagement processes (i.e., the system requirements), the DP subsystem must perform as a set of computational processes (usually asynchronous and interacting) which contribute to the system mission in a harmonious and non-interfering manner. The role of the process designer, therefore, is to devise a top-level software architecture and control scheme that satisfies the DP subsystem requirements and provides coordinating constraints on further software design.

Marker [18] separates the software in embedded computer systems (such as in weapon systems) into three distinct classes:

- System Support - the operating system
- Computational Modules - the application routines
- System and Process Logic - the process design.

The process design consists of three inter-related components that implement the strategy and tactics of the system:

a) The system decisions made based on the outputs of the computational modules. The decisions which comprise a weapon system firing doctrine or an electrical power routing policy are examples.

b) The software execution policy or scheduling criteria. In the absence of infinite computing resources, a choice must be made to determine which computational module gets executed next. This scheduling policy has a significant impact on the performance of the system, particularly during periods when the computer is saturated or nearly saturated.

c) The organization of the system data base and control of the data flow between computational modules. In most real-time systems, the data base organization and the control of data flow have an overwhelming influence on the responsiveness and load-degradation characteristics of the data processor and, hence, of the system. The design of the data base and data flow is, therefore, of top-level importance to the overall software design and is very sensitive to the load and to the strategy of the system.

This particular view has been useful in development of software for a single data processor (or for a multi-processor with stringent constraints on allowable concurrent activities). An implicit assumption is that there is a single process resident in the configuration. Distributed data processing necessitates a generalization of the concept to allow a hierarchy of processes, with concurrently active concurrent processes under control of a higher-level process (either conceptual or real).

The primary inputs to the process design phase are the refined functional and performance requirements on the DP subsystem as a whole (produced in the software requirements engineering phase), and the DP load profile or "path load timelines" (produced in the DP subsystem engineering phase), plus the hardware and software constraints imposed by earlier subsystem design decisions.

The output products of the process design phase are designs for the top-level software architecture and control scheme; and derived requirements for individual software processes and tasks. These include requirements for direct and indirect support software, as well as primary mission software. In order to state the requirements within the restrictions of the MIL-STD-490 specification hierarchy, processes are often designated as CPCI's while tasks are designated as CPC's. Thus, the process design phase will produce B5 and preliminary C5 specifications, to be finalized in the preliminary design phase. (The restrictions of the MIL-STD-490 format cause severe confusion between processes and programs, which will be discussed in 5.5.3.) If the specific development program has adopted a "software first" strategy, the process design phase will also produce software-imposed requirements constraining the selection or design of computers and communication hardware.

5.5.2 Content

The traditional activities of process design have been the following:

- Allocation and tracing of software requirements to application processes and the operating system.
- Decomposition and partitioning of application process software requirements into sets of requirements for independently scheduled tasks.
- Definition of scheduling/dispatching criteria, priority structure, inter-task communications, error handling and recovery, overload control mechanisms, and process control structure.
- Definition and control of global data base and refinement of external interface specifications.
- Definition of data management constraints and data access protocols necessary to maintain data base sanity.
- Estimation and allocation of timing and storage budgets for each task, and analysis of port-to-port thread timing, system responsiveness, and overhead.
- Identification and analysis of critical algorithms.
- Maintenance of requirements traceability from the subsystem requirements to the software design and impact analysis of requirements changes.
- Verification that the subsystem requirements will be met if the task timing, sizing, and accuracy requirements are met.

In the future, process design will be increasingly involved with multiprocessor architectures, distributed processing networks, hardware/software/firmware tradeoffs and concurrency issues not found in sequential processors. Advances in LSI and VLSI technology, lower hardware costs, and evolving DP architecture synthesis techniques will offer a wide range of problem-oriented DP system alternatives. The high performance requirements for weapon system applications will demand hierarchical sets of concurrent processes operating in distributed configurations under the supervision of multi-level operating systems. The process designer will be faced with an exponential explosion in the dimensionality of possible design decisions.

A real-time application process includes multiple stimulus-response paths that are activated by events outside the DPS. Thus, the order of demands for processing cannot be controlled by the DP designer. In sequential single processor systems, a major issue is how to partition the application into tasks in a way such that response time delays are distributed fairly across all stimulus-response paths. Partitioning of the application into numerous, rapidly executed tasks permits effective multiplexing of many processing paths on a single processor. However, dispatching overhead becomes significant as a task becomes smaller and response time delay eventually increases due to inefficiency. Thus, there is an optimum size task determined by the particular stimulus-response paths and the load profile on the DPS. Another factor in defining the tasks is that processing on different paths may not be independent (i.e., the paths must be synchronized at some point). Thus, some tasks may be designed to lie across two or more paths for the purpose of synchronization.

In distributed systems the problem becomes more complex, because the number of processors and their interactions must be considered. For a given set of processing paths and load profiles there is a maximum number of processors that can be effectively used (i.e., all stimuli are serviced instantly). As the number of processors are reduced, the load increases and saturation may be reached, indicating the minimum number of processors. There is a tradeoff between cost (number of processors, software difficulty) and other considerations (reliability, availability, threat expansion, system growth, vulnerability). For various numbers of processors and configurations, alternative allocations of stimulus-response paths to processors must be examined and optimum task sizes determined. The processor memory size and optimum task size should be compatible to avoid further complications.

In order to do a meaningful analysis, the process designer must know the structural relationships among the stimulus-response paths and must know the sequences of required processing steps to a level of detail such that reliable estimates of instruction counts and memory needs can be made. In addition, the arrival statistics of the stimuli for each path and the performance requirements on each path must be known. The maximum path execution speed (path instruction count x feasible processor rate) can be compared to the response time requirement to determine the slack time available for dispatching overhead and waiting for resources. At this point, the feasible degree of processing path partitioning can be determined and minimum-competition groups of paths can be identified. If slack time is low or non-existent, the affected path is designated a critical path and algorithm studies are initiated to improve performance.

Another important consideration in the partitioning of processing is the flow of data between processing steps and the access patterns within the system. Generally, processes and tasks should be structured so that inter-process and inter-task transfers are minimized and interfaces are simplified. The necessity for inter-process and inter-task communication leads to requirements for access protocols, process synchronization, and complex control logic in order to preserve data base coherence and sanity, and can severely limit the scheduling flexibility of the system. The added overhead limits available memory and degrades response time.

Arbitrary interfacing of tasks and processes by scattered groups of designers usually results in disaster. Thus, the process design group is responsible for centralized definition of the global data bases and configuration control of the data bases as development proceeds. As part of this effort, the process designers establish the allowable data structures (queues, data sets, etc.) and provide design rules for data access. These considerations lead to requirements for operating system services.

If adequate DP resources are available to support any set of system demands without delay, the scheduling of tasks is automatically accomplished by the external stimuli and by the predecessor-successor relationships among the tasks. Generally, however, DP resources are scarce for practical cost reasons and resource contention must be arbitrated by a scheduling algorithm. For any engagement scenario within the weapon system design load limits there exists a set of schedules of system actions that result in a "successful" engagement outcome. Embedded in each of these schedules is a schedule for necessary DP actions that support the system schedule. These schedules of DP actions are requirements upon the DP scheduling algorithm (i.e., every schedule produced by the algorithm must be a member of the set of "successful" schedules). One job of the process designer is to find an algorithm that meets these requirements.

A major difficulty is that computationally inexpensive techniques for determining the class of "successful" schedules for even a single engagement scenario are non-existent. Operations research investigations have shown that the search for an optimum solution to complex scheduling problems is usually impractical, although some heuristic methods provide near-optimal schedules. Thus, both the requirements for a scheduling algorithm and the formal identification of a satisfactory solution are difficult. Moreover, most optimal or near-optimal algorithms would be expensive to implement in terms of data processing overhead.

The usual approach taken by the process designer is to adopt a simple scheduling discipline (e.g., FIFO with or without preemption) and augment it with a task priority structure. The priority structure is tuned by trial-and-error simulation against representative engagement scenarios until a solution believed to be satisfactory is obtained. This procedure has usually been adequate for most systems, but its success is uncertain for future distributed systems.

When a viable software architecture and supporting design constraints have been identified, the process designer must develop requirement specifications for each of the independently-schedulable tasks. These requirements are either derived from the DPSS requirements or are induced by process design decisions. Since several tasks may participate in the satisfaction of a single DPSS functional or performance requirement, the DPSS requirements must be decomposed and restated so that they are meaningful in the specific process design context and reflect the data processing terms of the task designer. Response time requirements on the DPSS and memory constraints are decomposed in the sense that each task is given a specific execution time and memory budget. Failure to adequately decompose and restate the DPSS requirements results in ambiguity of design responsibility and traceability problems discussed in 5.6.3.

5.5.3 <u>Problems</u>

Process design, as a distinct technical discipline, originated within the Ballistic Missile Defense (BMD) community, and with few exceptions (e.g., OTH-B) has not yet been widely applied outside that problem area. Nonetheless, the principles and techniques of process design are generally applicable and are of significant benefit in the development of complex real-time systems. Unfortunately, the current literature on process design is fragmentary and based upon specific application assumptions. Most of the generalized knowledge is carried in the heads of practitioners and is undocumented. What is needed is a definitive text on the subject, collecting and generalizing existing knowledge to a wider range of application. Such a consolidation is necessary to provide a framework for extensions to distributed processing, and to define the use of recent software requirements engineering products in the process design phase.

Effective process design techniques for single computers have been empirically developed, but basic theory has been somewhat neglected. As we begin to consider distributed systems, however, we realize that even such basic terms as "process" are intuitive notions rather than precisely defined concepts. Until we develop a better understanding of non-trivial processes and concurrent interacting phenomena, the powerful new capabilities of concurrent programming languages (e.g., DoD-1) will not be used to full potential, or may lead to disastrous failures. Basic research is needed to clarify the concepts of real-world and abstract processes in a broader sense than is currently treated, to develop, and analyze systems of interacting concurrent phenomena, and to address problems of multi-process/multiprocessor real-time scheduling, process control, data management, dynamic reconfiguration, deadlock avoidance, and resource management. Without this research the process designer will soon be overwhelmed by the complexities of distributed systems.

Effective methods for describing and specifying processes and concurrent phenomena are sorely needed. The current MIL-STD-490 format for B5 specifications was defined over a decade ago in the world of second-generation computers, and views software as a static collection of programs. A process, however, is a dynamic entity, not simply a collection of application and operating system programs. It is a concept of action and behavior, not just components. Efforts to describe a process within the B5 format have not been entirely successful, because the dynamic qualities are not captured and confusion between processes and programs results.

In a distributed system, the DPSS may be spread across geographically distributed physical sites or nodes. Each node may contain one or more clusters of processors, depending on how a node is defined. Some conceptual processes may involve the activities of only one processor. Others may involve notions of competition for variable numbers of processors. Some processes may span multiple nodes. Others may "migrate" from node to node. Hierarchical structures of processes must be considered. The same locus of action may even belong to different processes simultaneously, according to the viewpoint of the particular analysis. To accommodate various perspectives and additional levels of design, new types of specifications must be considered for complete communication and traceability.

A process supporting a complex weapon system is a finely-tuned simultaneous solution to a large number of software requirements. If the system operating rules and process logic are implemented throughout tasks and low level routines, evolutionary changes in weapon system requirements, even small ones, can lead to extensive redesign of the software. If traceability from requirements to design is weak, staggering costs may be involved just to determine the software affected, as well as to modify it. One approach to avoiding these problems, a technique called "process construction", is explained by Marker [18]. The essence of the concept is centralized control of data and process logic by specific routines or macros. Interestingly enough, the necessary conditions defined by Marker for "process-constructible" software are reflected in at least two Higher Order Software (HOS) axioms.

Even with process construction and modern software requirements engineering, careful attention must be given to traceability during the process design phase. In previous phases, the software is viewed in terms of a problem-oriented description (i.e., "what the software does"). In subsequent phases, after process design has defined the software architecture, the software is viewed in terms of solution-oriented description (i.e., what the software is; how modules, data files interact). The process designer must follow disciplined rules for requirements decomposition to ensure that traceability between requirements, software components, and test procedures can be clearly maintained.

The quality of the process design is heavily dependent on the quality of the DPSS loading profiles, engagement scenarios, and system operating rules input to the process designer. If these are deficient, as they often have been, it is very difficult to validate the process design and ensure that the software architecture conforms to operational needs. More dangerous, the process designer may make assumptions about operating rules to fill in gaps in their definition. These assumptions may seem perfectly reasonable to make the process design more efficient, but may not be consistent with the needs of the weapon system as a whole.

5.6 PRELIMINARY DESIGN

5.6.1 Scope

Preliminary design can be factored into software preliminary design and hardware preliminary design activities. Generally, we are not concerned with the hardware design activity except in the cases where software considerations drive the hardware design or selection, or where significant hardware/software trade-offs are possible.

Software preliminary design can be divided into three components activities: application system preliminary design, operating system preliminary design, and support software preliminary design. Associated with these activities is test engineering, which determines a need for much of the support software.

The output of the process design phase consists of requirements for mission software and support software down to the CPC level. For each software process, the process design defines a set of independently-schedulable software "tasks", a scheduling algorithm that provides proper task synchronization and sequencing, a set of operating system services available to application programs, a global data base structure, and control algorithms that enforce the weapon system operating rules. The preliminary design phase addresses the internal design of tasks within the framework established by the process design.

The primary outputs of the preliminary design phase are designs for tasks, including sets of internal (informal) specifications for routines, evidence that each task can meet its requirements within the context of the process design, and preliminary test plans to demonstrate that the process design structure is testable. These outputs are formally presented and discussed at the Preliminary Design Review (PDR). During this phase, the B5 development specifications are finalized and updates to the preliminary C5 product specifications are generated. The preliminary design phase is ended when all action items from the PDR have been satisfactorily resolved, and the B5 specifications have been baselined.

5.6.2 Content

We will discuss the content of the preliminary design phase separately for each of the major software areas: application, operating system, and support. We will also discuss impacts on test engineering and hardware preliminary design.

5.6.2.1 Application System

Working from response time, accuracy and storage budgets, and a set of requirements to be satisfied by the task, the application task designer must devise a structure of lower-level modules (e.g., routines) that collectively meet the requirements, and a task data base structure that supports task private data and inter-routine transfers. The results of this design activity will also refine the definition of the global data base and will increase confidence that the task budgets can be met.

During preliminary design, problem areas and critical algorithms at the task level will be identified. Prototype code for certain key algorithms may be developed to benchmark alternative designs. The behavior of alternative module structures may be examined by functional simulation or emulation.

The work of the process designer should insulate the task designer from real-time interference problems (e.g., process deadlock, scheduling conflicts, memory access conflicts) and relieve application task designers from needing detailed knowledge of the operating system. Except for response time requirements, a task can be viewed as a stand-alone batch program.

Commonly, a task executes under control of a main program or task control routine. This routine handles all global data input transfers at the start of execution and provides all global data outputs at the end of execution. The structure of a task is usually hierarchical and in concert with structured software design principles.

A task typically performs a specific function (e.g., assimilation of radar returns, correlation of observations with tracks, scheduling of radar pulses). Usually the job to be done can be very well-defined, and the major design problem is to find a near-optimal way of doing the job within the timing, accuracy, and storage budgets.

While task design is still a creative process, many software engineering principles and tools are available to aid the designer. The earliest of these are the traditional flowchart and decision tables. Structured textual descriptions (e.g., Program Design Language) are augmenting graphical techniques. Principles of structure and organization are typified by Parnas' concepts of information hiding [4, 5] and program families [5]; the structured design strategies of Yourdon and Constantine [19]; Higher Order Software [20]; and the Michael Jackson Design Methodology [21]. Although structured programming constructs have proven to be more useful than previous unstructured constructs, new concepts of Functional Programming [22] show promise because they eliminate undesirable consequences of structured programming.

The requirements output of the application preliminary design activity is a set of requirements for each module within each task. These specifications are not deliverable in any MIL-STD-490 document. Hence, they are informal, and their quality is determined by the software engineering standards of the particular contractor. The module requirements may be indirectly reflected in the C5 specification which describes the design of each CPC. However, it is often difficult to separate design details from actual requirements under the current format.

5.6.2.2 <u>Operating System</u>

The most difficult problems in implementing the process design are often the responsibility of the operating system designer. While the application task designer is insulated from the problems posed by hardware and concurrent asynchronous software, the operating system designer is fully exposed to them.

The usual problem of the operating system designer in weapon system applications is to meet the needs of the weapon system application by augmentation of a general purpose commercial operating system, usually provided by the computer hardware vendor. The operating system designer must interface, on one hand, with the existing operating system and, on the other hand, with the application needs as specified by the process designer's requirements for scheduling and control algorithms, and operating system services. Other interfaces are defined by the characteristics of peripheral devices, and much of the operating system designer's job may be concerned with the development of special purpose I/O handlers and device controllers. The operating system designer must also be concerned with problems of interrupt structure, scheduling, task enablement and disablement, resource allocation, timing, deadlock prevention and resolution, data management, error detection, and error recovery. In most weapon systems, these functions are performed by executive or supervisor software operating in conjunction with the vendor-supplied general purpose operating system.

In rare cases, the operating system designer may be faced with the problem of designing an operating system from scratch, to interface an application with a specific set of data processing hardware. Even more rarely, the designer may be asked to design an operating system for an application and develop specifications for the hardware to execute the software. The rapidly dropping cost of hardware, the expanding capabilities of micro-electronics, and better understanding of distributed architectures may make the "software first" approaches more frequent in the future.

The operating system designer has fewer tools to help him than does the application system designer because concurrent concepts are hard to represent. The major issues faced in distributed processing have confronted operating system designers for some time, even in serial machines. Hence, operating system design has been a creative, highly experimental "black art".

The requirements outputs of the operating system preliminary design activity are much the same as those of application design, and are incorporated in B5 and C5 specifications. Sometimes the operating system will be designated as a separate CPCI. Often the system-specific operating system additions will be treated as an executive CPC within a CPCI covering the entire application. Requirements for operating systems have been difficult to write and are often fragmented due to representational difficulties. This does not mean that we should avoid stating requirements for operating systems. Rather, we should intensify efforts to find better representations.

5.6.2.3 Support Software

In addition to the mission software, other deliverable software packages must be provided to support development, operation, and maintenance of a weapon system. These may include software to construct software (e.g., compilers, assemblers, process construction programs, PA tools), software to exercise and test software (e.g., simulators, test drivers, diagnostic packages, data reduction programs) and software to exercise and test the system equipments (e.g., calibration software, system readiness verification tools, performance monitors). These needs must be considered in the DPSS and process designs because they are usually executed in the same hardware under the same operating system as the mission software.

The B5 and preliminary C5 specifications for deliverable support software are developed in the same manner as those for application software. However, the requirements for the support software are usually dependent upon both the requirements and design of the mission software and/or other system elements. Because much of the support software is needed to test the mission software as it becomes available, the support software development is often on the critical path of the schedule. Hence, the needs for support software requirements traceability and change response may be even more critical than those for mission software.

5.6.2.4 Preliminary Hardware Design

Hardware design is not an issue of interest to us unless the software design drives the hardware requirements or the software and hardware designs are being done concurrently with software/hardware trade-offs involved.

In the first case, the hardware designer needs to know characteristics of the software and its operations that form the basis of the hardware requirements. These include the instruction set used by the software, a definition of the operations commanded by these instructions, word sizes, data types, addressing modes, data manipulations, data stream, and instruction stream dimensionality, I/O interfaces, instruction speed, memory access speed, and memory size requirements. While some of these items are easy to describe (e.g., the functional requirements for a one-bit adder are described by a truth table) the operation of a complex system is better described through use of high order hardware description languages (e.g., SMITE).

Concurrent hardware/software design is a risky process justified only in cases of extreme need. Because it is highly interactive, coordination must be frequent, communication must be effective, and requirements traceability and change response needs are extreme. The Flexible Analysis Simulation and Test (FAST) facility concept [23] has been advanced as an exploratory solution to this class of problems.

5.6.2.5 Test Engineering

Among the items to be evaluated at the PDR is the initial version of the software acceptance test plan. This document contains test requirements and acceptance criteria, and information on classes of tests as follows:

- Test purpose
- Software requirements to be demonstrated
- Special software, hardware, and facility configurations to be used
- Generic test input environment and output conditions
- Critical analysis techniques relating test outputs to acceptance criteria.

By the time of Critical Design Review (CDR) the document must be refined to describe the test case structure and identify the following for each test case:

- Requirement to be demonstrated
- Test inputs
- Software and hardware configuration to be used
- Support software to be used
- Major software entities to be exercised by the test
- Test outputs
- Test output analysis method
- Uniquely identified test acceptance criteria.

Preliminary test engineering has been one of the most undisciplined activities in the software development cycle, primarily because of the free text, unstructured presentation of requirements in current specification formats. There has been no rigorous test engineering methodology, and probably there cannot be one without a rigorous requirements engineering methodology. The reason is clear: the ability to define an efficient, workable structure of test cases depends on the ability to identify structural relationships between the requirements being demonstrated.

5.6.3 Problems

Complexity should not be a major problem in preliminary design (except for software with stringent time/accuracy/storage budgets) given that the process design has been carefully done. The major problems are communication of precise requirements, traceability of requirements from higher levels, change response, and validation of the design.

A common practice in conventional software engineering procedure is to allocate software requirements to tasks. The statement of a requirement is generally excerpted verbatum from a B5 specification or some higher-level specification. It is not uncommon for a requirement to be allocated to several tasks, such that each task is to contribute to the total satisfaction of the requirement. Unless the requirement is carefully decomposed and restated as several sub-requirements, the responsibility of each task is ambiguous. Unfortunately, lack of time or manpower is a common excuse for omitting the extra effort of decomposition. Prevention of errors then rests on the degree of communication between the designers of the various tasks. However, one of the reasons for partitioning an application in the first place is to permit independent development without excessive coordination.

When a requirement is changed, the problem is amplified. What is the impact of the change on each task? What is the impact on routines within the task? Without a mechanism to decompose each requirement as necessary, and record the relationships between the requirement, the sub-requirements, and the tasks, the evaluation of change impact is an error-prone and time-consuming process.

A task will often participate in the satisfaction of several higher-level requirements. The means of satisfying these requirements will be further constrained by process design decisions, and assumptions about the behavior of other tasks. A precise and understandable statement of the task requirements must consider the relationships between the higher-level sub-requirements, design decisions, and assumptions. The failure to adequately decompose, trace, and relate requirements leads to inconclusive testing and validation at the routine and task levels. All that can be ascertained is that "the tester thinks that the coder did what the designer thought the process designer intended". Actual testing for satisfaction of requirements must then be deferred to system integration and acceptance tests, where subtle errors are difficult to find.

The major problems of preliminary design are inter-related, and stem from a single cause -- failure to state requirements in a decomposable, unambiguous, traceable, testable, structured representation. Without automated aids, a satisfactory requirements statement consumes substantial time and effort. Even with automated aids, sound requirements development will consume more resources than alloted in the past. However, a new benefit will accrue over the life of a project because testing, modification, and maintenance costs will be reduced.

5.7 CONCLUSIONS

Although many phase-specific issues appear during the course of front-end development, the underlying problems of complexity, communication, validation, traceability, and change response appear in different guises throughout the entire process. This is hardly surprising because the real nature of engineering, whether labeled requirements definition or design, is problem-solving and decision-making according to human thought processes. All human decision making, no matter how abrupt or hasty, seems to involve the following ten steps to some degree:

- Formulate the problem
- Search for key parameters and relationships
- Identify alternative solution candidates
- Predict consequences and side effects of alternatives
- Compare alternatives
- Evaluate sensitivities, uncertainties, risks

- Accept the risk of being wrong
- Make the decision (i.e., select one alternative)
- Accept the negative consequences
- Communicate the decision.

Effective aids to support this process would seem to involve the following for complex problems:

- Reliable methodologies for conceptual decomposition of problems into simpler, tractable sub-problems.
- Computer-maintained data bases for organization and retention of multi-dimensional information beyond the span of simultaneous human contemplation.
- Machine-readable languages for expression of ideas in terms of fundamental concept types: entities, attributes, relationships, and structures.

6.0 CANDIDATE TOOLS, TECHNIQUES, AND INTEGRATION APPROACHES

Given that the problems of the various front-end phases are fundamentally similar (with superficial differences), is there a set of tools and techniques existing today that can be integrated together to attack the major front-end problems of requirements definition and validation? To explore this question, we examined the characteristics of over fifteen systems, developed and/or used by TRW, or reported in the software engineering literature. Of these, nine were selected for further consideration as components of an integrated system. In addition, three current research programs advancing the state-of-the-art, and expected to yield future tools, are identified. Brief descriptions of the tool/technique systems evaluated, and the rationale for selection of the chosen nine are reported in Section 6.1.

In Section 6.2, we correlate the chosen systems with the front-end development phases and examine the applicability of the tools to each phase. In Section 6.3, we summarize the assessment of the tools against capabilities useful in the statement and validation of requirements.

In Section 6.4, we discuss the three approaches considered for integration of the tools. In Section 6.5, we summarize the rationale for the recommended approach.

6.1 TOOLS AND TECHNIQUES

To date, no single set of tools has been developed to support the entire front-end development process. However, many tools and techniques have been independently developed by university researchers to attack portions of the front-end problem (e.g., ISDOS), software organizations using Independent Research and Development (IR&D) funds (e.g., SADT, IORL), software organizations using contract funds to develop tools to solve specific problems (e.g., ALF, PERCAM). Lately, considerable funding has been provided by research-oriented DoD agencies to advance the state-of-the-art in software development -- RADC and BMDATC being the most significant ones.

Three categories of tool/technique systems are discussed in this section: selected systems, other systems, and expected systems. First, a set of selected tools are discussed which satisfy five criteria:

- Non-proprietary -- The systems are available for use by U.S. Government agencies.
- Maturity -- The techniques have reached sufficient maturity to be considered for use in a weapon system development.
- Demonstration -- The techniques have been used on real projects of sufficient size to be realistic.
- Tool availability -- The technique is supported by a computer-based tool.
- Capability -- The systems are unique, or incorporate most of the capabilities of similar systems.

For comparative purposes, a set of systems which have appeared in the software engineering literature, called "other systems", are then discussed and related to those selected. A set of applicable research programs which are expected to lead to additional tools and techniques are also discussed.

6.1.1 Selected Tool/Technique Systems

A review of the non-proprietary tools reported in software engineering literature led to the identification of nine systems which satisfy the five criteria identified above. These are discussed below.

- PERCAM -- For four years, TRW has used a system performance simulator called PERCAM (Performance and Cost Analysis Methodology) to support analysis and planning for U.S. Army tactical missile systems at the Missile Research and Development Command, Huntsville, Alabama. PERCAM was designed for modeling and analysis of combat engagement situations. A system is modeled with an Event Logic Tree (ELT) which describes the engagement functions and decisions within the defense system. A standard library of engagement components is used to define the status of the system, the logic for changing the attacker state as the engagement progresses, and, ultimately, system performance and resource consumption measurements. Using this approach, a system can be modeled initially at a high level and adapted to lower levels of detail as needed. The modular structure and standard library components permit a quick turn-around capability ideal for systems analysis support. PERCAM has been transferred to several organizations. The ELTs can be traced to the system operating rules.
- DP PERCAM -- When PERCAM is augmented to output the number of objects in each state as a function of time, a post-processor is used to calculate critical resource utilization (e.g., radar pulses per second, data processing instructions per second, etc.). This has been used to estimate DP resources of the systems engineering level to perform sensitivity and trade-off analyses.
- SREM -- The Software Requirements Engineering Methodology (SREM) was developed by TRW for the U.S. Army Ballistic Missile Defense Advanced Technology Center (BMDATC). SREM was designed to significantly improve the specification and validation of real-time software requirements for ballistic missile defense systems. Subsequent experience shows that SREM can be applied to broad categories of military sensor, and command and control systems. SREM includes a practical methodology; the Requirements Statement Language (RSL), an extensible, machine-processable language for stating requirements; and the Requirements Engineering and Validation System (REVS), an integrated set of tools for analysis, validation, and simulation of requirements. SREM has been transferred to several organizations.
- CARA -- The pioneer system for machine-analyzable software requirements is the ISDOS system developed by Professor Daniel Teichrow and his group at the University of Michigan. AF/ESD sponsored extensions to ISDOS under the Computer-Aided Requirements Analysis (CARA) program.

Basic CARA facilities include a User Requirements Language (URL) and a User Requirements Analyzer (URA). URA operates on URL statements to produce a number of fixed reports and summaries, including printer graphics in a convenient 8-1/2 x 11 format. Extensions of this approach are being addressed by the CADSAT Program. ISDOS was originally designed to support development of business information systems. Hence, many of the features designed into SREM, such as configuration management and simulation generation, are absent. Current work on CADSAT is aimed at extending the software for the military development environment. TRW has participated in this effort. Under contract to RADC, TRW designed a Consistency Checker to logically validate the CARA data base.

Comparison of SREM and CADSAT reveals that they are complementary, rather than competing systems. The more recent SREM drew heavily from the ISDOS experience, but was designed primarily for generation of real-time software requirements rather than analysis of requirements. SREM design decisions consciously limited the user's ability to express internal DP design concepts, so that the requirements engineer could not over-constrain DP design choices. CARA is far higher in design concepts, and is more suited to support of process design.

- PDL -- Program Design Language (PDL), developed by Caine, Farber, and Gordon, Inc., is supported by extensive documentation and a tool called the PDL Processor. Although labelled a program design language, PDL is really more of a design description language, used to express procedure flows in simple structured English. It supports top-down design in that single-line statements can be expanded to complex procedures in an orderly, cataloged manner. The PDL Processor is basically a text editor with cross-reference capability.

  Output reports are easy to understand and review, and are directed at non-programmers, managers, auditors, and customer personnel. Thus, PDL functions as a design documentation tool and communication medium rather than as a design aid. TRW has been using PDL for two years on a limited basis, and project experience has been favorable.

- PDS -- Texas Instruments has developed a Process Design System (PDS) under contract to U.S. Army BMDATC. PDS is designed to start where SREM finishes, and provides a set of tools for support of process design and software development. PDS incorporates tools for configuration management, library management, simulation control, data collection, and documentation. Models and techniques for monitoring of project costs and schedules are included. Facilities for compilation and process construction are provided. PDS is supported by the PDL2 language (not to be confused with Program Design Language discussed previously). PDL2 is a version of PASCAL extended to support operating system development and vector processing on the TI ASC computer. The PDS objective was complete support of software development from design trade-offs to final code. Early design issues, involving methods for decomposition of requirements and allocation to modules, have been particularly stubborn. Further research is needed to fill gaps between SREM and PDS. While PDS was designed for support of a specific language and computer, it serves as a prototype for more generalized tools.

- ALF -- The Analytic Load Formulator (ALF) was developed by TRW and used extensively on the Systems Technology Project (nee Site Defense Program) and subsequent projects to aid in the process design. It accepts the definition of a set of tasks, including such attributes as loading time, data to be accessed, data access times, execution time distributions, proposed scheduler parameters, and estimates (via analytical queuing analysis) the response time characteristics of the proposed process over a specified domain of arrival rates. This analytical technique has been found to be a cost-effective tool for process design in comparison to the cost and schedule to perform simulation analysis of the response times.

- HOS -- Higher Order Software (HOS) is a formal methodology developed by Charles Stark Draper Labs, and now actively advanced by HOS, Inc. Although it is promoted as a requirements methodology, HOS is actually an approach to decomposing systems and designing modular software structures. The methodology is based upon six axioms which explicitly define hierarchical control, where control implies responsibility, data access rights, and control authority.

  A system described as a tree structure in HOS can be analyzed for consistency on both a static and dynamic basis. A specification language and checker program are currently being developed to automate HOS. Although HOS has no current tools at present, it is sufficiently unique to be included in the list.

- SMITE -- The Software Machine Implementation Tool using Emulation (SMITE) is a higher order computer description language for programming the microprogram components of a diagnostic emulator. Its principle benefit is the ability to define (or modify) a DP architecture, and use this to emulate the execution of applications code to assess the ability of the DP to support the required load. This provides a high visibility approach to performing H/W trade-offs. It can emulate only serial uniprocessors, and has not been used on a real application (although it has been demonstrated for microprocessors, e.g., Z8080).

These nine systems span the front-end system development cycle from system analysis to DP Software/Hardware preliminary design, except for distributed processing design. Each has been in use for system design except HOS and SMITE, and the collection spans the capabilities of other systems in use. These other systems are discussed next.

6.1.2 Other Tool/Technique Systems

Other well-known tools and techniques addressed during the study include the following:

- SADT -- This manual technique for describing systems and software was developed by SofTech, and has been used for a number of years on a variety of projects. It has been discussed in several Software Engineering Conferences, but is not supported by any automated tools. It is similar in nature to CARA (describes a functional hierarchy), but has a rather extensive methodology.

- SAMM -- This computer-based technique will extend and automate the SADT approach with an automated data base and consistency checkers. It is still under development by Boeing for the Air Force as part of the ICAM project. Its capabilities are similar to those of CARA.
- IORL -- The Input-Output Requirements Language was developed by Teledyne Brown Engineering and is used in Huntsville on in-house projects. It is automated on a PDP-11 and is maintained as a proprietary product. It is similar in nature to SAMM.
- SVD -- System Verification Diagrams (SVD) were developed by Computer Sciences Corporation (CSC) as an aid to specification of top level system requirements and design. It is also similar to SAMM and IORL. The extent to which it is automated is not known, and it is proprietary.
- HDS -- The Hierarchical Design System was developed by Stanford Research Inc. (SRI) for the specification and design of software. It was sponsored in part by BMDATC, and is similar to SAMM and IORL, and is proprietary.
- Simulation Languages -- Simulation is known to be an important technique for validating system and software performance. Simulations are generally developed using Procedure Oriented Languages (e.g., SIMULA, SIMSCRIPT, SIMSCRIPT II), using general simulation support packages (e.g., GASP II, GASP IV, SALSIM), or specific problem oriented simulation packages (e.g., COMO is the standard simulation framework for U.S. Army air defense analysis). These simulation facilities were not selected due to their general lack of traceability to the design elements represented.

These techniques are used for the statement of requirements and software design at various places. A large number of similar capabilities exist at other places supported by proprietary software packages which will not be reviewed here. Similarly, the manual techniques for software design (e.g., HIPO Diagrams, Nassi-Schneidermann charts, Top-Down Design, the Michael Jackson Design Methodology, Yourdon's Structured Design) are not addressed.

6.1.3 Research Programs

There are a number of research programs underway which promise the development of new tools and techniques to deal with the front-end system problems. Although not exhaustive, the following projects are significant in terms of scope of effort and unique approaches.

- ARE -- The Axiomatic Requirements Engineering project is being sponsored by Ballistic Missile Defense Advanced Technology Center (BMDATC) to address the front-end problems of specifying the data processing requirements at the systems engineering level. Parallel programs are being funded to TRW, Systems Control Incorporated (SCI), and General Research Corporation (GRC). This research is still in the conceptual stage, and has not yet resulted in the development of computerized tools.

- DDP -- The Distributed Data Processing program is also sponsored by BMDATC to address the problems of selecting distributed processing hardware and specifying and developing distributed software. No tools have yet been developed and demonstrated by the two contractors, GRC and TRW.
- ADPC -- The Advanced Data Processing Concepts program is being funded by BMDATC to address the top level estimation of cost and performance of data processing solutions to BMD problems. It is synergistic with TRW's ARE program in providing a data base to support the more theoretical ARE research approach and using early ARE research results.

6.2 CORRELATION WITH DEVELOPMENT PHASES

The correlation of the front-end development phases against the initially selected set of tools are presented in Table 6.1. As indicated by the legend, a "U" in an intersection of a tool and a development phase signifies the current use of the tool on one or more projects for that development phase.

A "U" was assigned to PERCAM, DP PERCAM, SREM, PDL, and ALF because they are currently in use on several projects by TRW. PDS is in use at the Naval Research Laboratories and by Texas Instruments, although it is not supported by TI as a product, and HOS has no current support tool. CARA and ISDOS are used for design purposes by a large number of companies.

A "P" signifies that a tool is currently potentially useful for a phase, but has not been used on a real project. DP PERCAM is assessed as potentially useful for establishing DP and communications loads during the Requirements Engineering and Distributed Process Design phases. CARA was assessed as potentially useful because of its ability to express hierarchies of functions with inputs and outputs for systems, software requirements, and preliminary design. SMITE was developed to address the impact on overall DP performance of changes to the DP hardware architectures, but has not been used on an actual project to date.

A "C" concepts potentially useful was awarded to HOS for preliminary design due to its six axioms for module definition. Although claims have been made for its application at the system level, its utility has yet to be accepted. Similarly, PDL, PDS, and ALF were awarded "C's" for Distributed Design because their concepts appear to be useful in describing distributed designs.

An "E" was awarded to SREM, PDL, PDS, ALF, and SMITE because their capabilities appear to be extendable to other phases.

6.3 ASSESSMENT OF TOOLS

The existing tools have currently known deficiencies even for claimed applicabilities. Table 6.2 presents an overview of the features which existing tools are claimed to address. The capabilities addressed here are the following:

Table 6.1 Development Phase/Tool Correlation

TOOLS / FRONT END DEVELOPMENT PHASES	PERCAM	DP PERCAM	SREM	CARA-CC	PDL	PDS	ALF	HOS	SMITE
SYSTEMS ANALYSIS	U	U	E	P					
SYSTEMS ENGINEERING	U	U	E	P				C	
DP SUBSYSTEMS ENGINEERING	U	U	E						
SOFTWARE REQUIREMENTS ENGINEERING		P	U	P					
DISTRIBUTED PROCESS DESIGN		P	E	P	C	C	E	C	E
PROCESS DESIGN			E		E	PE	U		P
PRELIMINARY DESIGN			E	U	U	U		C	

LEGEND

U -	CURRENT TOOL USED
P -	CURRENT TOOL POTENTIALLY USEFUL
C -	CONCEPTS POTENTIALLY USEFUL
E -	EXTENSIONS OF CURRENT TOOL POTENTIALLY USEFUL

Table 6.2 Current Tool Capabilities

	FUNCTIONS	PERFORMANCE	CONSISTENCY/ COMPLETENESS	SIMULATION	ALLOCATION/ TRACEABILITY
SYSTEMS ANALYSIS	CARA		CARA	PERCAM	
SYSTEMS ENGINEERING	CARA		CARA	PERCAM	
DP SUBSYSTEMS ENGINEERING	CARA		CARA	DP PERCAM	
SOFTWARE REQUIREMENTS ENGINEERING	SREM CARA	SREM	SREM	SREM	SREM
DISTRIBUTED PROCESS DESIGN	CARA				
PROCESS DESIGN	PDS CARA		ALF PDS	ALF SMITE	
PRELIMINARY DESIGN	CARA PDL PDS HOS	PDS	PDS	PDS	

RADC79-005

- Functions -- CARA is claimed to be able to state function hierarchies at the system and software levels, although techniques are not described for linking the separate data bases. SREM addresses the statement of DP functional requirements, while PDS and PDL address the statement of functions of a software design. No current tool is currently used specifically to state the distributed processing functions, although CARA is claimed to be applicable here also.
- Performance -- SREM is the only tool which is claimed to be able to state testable performance requirements for any phase of requirements development. PDS allows for the statement of software budgets for procedures, while CARA and PDL allow such statements in a textual format.
- Consistency/Completeness Checking -- SREM contains a fairly complete set of tools for static checking for various types of completeness and consistency. PDS allows the user to request a set of cross-reference tables to be used to perform such analyses. CARA has added that static checking to CARA, but because SREM allows the definition of precedence, its consistency checking is much more complete than that of CARA; however, even SREM does not include consistency checking of processing of legal sequences of messages.
- Simulation facilities are provided by PERCAM, DP PERCAM, SREM, PDS, and ALF. Traditionally, stand-alone problem-oriented simulators have been used; only SREM, PDS, and ALF tie the simulation to specific design or requirements statements. SMITE provides a simulation of a given DP architecture.
- Representation of the allocation of requirements to design elements and its traceability is represented only in SREM (originating requirements to functions and performance), and ALF (task budgets). Traceability capabilities exist between levels only in terms of stand-alone capabilities.

This table is the source for several observations. First, there are a number of current deficiencies: the statement of performance requirements for systems and distributed systems, the traceability between requirements and design, and consistency checking of designs are not well addressed. Second, except for SREM and PDS, simulations are not well tied to the requirements or design; hence, the statement of requirements and design generally proceed independently of their validation via simulation. Finally, no tool can be used through the various phases of the system development. CARA comes the closest, but it does not provide a traceable link from one phase to another. These conclusions hold true for the other tools examined as well.

Another significant feature of these techniques is the availability of a specific methodology to obtain maximum effect from the usage of the tools. At the current time, ALF, CARA, PDL, and SMITE do not have documented methodologies for their use. CARA particularly has been subject to severe criticism due to this fact. On the other hand, SREM, PDS, PERCAM, and DP PERCAM have methodologies documented to some degree. The availability of a methodology has a profound impact on the ability to transfer technology effectively.

6.4 INTEGRATION APPROACHES

There are various approaches for combining the capabilities of a set of tools and techniques to address the front-end problems. Even after an underlying methodology is developed for addressing all phases of the front-end development, and the role of each tool is identified, there remains the problem of how these tools are to be tied together operationally. The critical problem to be addressed is how the information resident in one tool is to be translated to be available for use in the next tool to be used (e.g., if CARA is used to state requirements for a system, and PERCAM is used to simulate the system performance, information should be translated between the two). Three approaches are considered for accomplishing this translation: user translation, automated aids, and full integration. Each approach is discussed below.

6.4.1 Manual Translation

The cheapest and least desirable approach is for an analyst to use the tools in a stand-alone fashion, with the analyst providing the translation capabilities. This approach is undersirable for a number of reasons.

- Efficiency -- A great deal of effort and time may be required to accomplish the translation, and the translation effort may become a bottleneck. This is particularly true of the development of large scale systems, where the software specifications may involve the statement of 1,000 to 10,000 separate requirements.
- Training -- Training analysts to use a number of different tools and to become proficient in their separate idiosyncracies and usage can present a large initial start-up training effort.
- Reliability/Traceability -- When translation is handled manually, the reliability of the translation is subject to question; moreover, the additional effort to provide traceability (particularly for system modifications) can become prohibitive.
- Completeness -- As previously described, the current set of tools and techniques do not contain all of the required capabilities. Thus, use of current stand-alone capabilities would still require the development of additional tools and techniques.
- Methodology -- A significant existing problem is that there is no underlying methodology for tying current capabilities together, and some capabilities have no documented methodology. Thus, even if existing tools were used in a stand-alone fashion, a significant effort would be required to develop and validate an integrated methodology to use them effectively.

For these and other reasons, some automated mechanism is desirable for tying the tools together in an integrated framework. The use of ad-hoc automated translators is discussed next.

6.4.2 Ad-Hoc Translators

When a substantial effort has already been invested in a set of existing tools, a cost-effective solution to integrate these tools sometimes lies in the area of the development of a set of ad-hoc translators which leave the tools invariant. In some cases, this approach is simply not possible. For example:

- Augmentation of CARA to provide a simulation capability has been found to require an extensive modification of its basic concepts of stating requirements in terms of elements, attributes, and relationships (these cannot easily express the notions of parallelism and precedence necessary for simulation).
- Translators alone cannot provide the required traceability linkages between the different levels of requirements and design. For example, CARA states a design in terms of a design hierarchy; traceability between the hierarchy of system functions (e.g., tracking, discrimination) and the hierarchy of processing functions (e.g., radar returns processing is used for both tracking and discrimination) requires that both be expressed in the <u>same</u> data base.

This approach does address the problems of efficiency and reliability/traceability, but leaves the more fundamental problems of training, completeness, and methodology unaddressed. Thus, the automated translator route is unsatisfactory, and cost advantages of using current tools must be compared to its deficiencies. The integrated tool approach is addressed next.

6.4.3 Common Tool Approach

The advantages of a common tool in terms of efficiency of training and operational usage, traceability, and reliability are obvious. The largest problems for devising a common tool for the front-end system development lie in the problems of feasibility, extensibility, cost, and methodology development. Such questions as the following need answers.

- Is enough known about the problem to devise a comprehensive tool?
- Can additional facets of the problem be easily incorporated into the tool at a later date?
- Are the development costs for such a tool prohibitive?
- Would the cost of using a comprehensive tool in an operational environment be excessive over those of separate stand-alone tools?
- Would the additional capability of the comprehensive tool be worth the additional cost over current capabilities?
- Will the knowledge gained on current tools be lost in the transition?
- What is the risk of developing such a comprehensive tool in terms of cost, schedule, and even risk of completing it at all?
- How can such a tool be transitioned into an operational environment?

The questions of feasibility and extensibility can best be addressed by the identification of a common underlying structure of the front-end system development discussed in the Final Report in some detail: that the front-end development process can be described in terms of concepts of:

- Functions (or transformations)
- Inputs/outputs of functions
- Sequences/parallelism of functions and inputs/outputs
- Decomposition of functions
- Decomposition of inputs/outputs
- Performance of functions
- Allocations of functions to subsystems
- Projection of inputs/outputs to a common interface
- Traceable simulations obtained by mapping functions onto simulation procedures (e.g., as accomplished by REVS and PDS).
- Recording of decisions and alternatives evaluated.

Because of this underlying structure, it is possible to express all of these concepts in terms of language with a simple meta-language consisting of only:

- Elements (the "nouns" of the language, e.g., DATA, FUNCTION).
- Attributes (the "adjectives" of the language, e.g., DATA has UNITS and TYPE).
- Relationships (the "verbs" of the language, e.g., FUNCTION INPUTS DATA).
- Structures to express parallelism, sequentiality, and precedence relationships in a compact form (e.g., first A, then B and C or D).

It is noted that all of the Requirements Statement Language of REVS is stated in terms of these concepts, while CARA concepts are stated in terms of the elements, attributes, and relationships alone. Moreover, the current utility of SREM and CARA, and the research results of the ARE and DDP programs suggest that such a meta-language is sufficient for the statement of all levels of functional and performance requirements, design allocations, and traceability. Experience with SREM, ALF, and PDS, and research results of ARE suggest that simulations can be generally developed from a statement of requirements and design which include the structural components expressing parallelism, precedence, and inputs/outputs.

Current experience with SREM and CARA-CC, and the research results of ARE, suggest that much consistency checking can be accomplished using only the element/attribute/relationship features, while others require structures of precedence relationship.

This covers the entire range of capabilities presented previously in Table 6.2. Thus, all currently envisioned capabilities of such an integrated tool appear to be possible if based on a common meta-language.

The structure of such a common tool, and its feasibility, can be discussed in relations to the current organization of REVS presented in Figure 6-1. Currently, all user input (except for direct graphics input) is routed through the REVS Executive. REVS supports an extensible RSL language by allowing additional elements, attributes, and/or relationships to be added via the RSL Extension Translation function. These extensions are added to the REVS data base (ASSM), and are used by the RSL Translation function and the generalized query capabilities of the Requirements Analysis and Data Extraction (RADX) function. The Simulation Generation facilities access this data base to create simulators coded in PASCAL.

To implement a generalized front-end tool, this same structure would be possible with the following augmentations:

- The current structure segments are fixed; some mechanism is needed for adding new types (e.g., add new structures to the translator, or add the capability of structures of user-defined structures to the Extension Translator).
- Extend the structure checking capabilities of RADX to check the new structures for completeness and consistency.
- Modify the interactive R-Net Generation function to generate the new structures and check them for the new structural rules.
- Modify the simulation generation function to utilize the new structures.
- Modify the REVS Executive to recognize new components (e.g., a systems level simulation generator for PERCAM).
- Add additional tools to accomplish specific analyses (e.g., ALF).
- Add additional tools to compare data bases for configuration management.

To avoid the creation of an unwieldy data base, the output capabilities of RADX could be used to extract the relevant portions of a systems level data base to allow the definition of the software requirements; and a subsequent extraction of the end requirements to allow definition of traceability and the software design. This process is illustrated in Figure 6-2. A compare capability is shown to enable the comparison of two data bases to identify differences -- necessary for configuration control.

6.5 ASSESSMENT

Assessment of the previous discussion leads to the following conclusions:

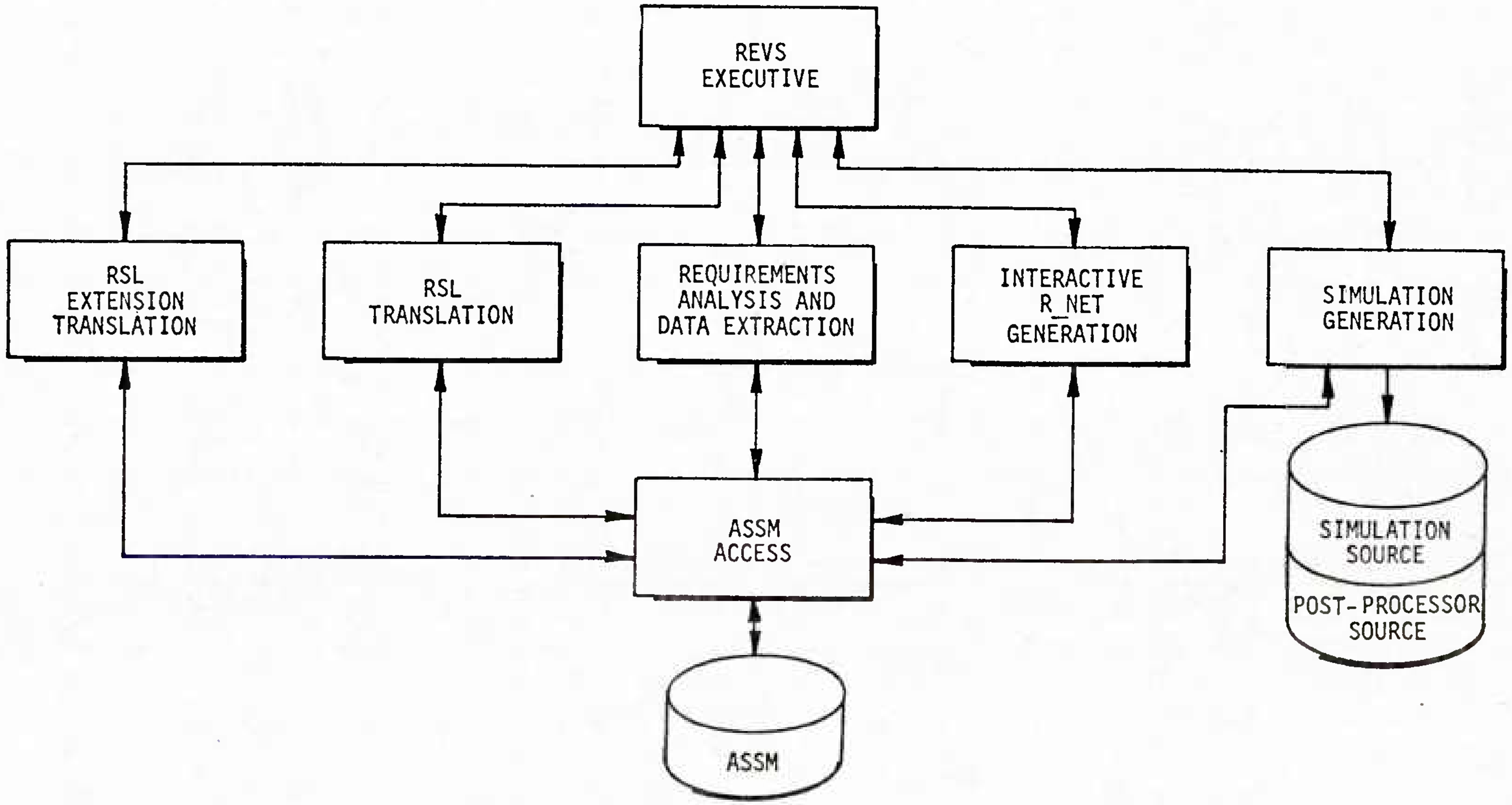


Figure 6-1 REVS Functional Overview

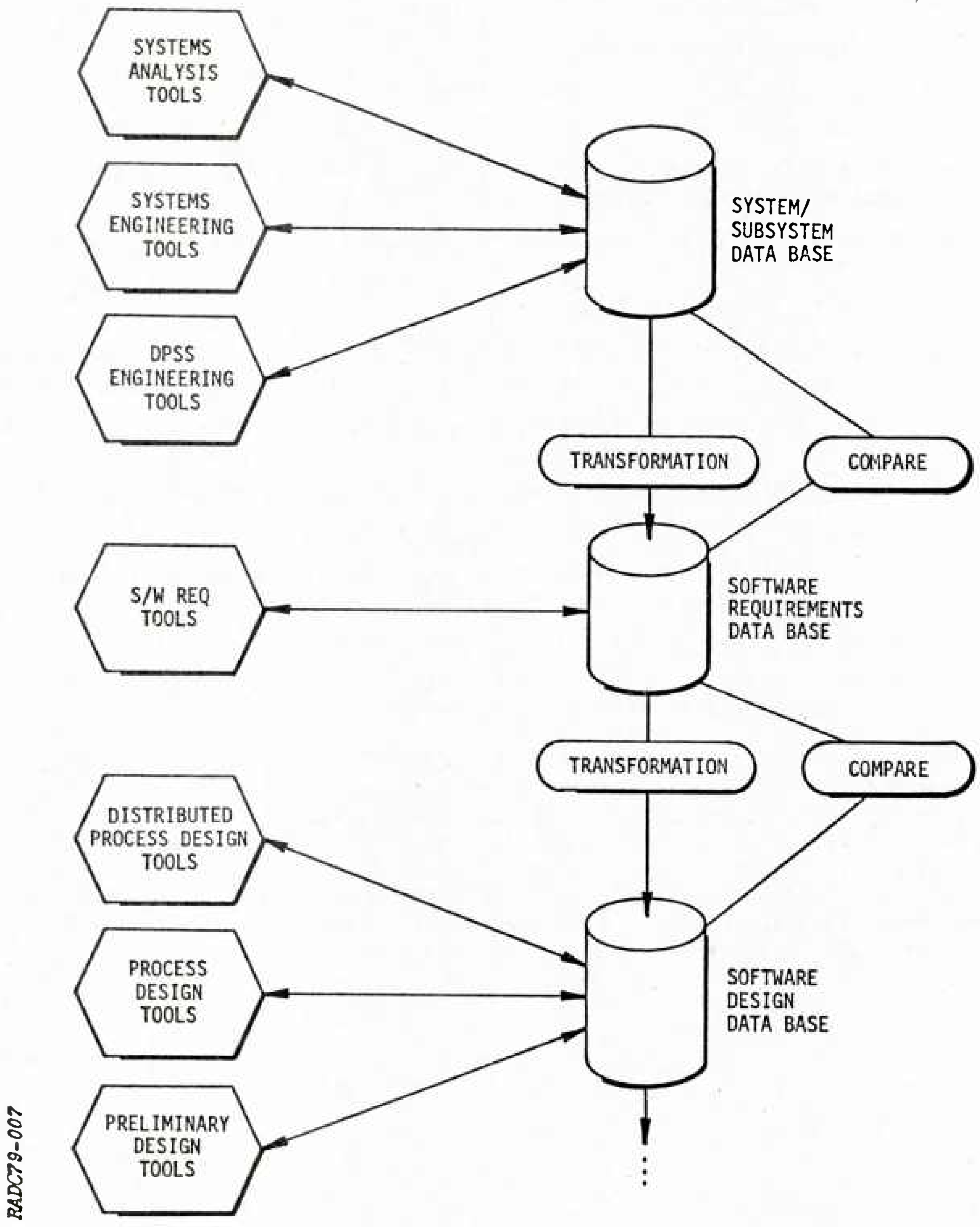


Figure 6-2 Conceptual Development System Structure

- Development of a common tool based on REVS is
  - Based on an underlying methodological structure
  - Feasible
  - Extensible and flexible
  - Low risk (based on current REVS design)
  - Comprehensive.
- All but the structure and simulation facilities are immediately available via the extension capabilities of RSL today.
- Development of such a tool could take advantage of current experience using
  - SREM (because it is based on REVS)
  - CARA (because of an overlap of meta-languages)
  - PERCAM (simulation obtained by translations of the system logic)
  - DP PERCAM (simulation post-processor based on translation of the system subfunctions)
  - PDL (RADX can provide a more extensive data extraction capability)
  - PDS (RADX could extend the completeness/consistency checking)
  - ALF (RADX could provide the ability to tie the design to the PDS implementation)
  - HOS (concepts useful for software design)
  - SMITE (used as an off-line analysis tool to estimate resource requirements for the preliminary design).

Because of the existing structure of REVS, the approach of using REVS as the baseline for augmentations appears to be cost-effective for adding new tools or for using automated translators: RSL/REVS appears to provide an existing structure for low cost augmentations, while RADX may well be sufficient as a base for the automated translators. The advantages in efficiency, training, and reliability are obvious. A recommended approach for development of the integrated set of tools is presented in a separate evolutionary development plan, CDRL Sequence No. A002 of this contract [24].

7.0 REFERENCES

1. Graham, Ronald L., "The Combinatorial Mathematics of Scheduling", Scientific American, Volume 238, No. 3, March 1978, pp. 124-132.

2. Bell, T.E. and T.A. Thayer, "Software Requirements: Are They Really a Problem?", Proceedings of the Second International Conference on Software Engineering, October 1976, San Francisco, Ca. (IEEE Catalog No. 76CH1125-4C).

3. Pippenger, Nicholas, "Complexity Theory", Scientific American, Volume 238, No. 6, June 1978, pp. 114-124.

4. Parnas, L.D., "On The Criteria Used for Decomposing Systems into Modules", Communications of the ACM, Volume 15, pp. 1053-1059, December 1972.

5. Parnas, L.D., "On the Design and Development of Program Families", IEEE Transactions on Software Engineering, Volume SE-2, No. 1, March 1976.

6. Couger, J.D., "Evolution of Business System Analysis Techniques", ACM Computing Surveys, Sept. 1973, pp. 167-198.

7. Rudwick, B.H., "System Analysis for Effective Planning: Principles and Cases", John Wiley & Sons, New York, 1969.

8. Quade, E.S. and W.I. Boucher, eds., "Systems Analysis and Policy Planning -- Applications in Defense", American Elsevier, New York, 1968.

9. Fisher, G.H., "Cost Considerations in Systems Analysis", American Elsevier, New York, 1971.

10. Boehm, B.W., "Software and Its Impact: A Quantitative Assessment", Datamation, May 1973, pp. 48-59.

11. Davis, C.G. and C.R. Vick, "The Software Development System", IEEE Transactions on Software Engineering, Volume SE-3, No. 1, January 1977, pp. 69-84.

12. "DoD Weapon Systems Software Management Study", The Johns Hopkins Univ. Applied Physics Lab, May 1975.

13. "DoD Weapons Systems Software Acquisition and Management Study", Volume 1, MITRE Findings and Recommendations, the MITRE Corp., May 1975.

14. BMDATC Specification Technology Evaluation Program (STEP) Panel, 1976.

15. Salwin, A.E., "A Test Case Comparison of URL/URA and RSL/REVS", Fleet Systems Dept., The Johns Hopkins Univ. Applied Physics Lab., July 1977.

16. "Technical Report - SREM Evaluation Results Summary", Report MDC-G7750, McDonnell Douglas Astronautics Co. - West, October 1978.

17. Alford, M.W., "Software Requirements Engineering Methodology (SREM) at the Age of Two", Proceedings - IEEE COMPSAC78, Chicago, Ill., November 1978.

18. Marker, L.R., "Process Construction - An Overview", Proceedings - IEEE COMPSAC78, Chicago, Ill., November 1978.

19. Yourdon, E., and L.L. Constantine, "Structured Design", Yourdon Inc., New York, 1976.

20. Hamilton, M., and S. Zeldin, "Higher Order Software - A Methodology for Defining Software:, IEEE Transactions on Software Engineering, Volume SE-2, No. 1, March 1976, pp. 9-32.

21. Jackson, M.A., "Principles of Program Design", Academic Press, New York, 1975.

22. Brown, J.R., "Functional Programming - Final Technical Report", TRW Defense and Space Systems Group, Redondo Beach, Ca., July 1977.

23. McClean, R.K., and B. Press, "The Flexible Analysis, Simulation and Test Facility: Diagnostic Emulation", TRW Software Series, TRW-SS-75-03, October 1975.

24. "Evolutionary Development Plan for an Integrated Requirements Engineering System", TRW Report No. 32697-6921-001, TRW DSSG, Huntsville, Alabama, February 1979.

PART III

1.0 INTRODUCTION

This portion of the report addresses the description of an overall unified methodology to address the system development front-end problems described in the previous section. The approach is based upon formal foundations developed under the Axiomatic Requirements Engineering (ARE) program supported by the Ballistic Missile Defense Advanced Technology Center (BMDATC) in Huntsville, Alabama. The key to this approach is a formal definition of decomposition and allocation, and definition of a framework in which systems analysis and design decision can be described and compared for different decompositions. This leads to the definition of a unified methodology to be applied from the front-end needs analysis down to the definition of the software design for each of the data processors. This in turn leads to the identification that a common metalanguage is possible which can be used to define languages for the precise statement of requirements at each level of the hierarchy, thereby providing the basis for a common set of methods and procedures.

The purpose of this report is not to define the completed methodology (that is beyond the scope of the effort); the purpose is to identify the formal foundations and to outline the features of the methodology which should be developed in detail.

The purpose of a methodology is not to provide a mechanical set of steps which, if followed, will result automatically in an optimal end product; no matter how good, a methodology cannot make up for deficiencies in engineering. It is commonly agreed that any design effort is comprised of one percent inspiration and 99 percent perspiration; it is the purpose of the methodology to assure that the inspiration is not drowned by the perspiration. In other words, the purpose of the methodology is to identify the steps which should be passed on the way to the end product, and the properties of the intermediate and final milestones which should be reached in order to have an acceptable end product. This provides the foundations from which requirements for automated tools are developed to aid in the requirements and design process.

1.1 BACKGROUND

Before defining a new methodology for systems engineering and design for the front-end of system development, it is useful to review the systems engineering literature to determine current deficiencies. Table 1.1 presents a set of the systems engineering and software engineering methodologies and their authors. Table 1.2 presents a comparative overview of the major steps of these methodologies, showing rough equivalence of overall contents and sequences of steps, although different in the individual step definitions. A more detailed review leads to the conclusion that all of these methodologies lack a formal definition of a consistent set of system properties, and tools to address those properties. All of the methodologies offer only generalized procedures for accomplishing the methodology steps (e.g., none offer specific procedures for checking the consistency of a decomposition). None offered the identification of specific steps of the development process. This result

Table 1.1 Representative Methodology Frameworks and Techniques

● SOME SYSTEM ENGINEERING METHODOLOGIES	
- SYSTEM DESIGN PROCESS	LIFSON, KLINE
- PROBLEM SOLVING	HALL
- DESIGN PROCESS	ASIMOW
- ANATOMY OF DESIGN	ROSENSTEIN, ENGLISH
- DESIGN PROCESS	GOSLING
- ENGINEERING DESIGN PROCESS	ALGER, HAYS
- SYSTEM ENGINEERING PROCESS	AFFEL
- PHASES OF OPERATIONS RESEARCH	CHURCHMAN, ACKOFF, ARNOFF
● SOME SOFTWARE ENGINEERING METHODOLOGIES	
- SUCCESSIVE REFINEMENT	DIJKSTRA
- PROGRAM DESIGN	JACKSON
- STRUCTURED PROGRAMMING	DAHL, DIJKSTRA, HOARE
- S/W DEVELOPMENT SYSTEM	DAVIS/VICK

RADC79-038

Table 1.2 Representative System Engineering Methodologies

	Lifson, Kline	Hall	Asimow	Rosenstein, English	Gosling	Alger, Hays	Affel	Churchman, Ackoff, Arnoff
MIL STD	System design process	Problem solving	Design process	Anatomy of design	Design process	Engineering design process	System engineering process	Phases of operations research
FUNCTION ANALYSIS	Information gathering and organizing (need)			Information collection and organization; identification of need	Description of input and environment	Recognizing		
	Formulation of value model	Problem definition, selecting objectives	Analysis of problem situation	Identification of system variables; criteria development	Measure of value (system worth)	Specifying	Problem definition; selecting system criteria	Formulating the problem
SYNTHESIS	Synthesis of alternative solutions	Systems synthesis	Synthesis of solution	Synthesis	Formation of system models	Proposing of solutions	System synthesis	Constructing mathematical model
	Analysis and/or test	Systems analysis					System analysis	Deriving solution from model
EVALUATION & DECISION	Evaluation	Selecting the best system	Evaluation and decision	Test and evaluation	Realization	Evaluating alternatives		Testing model and solution derived from it
	Decision			Decision	Optimization	Deciding on solution		Establishing controls over solution
DESCRIPTION	Optimization (iteration)		Optimization (revision)	Optimizing (iteration)				
	Communication	Communicating results	Implementation	Communication and implementation	Description of outputs	Implementing	Implementation	Putting solution to work; implementation

is typical of the "heuristic" methodologies, including the MIL-STD Systems Analysis techniques.

At the other end of the spectrum are the mathematical descriptions of General Systems Theory. Mesarovic *et al.* [1, 2] is typical of attempts to describe general systems properties based on mathematical formalism; unfortunately, these attempts are too limited to be successful. For example, Mesarovic [1] addresses the formal description of one system decomposition at one level -- this is insufficient to address multiple possible designs to meet a set of requirements for a complex system ultimately composed of a large (e.g., 100,000) number of parts containing complex data processing functions.

These conclusions (lack of a consistent set of formally defined properties, lack of unified tools) hold true at all levels of the front-end of system design, from Systems Analysis to Data Processing Engineering to process design.

1.2 OUR APPROACH

Our approach to defining a unified methodology and supporting tools is illustrated in Figure 1-1. First, the underlying formal foundations are identified, a methodology based on those foundations is described, the methodology is demonstrated on example problems, and then tools to support the methodology are developed and demonstrated.

Section 2.0 provides the formal foundations in terms of definitions of decomposition and allocation to subsystems. Section 3.0 provides an overview of a methodology based on these foundations. First, a generic methodology is provided, and then it is particularized to the design of data processing subsystems to the software preliminary design level. These concepts are solidified using examples from a preliminary analysis of a strategic surveillance system. Section 4.0 presents conclusions. Section 5.0 presents the references for Part III of this report.

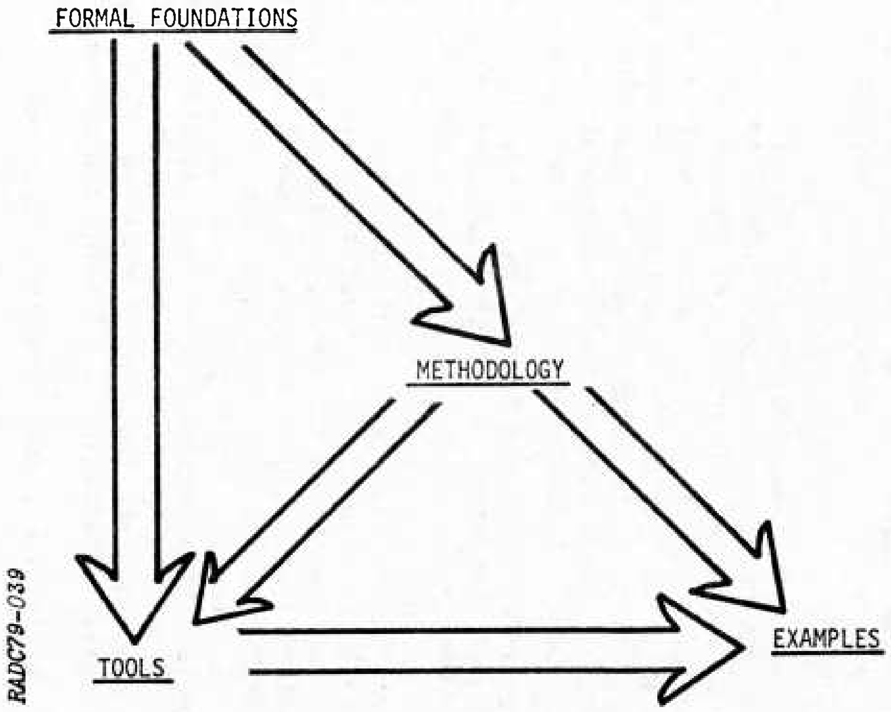


Figure 1-1 Overall Approach

2.0 FORMAL FOUNDATIONS

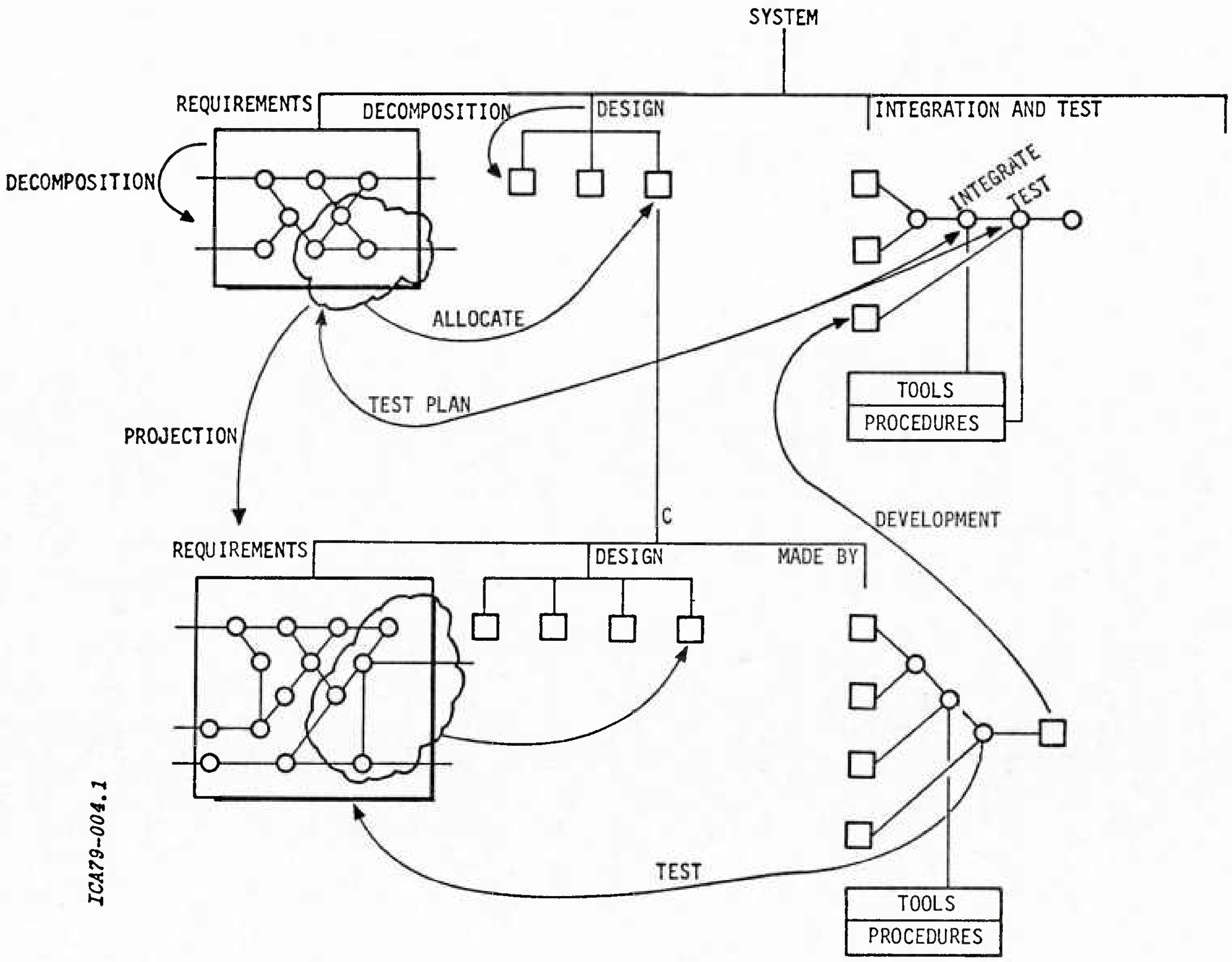
2.1 OVERVIEW

A system is viewed in many different ways during the system development process. Figure 2-1 presents three specific views and their relationships:

- The system requirements (i.e., what the system does). This is described at various levels of detail, from a system mission level (e.g., save the world for democracy), and at more detailed levels (e.g., input message A). These levels of detail are usually expressed hierarchically in terms of "decompositions" of one level into another.
- The system design, i.e., the physical pieces of the system, variously called subsystems, critical items, assemblies, or parts. The system can thus be viewed as a hierarchy of these elements terminating at the bottom of a set of "parts". At any level of this hierarchy, there is an allocation of the requirements of what the overall element (or system) is to do in terms of what the sub-elements (or subsystems) are to do. The data base of such requirements can get very large (e.g., 100,000 to 1,000,000 parts for an aerospace system).
- The system integration and test plans, i.e., the system is not a collection of subsystems, it is an integrated collection of subsystems. Resources (e.g., time, cost) are required to integrate the sub-elements into the overall element (e.g., bolt them together) and to verify that the overall element requirements are satisfied (i.e., test). An integral part of the system's design process is the identification of how the parts are assembled and tested. Note that the overall element is tested against the element specification, not the sub-element specification. Note also that, in order to test the element, test tools and test procedures must be developed. The cost and schedule of the development is thus calculated from the cost and schedule of developing the sub-elements, plus the cost and schedule of developing the test tools and procedures, plus the cost and schedule of actually integrating and testing the element from sub-elements.

These three views of a system should have the properties:

- The detailed requirements at one level should be "decompositions" of the initial requirements at that level.
- The allocation of requirements at a level onto sub-elements should be unique.
- The test plan should identify the sequence and manner of verifying that each of the system actions are accomplished by the cooperating sub-elements.
- The integration and test plan at one level should be a "decomposition" of that of the previous level.



ICA79-004.1

Figure 2-1 Three Views of a System

Note that four concepts are central to this discussion:

- The concept of a function (of a system, of a subsystem, or of the integration and test process).
- The concept of sequences of functions (of system actions, of tests).
- The concept of decomposition, i.e., of describing a function in terms of a collection of more "detailed" functions.
- The concept of allocation (of functions to pieces and tests).

To achieve the definition of system properties and relationships, these concepts must be formally defined.

Note further that these views of a system are not static, i.e., that decomposition and allocation are non-unique mappings. Thus, changes in the system requirements ripple into changes in the design, and into changes in the integration and test plan. Similarly, if a part cannot be fabricated for a specific cost and schedule, an alternate design may be necessary. During the life of the system, the whole information structure of the system is in a continuous state of controlled change -- continuous change in response to changes in requirements, design, or testing capabilities, but controlled and managed to achieve specific ends (i.e., delivery of systems with agreed-to performance, cost and schedule).

Formalizing the above, we have the following definition: A system set S is a five-tupal, $S = (R, \hat{D}, T, W, Z)$ where

R = A set of requirements for system actions.

$\hat{D}$ = A set of design elements $(SS_1, \ldots, SS_n)$ each of which is a system, and a description of the environment E.

T = An integration test and plan.

W = A set of estimating relationships.

Z = A set of preference rules for comparing systems.

Each of these will now be examined in turn.

2.2 SYSTEM FUNCTIONS

We start with the definition of structures of data identifiers, systems, and then formally define the notion of decomposition.

Definition 1: (SDT). A Structured Data Tree (SDT) is a triple (S, d, T) where T is a tree with nodes which are identifiers from the set, S, and d is a function on the nodes of the tree such that d maps nonleaf nodes into (+,&,*,@), where:

+ indicates that exactly one of the subtrees are included.

& indicates that all subtrees are included in parallel.

* indicates that the subtrees are replicated some number of times.

@ indicates that the subtrees are included in a left-to-right sequence.

Remark. This notation is a variation of Jackson's [3] terminology which will prove useful later. Figure 2-2 gives a graphical representation of an SDT with the following interpretation. The identifier A is composed of the identifiers B, C, and D in that sequence. B consists of either B_1 or B_2. D consists of two parallel streams, E and F, where E consists of a sequence E_1, E_2, ..., and F consists of the sequence F_1, F_2, ... From this, the tree, T, the identifier set, S, and the mapping, D, can be constructed for the SDT, with root identifier, A. In general, the name of the SDT will be the same as the name for the identifier of the root node.

The structure of identifiers is used to represent inputs/outputs, system parameters, and system performance indices of a system. These identifiers may themselves have values, or may represent a subtree of other identifiers. This concept will prove useful in providing structure to the inputs and outputs of a system.

Definition 2 (Function). A system function, F, is a six-tupal, F = (I, O, U, P, D, C), where

I = an SDT of inputs.

O = an SDT of outputs.

U = an SDT of system parameters.

P = an SDT of performance parameters.

D = the definition of a transformation, D: $(I, U) \rightarrow (O, P)$

C = a completion condition.

Remark. A system function, F, is viewed here as a "black box" which has inputs I and outputs O. The input is assumed to contain any relevant environment parameters, e.g., rain. The system function is viewed as incompletely defined until the system parameter set, U, is specified: thus F can be viewed as a family of functions, where the selection of U will result in the selection of exactly one transformation of inputs and environment into the function system outputs and performance, P. The transformation will continue until a completion criterion, C, is satisfied.

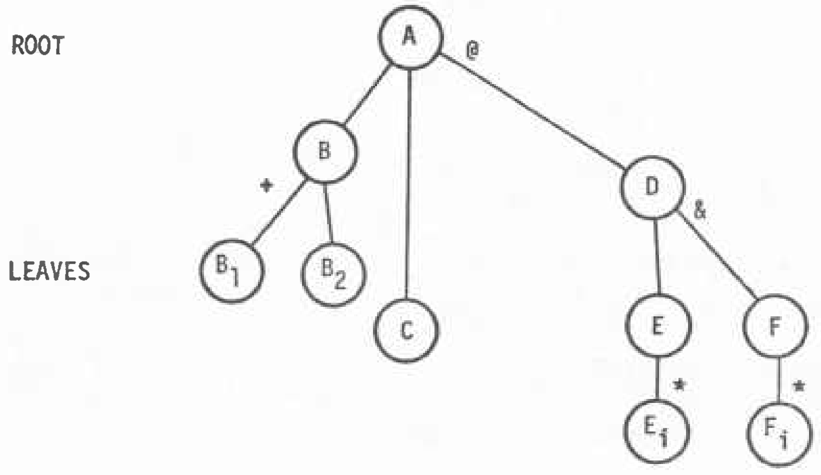


Figure 2-2 An Example SDT

For example, we may describe a system function for detection of an object. The input, I, might consist of the real object position and radar cross-section. The environment, e, might represent the rain rate in inches-per-hour which will tend to attenuate the radar detection capability. The performance, P, might be the probability of detection, while the system parameters might be the radar power-aperture product and receiver noise level. From these system parameters, object location, and rain rate, the probability of detection can be estimated analytically.

2.3 COMPOSITION

The concept of decomposition involves the notion of one function being described in terms of a number of other functions; in other words, a number of functions are "composed" into another function, and it is this function which may have a decomposition relationship with the original function. The manner of this composition is defined precisely below.

Definition 3 (GMF). Let $F_0, F_1, \ldots, F_n$ be functions, where F_0 is an external node, and $F_i = (I_i, O_i, U_i, P_i, D_i, C_i)$ $i = 1, \ldots, n$. Let G be a directed graph with nodes F_i and edges E_{ij} such that:

1) Edge E_{jk} connects two nodes, i.e., $E_{jk} = (F_j, F_k)$ for some j, k.

2) There exists mappings B_I and B_O with

$$B_I : F_i \rightarrow \begin{cases} + & \text{indicating the function } F_i \text{ is initiated by activating any of the input edges connected to it.} \\ * & \text{indicating the function } F_i \text{ is initiated when all of the input edges connected to it are activated.} \end{cases}$$

3)

$$B_O : F_i \rightarrow \begin{cases} + & \text{indicating } F_i \text{ causes activation of exactly one edge when } C_i \text{ is satisfied.} \\ \& & \text{indicating } F_i \text{ causes activation of all output edges when } C_i \text{ is satisfied.} \end{cases}$$

4) If $B_O : F_i \rightarrow +$, then C_i maps (I, U) onto j, indicating that edge E_{ij} is to be activated.

Remark. This definition is an adaptation of the definition of a Graph Model of Computation in Computer Science (e.g., see [4]). The graph consists of edges which define precedence relationships among functions. The concept of precedence or sequence is defined in the following ways. Assume $E_{ij} = (F_i, F_j)$.

Then when C_i is satisfied, one of the following is the case:

- If $B_0 : F_i \rightarrow *$ and $B_I : F_j \rightarrow +$, F_j will be immediately initiated.
- If $B_0 : F_i \rightarrow +$ and $B_I : F_j \rightarrow +$, then F_j may be initiated if that edge is selected by C_i.
- If $B_0 : F_i \rightarrow *$, $B_I : F_j \rightarrow *$, then F_j will be activated if, and only if, F_i selects edge E_{ij} and all other edges to F_j are activated.
- If $B_0 : F_i \rightarrow +$, $B_I : F_j \rightarrow *$, then F_j will be activated if, and only if, F_i selects edge E_{ij} and all other edges to F_j are activated.

This allows the synchronization of functions to be specified. Because of the close similarity of the GMF and GMC, many results of the GMC (e.g., liveness) can be used without modification; these will not be discussed here.

Example. Figure 2-3 illustrates a GMF. Note that G has two input edges and three output edges. It can be initiated by edges E_1 or E_2, leading to F_3. After F_3, all of F_4, F_5, F_6, and edge E_6 are activated. When all of F_4, F_5, and F_6 are completed, F_7 is initiated, leading to activation of F_3, or edges E_{11} or E_{12}.

Definition 4 (Composition). Let G be a GMF over nodes $F_1, \ldots, F_n$, with a single entry. Define $F_o = (I_o, O_o, U_o, P_o, D_o, C_o)$ where

$I_o = G(I_i) - G(O_i)$.

$O_o \subseteq G(O_i)$.

$U_o = \bigcup_i U_i$.

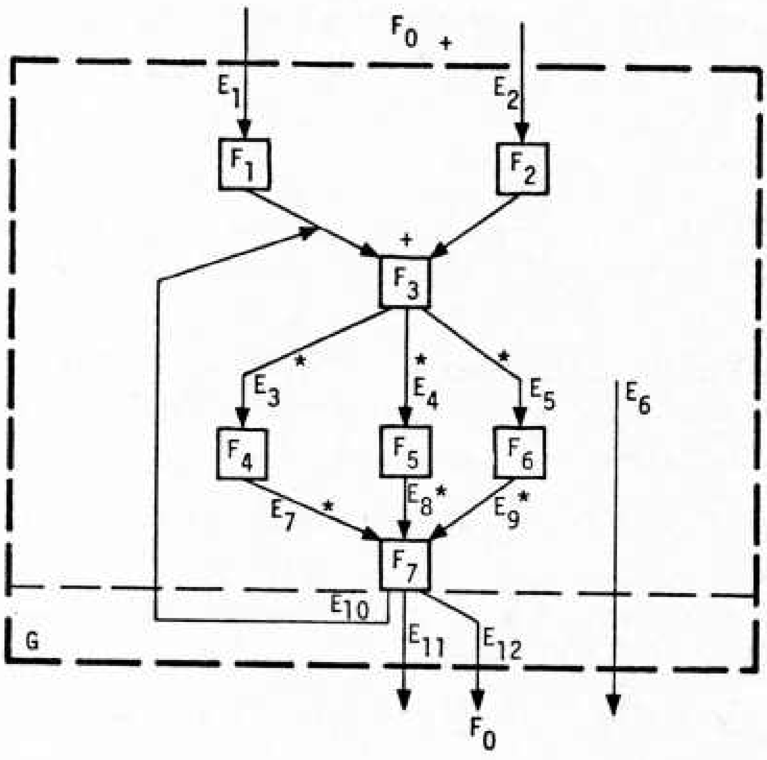
$P_o = \bigcup_i P_i$.

$D_o = (G, F_1, \ldots, F_n)$.

$C_o = G(C_1, \ldots, C_n)$.

The F_o is called a composition of $F_1, \ldots, F_n$.

Remarks. Composition is the technique used to compose a "Super Function" F_o, from a set of functions, $F_1, \ldots, F_n$. The inputs I_o to F_o are the inputs to an F_i which are not outputs of other F_j. The outputs O_o of F_o are some subset of all outputs of all F_i. The system parameters U_o and performance indices, P_o, are simply the union of those of the F_i. The description, D_o,



ICA79-001

Figure 2-3 Example Graph Model of Functionality

of the transformation of F_o is the Graph over the F_i. And the stopping conditions, C_o of F_o, are determined from the graph over the stopping conditions of the F_i.

The inputs, I, and outputs, O, of F_o are more than just a simple union of the inputs, I_i, and the outputs O_i, of the F_i. The notions of sequence and parallelism must be preserved also. Figures 2-4, 2-5, and 2-6 present typical data compositions derived from the GMF with sequential, selection, parallel functions. Note that the input data, I_i, to a function, F_i, may be composed of subtrees output by previous functions and subtrees with an external source. Thus the definition of the input data, I_i, for each function F_i, and the data available from preceding or concurrent functions. The output, O_o, is a subtree of a maximal output SDT; the selection of the specific subtree is a design decision.

2.4 DECOMPOSITION

The approach of defining inputs and outputs in terms of SDTs (rather than sets of data identifiers) allows the definition of decomposition of data, or refinement.

Definition 5 (Data Refinement). Let d_1 and d_2 be SDTs. Then d_2 is said to be a refinement of d_1 if d_2 can be constructed from d_1 by adding subtrees to a subset of the leaves of d_1, denoted by $d_1 \downarrow d_2$.

Example. In Figure 2-7, d_2 is a refinement of d_1.

Remark. This definition of refinement of data satisfies our intuitive notion that the refinement of a refinement should be a refinement. The following theorem is a confirmation.

Theorem 1 (Transitivity of Refinement). If $d_1 \downarrow d_2$, and $d_2 \downarrow d_3$, then $d_1 \downarrow d_3$.

Proof. The subtrees to add to d_1 to yield d_3 are obtained by combining the constructions for $d_1 \downarrow d_2$ and $d_2 \downarrow d_3$.

Remark. The concept of refinement of data provides the critical concept for the definition of decomposition.

Definition 5 (Decomposition). Let F and F_o be system functions

$F = (I, O, U, P, D, C)$

$F_o = (I_o, O_o, U_o, P_o, D_o, C_o)$

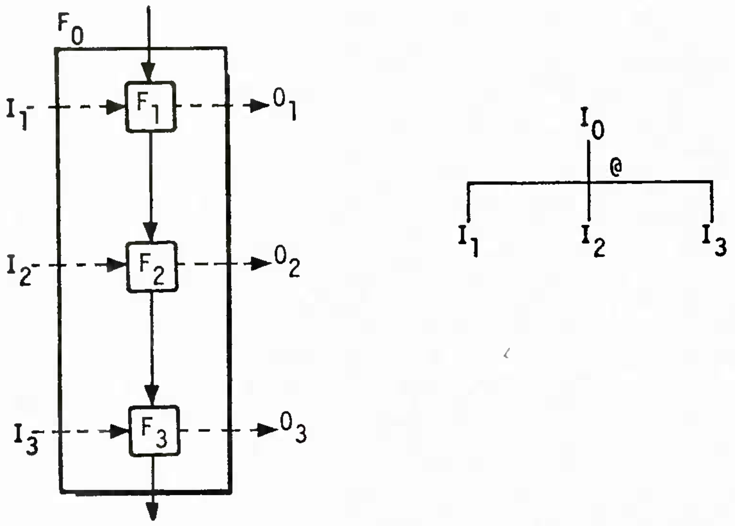


Figure 2-4 Composition of Sequential Data

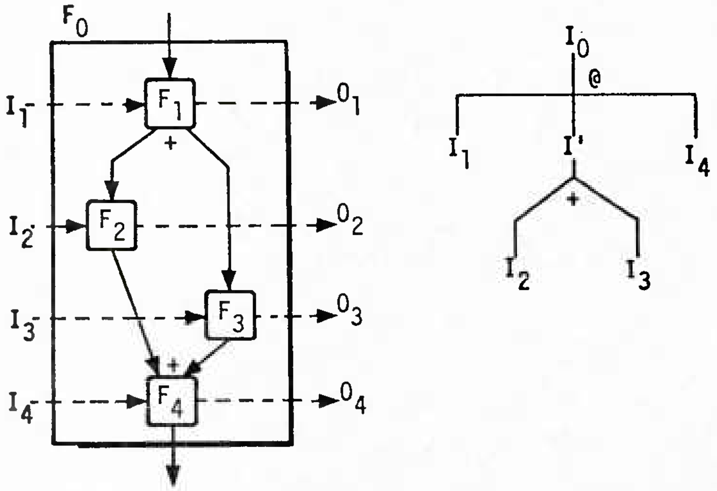


Figure 2-5 Composition of Selected Data

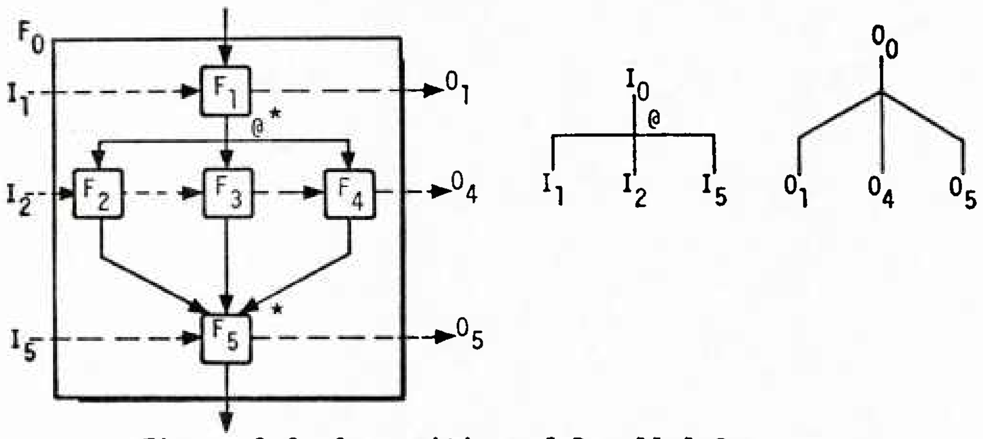


Figure 2-6 Composition of Parallel Data

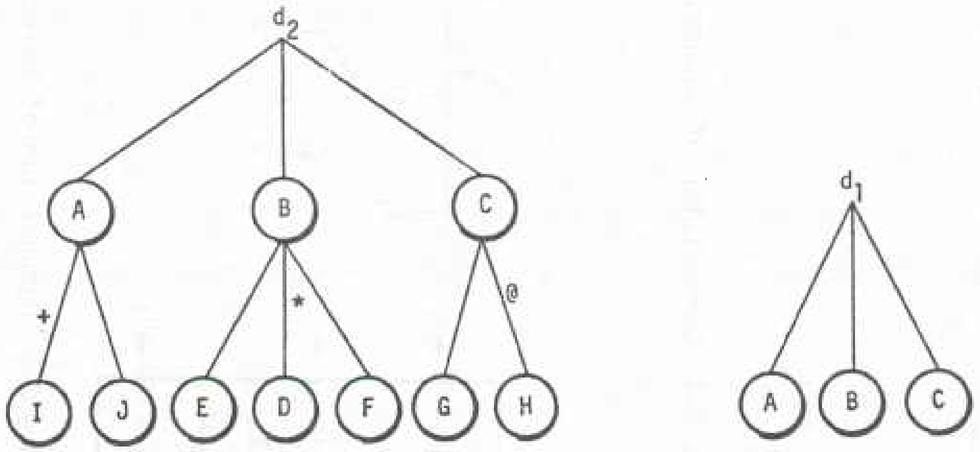


Figure 2-7 Example of Refinement

Then F_0 is said to be a decomposition of F, denoted by $F \downarrow F_0$, if and only if:

a) $I \downarrow I_0$

b) $O \downarrow O_0$

c) $U = \phi(U_0)$

d) $P = \Psi(U_0, P_0)$

e) $C = C_0$

f) D is a projection of D_0, i.e.

   if $i \in I$, $i_0 \in I_0$, $i \downarrow i_0$

   $o \in O$, $O_0 \in O_0$, $O \downarrow O_0$,

   then

   $D_0\ (i_0) = O_0$

<u>Remark</u>. This definition of decomposition requires several criteria to be satisfied:

a) The input and output data of F_0 must have the same structure and sequence as that of F (i.e., $I \downarrow I_0$, $O \downarrow O_0$). Thus, I may be defined as a sequence of "messages", and I_0 may define the contents of those messages, with the requirements that I_0 have the same top-level structure as I. Similarly, O can only be elaborated by O_0.

b) The system parameters U must be calculatable from the U_0. This provides the link between the system parameters selected at one level of analysis and the system parameters selected at the next, more detailed level. Similarly, the performance indicates P must be calculatable from the performance indices P_0 and the system parameters. U_0.

c) The completion criteria C and C_0 must match. This requires that

   i) both F and F_0 complete at the same time; and

   ii) a branch selected by C be also selected by C_0.

d) If D maps an element of I onto O, then D_0 must preserve this mapping; and the associated performance indices. This assures that the nature of the mapping not change from level to level of analysis.

On the other hand, it is also clear that a large number of decisions are made with each decomposition step:

a) The subtrees of I_o and O_o.

b) The functions $F_i, \ldots . F_n$.

d) The sequence of these functions defined by G.

e) The completion criteria C_i.

f) The nature of the transformation D_i.

Note that none of these are unique, and that each requires a specific decision. Thus, with each decomposition $F \downarrow F_o$, we can identify a rationale for its selection.

<u>Theorem 2 (Decomposition Transitivity)</u>. If $F_1 \downarrow F_2$ and $F_2 \downarrow F_3$, then $F_1 \downarrow F_3$.

<u>Proof</u>.

a) From Theorem 1, $I_1 \downarrow I_2$ and $I_2 \downarrow I_3$ imply $I_1 \downarrow I_3$

b) Similarly, $O_1 \downarrow O_2$ and $O_2 \downarrow O_3$ imply $O_1 \downarrow O_3$

c) Since $U_1 = \phi_2 (U_2)$ and $U_2 = \phi_3 (U_3)$,

Then $U_1 = \phi_2 (\phi_3 (U_3))$.

d) Similarly

$P = \Psi_2 (U_2, P_2)$

$= \Psi_2 (\phi_3 (U_3), \Psi_3 (U_3, P_3)) = \Psi_2^1 (U_j, P_3)$

e) If $i_1 \downarrow i_2 \downarrow i_3$, $o_1 \downarrow o_2 \downarrow o_3$, then

$D(i_1) = o_1$ implies $D_2 (i_2) = O_2$, and

$D_2(i_2) = O_2$ implies $D_3 (i_3) = O_3$, hence

D_3 is a projection of D_1.

Remark. This concept of decomposition as a transitive relationship on system functions induces a partial ordering on any set of system functions which represents a hierarchy of decisions. Figure 2-8 illustrates such a tree with a root node F_o, with $F_o \downarrow F_1$, $F_o \downarrow F_2$, $F_o \downarrow F_3$ representing alternate decompositions. Such a tree graphically portrays the relationships of the F_1 in terms of the alternatives which were examined at each level of decomposition. The tree also allows the identification of the specific decisions which led to the form of the function (i.e., F_{10} is the result of the decisions $F_o \downarrow F_3$, $F_3 \downarrow F_7 \downarrow F_{10}$).

The concept of decomposition is common-place in systems theory literature. A treatment of this subject is given by Mesarovic, Macko, and Takahara for hierarchical systems [2] and general systems [1].

It is treated by Softech [5], Peters [6], Hamilton and Zeldin [7], Dijkstra in his description of successive refinement [8], Fitzwater [9], Wymore [10], and in military specification standards [11]. None of the treatments simultaneously address the concepts of I/O sequence, functional sequence, system parameters, and performance.

Several special cases of decomposition are of interest. These are summarized in Figure 2-9 and discussed briefly below.

1) Functional Refinement. In this special case, $I_0 \downarrow I_1$ and $O_0 \downarrow O_1$, but F_1 has the same graph as F_0 and $U_1 = U_0$, $P_1 = P_0$. The refinement of I_0 and O_0 induces the need for a refinement on F_1 which has an elaborated range and domain of its transformation D_0 and completion criteria C_0. This sometimes is used to provide a motivation for further decompositions.

2) Concurrent Decomposition. In this case, F_o is represented as the interaction of three concurrent functions, F_2^1, F_2^2, and F_3^3, which interact via exchange of inputs and outputs, and set together to accomplish the transformation. This is similar to traditional definitions of decomposition in [5], [6], [10], and [11].

3) Hierarchical Decomposition. In their book on Hierarchical Systems Theory [2], Mesarovich, Macko, and Takahara elaborated the point that any system which has inputs I_o which can be decomposed into streams I_1^1, I_2^1, ...I_i^1, can be represented as the interaction of functions F_i^1 which deal with inputs I_i^1 and measurements system control parameters U_i, and have outputs O_i and measurements M_i; this can be done as long as a function F_o^1 is defined (called a control or coordination function) which has as inputs $(M_1^1, \ldots M_n^1)$ and as outputs $(U_1^1, \ldots U_n^1)$. Note that this can be viewed as a special case of the concurrent decomposition.

$$\begin{tikzcd}
 & & & F_0 \arrow[dll] \arrow[d] \arrow[drr] & & & \\
 & F_1 \arrow[dl] \arrow[dr] & & F_2 \arrow[d] & & F_3 \arrow[dl] \arrow[dr] & \\
F_4 & & F_5 & F_6 & F_7 \arrow[dl] \arrow[d] \arrow[dr] & & F_8 \\
 & & & F_9 & F_{10} & F_{11} &
\end{tikzcd}$$

ICA79-009

Figure 2-8 Tree of System Functions

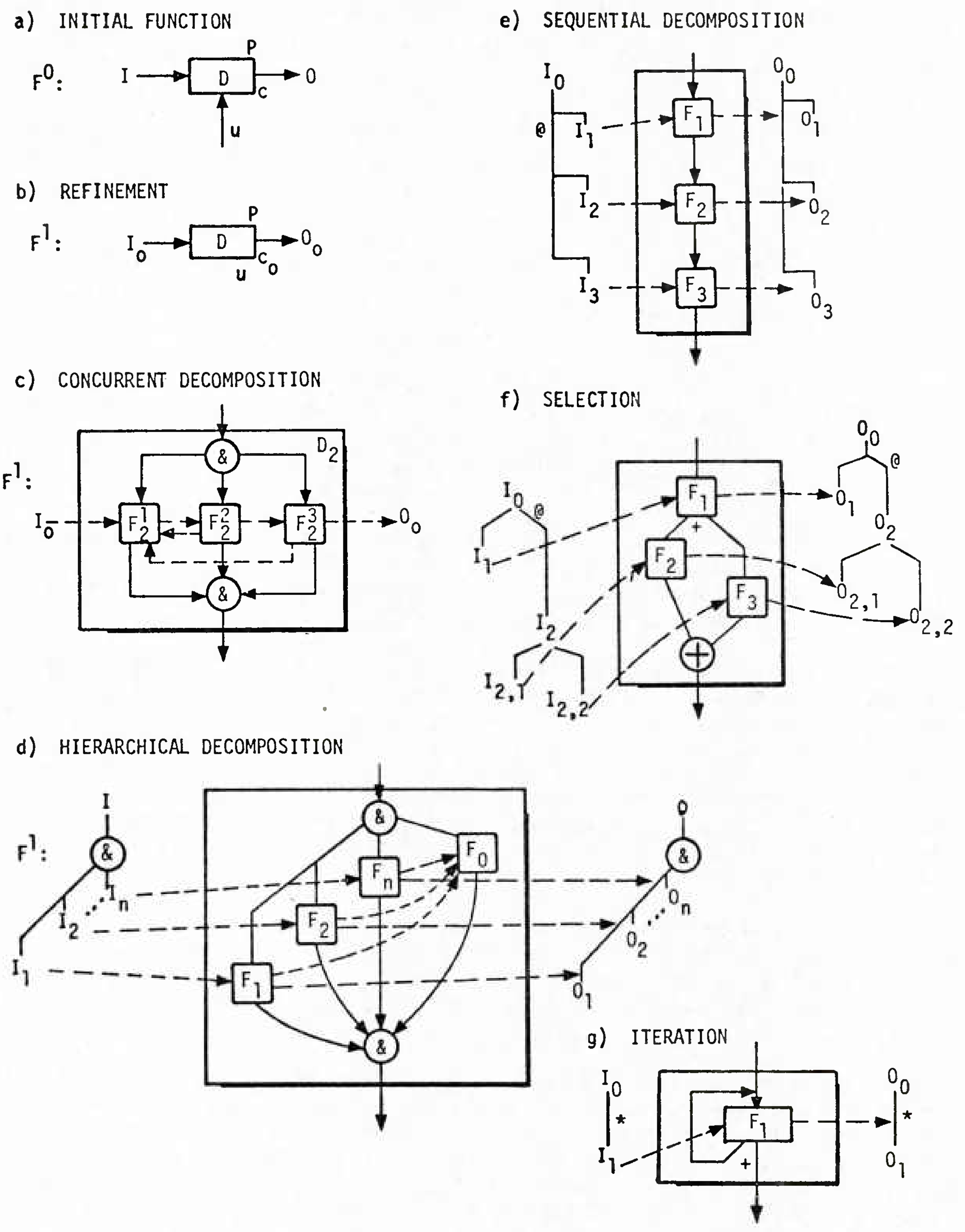


Figure 2-9 Special Cases of Decomposition

ICA79-007

4) Sequential Decomposition. If I consists of the sequence I_1, I_2, I_3, then we can construct the sequence F_1, F_2, F_3 which has these as inputs. This type of decomposition is used most often in simulation modeling of a system.

5) Selection. If I consists of inputs I_1 followed by I_2, where I_2 is either $I_{2,1}$ or $I_{2,2}$, then F can be represented as F_1 followed by either F_2 or F_3.

6) Iteration. If I consists of a sequence I_1, I_2,..., I_n, then a function F can be defined in terms of a sequence of functions F_1 which map I_1 onto O_1.

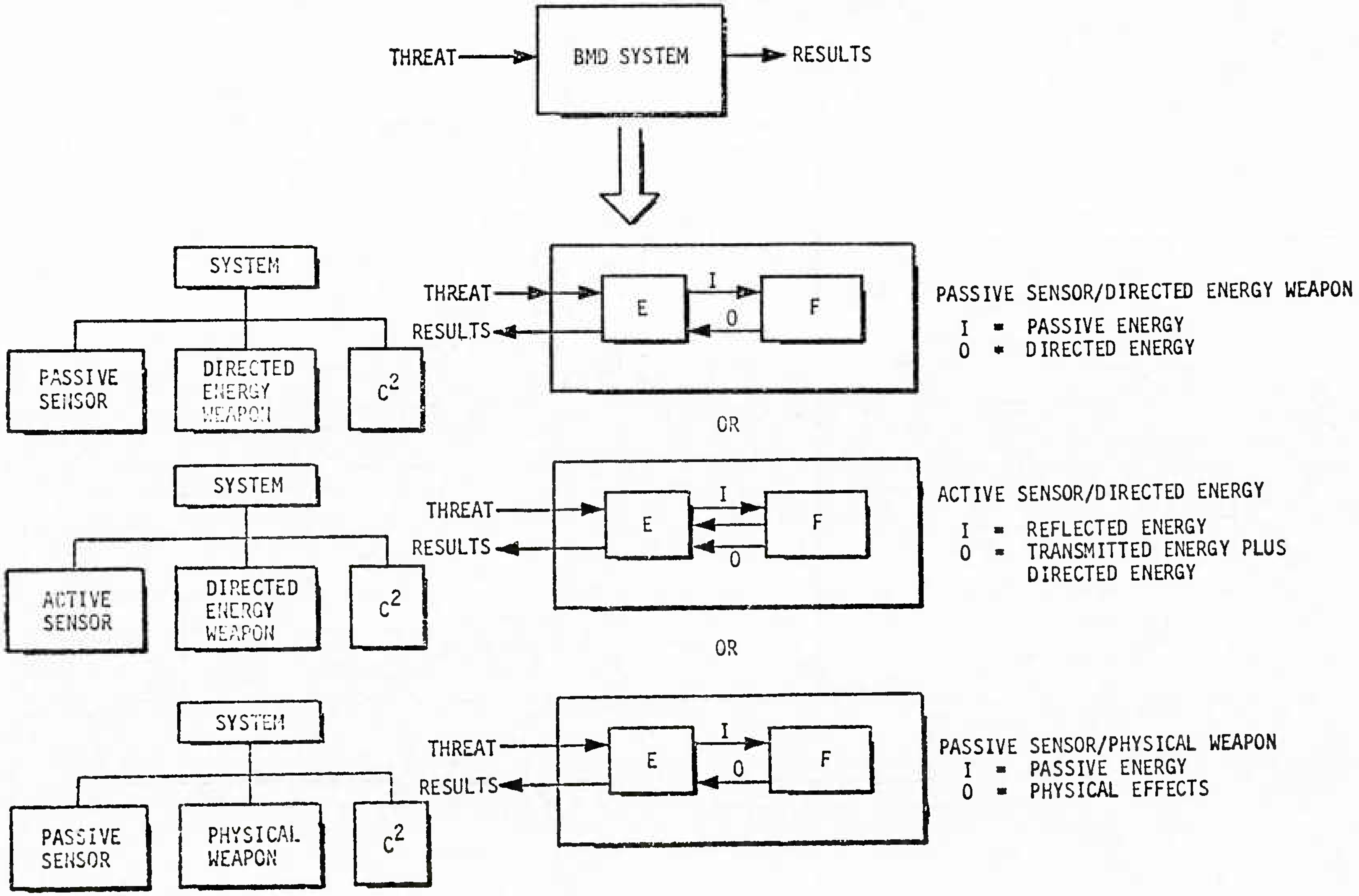
We call these special cases the "structured decompositions", and conjecture that any system function can be derived by a sequence of structured decompositions. Note that structured programming is based on the concepts of sequential, selection, and interaction decompositions, and have no analogs of hierarchical, concurrent, and refinement decompositions.

Example. The relationship between system classes, data refinement, and function decomposition can now be made clear. Figure 2-10 illustrates how a pair of top-level functions (one for the system, one for the environment) are refined, based on the assumptions on the system classes.

If we postulate a passive sensor (e.g., passive optics) and a directed energy weapon, then the system inputs, I, consist of passive energy and the outputs are directed energy (e.g., laser, neutrons, X-rays). If we postulate an active sensor, then the inputs consist of the energy reflected by the threat objects, and the outputs consist not only of the directed energy but also the transmitted energy (e.g., laser radar, radar). If we assume a passive sensor and physical weapons, then the inputs might be the passive energy, and the outputs of the system might be the physical effects of the weapon at impact or detonation time.

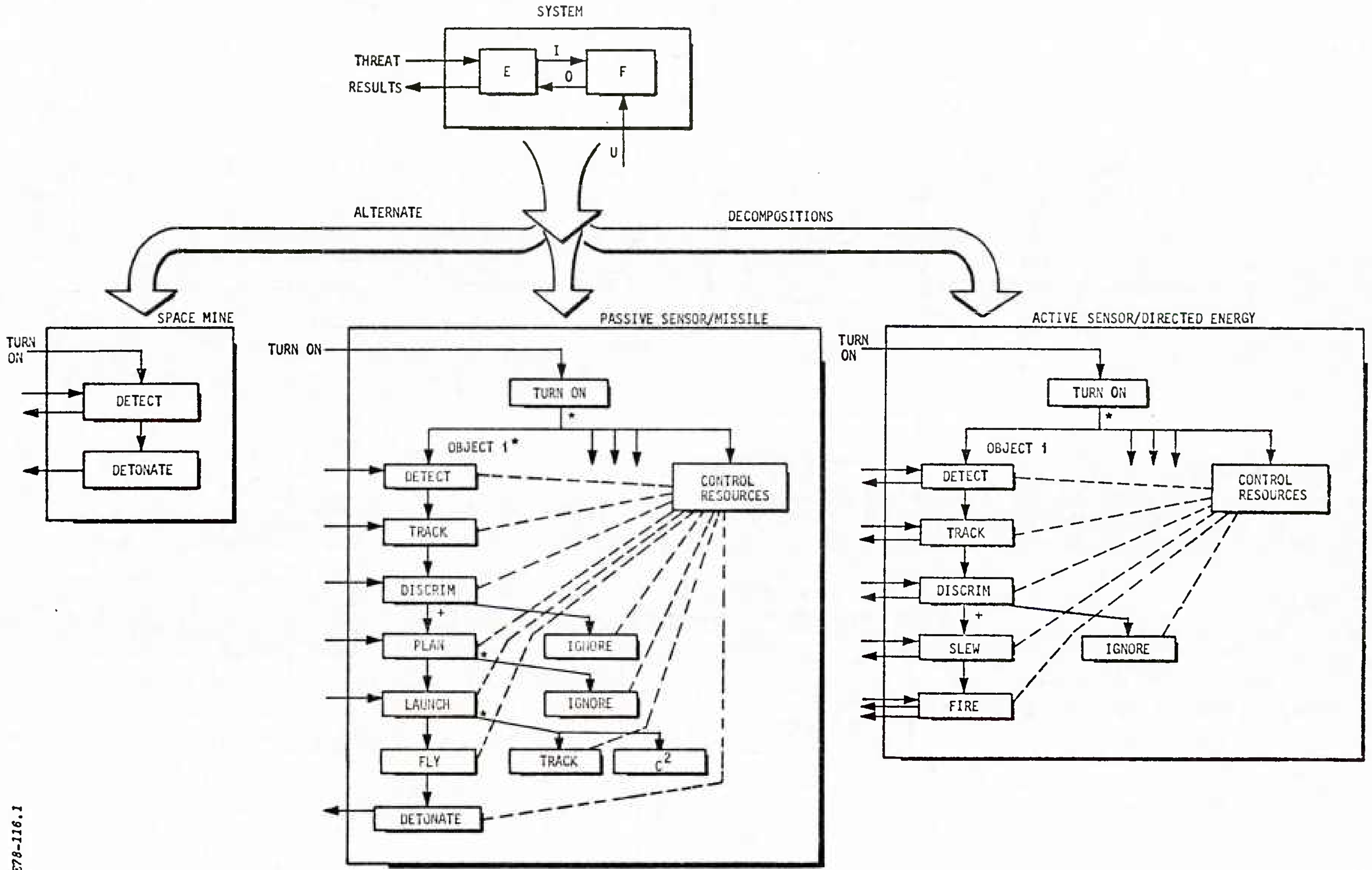
Note that whenever we refine the inputs and outputs, we have in mind a class of device which accepts the inputs and produces the outputs. It is an open question of which comes first; our concept of the physical device which might correspond to such transfer functions, or the identification of the class of input.

In the same fashion, Figure 2-11 illustrates how a decomposition into system actions depends on the nature of the components taking those actions. If a weapon system consists of a sort of "mine" or unguided round with a proximity fuze, then its function is to detect a target and explode. If the weapon system has a passive sensor which is used to guide a missile, the system actions include the functions of tracking, discriminating, launching and guiding a missile to detonation. If the sensor is active and the weapon uses directed energy, the target is detected, track, and the weapon is



ARE78-115.1

Figure 2-10 Effect of Top-Level Refinement on BMD System



ARE78-116.1

Figure 2-11 Alternate Decompositions of BMD System

physically aimed and fired. Thus the system actions (nature, sequence, inputs, and outputs). All depend on the nature of the components to take them.

2.5 SIMULATION

It is well known that simulation is an important tool for performing the estimating of the performance indices P given approximations of the inputs system parameters, and environment. It is also well known that the simulation should somehow be "traceable" to the system description. These concepts can be formalized as follows:

Definition (Typed SDT). Let I be an SDT. A typing of I, denoted by I^+, is the refinement of I obtained by assigning a type (e.g., boolean, real) and range of values to the leaves of I.

Definition (Simulator). Let F be a system function. F^+ is called a simulator of F if

$F^+ = (I^+, O^+, D^+, U^+, P^+, C^+)$ where

I^+, O^+, u^+, P^+ are typed SDTs

and D^+ is a procedure which maps

(I^+, U^+) to (O^+, P^+), and

C^+ is a procedure which maps I^+ onto the set of output edges of F.

Definition (Simulator traceability): A simulator F^+ is said to be traceable to function F_m if

$D^+ = (G, F_1^+, \ldots., F^+)$

i.e., if both have the same precedence graph.

Remark. This definition of traceability is quite strong: it requires that procedures exist for each system function F_k, and traceability between system inputs, outputs, system parameters, and performance indices. This definition attempts to formalize the approach used to build simulations of software requirements written in the Requirements Statement Language (see Bell [12]).

Note that various levels of simulation exist as a consequence of having different levels of decomposition of the system requirements. This provides a key link for concepts of validating one level of simulation by lower levels of simulation.i.e., one should be a decomposition of the other).

2.6 ALLOCATION TO SUBSYSTEMS

We start by defining the Systems Requirements, R, in terms of the root function, F_o.

Definition (Functional Requirements). A system, S, is said to have functional requirements, R, if

$$R = (X, Y, F, \overline{P})$$

where F is the root function of the system, F = (I, O, D, U, P, C).

X = domain of input space I

Y = required range of the output space O

$\overline{P}$ = range of performance indices P.

A second view of a system is "what it is", i.e., as a set of interacting parts with a common goal. These parts are here called "subsystems", denoted by $SS_1, SS_2, \ldots, SS_n$, and the description of the environment, E.

The description, E, of the environment plays a special role in the development of a system. The root function, F, maps a threat scenario onto the engagement results; the nature of the interaction between the environment and the system components depends on the nature of the components. Thus, E will describe radar reflection properties of the threat if a radar is a component of the system, while E will describe optical emissions if the system includes an optical sensor. The subfunctions to describe the environment must, therefore, be developed as part of the decomposition process, and then "allocated" to the environment description. Note that F describes the closed system, while F-E describes the open system, $\hat{F}$, which maps all environment inputs (e.g., radar returns) onto all system outputs (e.g., radar pulses, blast effects).

The relationship between the subsystems and the system requirements should include the following:

- The subsystems $SS_1, SS_2, \ldots, SS_n$ and the environment model E should collectively perform all of the system actions defined by some decomposition F_m.
- The subsystems have interconnections between them to transmit precedence relationships and information.
- Each subsystem SS_i can be considered as a system.

The implications of these properties are profound: if SS_i can be considered as a system, then the decomposition and allocation process can be repeated successively to yield a hierarchical structure for defining a system in terms its lowest level parts, subassemblies, assemblies, critical items, prime items, and top-level subsystems. If the collection of subsystems must perform all of the actions of F_m, then each subfunction of F_m should be uniquely allocated to exactly one subsystem. This provides the stopping criterion for the decomposition process; in effect, decomposition stops when each subfunction of a functional decomposition can be mapped uniquely to one subsystem SS_i. Finally, the interfaces between the subsystems must pass all information and enablement information between the subfunctions allocated to the subsystems; this provides the requirements for the interface design. These concepts are formalized below.

Definition (Subsystem Allocation Relationship). Let M be a relationship between the subfunctions of F_m and the subsystem elements SS_1, SS_2,...., SS_n and the environment model E. Then M is said to be a subsystem allocation relationship.

Definition (Subsystem Allocation). If M maps each lowest level subfunction F_m onto one subsystem or the environment model E, then M is said to be an allocation.

Remark. A subsystem allocation relationship can exist between any level of decomposition of the system function F_0 and the subsystem (including the environment model E). Note that, as F_0 is further decomposed, eventually M becomes single valued, and hence, an allocation. Further decomposition of F_0 after M is an allocation, explicitly imposes "design constraints" on the subsystem to which the subfunctions are allocated. This provides the key to the definition of "design freedom" as a quality. Note further that the qualities of design freedom and allocatability (i.e., several different allocations to subsystems can be explored without further decomposition) appear to be incompatible -- the further F is decomposed, the finer granularity can be considered for allocation to subsystems, but then "neighboring subfunctions" will be allocated to one subsystem, thereby giving a finer description than "necessary". It appears that one may "recompose" the subfunctions into higher level functions after the allocation has been made to reduce the design constraints of the subsystem requirements; the "validation points" on the R-Nets of the Software Requirements Engineering Methodology have this property.

Note that an allocation, M, not only allocates the subfunctions to the subsystems, but induces an allocation of system parameters and performance functions to the subsystems SS_i and the environment E, and induces the mapping on inputs and outputs of the subfunctions onto the inputs and outputs of the subsystems.

This is formalized below.

<u>Definition (Subsystem Graph)</u>. Let M map F_m onto subsystems $SS_1, \ldots., SS_m$. The subsystem graph $\hat{G}_i$ is the graph

$$\hat{G}_i = (e_{rs}),$$

where $e_{rs} = (F_r, F_s)$ where F_r or F_s (or both) are in SS_i.

Thus $\hat{G}_i$ contains all edges connected to any F_i mapped by M into the subsystem SS_i.

<u>Definition (Subsystem Composition)</u>. Let M map F_m onto subsystem $SS_1, \ldots., SS_n$, E having subsystem graphs $\hat{G}_1, \ldots., \hat{G}_n$, $\hat{G}_E$. Define

$$\hat{SS}_i = (\hat{I}_i, \hat{O}_i, \hat{U}_i, \hat{P}_i, \hat{D}_i, \hat{C}_i)$$

where

$$\hat{D}_i = (\hat{G}_i, \{F_j \text{ where } F_j \text{ is in } SS_i\})$$

$$\left.\begin{array}{ll} \hat{I}_i & = \hat{G}_i(I_j) - \hat{G}_i(O_j) \\ \hat{O}_i & = \hat{G}_i(O_i) \\ \hat{U}_i & = \bigcup_j \hat{U}_j \\ \hat{P}_i & = \bigcup_j P_j \\ \hat{C}_i & = G_i(C_j) \end{array}\right\} \quad \text{for all } j \text{ such that } F_j \text{ is in } SS_i$$

<u>Remarks</u>. Given the mapping M, the actions of SS_i can be specified in terms of the graph of its subfunctions (the subsystem graph), and then the inputs, outputs, system parameters and performance indices, and completion criteria follow. The interfaces between the subsystems now consists of the common edges of G and the common input/output information between functions of the subsystems. This information transfer can be accomplished with one or several different "links" between subsystems. These concepts are explored below.

Definition (Link): A Link L_{ijk} between subsystems SS_i and SS_j is a pair

$$L_{ijk} = (C^*_{ijk}, C^{**}_{ijk})$$

where C^*_{ijk} and C^{**}_{ijk} are communication functions assigned to SS_i and SS_j; respectively. The interface between SS_i and SS_j can consist of a number of links.

Definition (Interface Design). Let M be an allocation of F_m onto $SS_1, \ldots, SS_m$ and E. Let Q_{ij} be a collection of messages,

$$Q_{ij} = \left\{ \begin{array}{l} q_{rs} \text{ is an output of an } F_r \text{ belonging to } SS_i \text{ and } q_{rs} \text{ is input} \\ \text{to an } F_s \text{ belonging to } SS_j \end{array} \right\}$$

Let E_{ij} be a collection of edges between SS_i and SS_j,

$$E_{ij} = \left\{ e_{rs} = \text{edge } (F_r, F_s) \text{ where } F_r \in SS_i \text{ and } F_s \in SS_j, \right\}$$

Let N be a relationship between $(Q_{ij}, Q_{ji}, E_{ij}, E_{ji})$ and $(L_{ij1}, \ldots, L_{ijk})$

If N is a mapping, then N is called an interface design.

Remarks: N is a design because it specifies that all interfaces between subsystems will occur across well-defined links. It is the function of the communication functions C^*_{ijk} and C^{**}_{ijk} to accomplish the transfer of precedence and information between functions. Note that C^*_{ijk} and C^{**}_{ijk} must have consistent decompositions, i.e., both must be decomposed in a consistent fashion. Some links between subsystems are highly serial (e.g., using a single wire), while others can be highly parallel (e.g., buffer storage). In any case there are general requirements to merge data, send the data to the other function, where it is sorted and communicated to the appropriate functions. Note therefore, that two kinds of decisions are necessary to define the SS_i: the boundaries of the SS_i, and the nature of the interfaces between the subsystems (e.g., the number of input/output links determines whether inputs to a subsystem are parallel or interleaved). Thus, we arrive at the following:

Definition (Design). A design $\hat{D}$ of a system is the 4-tupal

$$D = (F_m, M, N, SS)$$

where F_m is a decomposition of F,

SS is the set $(SS_i,\ldots,SS_n, E)$

M is a mapping of F_m onto SS

N is an interface design.

Example. In Figure 2-12, we see the allocation process in action. In Figure 2-12, we have a function defined in terms of a decomposition of two parallel sequences; in b), SS_1, and SS_2, and the interface A are defined in a straight-forward way.

In Figure 2-13, the inputs are parallel and the functions F_1, F_2, and F_3 are also seen as parallel. When M allocates the outputs A_1, A_2, and A_3 to a single interface link, then we create the need for a SORT function to examine the input stream into its constituents A_1, A_2, and A_3, which are then input to F_4, F_5, and F_6. This SORT function is not part of the original requirement, but is made necessary by the mapping of the inputs to SS_1 onto a single input link; thus, SORT is a derived requirement.

Note that the definition of SS_i can be described in terms of hierarchy of functions which are different than those of the requirements. And note that the SS_i becomes the initial requirement for any further decompositions.

The design of a system can thus be viewed as a series of decompositions leading to an allocation to subsystems; the subsystem functions are in turn decomposed and then allocated to critical items; the critical item functions are decomposed and allocated to smaller items, assemblies, and so forth, until the lowest level part is identified.

As previously discussed, the allocations are not unique. What then leads to the selection of one allocation over another? It appears that properties of reliability, modularity, testability, use of existing pieces, interface complexity, and producibility are dominant considerations.

It is further noted that some system indices are properties of the pieces of the system alone -- reliability is such an index. The reliability of the system is calculated from the reliability of the subsystems -- the reliability of the system is not normally calculated in terms of the reliability of its functions.

Finally, some of the most important indices (cost and schedule) are functions of the subsystems and the resources necessary to construct the system from the subsystems. These are addressed next.

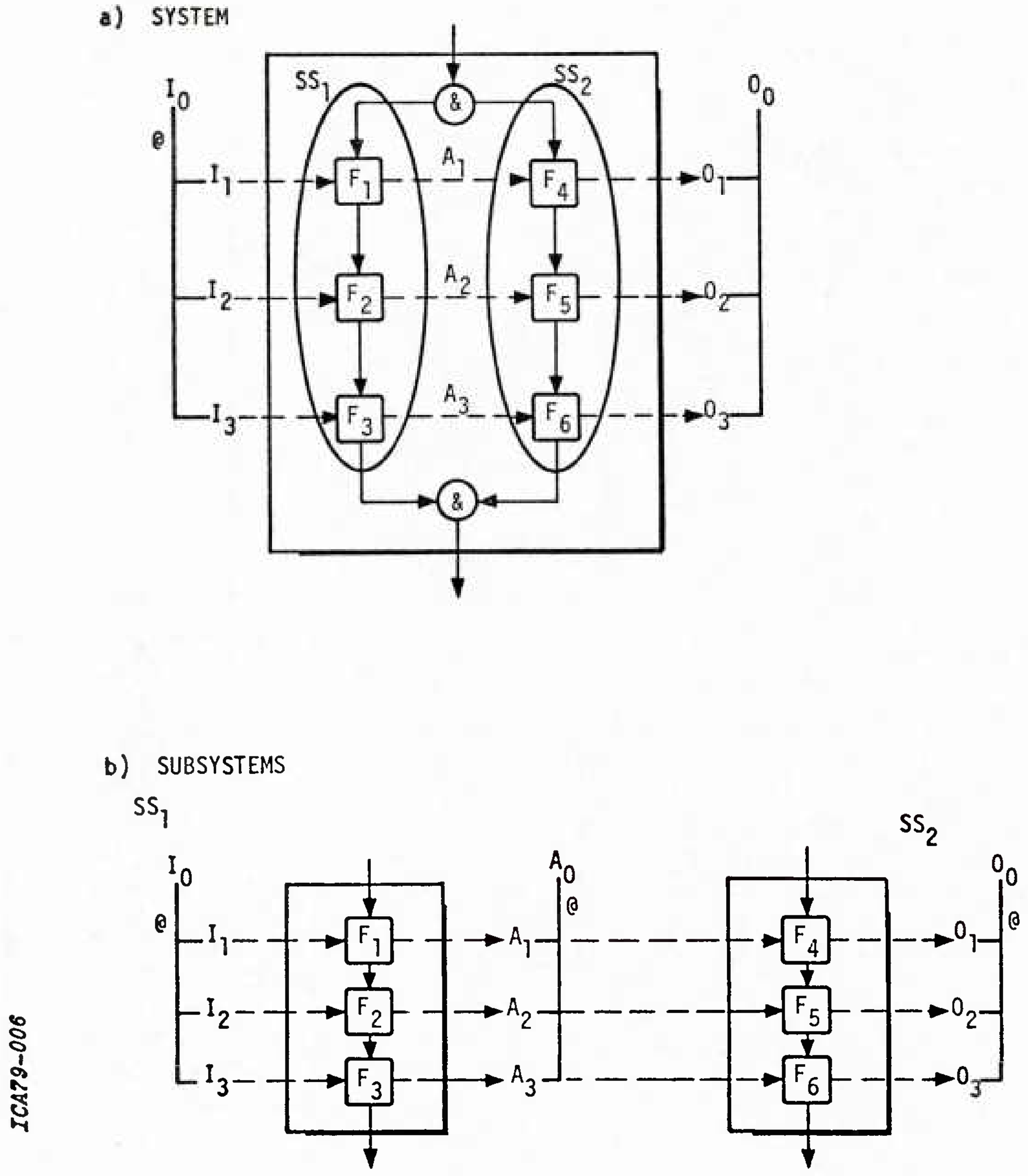
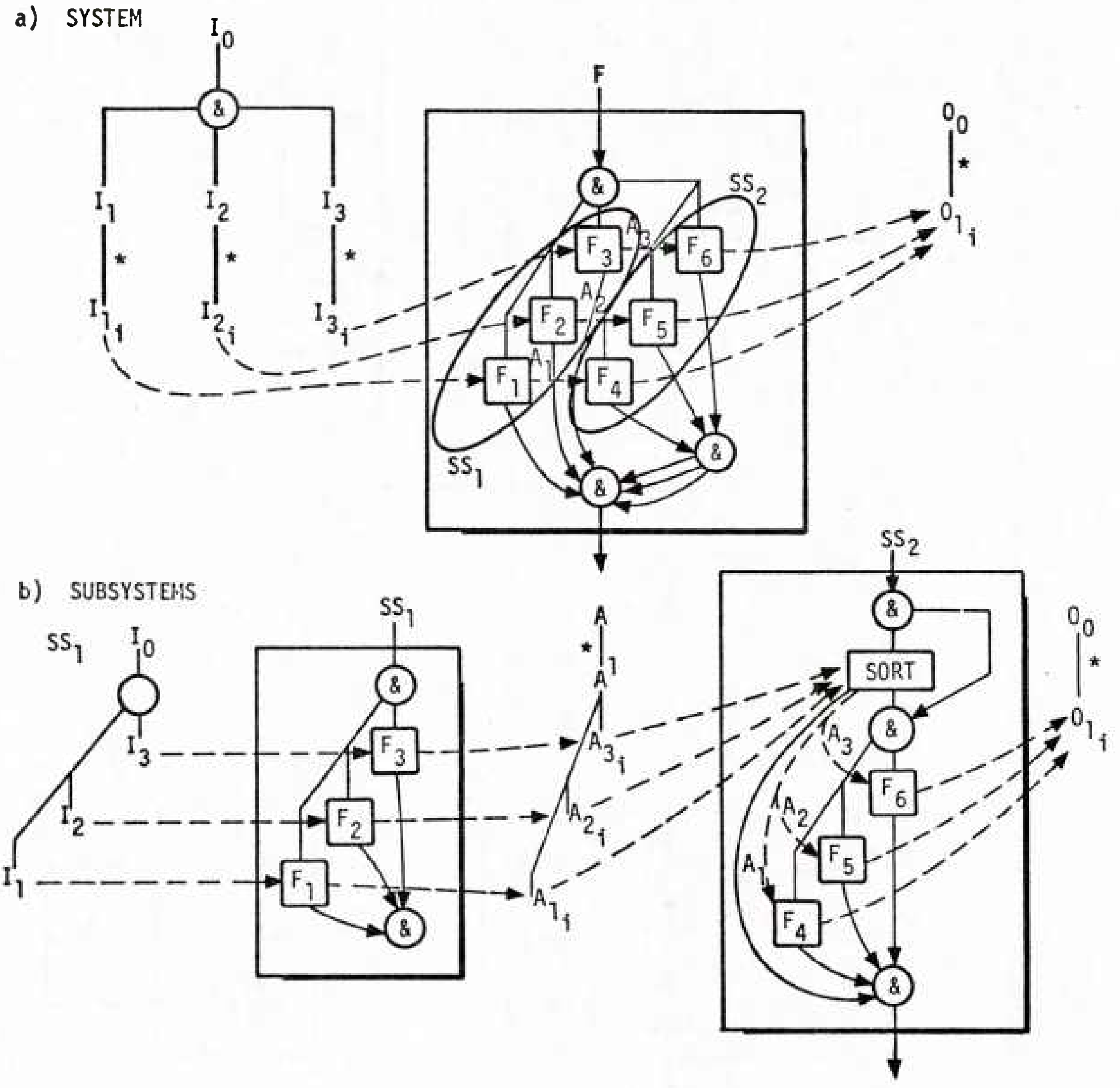


Figure 2-12 Allocation of Sequential Functions



ICA79-014

Figure 2-13 Allocation of Parallel Functions

2.7 INTEGRATION AND TEST

A system is not merely a collection of its constituent subsystems: time and resources are required to integrate the pieces together and to verify that they work together to accomplish the system functions. Several things are required to accomplish the construction of the system:

- The constituent subsystems.
- A set of test tools.
- A set of test procedures.
- A sequence of integration and test steps.

The integration and test of a system can be considered as a system decomposed into a sequence of steps which have inputs, outputs, and performance (in particular, cost and schedule). Several points are significant:

- After integrating two or more subsystems, these subsystems can be tested according to a test procedure. The test procedure has as its objective the verification that one or more specific system functions are satisfied. Consider a system function, F, decomposed into the interacting functions, F_1 to F_7, which are allocated to SS_1, SS_2, and E, as indicated on Figure 2-14. Then some specific tests should verify that function, F, is in fact accomplished by the cooperative action of the subsystems if the environment performs the functions assigned to E, and its performance index, P, is satisfactory.
- To verify that F is performed satisfactorily, it may be necessary to have a test tool to provide the initial data, I_1, to the function, F_1, and to emulate the environment, E, by accepting an I_4 and an I_{34} to produce the I_{45} to SS_2. The test procedure then defines the inputs, the expected outputs, and the criteria for accepting the performance of the function, F. The definition of the characteristics of the functions assigned to the environment (in our case, F_4) then become the requirements on the test tool.
- Resources (e.g., time, manpower, computer time, costs) are required to develop both the test tools and the test procedures. Thus, the development of the test tools and the test procedures are the distinct steps in the integration and test plan.

These concepts are formalized in the following.

Definition (Integration and Test Plan). An Integration and Test Plan, T, is a six-tupal, T = (I, O, U, P, D, C) where

I = inputs necessary for construction of the system, e.g., raw materials, labor, machine resources, and specifications.

ICA79-003

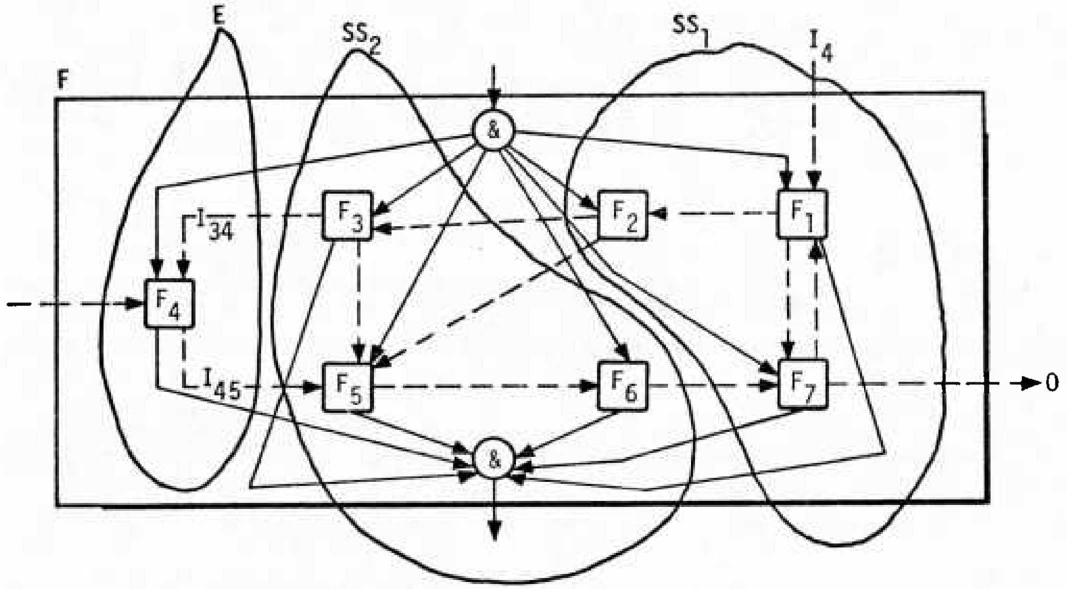


Figure 2-14 Example Decomposition

O = outputs, i.e., the system.

U = the parameters affecting the system development, including the parameters U_n.

P = system development performance indices including cost, schedule, and utilization of critical resources.

C = completion criteria, including the test acceptance criteria.

D = the sequence of steps necessary to develop and test the system.

Remarks. Figure 2-15 presents an example of the first level of decomposition. The steps to develop SS_1, SS_2, and SS_3 have the definitions of SS_1, SS_2, and SS_3 as input. The Test Procedures are developed using the definition of the system function decomposition, F_n, as input; the test tools are developed from the definition of E and the test procedures. In this case, an SS_1, SS_2, and SS_3 must be available to integrate, and the system must be available with test tools and procedures to test it. The Completion criteria, C, is the satisfaction of the tests.

The first steps are always the development of the subsystems; the next level of decomposition the system development will decompose this into the development of the constituents of SS_1 plus their integration and test. Also, the definition of the test procedures can be initiated, given only the F_m and the allocation, M.

How far is the testing decomposed? This question is similar to that of how much software testing is necessary. Several criteria are possible:

- All system functions are exercised at least once.
- All paths through the system logic are exercised at least once.
- The boundaries between the input regions resulting in different system paths are verified.

2.8 ESTIMATING RULES

Given a decomposition F_m of the system requirements which has system parameters U_m, given values of U_m, we can map the U_m onto the performance indices P using either analysis or simulation. However, to perform cost/performance tradeoffs, we need some technique of estimating the cost of a system which has those values of U_m. This results in the need for estimating rules, designated as the set W. The estimating rules have to be a function of the following:

ICA79-008

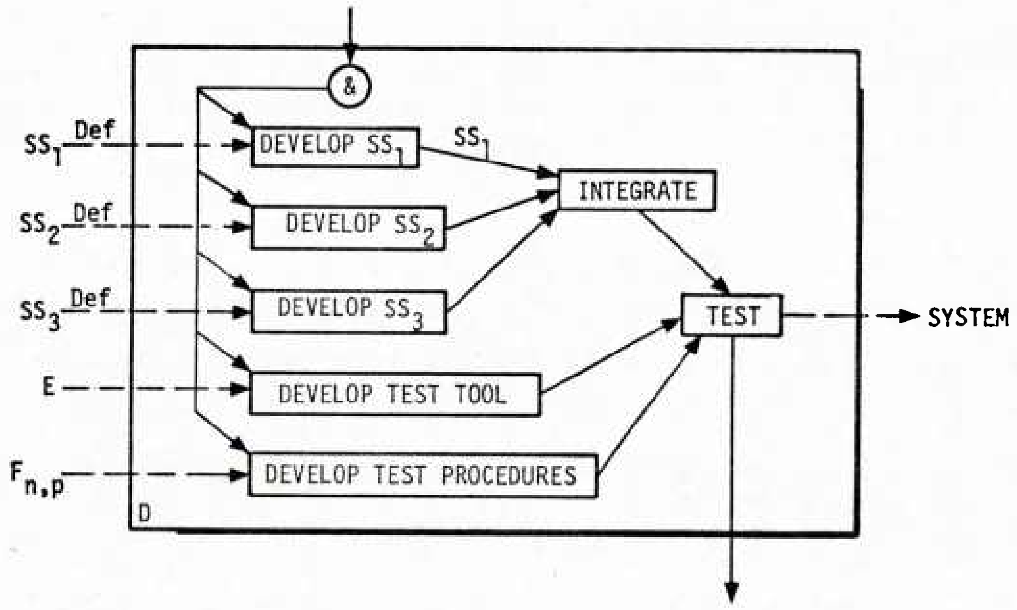


Figure 2-15 Example Integration and Test Sequence

- As the design $\hat{D}$ changes, subsystem costs and schedules should change (i.e., due to differences in interfaces and subfunction allocation).
- Given a design $\hat{D}$, there are still cost/performance relationships which exist for the subsystem (i.e., you can buy a cheap unreliable low-performance component or an expensive high-performance ultra-reliable component for any value of U_m).
- Given subsystem component estimates given U_m, the cost of the system must still include the cost of integration and test; hence, estimating rules must exist for constructing integration and test tools, devèloping the testing.

The estimating rules, W, must map the system parameters U onto the set of performance indices of the integration and test functions; the decomposition then can be used to map these back onto the resource constraints for the development system, T. This is the way to change the development, integration and test cost, and thus yield cost/performance trades.

2.9 PREFERENCE RULES

The term "preference rules" is used to denote the set of rules by which one system is compared to another; ultimately, one system is preferred to another. And thus the rules for determining preference are an important factor in the system design.

The content of the rules includes as a minimum the performance indices of F and the resource indices of T. The rules should identify the desired relationships of these indices (e.g., maximize performance for specified cost, minimize cost for a specified performance and development risk and schedule, or get the best performance for the least resources). These rules can merely state preference, or can rule out catagories of solution (e.g., if the cost is over a specified amount, don't consider it). The rules can also include design constraints such as "use component type XYZ" or more nebulous factors such as "use component type XYZ unless performance penalties are 'too severe'", or include factors such as growth potential, modularity, etc.

The set of preference rules for system selection are usually never written down fully; it is mandatory that some of them be written down in order to reduce the size of the design and test space to be examined (e.g., maximize performance for fixed cost and deployment schedules), but factors such as modularity are usually handled subjectively and informally.

2.10 CONCLUSIONS

In the overview, a system set was defined as the five-tupal

$$S = (R, \hat{D}, T, W, Z)$$

where R identified the requirements set, $\hat{D}$ the design set, T, the test and integration plan set, W the set of estimating rules, and Z the preference rules.

We see from the analysis that these factors are not independent: the decompositions of R are mapped onto the subsystems defined by $\hat{D}$; the T describes how these specific subsystems are integrated and tested, and decomposes the resource requirements into those of the integration and test steps; W must contain estimating relationships for the resource requirements for subsystem development and the integration and test; and Z must identify preference rules for all of the performance indices and resource requirements.

The above discussion concentrates on the features of performance, development cost and schedule, and deployment cost and schedule; the factors of life-cycle cost (including logistics and maintenance costs) require the addition of additional factors; preliminary indications are, however, that these factors can be handled in a manner analogous to those of the development resources.

Note the dimensionality of the system design problem even at this one level. There are a large number of decompositions of F; given a decomposition F, there are a large number of ways of packaging these functions into subsystems and their inter-connections; there are a large number of possible values for the system parameters; there are a large number of possible ways of integrating and testing the system; and even if the cost-estimating relationships are fixed, different system designers might have different rules for selecting one design over another which he unconsciously applies during the design process. Because of this large dimensionality, it is necessary to have a methodology to assure that all factors have been taken into account without enumerating all possible designs. This is the subject of the next section.

3.0 METHODOLOGY OVERVIEW

The foregoing formal foundations laid the groundwork for a methodology for allocating system requirements to subsystems by precisely defining basic concepts of precedence relationships of actions, decomposition, allocation, simulation, and interface design. In this section, the implication of these foundations on a methodology for the front-end system design is discussed.

A methodology for the front-end system design must address two types of issues: the selection of the sequence of design elements (e.g., system, subsystem, prime item, critical item), and the tools and techniques used to define requirements for a component and allocate requirements to the next level of components. The classes of components are established by the state-of-the-art of the application (e.g., radars, radar transmitters, data processors divided into hardware and software). The specific nomenclature for the system functions is application-specific. However, the general outline of the steps for performing the requirements definition and allocation to components is essentially application independent. Although a detailed methodology has not been fully worked out for the front-end system design activity, a top-level description appears below.

3.1 OVERALL APPROACH

Table 3.1 and Figure 3-1 present a top-level view of the methodology implied by the preceding formal foundations. At any level of component design, the functions and performance of the component as a whole are first identified: for the top-level system, this includes the systems analysis step of performing the needs analysis, defining the mission and threat, and defining the top-level performance indices and preference criteria (e.g., minimum cost, fixed deployment date), for systems which are components of other systems, the relevant allocated requirements are identified.

The second step is the identification of the appropriate component types. There are generally classes of components which could be considered as candidates for inclusion into the system design. Depending upon the preference criteria, whole classes may be excluded (e.g., due to weight restrictions, deployment dates, reliability factors), or the class of components may be subject to design restrictions (e.g., consider only specific approved data processing hardware).

Based on a specific set of component types, there are actions appropriate for a system which has that set of components to address the objects of the system's environment. This is first done in terms of describing how the system addresses each object that it deals with (commonly referred to as stimulus/response relationships), and then in terms of subfunctions which are allocatable to the components. In both cases, decomposition is used to express the higher level actions in terms of lower level ones. In both cases, analysis and/or simulation is used to predict the performance of the system, and the utilization of system resources.

Table 3.1 Overall Methodology Steps

STEP 1	DEFINE MISSION
STEP 2	IDENTIFY COMPONENT TYPES
STEP 3	DECOMPOSE TO SYSTEM LOGIC
STEP 4	DECOMPOSE TO ALLOCATABLE SEQUENCES
STEP 5	ALLOCATE AND ESTIMATE FEASIBILITY
STEP 6	IDENTIFY CRITICAL ISSUES AND RESOURCES
STEP 7	IDENTIFY RESOURCE MANAGEMENT RULES, ALTERNATE LOGIC PATHS, AND ALTERNATE DECOMPOSITIONS
STEP 8	OPTIMIZE OVER COMPONENT CLASSES
STEP 9	PLAN INTEGRATION AND TEST

RADC79-041

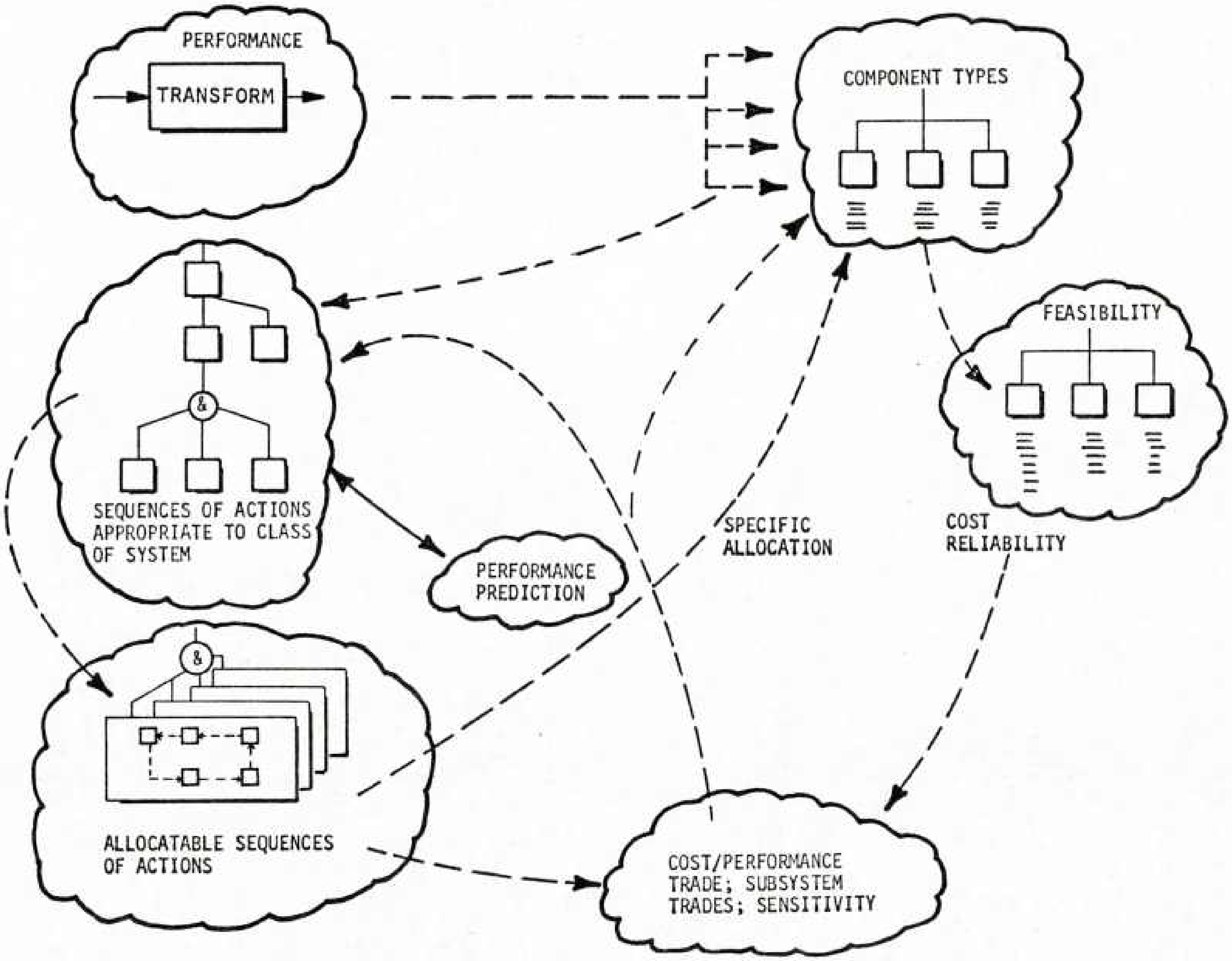

PERFORMANCE
TRANSFORM
COMPONENT TYPES
FEASIBILITY
&
SEQUENCES OF ACTIONS APPROPRIATE TO CLASS OF SYSTEM
PERFORMANCE PREDICTION
SPECIFIC ALLOCATION
COST RELIABILITY
&
ALLOCATABLE SEQUENCES OF ACTIONS
COST/PERFORMANCE TRADE; SUBSYSTEM TRADES; SENSITIVITY

Figure 3-1 Overall Methodology Approach

ARE79-033.1

The next step of the methodology is the allocation of subfunctions to subsystems, and application of the estimating relationships to predict system cost, schedule, etc. These are then evaluated using the preference rules to identify a feasible system (if any) which best satisfies the subsystem and system objectives.

The cornerstone of system engineering is the identification of critical issues, and working of the critical issues first -- without their resolution, there may be no solution. This rule, and the knowledge of the state-of-the-art, guides the analysis in terms of restricting the classes of system components considered, the range of the system design parameters, and the inclusion in the first model of control rules for the critical resources of the system with given configurations. It is not necessary to identify all system action models, followed by all possible subfunctions for their implementation, followed by all possible allocations, followed by the selection of the "best" of all possibilities. In fact, considerable iteration usually takes place between the selection of component types, the definition of the system logic, the selection of candidate system parameters, and the prediction of performance using simulation. Whole classes of systems are eliminated from consideration using quite crude estimation rules and performance predictions. Because of the state-of-the-art in components, only a subset of the possible allocations are considered at first, in order to assure feasibility and to identify the possible range of the cost-performance relationships, and to identify the critical issues -- these critical issues then become the drivers of the analysis process.

There is a feedback between the preliminary identification of the critical issues, critical resources, and cost-performance relationships of the subsystems, and the definition of the system logic. When critical resources are identified, the rules for the allocation of these resources must be identified. Alternate paths may be needed in the system logic to implement the resource management rules. In the case of a surveillance system, an object may not be placed into track if insufficient radar and data processing resources are available to maintain the track -- this results in resource allocation rules (e.g., allow only 10 objects in track) and additional system logic (e.g., if 10 objects are in track, and an additional object is detected, drop track on the object and modify the search volume to assure redetection). This will, in turn, modify the prediction of the system performance in high load situations, thereby changing the cost-performance relationships. If the performance degradations are severe enough, new classes of components may have to be considered to find a feasible solution.

A second general guideline is to parameterize as much as possible. Thus, the classes of components will be expressed hierarchically with parametric performance and the class of active sensor systems (e.g., radars) may be compared to the class of passive sensor systems (e.g., optics) before consideration of specific waveforms of the radar. In either case, the frequency of the sensor would be a system parameter whose value would determine the approximate cost estimating rules.

Several types of computerized tools would be useful in support of the above type of methodology:

- A formal language for stating the original requirements.
- A language for expressing the decompositions, and tools for checking these decompositions for consistency.
- Tools for defining simulations based on a system function decomposition. The simulation would then be guaranteed to be traceable to the system performance degradation due to resource, constraints and to validate the system resource management rules.
- Tools for defining the allocations, the specific interface design, and aiding in the presentation of the resulting subsystem specification.

These types of tools are appropriate for all levels of the system design, and prototypes of these tools exist for many levels of analysis.

To illustrate these concepts, an overview of their application to a specific problem is presented below. The problem chosen is that of performing surveillance of aircraft in a large area, with missions of identification, providing navigational assistance, and providing tactical control of aircraft interception of unknown objects -- this problem was chosen because it typifies many of the features of Air Force systems. The following analyses are illustrative, and are not meant to reflect actual performances or costs.

3.2 SYSTEM ANALYSIS

The purpose of the system analysis step is to identify the system mission in quantified terms (e.g., deployment date, performance bounds, expected environment for operation), and to assess feasibility. This phase is characterized by the identification and analysis of a large number of potentially feasible system classes, and their evaluation to yield a smaller number of preferred constructs. The overall methodology described in the last section can be followed as described to perform the systems analysis using fairly crude rules of approximation to narrow the class of solutions.

3.2.1 Step 1 -- Define Mission

Assume that the purpose of the system is to perform surveillance of aircraft in a large area,to identify aircraft (i.e., friently or hostile), provide navigational assistance as requested, and provide tactical control of interceptor aircraft to intercept unknown objects. A reasonable set of performance and life-cycle indices include the following:

- Cumulative probability of detection versus time.
- Probability of identification versus time from first detection.
- Track accuracy versus time from first detection.
- Development schedule and cost.
- Maintenance cost versus time.

The first three items are related to the ability to perform the mission of identifying the aircraft (whether friendly or hostile) and the accuracy of the navigational and interceptor guidance data; the last two might have constraints or minimum cost goals. The system inputs would be the aircraft entering the monitored air space, the interceptors entering the observed air space, and the range of environmental factors (e.g., rain, blizzard, hail). The end product system outputs would be the aircraft track data and the guidance data to the interceptors -- these are summarized in Figure 3-2a.

3.2.2 Step 2 -- Identify Component Types

The types of aircraft to be monitored by the system can be initially identified as friendly aircraft (with or without beacon transponders for identification) and unfriendly aircraft (with or without jammers). The system is to detect all types, identify all types, track all types, and provide navigational information to interceptors and friendly aircraft.

Figure 3-2b presents the class of components which might make up the system. The candidate classes include a set of individual sites, a communication net linking these sites to a central control, and a central control point with consoles and operators. The individual sites must have sensors to detect the aircraft and beacons to interrogate the aircraft transponders; constraints on the state-of-the-art and the nature of the environment (e.g., rain, blizzards) quickly reduces the total class of sensors to the class of radars of specified frequency bands. Fault detection equipment is included to quickly detect faults in the radar and other equipment; data processing (hardware and software) is contained to process all of the radar data, command the fault detection equipment and analyze the results, monitor the facilities equipment, and communicate with the central control. Facilities include the building power generation, air conditioning, etc.

Although classes of components need to be identified in order to perform the decomposition of the system functions, we need not yet specify design details of that equipment (e.g., radar frequency, waveforms); in this way, classes of designs are considered before specifics -- these classes are characterized by parameters (e.g., radar effectiveness for large classes of radars can be described by frequency, power-aperture product, noise level, and resolution). Analysis occurs at this level before specific designs are addressed.

This level of description is sufficient to perform the first few levels of decomposition, discussed next.

3.2.3 Step 3 -- Decompose to System Logic

Figure 3-2c presents a first possible decomposition of a system having such components. The overall surveillance function of the system can be described in terms of the surveillance performed by each radar site plus the interactions of coordination and control. Note that coordination is necessary to cope with the possibility of overlapping radar coverage, aircraft passing from one radar region to the next, and coordination and control of interceptors. Even at this level, there are a large number of tradeoffs which can be made. Figure 3-3 presents two different deployment concepts for the same region using

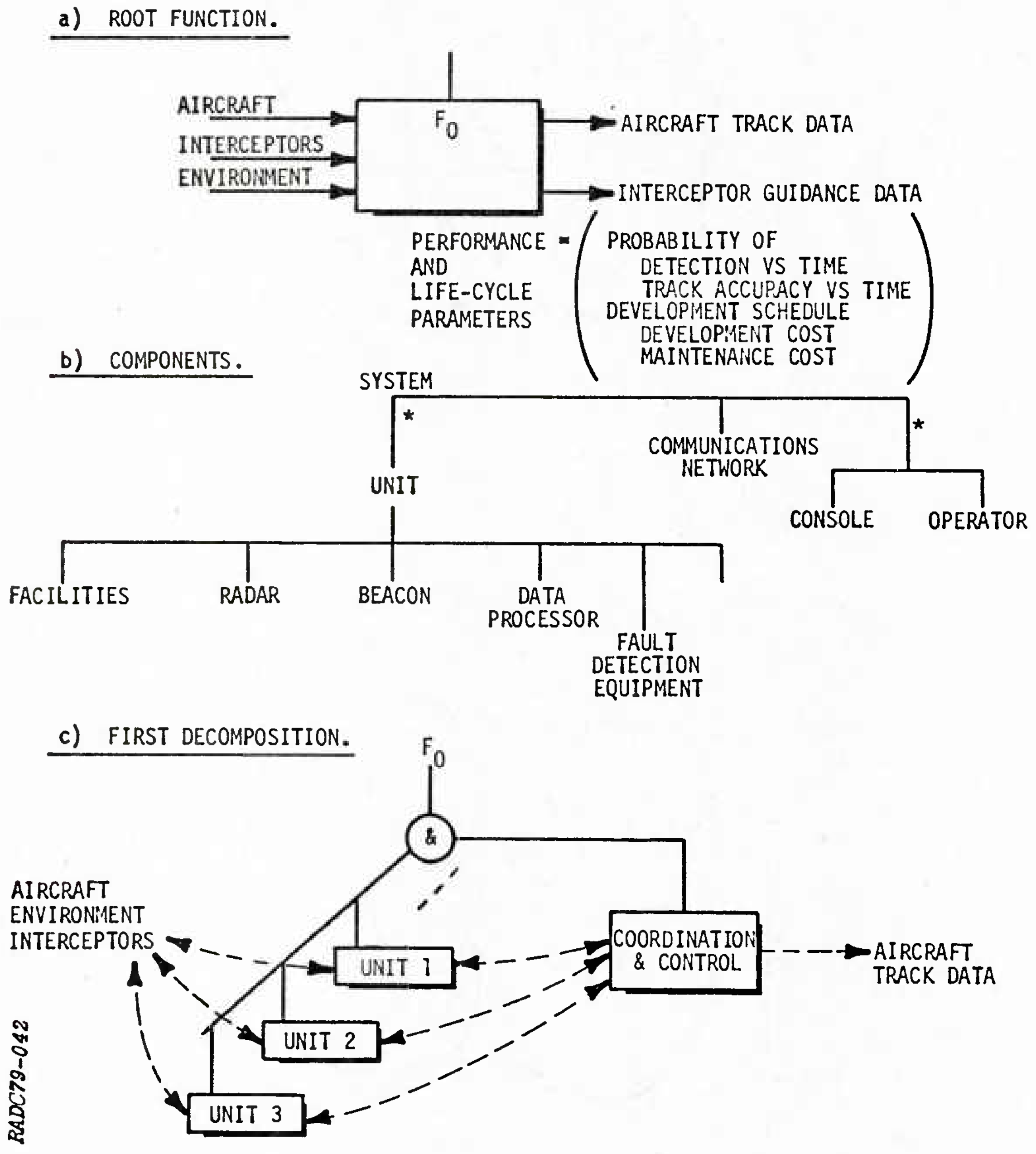


Figure 3-2 Surveillance System Functions and Components

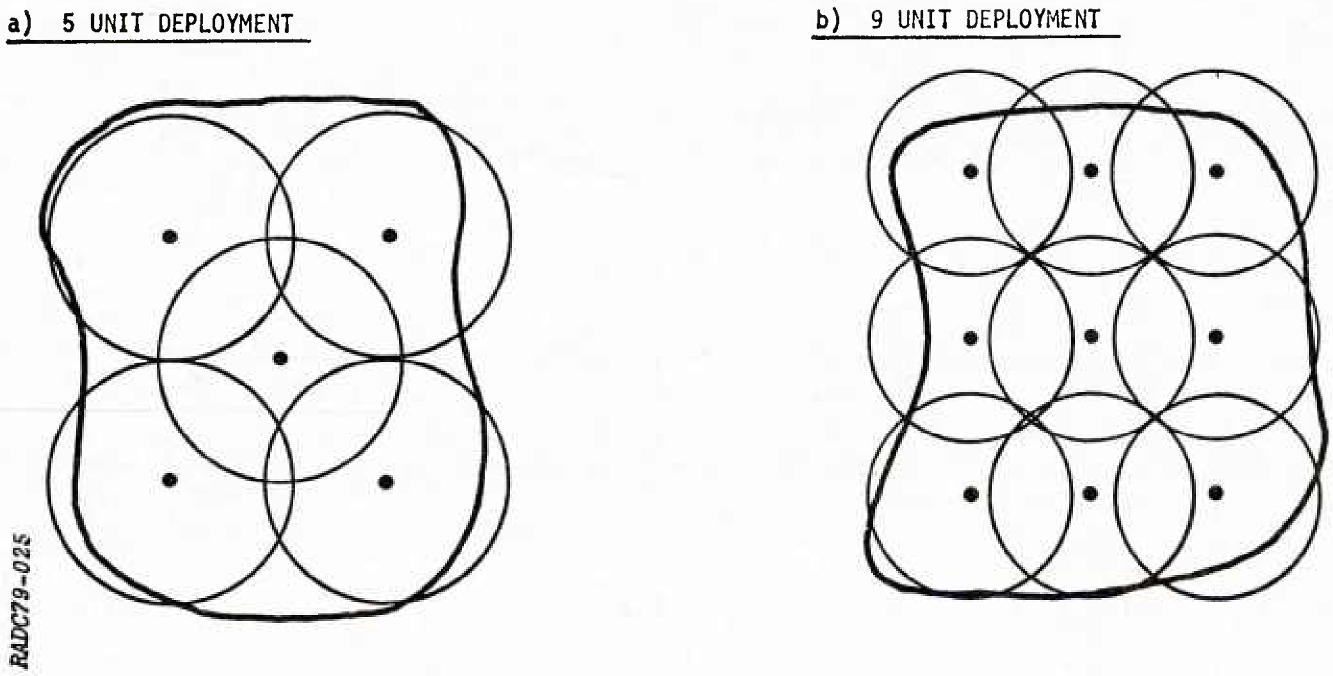


Figure 3-3 Deployment Alternatives

five and nine radar sites, respectively. Note that a given site should be dependent upon the effective range of its operation, and hence different numbers of sites which cover the area should give rise to different costs for individual sites, and hence different total development costs and maintenance costs. Note that, if siting costs are high (for facilities or maintenance), this will drive the design towards a single radar. On the other hand, since radar power is a function of the fourth power of required detection range, if power costs are dominant, this will drive the design towards a large number of smaller radars. The optimal design will balance these two factors to yield a design region (e.g., nine to 15 radar sites) for further analysis. Note that this type of analysis is not in terms of component design factors, but in terms of their costs. This type of analysis is not new, but its need is immediately obvious when the decomposition requires expression of total performance and cost in terms of the unit's cost/performance and the number of units deployed.

The next level of decomposition is presented in Figure 3-4. In this decomposition, the actions of a unit of the system are described in terms of the actions with respect to each aircraft in its surveillance volume, plus the coordination function (necessary for the allocation of critical resources). Note that the performance of the system against an aircraft depends on resources available which depends on the total system load, and the resource management rules. The nature of the critical resources has yet to be determined. Figure 3-5 presents a decomposition of the unit engagement of an aircraft in terms of the actions of the system. The system actions depend on the type of aircraft being engaged:

- Friendly aircraft with beacons are to be identified via beacon response, and provided navigation assistance as required.
- Friendly aircraft without beacons are to be identified by radio or by interceptor visual identification, and provided navigational assistance as required.
- Hostile aircraft with jammers are to be immediately identified and tracked by triangulation of radar units and interceptor visual identification.

The left side of Figure 3-5 describes the actions of the aircraft. Friendly aircraft with beacons reflect radar, respond to beacon pulses, and may request navigational assistance. Friendly aircraft without beacons, which respond to radio, reflect radar and may request navigational assistance. Friendly aircraft without radio response merely reflect radar (and require visual inspection). Hostile aircraft may turn on a jammer.

The right side represents the actions of the system. Detection can occur by radar or by jammer. A radar contact will result in an initiation of track and a beacon pulse. If a beacon response occurs, identification is complete and the aircraft is tracked. Navigational assistance is provided as required. If no beacon response occurs, radio contract is attempted. If contact occurs, the aircraft is tracked as before. If no contact occurs, aircraft are scrambled and information is provided. If the detection is by jammer

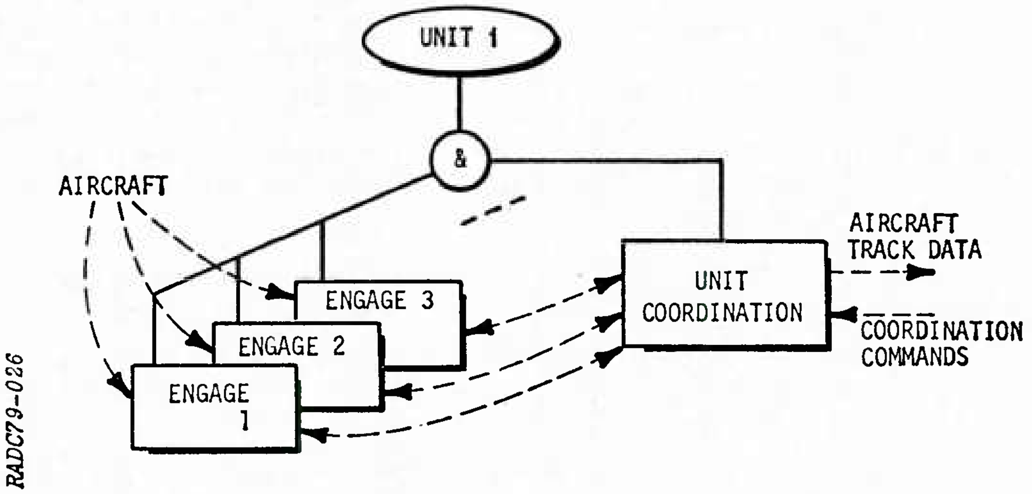


Figure 3-4 Second Level Decomposition

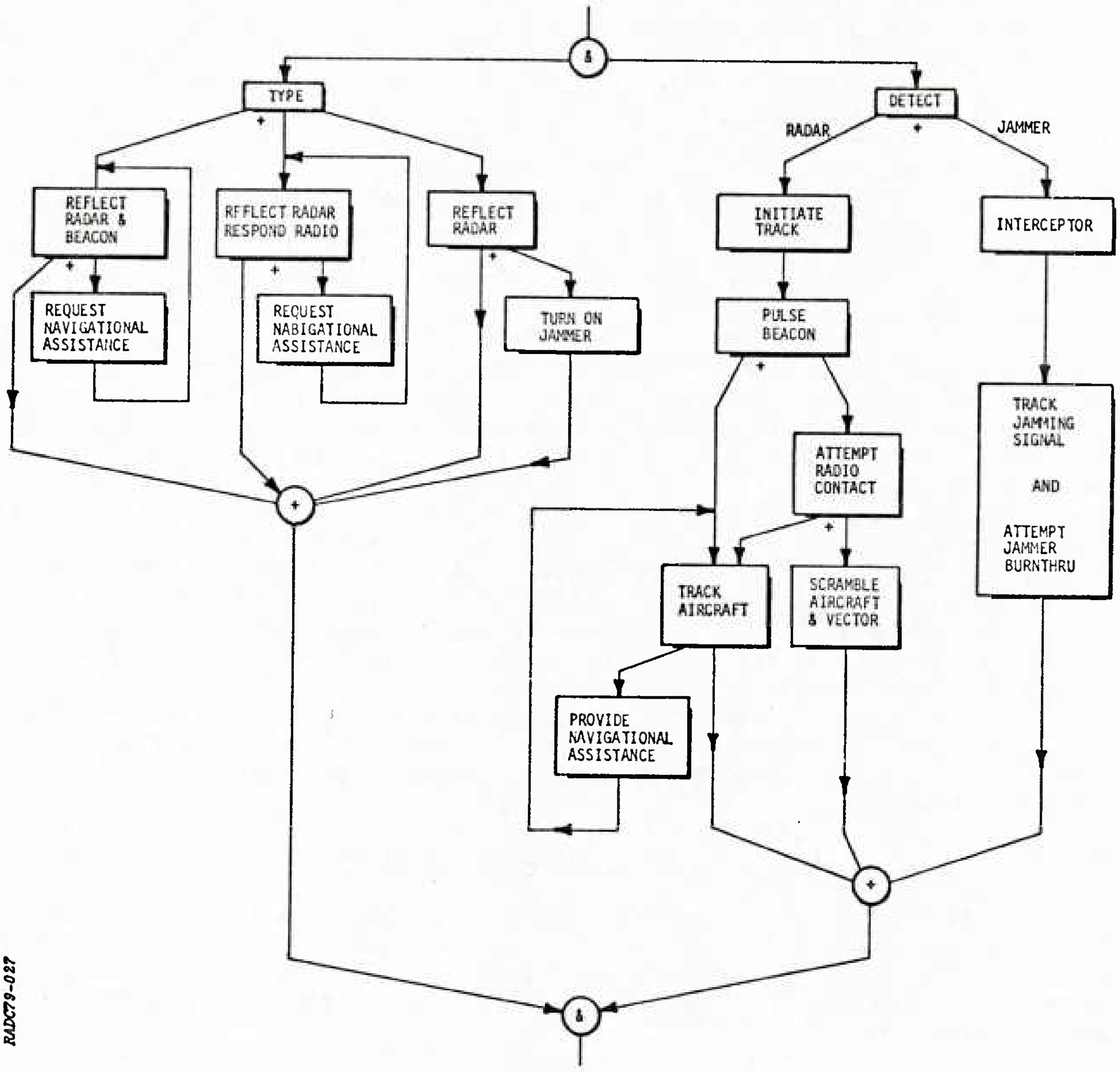

&
TYPE
+
REFLECT RADAR & BEACON
+
REQUEST NAVIGATIONAL ASSISTANCE
RFFLECT RADAR RESPOND RADIO
+
REQUEST NABIGATIONAL ASSISTANCE
REFLECT RADAR
+
TURN ON JAMMER
+
DETECT
RADAR
+
JAMMER
INITIATE TRACK
PULSE BEACON
+
ATTEMPT RADIO CONTACT
+
TRACK AIRCRAFT
SCRAMBLE AIRCRAFT & VECTOR
PROVIDE NAVIGATIONAL ASSISTANCE
INTERCEPTOR
TRACK JAMMING SIGNAL AND ATTEMPT JAMMER BURNTHRU
+
&
RADC79-027

Figure 3-5 Third Level Decomposition

detection (yielding angle but no range information), interceptors are scrambled (if not already scrambled), and the jamming signal is tracked and jammer burn-through is attempted by the radar. Coordination is required at the multi-unit level to triangulate jammer tracks between units.

Note that this logic assumes that hostile aircraft do not have transponders, and can be identified from radio contact. If a hostile aircraft has a normal-appearing transponder, it will be classified as friendly and not contacted, or, if it does not have a transponder and it responds correctly to radio contact, it will be classified as friendly. Thus, if the definition of the threat is to include hostile aircraft with bogus transponders and/or correct radio responses, additional system logic is necessary. A solution to this problem might lie in the area of comparing aircraft locations and transponder identifications with pre-filed flight plans available at the central control location -- this would result in an additional step in the logic to compare locations and transponder locations with planned flights, with additional paths to handle unauthorized deviations from flight plans. This would have the effect of changing the mission to control of air-space rather than simple surveillance.

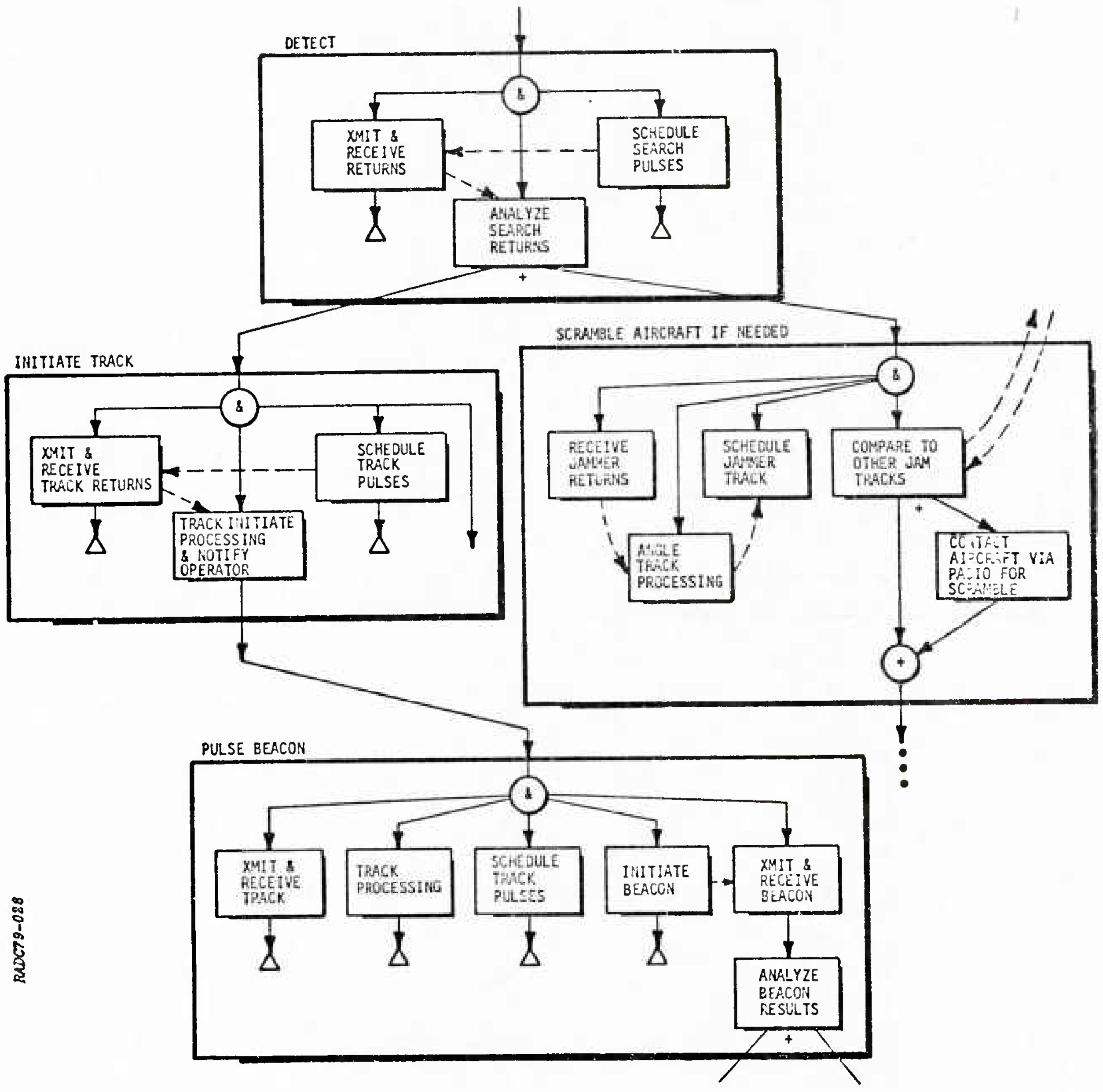
This level of decomposition is sufficient to enable the prediction of system performance as a function of the system parameters of the system actions, e.g., probability of detection as a function of range and radar cross-section, track accuracy as a function of time and track rate, time to accomplish beacon interrogation and analysis, etc. The creation of such a simulation is a useful technique for assuring that relevant threat, environment, and system parameters and their relationship to the function and system performance, have been identified. The use of such a simulator requires definition of threat scenarios, threat parameters (e.g., radar cross-section values), site locations, and values of system parameters (e.g., scan rate, time delay to interrogate a beacon and analyze the results), thus identifying feasible ranges of such parameters. The results of the simulations are used to identify feasible classes of solutions, identify typical loading parameters (e.g., a worst case unit must track 100 aircraft simultaneously), and provides information of the sensitivity of system performance to the parameters, thus allowing the identification of the critical system issues.

3.2.4 Step 4 -- Decompose to Allocatable Subfunctions

Figure 3-6 presents a possible decomposition of the system actions into the subfunctions appropriate to data processors, radars, beacons, and operators. Note that this particular decomposition explicitly assumes a "schedulable" radar; a "track-while-scan mode" radar would be decomposed into a different set of subfunctions.

3.2.5 Step 5 -- Allocation and Feasibility Estimation

Figure 3-7 presents an allocation, designated in the upper right hand corner of each subfunction. Note that alternate decompositions are possible: for example, the radar pulse scheduling functions could be allocated to the radar.



RADC79-028

Figure 3-6 Example Decomposition to Allocatable Subfunctions

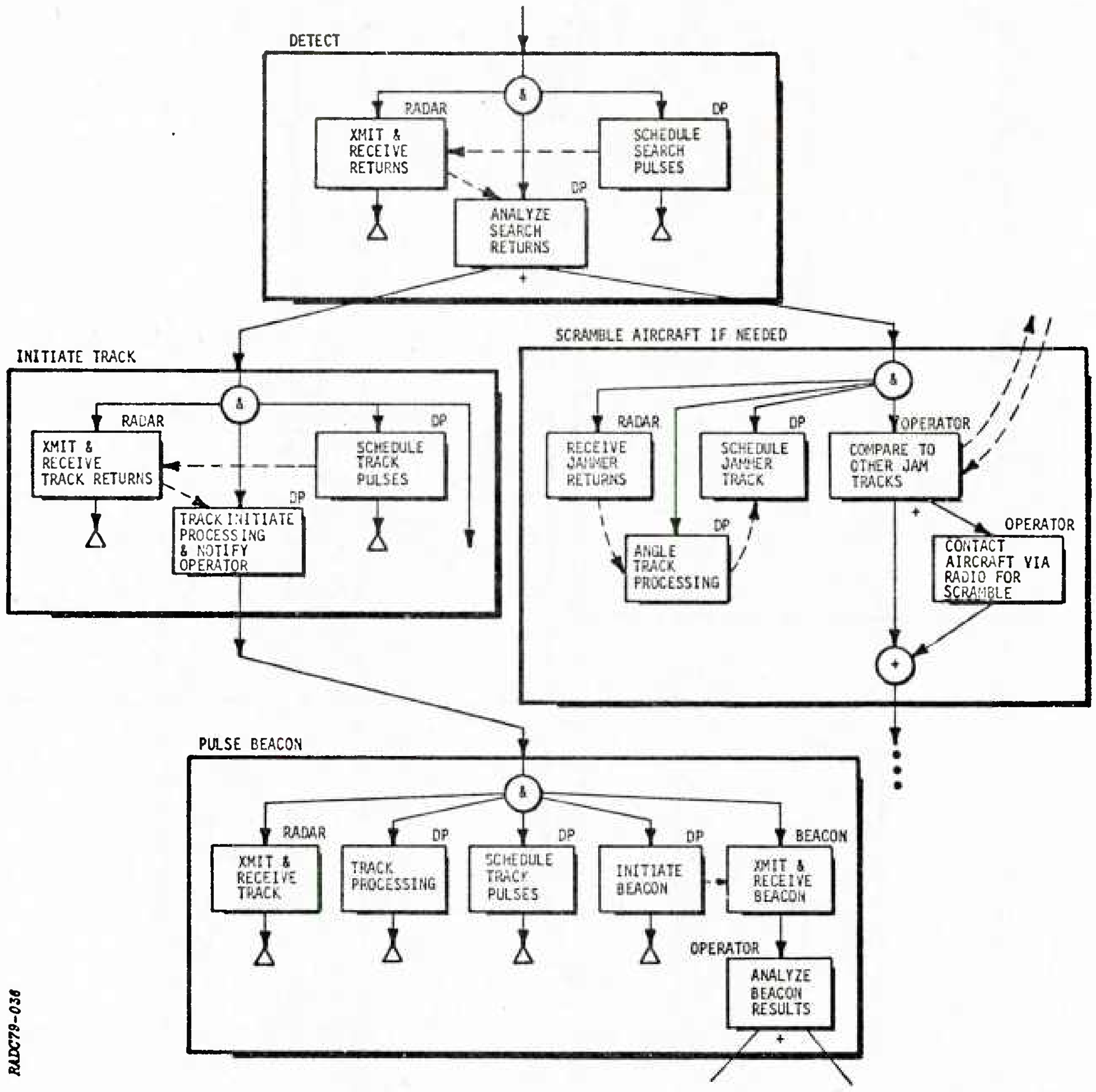


Figure 3-7 Example Allocation to Subsystems

Until now, the methodology being described is common for all subsystems. At this point, the feasibility and resource requirements for the subsystems are to be determined. Estimation techniques are the proper study of the various technology areas (e.g., radar, weapons). For the data processing subsystems, the estimation of feasibility and resource requirements has been based on simplistic measures of MIPS (Millions of Instructions Per Second) and estimates of memory size. In today's technology of PROMS, ROMs, federated microprocessors, bubble memories, etc., this technique has been found to be too simplistic. For example, a problem requiring 100 MIPS might require a 100 MIPS serial processor, or may be decomposable into the independent execution of around 100 1 MIPS microprocessors -- the latter is clearly feasible, while the former is beyond the state-of-the-art. To close this gap, we offer the following methodology for the data processing and communications (DP/C) estimation.

3.2.5.1 Step 5a -- Allocate Subfunctions to DP/C

In this step, the subfunctions to be allocated to the DP/C network are identified. The data processing and communications subsystems are combined at this point to be able to later perform local tradeoffs between the data processing facilities and the communications capabilities (e.g., increasing data processing load to pack information and decode it at the other end of the link will decrease required communications transfer rate).

3.2.5.2 Step 5b -- Summarize DP/C Requirements

In this step, the required data processing and communications are summarized in the form of data flow diagrams which abstract out all of the precedence information by combining like functions for each of the objects being engaged. Figure 3-8 presents such a data flow diagram. Note that the functions of SORT and MERGE have been added as a consequence of assuming that a single radar will be used to transmit the radar pulses and process the returns (an alternate interface assumption might be to separate the search returns, track returns, and jammer returns into separate channels).

Note that the data flow diagrams are derived from the previous allocation of system subfunctions (which decompose the system logic); they are not simply asserted as today's technology does. This provides an important link of traceability between the system design and the initial requirements for the data processor and communications subsystems.

3.2.5.3 Step 5c -- Identify Maximum Dimensionality Architecture

In this step, analysis or simulation is used to identify the maximum number of processors which could be used to perform the data processing, and the maximum number of communications links which could be used to transfer the data among the geographical locations. The purpose at this step is to preserve the dimensionality of the problem to assess the feasibility of using some type of parallel data processing architecture to solve the problem. In this way, the potential parallelism of the data processing architecture is derived from the parallelism of the problem to be solved, rather than trying to analyze it

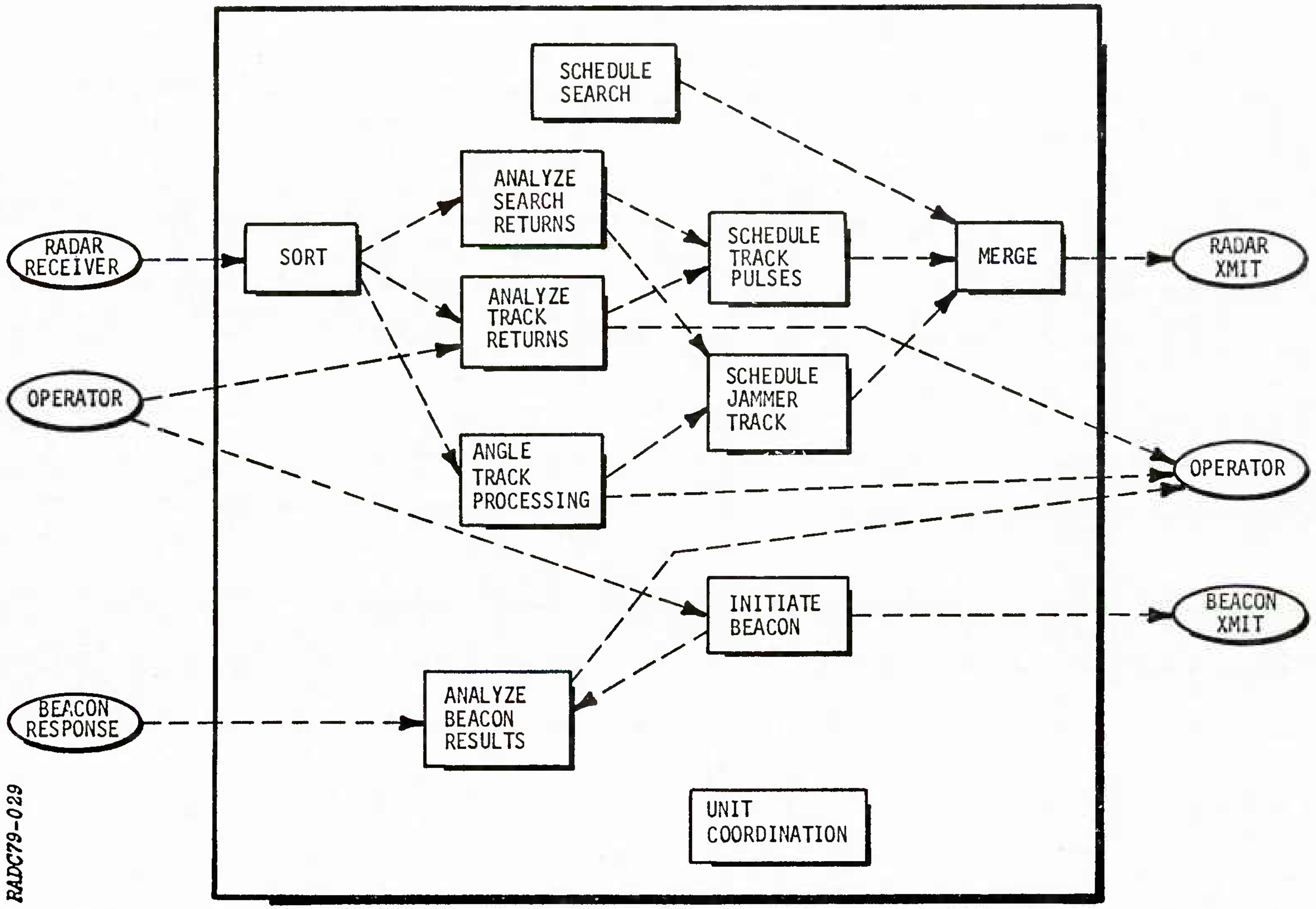


Figure 3-8 Example DP Allocation

after the problem has been stated in serial terms. This allows the identification of applications of vector processing, Single Instruction Multiple Data architectures, etc.

The approach to discovering the maximum dimensionality is to combine the estimates of the parallelism of the processing with the parallelism of the subfunctions, and the estimates of the load, and the allocated response times. These factors combine in the following way:

- If a processing subfunction has an inherently parallel transformation (e.g., compare an aircraft position against those of all aircraft in track), the processing subfunction has a "natural dimensionality".
- Depending on the scenario, there is a maximum number of objects in each phase of the engagement.
- The total dimensionality of each phase depends on the total number of objects and the required response times, e.g., if 100 objects are in track, but the response time is 1 millisecond for tracking, then the track dimensionality is one, (i.e., only one processor could be used to do track processing).

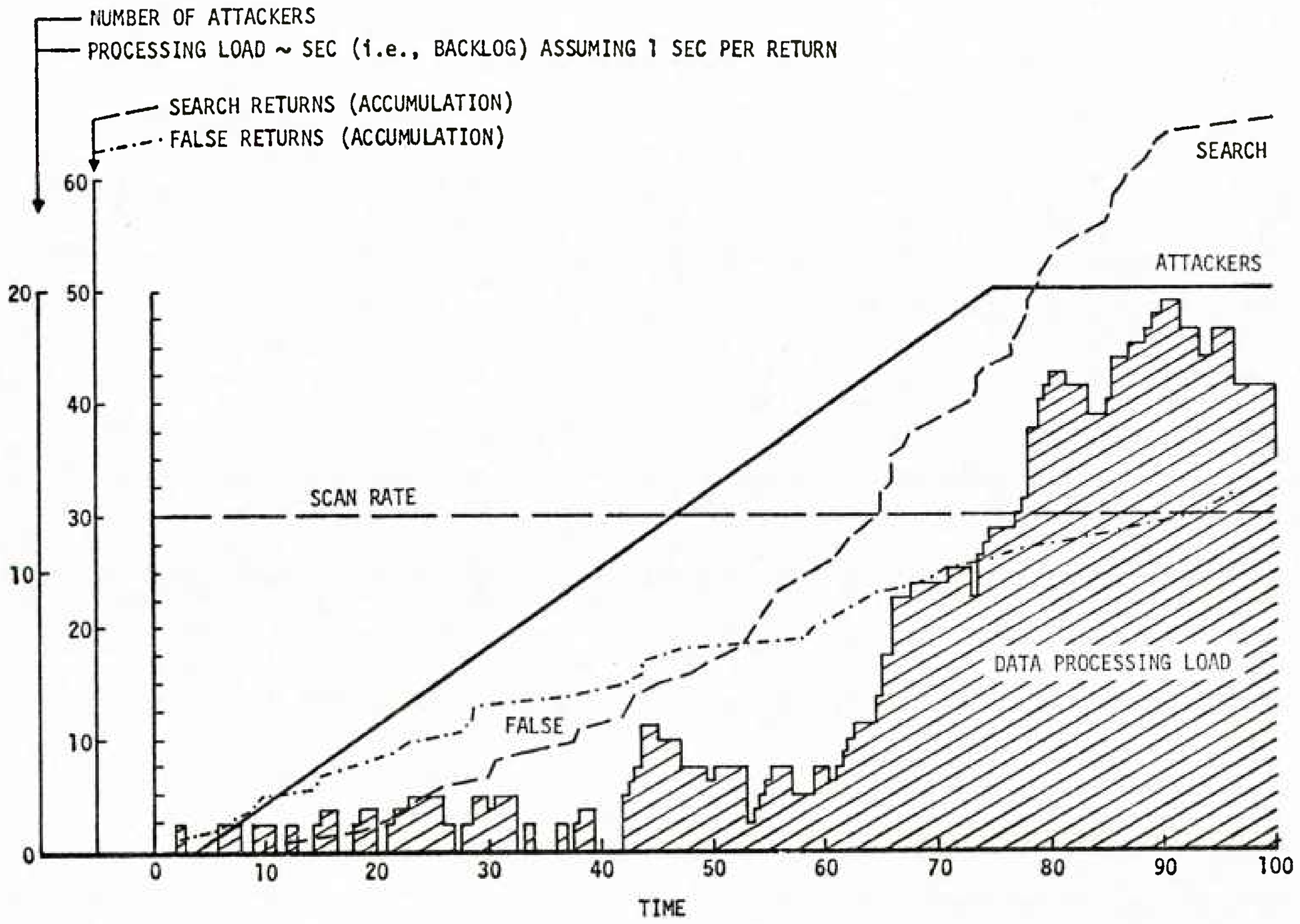
Figure 3-9 presents the results of such an estimation using a simulation. For a specified scenario, Figure 3-9 presents the time history of the search rate, number of aircraft in the search volume, number of search returns (both total and false), and the number of objects to be tracked. This information is used to estimate the maximum dimensionality of the data processing and communications by establishing the maximum number of objects in each phase of the engagement versus time.

Figure 3-10 presents the results of such an analysis. Assume the following:

- All subfunctions have inherent dimensionality one.
- Threat scenarios are not allowed to contain aircraft flying in formation.
- A maximum of 100 aircraft are visible at any time.
- Ten second search scan time.
- Track rate of one per second per aircraft in track.
- Search and track response times allocated to the data processor are 20 milliseconds.

Under these assumptions, Figure 3-10 presents the maximum number of processors which could be used to accomplish each subfunction. This results in a total of 29 processors and is derived in the following way: assuming 100 aircraft in track at one track return per second, with a 20 millisecond response time, a maximum of five track returns could be processed simultaneously and still meet response time objectives, and 3 search returns, one jammer track, and one beacon analysis. Because radar returns are coded, as many as nine processors might be used to perform the sort (i.e., one for each simultaneous

ASSUMPTIONS: CLOSED LOOP SYSTEM.
A SEARCH RETURN MUST BE PROCESSED BEFORE ANOTHER RETURN IS RECEIVED FROM THE SAME TARGET (i.e., A PROCESSED SEARCH RETURN PROVIDES THE RADAR COORDINATES FOR ANOTHER RETURN (TRACKING).



SGD78-020

Figure 3-9 DP Load Versus Time

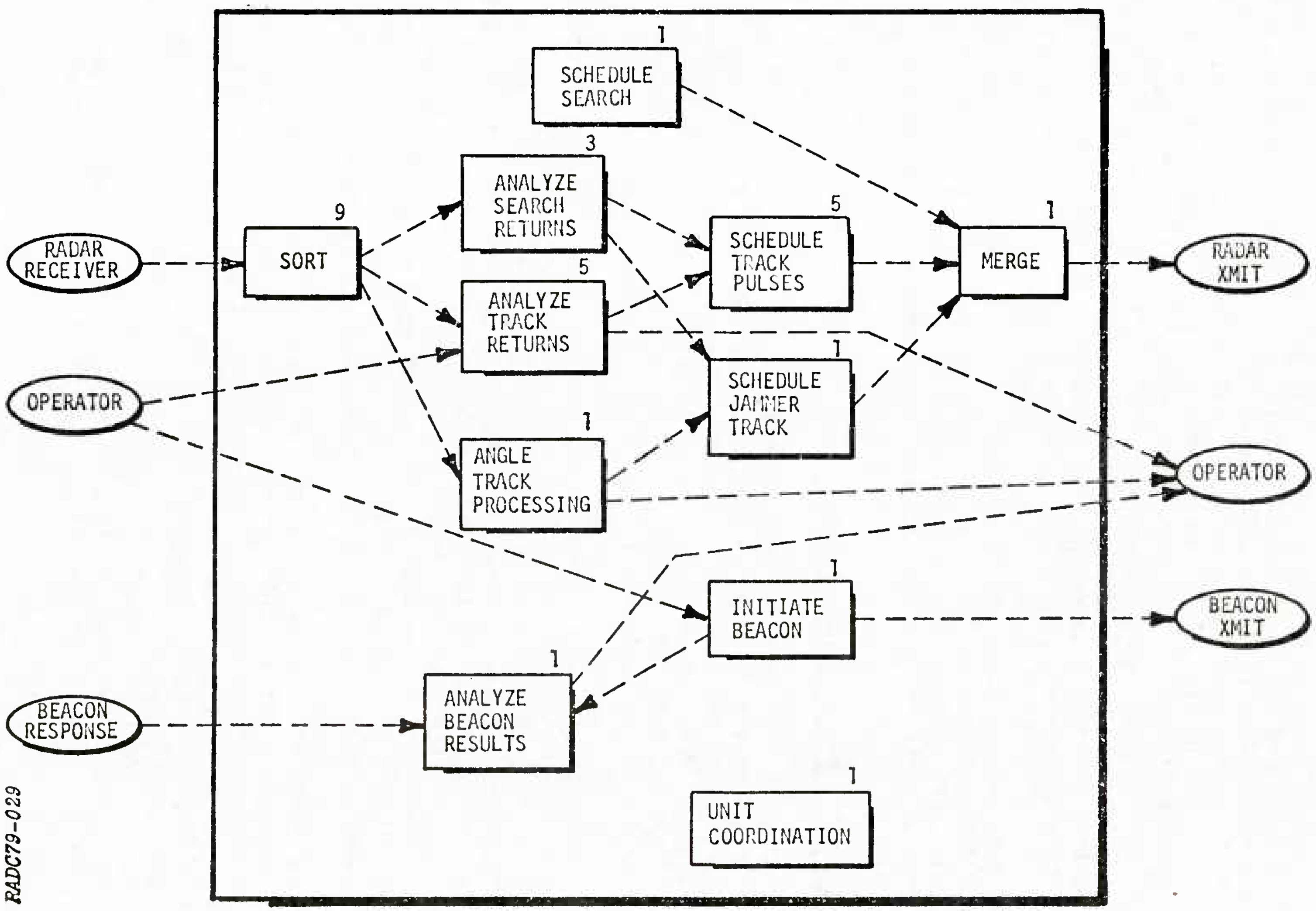


Figure 3-10 Example Maximum DP Dimensionality

return). Only one processor can be used to schedule the search pulses and merge the track commands together, and perform unit coordination.

Further analysis along each path (e.g., analyze track returns and schedule track must be performed within 20 (milliseconds) reduces this number to 14 processors: one for scheduling, one for merging, three for searching, five for analyzing track returns and scheduling track pulses, one for angle track, one for sorting, one for beacon scheduling and analysis, and one for coordination. This strongly suggests that vector processors and parallel processors with a single instruction stream (which generally require 20 or more parallel streams of data to process in parallel to be efficient) are not applicable to this problem, thus ruling out whole classes of data processing architectures. However, multi-processors or federated processing architectures are still feasible.

This information is important for identifying the classes of architectures for which estimates of processing time for each subfunction is necessary -- estimates of processing time for vector processing is estimated differently than for serial processors, and these estimates are needed to perform the data processing sizing.

3.2.5.4 Step 5d -- Estimate DP/C Loads

In this step, the processing requirements for each unit are projected using estimates of required processing for each subfunction, and communication requirements are estimated using estimates of the size of each information transfer between subfunctions and the allocated response times. To include all of the engagement interaction effects, this is best done in a simulation in which resource utilization for each system function is identified.

The average data processing load is not sufficient for processor sizing. If a processing step requiring 20,000 instructions is executed once per second, its average rate is 20,000 instructions per second (0.2 MIPS). However, if this subfunction has a response time of 1 millisecond, then the required rate is 20 MIPS -- this shows the effect of response time requirements on required data processing capacity. Similarly, if one message having 1000 bits is required to be sent once per second, this requires an average communication capacity of 1 kilobit per second; but if this must be sent in one millisecond, this results in a required capability of one megabit per second.

Table 3.2 presents an example analysis of the required average and peak data processing load. The columns indicate the average execution rate of the processing functions (e.g., number of search pulses, number of track returns obtained from simulations), the number of serial instructions per execution (obtained from Data Processing engineers knowledgeable in the state-of-the-art), and the resulting average MIPS. The right-hand side of the table includes the peak number of executions required in the indicated response time, which results in the calculation of the peak MIPS. Totals are included for convenient reference. Note that for this case three kinds of critical issues are identifiable:

- The peak instruction rate is due to track returns processing, thus the number of objects in the surveillance volume and track rate are critical drivers of DP capacity.
- The effect of the 20 millisecond response times is to more than double the required data processing capacity -- from 0.4 MIPS to over 1.0 MIPS. Reducing the response time (either by reducing the overall requirement or by reducing the DP allocation) would reduce the required DP capacity.
- If the estimate on track processing rate were off by 25 percent, the required DP size would have to be increased almost as much. On the other hand, estimates on jammer track could be off by a factor of two without changing the DP size significantly.

From this level of information, it is possible to identify critical issues. If the estimate of tracking instructions is firm, if the estimate of number of aircraft to be tracked is firm, and if the response times are not reduced, then the processing can be performed by a serial processor of approximately 1 MIPS capacity -- this is on the ragged edge of technology, but appears to be feasible. Candidate architectures to accomplish this are identified next.

Table 3.3 presents an analogous analysis for the communications sizing. If the number of bits per track message is 200 bits, this results in an average communications rate of 20 Kilobits per second (KBS). However, if there is a requirement to pass this information to the central control point in 10 milliseconds, this results in a peak rate of 100 KBS. Other subfunctions communicating with central control include the jammer track and others with communications at a much lower rate which are included in the miscellaneous estimate. Thus the average communication rate is 21.5 KBS and the peak is 111 KBS. Since the human perception time is close to 100 milliseconds, and the track rate is one per second, if the response time were relaxed to 50 milliseconds, the peak communications rate would reduce to the average of 21 KBS. This communication rate is feasible with today's technology.

3.2.5.5 Step 5e -- Postulate DP/C Functional Architecture

In this step we identify feasible architectures for the data processor and communications subsystems which are consistent with the above loads and parallelism. From this, rough approximations of costs, deployment schedules, power requirements, etc., can be derived. This is done by consulting a data base of existing and proposed subsystems.

For data processing, a MIL-STD processor with serial processing capacity of over 1 MIPS is available from RCA (ATMAC has an advertized capability of 1.4 MIPS at a cost of $35K). At this price, one could afford to have two spares for a cost of about $100 K. If this option were not available, a next choice might be a 4PI/MLI processor having about 0.38 MIPS for a cost of about $30 K each. Since the peak processing requirement has dimensionality of about nine, a federated architecture containing three or four of these processors appears to be feasible, with reconfiguration logic to enhance the reliability of the combined set. Thus the hardware would cost between $100 K and $150 K per unit.

Table 3.2 Expected DP Instruction Load

	AVERAGE NO.		INSTRUCTIONS	MIPS (AV)	PEAK NO	ΔT	PEAK MIPS
SEARCH	300	x	100	30,000	—	—	30,000
SEARCH PROCESSING	12	x	1200	14,400	3	.02	180,000
TRACK	100	x	3000	300,000	5	.02	750,000
JAMMER TRACK	5	x	1000	5,000	1	.02	50,000
COORDINATION	1	x	50,000	50,000	—	—	50,000
				399,400			1,060,000

ADP79-154

Table 3.3 Expected Communication Loads

	AVERAGE NO	BITS	AVERAGE KBPS	PEAK NO	ΔT	PEAK KBPS
TRACK	100	200	20	5	.01	100
JAMMER TRACK	5	100	.5	1	.01	10
MISCELLANEOUS	1	1000	1	1	.01	1
			21.5			111

RADC79-037

The communications system to carry a load of 100 KBS could be built using RF links with repeater stations (on the ground or via satellite) at a cost in excess of $400 K per unit. If the load were more like 20 KBS, then standard telephone links might be possible for appreciably less. This is to be compared to radar and facilities costs of perhaps $1 Million per copy.

3.2.6 Step 6 -- Identify Critical Issues and Resources

In this step, the critical issues are identified. The critical data processing issues have been discussed -- the processing appears to be feasible under the worst conditions, but increases in estimates of the instructions required for tracking or decreases in the required response time allocation would be critical issues. Under these conditions, the data processor hardware does not drive feasibility, cost, schedule, power, or any other resource constraints. Various serial and distributed processing architectures are applicable.

Communications appear to be feasible but critical; response time allocations are definitely critical, and would require more analysis of alternate configurations.

Note that additional data processing and communication requirements will stem from the design of the fault diagnosis, fault isolation, and reconfiguration requirements for the facilities, radar, communications equipment, and data processing design. If high reliabilility were a preference factor (e.g., Mean Time To Failure of 1000 hours for any component, Mean Time To Failure for the system of three years, Mean Time To Repair of 1 hour), then the detailed radar design and test equipment would have to be developed and the test procedures would have to be developed to complete the data processing requirements. This would probably not be a large DP load, but would require a large amount of instructions. In this case, the development of the software would then become a critical item in both development cost and, particularly, development schedule.

3.2.7 Step 7 -- Identify Resource Management Rules

In this step, rules to allocate the critical resources are identified -- this includes how the resources are to be allocated (e.g., when radar resources approach maximum, reduce load by reducing track rates on objects in track), and any additional system logic paths (e.g., if load is at maximum and track rate per object is at a minimum, drop track on objects near to exiting the track volume). These rules become incorporated in the system logic by decomposition of the "coordinate unit" function, and the effect of limitations of critical resources on system performance can be estimated by additional simulations. This provides sensitivity information for later optimization.

3.2.8 Step 8 -- Optimization

In this step, the sensitivity of cost versus capability and capability versus system performance are used to perform cost/performance trades for subsystems. For our surveillance system, such trades include the following:

- Radar pulse rate versus system performance
- Data processor instruction rate versus-system performance
- Allocation of response time between
    - Data processor
    - Communication system
    - Console display
    - Operator.

The costs of different types of decompositions are compared (e.g., track-while-scan versus directed-track radars).

The result of these analyses should be an identification of classes of system configurations which are feasible, and are worthy of further analysis.

3.2.9 Step 9 -- Plan Integration and Test

In this step, the foundations for the development plans are laid, including the identification of the required resources to develop, integrate, and test the subsystem. The necessity for such a step was discussed in the formal foundations -- to account for all of the resources required to develop a system, the resources necessary for integrating the subsystems, developing the test tools, and developing and applying the test procedures must be included in the total resource estimates. These factors will in part be configuration dependent (e.g., the costs of building test tools), but may in part be configuration independent (e.g., a prototype of the system may be built and tested before production is authorized for any configuration).

There may be considerable feedback between the decomposition and allocation of the system requirements and the definition of the integration and test plans. For the case of complex systems, as much as 40 percent of the development effort may be spent in this phase of the system development; thus selection of the system configuration should be influenced by these considerations.

It is noted that in the standard methodologies for systems development, there is little recognition that integration and test requirements should influence the final configuration. Although this step comes last in this discussion, it is clear that such considerations should enter the system development process no later than the allocation and feasibility estimation step, and that the system design is not complete until the feasibility and costs of the integration and test of the system have been considered.

3.2.10 Discussion

Note that this approach surfaces critical data processing issues early in the system analysis phase. The traditional approach to data processing sizing is to focus on the average processing time. The above analysis clearly identifies that the effects of response time are critical, and can be addressed during the system analysis phase. The failure to address the response time issues is partly responsible for the "data processing growth" which occurs during the development of the system. The data processing estimates are usually small to begin with, and when the response time effects are finally incorporated into the data processing design, the data processing size "suddenly grows" by factors of two or more, and the system and subsystem designs are so far along that changes in the response time allocations become painful to implement.

Secondly, note that the data processing requirements and the feasibility estimates are strictly traceable to the system logic by a string of decompositions and allocations. Changes in the system components result in changes to the system logic and lead to traceable changes in the data processor sizing estimates. This strict traceability is a new feature of front-end system design, and is a fallout of the formal definition of decomposition and allocation.

Finally, the inclusion of integration and test requirements into the preference relationships should highlight the relationship between allocation of data processing to subsystem components and its impact on integration and test. For example, if one allocates all radar scheduling to the data processor, special purpose scheduling software must be developed to be able to test the radar or the testing might be delayed until the data processor was completed in order to test the maximum tracking rate of the operational system. This might heavily influence where the scheduling software was allocated.

3.3 SYSTEM ENGINEERING

The system engineering phase has the same methodology steps as the system analysis phase -- starting with the mission requirements and the candidate subsystem classes from the system analysis phase, identify requirements for the subsystems. The critical differences are in intent and depth: the intent of the system analysis phase was to identify feasible classes of solutions, while the system engineering phase has the goal of finding the best configuration (where "best" is with respect to all of the preference rules), freeze the allocation of the requirements to the subsystems, and definitize the integration and test plans. This means that fewer gross classes of configurations will be addressed, but the system logic will be scrubbed to identify all possible paths (including the failure mode paths), alternate decomposition and allocation decisions will be examined to assure the most preferred boundaries between the subsystems, and alternate subsystem configurations will be analyzed to re-estimate the performance and resource requirements for the data processing and communication subsystems.

Thus the same basic nine steps will be followed, but more alternative decompositions, allocations, test plans, and subsystem configurations will be analyzed to assure the most preferred system has been identified. The results will then be incorporated in the equivalent of a Type A Systems Specification with supporting documentation.

3.4 DP SUBSYSTEM ENGINEERING

In this phase, the requirements allocated to the Data Processor/Communications subsystems are analyzed and allocated to hardware (e.g., analog processors, special purpose processors like Fast Fourier Transform boxes, Surface Acoustic Wave Devices (SAWD), special purpose communications hardware including crypto processors, and the general purpose processing nodes). The end result will be the identification of the processing and communication hardware components and software processes which utilize the programmable components. In some cases, classes of components will be identified whose parameters (e.g., number of federated processors) will be later established in the process design. Included in this phase is the tradeoff between analog and digital processing, general purpose versus special purpose processing, digital versus communications capabilities, and the selection of the approach to meet reliability constraints (e.g., fault tolerance versus high reliability components). The same basic nine steps apply.

3.4.1 Step 1 -- Define Mission

The mission requirements for the DP/C subsystems are the system requirements allocated to the subsystems, including functions, loads, and response times. The particular version of these requirements is baselined for analysis.

3.4.2 Step 2 -- Identify Component Types

A larger class of component types are identified for this phase of the analysis than were considered in the systems analysis phase. Thus the whole range of analog devices (including SAWD), special purpose digital devices, special purpose microprocessors, and classes of general purpose processors and components (vector processors, parallel processors, multiprocessors, federated architectures, ROM memories, bubble memories, etc.) are open for analysis. These will be subjected to the preference criteria to limit the range of components considered (e.g., optical processors are not yet feasible, but bubble memories are now available).

3.4.3 Step 3 -- Decompose to System Logic

In this step, the actions of the DP/C subsystems are decomposed to identify their actions with respect to the objects they deal with. Two critical types of objects are the interfaces with other subsystems and the messages crossing those interfaces. Two examples are used to illustrate this type of analysis.

First, consider a requirement to transfer information from point A to point B. Because response time and reliability of transfer are relevant factors, one approach is to simply send the message -- another is to send the message periodically until an acknowledgement is received. Yet another is to send the message along different routes of a communications network and wait for an acknowledgement. There are a large number of possible actions of the system to send the message and assure that it is in fact correctly received -- these are commonly referred to as communications protocols, and each has its own data processing load and reliability characteristics across communication links of specified reliability.

Next, consider the processing of measurements. Depending upon the measurement device characteristics, one might be able to use an analog filter followed by an A/D converter to sample the data, or digitize the data and use the general purpose processor to filter the data using a digital band pass filter. Each of these combinations of components can be used to sample and filter the data, and the system actions differ depending upon the components utilized.

3.4.4 Step 4 -- Decompose to Allocatable Subfunctions

Further decompositions are performed to analyze at a fine level of detail where to draw the boundaries. Thus, message formatting might include encryption, or encryption might follow the message formatting. Both decompositions are feasible, and both could be performed by the data processor or a specialized communications processor (e.g., the IMP of ARPANET).

3.4.5 Step 5 -- Allocation and Feasibility Estimation

This step is where the different allocations are identified, and the resource requirements are estimated. This requires the same type of DP architecture estimation schemes as discussed in the system analysis phase, but with an expanded scope (e.g., costs of SAWD along with reliability, development schedules, etc., must be available if such devices are to be considered).

3.4.6 Step 6 -- Identify Critical Issues and Resources

As before, critical issues and resources are to be identified. For the case of our example, communications resources to accomplish communication of information within the required response time might be a critical issue.

3.4.7 Step 7 -- Identify Resource Management Rules

As new classes of devices and their functions are introduced, new critical resources are identified, and new resource management rules are needed. In our example, if communications resources are identified as a critical issue, then a priority scheme might be introduced to assure that high priority messages meet their response times by slightly delaying the other non-critical traffic.

3.4.8 Step 8 -- Optimization

In this step, the different configurations are compared using the system preference rules to select the best subsystem configurations. This in turn is used to update the feasibility estimates and sensitivities back to the system engineering level to assure that the proper choices have been made -- if not, this will call for a further iteration at the system engineering level.

3.4.9 Step 9 -- Plan Integration and Test

In this step, the impacts of testing the combined subsystem are analyzed in terms of the integration and test resource requirements. These elements are factored into the optimization process to assure that the preferred solution is testable and has test resources included in the total development resource requirements.

3.4.10 Discussion

The consideration of alternate designs incorporating special purpose analog and digital devices in addition to more exotic data processing architectures is a necessary one to assure the subsystem designs have considered the classes of alternatives available with the current state-of-the-art. The identification of the alternatives considered is the proper subject of a design review to assure that a wide enough set of subsystem components have been considered to yield a preferred design.

The completion of this phase may require significant interaction with the software requirements, distributed processing design, and process design phases to assure the feasibility estimates. When completed, however, those efforts must take their requirements from the baseline requirements output by this phase.

3.5 SOFTWARE REQUIREMENTS ENGINEERING

In this phase, the requirements allocated to the Data Processing/Communications subsystems are refined into testable stimulus-response relationships. Figure 3-11 illustrates this type of analysis.

Figure 3-11a presents the subfunctions allocated to the data processor for each object engaged related to tracking. The track processing function is decomposed in Figure 3-11b into an iterated function which processes a single track return from the object. Since it is always possible to describe a function with sequential inputs in terms of a repeated function with a single input, this decomposition is always possible.

Figure 3-11c presents a partial path of an R-Net which processes a single track return for a single object. When a track message arrives from the radar, it is converted to R-A-E coordinates, its quality is assessed, and if good, the track state estimate is updated. Note that the last track estimate must be stored between pulses for each object in that phase. The usage and quality of the track are used to determine whether another track pulse is to be sent (the time of which depends on the track rate assigned by the unit coordination function), or the object is set to an "exit" status to exit this function.

Figure 3-11d presents an R-Net fragment which integrates these paths into other paths. When a message arrives from the radar, other radar messages are sorted out and only track returns are routed down this path. The specific track command information is accessed to obtain the information to translate the return information (e.g., time of return, amplitude, off-axis azimuth and elevation) into range, azimuth, elevation, and radar cross-section. Objects in other states are routed to other paths (e.g., in track initiation), and the track information for this specific object is accessed, denoted by the

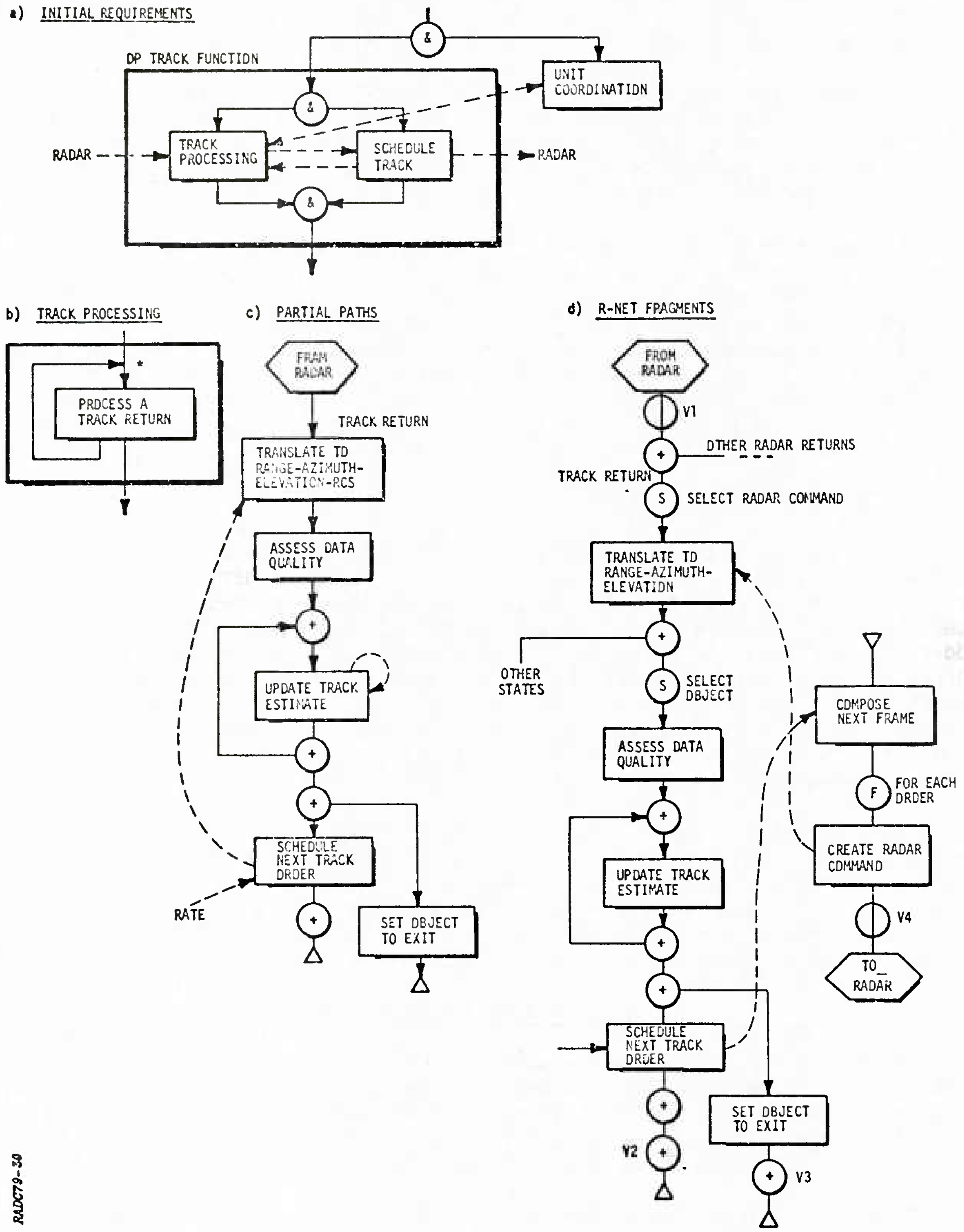


Figure 3-11 R-Net Fragment Derivation

select node, (S). The R-Net for composing a frame of radar information is identified, and the radar commands are created and transmitted to the radar and saved for later retrieval. Validation points are appended and performance requirements are written in terms of data accessible at the validation points.

Note that these requirements are fully traceable to Figure 3-11a via two levels of decomposition and three levels of integration (integrate over all objects, all phases, and all return message types). This results in the R-Nets which form the key to the Software Requirements Engineering Methodology (SREM) technique of specifying processing requirements.

The requirements engineering phase thus starts with the output of the Data Processing Subsystem Engineering phase which identifies the digital processing to be performed in terms of functions, inputs, outputs, interfaces, performance requirements and loads, and decomposes the processing to the stimulus-response requirements for the DP/C subsystems as a whole. This is equivalent to Step 3 of the overall methodology for decomposing the initial requirements into the system logic: these requirements must be met no matter how the processing is distributed between the processing nodes and the communications subsystem. The actual allocation is addressed in the next phase.

3.6 DISTRIBUTED PROCESS DESIGN

This step was not explicitly identified in Part II as a necessary phase of the front-end system development, but was included as part of the overall process design. The distributed process design is called out here as a special step in order to emphasize the importance of the decisions -- this phase addresses the issues of balancing the data processing between geographically separated nodes, tradeoffs between the data processor and communications capabilities, issues of system vulnerability and reliability as affected by distribution of processing and data base among the data processing nodes to result in the identification of testable requirements for the processing nodes and requirements on the individual communication links.

Step 1 of this analysis identifies the baseline requirements from the data processing subsystem engineering phase. Step 2 corresponds to the identification of the classes of data processing and communications subsystems left open in the data processing subsystem engineering phase. Step 3 corresponds to the definition of the R-Nets in the software requirements engineering phase. We now go into the methodology proper.

3.6.1 Step 4 -- Decompose to Allocatable Subfunctions

The R-Nets specify all processing to be performed by a combination of the Data Processor/Communications network and forms the basis for testing such networks. These R-Nets are partitioned to form packages of processing allocatable to subsystem classes. The subsystem classes in this case are the DP nodes and the communication links. Figure 3-12 illustrates this process.

Figure 3-12a presents a fragment of an R-Net which processes search returns, sorts out the images of objects already in track, and prepares to put the new objects into track. The comparison of the new detections to

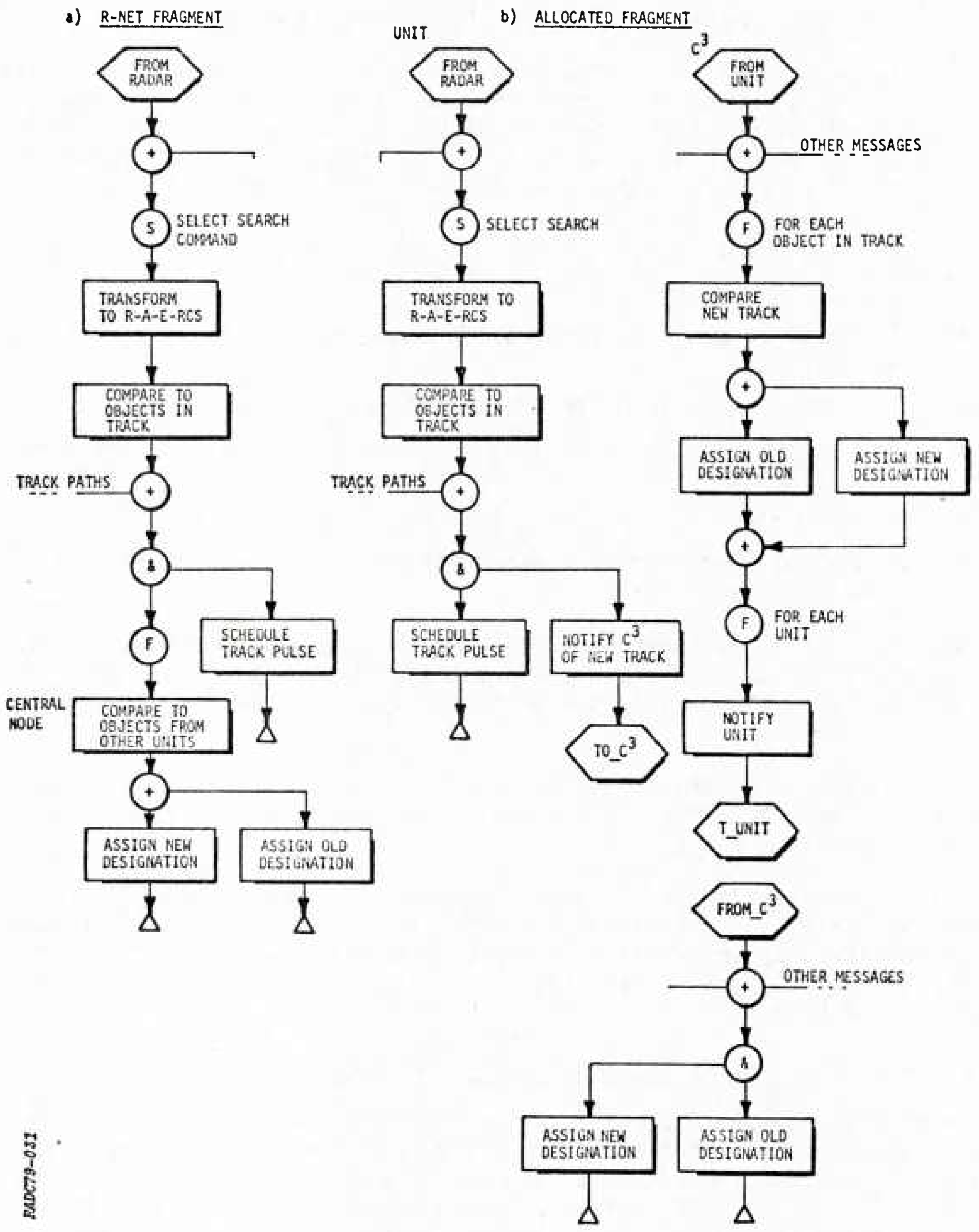


Figure 3-12 Example Allocation to Nodes

objects in track by other units could be performed at the unit or at the central coordination point: in this case the comparison is allocated to the central node, while the others are allocated to the unit node.

3.6.2 Step 5 -- Allocation and Feasibility Estimation

In this stage, candidate allocations of processing steps and data are made to the data processing nodes -- this results in an allocation of communication rates to communication links. Several different allocations are possible, resulting in different R-Nets for each of the nodes. In addition, the response times for the process of a whole are allocated to the nodes and links.

Figure 3-12b presents a resulting allocation of stimulus response relationships to:

- The Unit Processor -- process the data and form a message to the central node.
- The Comm Link -- transfer the message to the central node.
- The Central Node -- accomplish the comparison and form messages to each affected unit.
- The Comm Links -- transfer the messages to the units.
- The Units -- update the information base at the units with the correct designation.

Note that this level of information is necessary to completely size the data processors at the nodes and the communication links between them. The trade-off analyses can utilize the DP PERCAM type of simulations to great advantage.

The end result of the analysis is a set of R-Nets for each node which are traceable to the R-Nets for the network, and a set of definitized communications link requirements.

For each allocation, a feasibility estimate is performed to yield estimates of cost, schedule, vulnerability, and other preference factors. This may require considerable interaction with the process design activity to establish feasibility of meeting response times for all nodes by performing a process design for the node.

3.6.3 Step 6 -- Identify Critical Issues and Resources

In this step, the critical issues of the distributed design are identified. Critical issues include the effect of distribution of response times to the nodes and links, and their effects or overall data processing and communication costs. Critical resources particularly include the communication rates, and the distribution of the data base elements among the processing nodes and the impact of this distribution on system vulnerability, and the design of the communication network and its relationship to cost and vulnerability.

3.6.4 Step 7 -- Identify Resource Management Rules

In this step, the rules for managing the critical resources are identified. Communication protocols and priority rules for message transmissions are defined to control communications. Schedulers for the processors are defined using the tools and techniques of process design to message processor resources. This may result in the identification of additional logic (e.g., additional R-Net paths to define the protocols), reallocation to the nodes and links, and re-estimation of feasibility.

3.6.5 Step 8 -- Optimization

In this step, the different designs are compared using the preference rules. This is the place where the performances and resource requirements of the different allocations and the best designs of each class are compared to yield the "best" configuration, determine the sensitivity to the assumptions, and validate the previous level of feasibility estimation.

3.6.6 Step 9 -- Plan Integration and Test

Again, the effects of integration and test are factored into the optimization. In particular, the relationship between the tools to test the subsystems and the tools to test the integrated data processing/communications network need clarification.

3.6.7 Discussion

The techniques to perform distributed process design are new, and the proper subject of further research. The above methodology outlines the appropriate steps, but the definition of the performance indices for survivability and reliability, and the tools and techniques to identify allocations and evaluate alternatives require further development.

3.7 PROCESS DESIGN

The process design for a processing node addresses the definition of the schedulable units of software, the allocation of these units to processors, and the definition of the scheduling techniques to assure satisfaction of the functional and performance requirements, especially the response times. The requirements are the R-Nets for the node: "subsystem classes" are the peripheral processors (to handle the input/output), the executive, the applications code modules and the scheduler. Figure 3-13 presents a partial set of modules for our example problem:

- A radar input handler to input all radar returns and store them in memory.
- A radar output handler to output all radar commands from a queue in memory.
- A scheduler to schedule the next applications task.

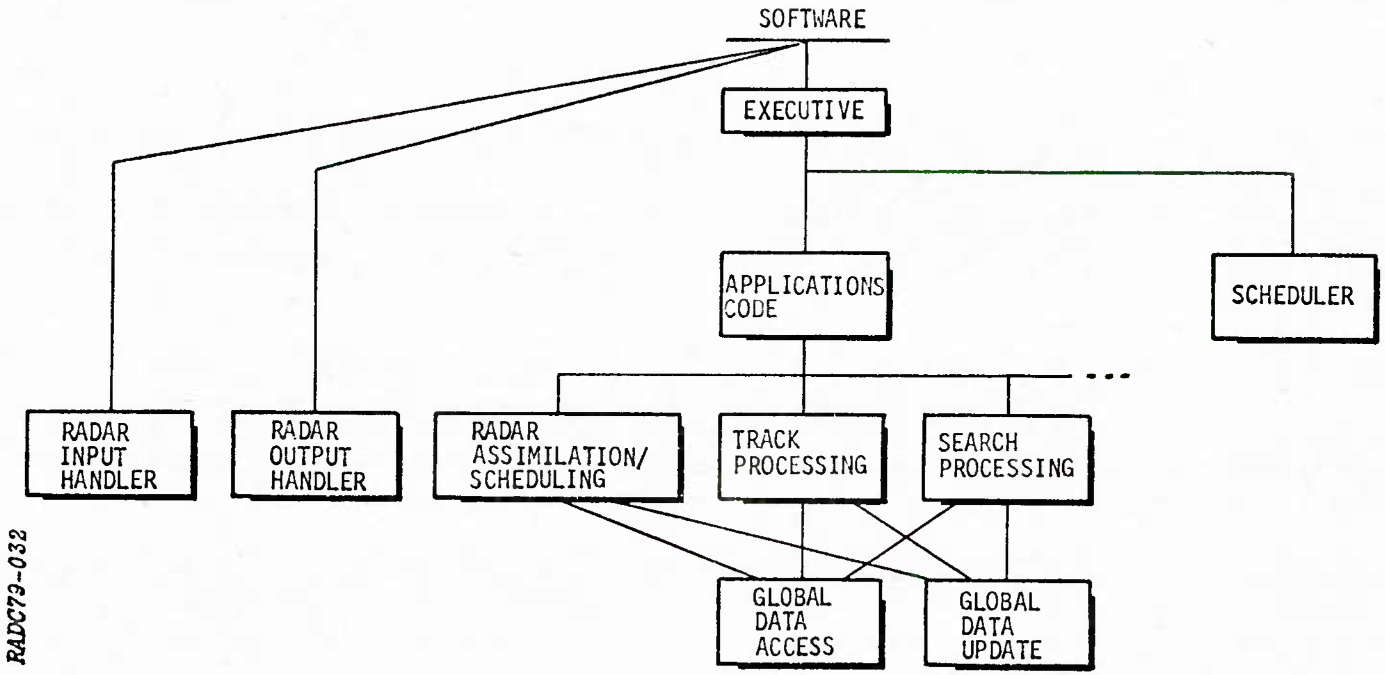


Figure 3-13 Example Process Design

- Application task modules are defined to:
  - sort all radar returns and schedule new commands -- these functions are merged into one module due to the close data interactions.
  - perform track processing.
  - perform search processing.
- Operating system utilities to control access and update of data communicated between modules.

The same general methodology is followed.

3.7.1 Step 1 -- Define Mission

The applicable R-Nets are identified, along with the stressing design scenarios.

3.7.2 Step 2 -- Identify Component Types

The classes of "subsystems" include the software modules on peripheral processors, operating system modules for the executive. I/O handlers, data access/update, and the applications code modules.

3.7.3 Step 3 -- Decompose to System Logic

In this step, the R-Net processing steps are expanded to identify additional requirements, protocols, access and storage and data queues applicable to specific machine architectures.

3.7.4 Step 4 -- Decompose to Allocatable Subfunctions

In this step, the R-Net processing steps are decomposed to show the access update of data, and algorithms are decomposed to "chunks" with processing times small enough to be allocated to modules.

3.7.5 Step 5 -- Allocation and Feasibility Estimation

In this step, candidate allocations are made to classes of modules, execution times are estimated (including data access/update times for specific data storage structures), and total DP laods and response times are predicted for the design scenarios. Because of the complexities involved, some sort of analytical load analyzer (e.g., ALF) and/or simulator is useful to aid in such predictions. A variety of scheduling techniques (e.g., priority schemes, polling schedulers, and use of pre-emption) may be necessary to achieve all response times for a given allocation. Multiple allocations to different machine sizes are also explored.

3.7.6 Step 6 -- Identify Critical Issues and Resources

In this step, the large contributors to DP load and the driving factors for satisfaction of response times are identified. Processing time on modern processors can be divided into three categories: idle time (processor is doing nothing), load time (processor is loading a task to be executed into executable memory) and execution time (processor is actually transforming the data). The load time is generally a function of the task and scheduler design, while the execution time is load dependent. The whole trick to process design is to meet all response times while making the load time as small as possible: efficiency is obtained by processing many instances of a task at a time in order to spread out the load time over many instances; this results in a long queue time waiting for enough instances to arrive, and results in a longer average response time for each instance. Short response times require more frequent execution, thus increasing total load time; in time-sharing systems design this is known as thrashing.

The critical issues step identifies the response time and load characteristics which make the scheduling difficult. Other critical resources may include I/O channel times, memory, etc.

3.7.7 Step 7 -- Identify Resource Management Rules

In this step, the data access rules to prevent deadlock, lockout, and the rules for allocation of data to memory modules are formulated. Detailed simulations may be necessary to validate their operation. The precise scheduling algorithm constants for various design scenarios may be necessary to demonstrate feasibility. Special failure mode identification and reconfiguration rules are formulated to meet reliability requirements.

3.7.8 Step 8 - Optimization

In this step, different allocation approaches are considered and a "best" allocation is selected. This allocation will then be the initial one for the preliminary design step. This includes optimization over allocations to several candidate hardware configurations, and includes preference factors of growth, graceful degradation, etc.

3.7.9 Step 9 -- Plan Integration and Test

Part of the design job is the identification of the design verification techniques. A persistent problem for load testing is that of recording data required to measure data processing resource utilization -- thus the tactical process is difficult to observe without disturbing its operation. The specific scenarios for load testing are identified, and the techniques for inputing the scenario, recording the results, and analyzing the results for satisfaction are identified. This results in a set of requirements for the test tools including general utilities for data logging (generally allocated to the operating system) and post-process analysis.

These tools are generally ignored until they are needed for test -- and then they are needed badly.

3.7.10 Discussion

The techniques of process design for real-time processes are just emerging into the general state-of-the-art. Tools like ALF have been found to be indispensible for performing design studies, and then the designs are later validated with design simulations. This allows the design analyses to be performed with quick-reaction tools, and validated with the high fidelity, longer running simulations.

The techniques for process design of non-real-time software concentrate on efficiency and wall-clock-time to complete a normal run. These techniques are discussed in the literature (e.g., Don Knuth's Empirical Study of FORTRAN Programs discusses how such programs can be optimized).

3.8 PRELIMINARY DESIGN

In this phase, the allocated processing requirements for each task (or schedulable software module) are identified and allocated to a hierarchy of software routines, subroutines, and procedures to a level where each can be described and sized. The overall approach for development, integration and test are identified for applications code and operating system modules. The requirements for test tools are expanded, and the test processes are defined. For the DP hardware, the details of the hardware design are solidified (e.g., bus structure), and the performance estimates used in the process design are validated by the more detailed designs. If significant differences occur, the process design is reiterated.

The activity of preliminary design for a task is fairly well understood by today's software practitioners. Dijkstra, Jackson, and Yourdon and Constantine all have methodologies for this level of design, to name but a few. The nine-step approach for the design does, however, suggest a shift in emphasis:

- Step 1, to identify requirements, is recognized by all.
- Step 2, to identify alternate types of modules for allocation, is not generally discussed. For example, software can be divided into modules for different types of action (refinement), or in different levels of action (e.g., all software is isolated from memory by a data manager). Both types of modules should be addressed.
- Step 3, different mathematical approaches should be examined for accomplishing a function.
- Step 4, the approaches are decomposed into steps to be allocated to modules.
- Step 5, different allocations are proposed, especially from the viewpoint of testability.
- Step 6, feasibility is estimated, particularly from the accuracy, execution time, and memory budget points-of-view.
- Step 7, the resource requirements are estimated (e.g., total instructions, total memory, development time).
- Step 8, a best design is selected. This is the place where execution time, development time, and memory size are traded-off.
- Step 9, integration and test plans, test tools, and their impact on design are identified, including the approach for testing the completed

code. This can include program correctness proof techniques, use of code analyzers, etc., during the design stage.

3.8.1 Discussion

Note that the nine-step approach is consistent with many of the design methodologies, but does explicitly include features which are ignored by many other methodologies. If the problem is stated as one of allocating requirements to classes of design elements, this tends to give the software design process a new dimension which has not been fully explored in the literature (e.g., the HOS design rules address the allocation of responsibilities for input data checking between modules are in the right direction). Existing tools to support these phases include PDL and PDS to describe the design, and PDS to aid in simulating the total effect of the design and maintain configuration management.

3.9 CONCLUSIONS

The overall methodology discussed in Section 3.1 appears to be sufficiently general to describe the system design front-end activities by its repeated application to four levels of allocation:

- System to DP/C
- DP/C to DP nodes and communication links
- DP node to process design
- Process to preliminary design.

The concerns of the steps are changed as the nature of the "subsystems" to be allocated change, but the sequence and types of activities remain the same.

The methodology has been found to work rapidly and quickly during the system analysis phase, and to highlight the data processing issues at the earliest possible time. This represents a significant advance towards early identification and resolution of DP issues in the system design front-end. Explicit phases and steps are included to address known problems (e.g., distributed process design, influence of test planning) unaddressed elsewhere.

Prototype tools exist for aiding many of these steps. The role of PERCAM and DP PERCAM in the system analysis and engineering phases, the role of SREM in the requirements engineering phase, the role of ALF and PDS in the process design phase, and PDL and PDS in the preliminary design phase has been discussed.

The generality of this methodology suggests that a series of languages based on a common meta-language could be developed to exploit the similarities to yield a common tool framework. This meta-language would have the following concepts:

- Elements -- containing the functions, subfunctions, structured data trees, performance indices, and system parameters at each level.

- Attributes -- providing the descriptions of the elements, type for data (including units and range), etc.
- Relationships -- providing linkages between elements such as functions INPUT data, traceability, decomposition between functions, allocation to subsystems, etc.
- Structures -- providing the graphs of the functions and sequences of the structured data trees.

If such a series of languages were developed on a common base, a common set of tools which utilized this deep structure could be developed to manipulate the data bases regardless of level of analysis. This would simplify the development and use of automated tools to perform consistency/completeness checking, traceability analyses, and simulation generation.

The above comments form the justification for the approach taken in the Evolutionary Development Plan published earlier.

4.0 CONCLUSIONS

In this part of the report, we have presented formal foundations for a methodology, and the overview of a methodology based on those foundations. The concepts of decomposition and allocation were found to be key concepts of the methodology for elaboration of the requirements, relating the requirements to the design, and for the definition of integration and test tools and the test plans and procedures.

This approach addresses the five issues previously identified in the following way:

- Complexity -- the complexity inherent in the many possible subsystem classes, system logics, and allocations is addressed by first identifying feasible solutions for subsystem classes, identifying the critical issues, and then optimizing to alleviate or eliminate these critical issues. The natural level hierarchy of system, subsystems, critical items, etc., are used, with strict decomposition and allocation used to link these levels together.
- Communication -- emphasis on testability and performance decomposition leads to the description of the system actions in terms of sequences of functions with specified inputs, outputs, and performances. This is a "natural" way to describe the actions of the system in terms of the integrated effect of subsystem actions; the precise definition of the decomposition also tends to eliminate ambiguity. The definition of a extensible machine-processible language for the expression of these concepts (e.g., the Requirements Statement Language of SREM) will further aid communication by restricting the vocabulary to a precisely defined set.
- Validation -- emphasis on formal decomposition and allocation leads not only to naturally testable requirements, but to the identification of verifiable properties of the specifications (e.g., consistency of inputs/outputs defined for requirements written in RSL). In addition, the emphasis on simulations to verify performance predictions which are traceable to the statement of requirements provides validation of the dynamic behavior of the system. Finally, the emphasis that the allocation is not complete without the integration and test plan leads to early emphasis on validation.
- Traceability -- the rigorous definition of decomposition and allocation provides an unbroken chain of decisions from any requirement or design feature back to the mission requirements. In addition, incorporation of these concepts in an automated data base (e.g., the REVS data base) provides the tools to perform the traceability analysis upward or downward.
- Change Response -- the definition of the requirements in an automated data base and the availability of tools to aid in the traceability analysis provides the capability to extract traceability information rapidly, perform an analysis of the impact of a change in top-level requirements or the infeasibility of meeting lower-level subsystem constraints on the total set of requirements. The ability to copy

the data base, modify it, perform automated analysis for consistency, completeness, traceability, and to generate simulations to predict system performance provides to the system analysts the capability to quickly identify impacts, make changes, and verify their impacts.

A comparison of the types of tools (e.g., automated data base of requirements, automated consistency/completeness checking, automated simulation generation from the requirements statements at various levels of the system design) needed to support such an overall methodology with the tools which currently exist suggest that prototype of many of these tools already exist:

- RSL is a prototype of the user-extensible requirements definition language for the definition of the system fucntions which incorporates most relevant URL features.
- REVS is a prototype of the types of tools to verify the static properties of the requirements, and to generate simulations of the specified system actions, which also incorporate most of the features of CARA.
- PERCAM is a prototype of the simulator to develop system-level simulations traceable to the system logic.
- DP PERCAM is a prototype of the simulator to develop system-level simulations of critical resource utilization.
- ALF is a prototype of the types of tools needed to predict the sufficiency of a process design.
- PDL and PSL are prototypes of tools to define the software preliminary design, and to simulate its performance.

The availability of these tools suggests that the consolidation of these tools into a unified set, and the detailed definition of a methodology which uses them is feasible. The details of tool consolidation, tool extension, methodology development, and technology transfer are discussed in the Evolutionary Development Plan previously published.

5.0 REFERENCES

1. Mesarovic, M.D. and Y. Takahara, "General Systems Theory Mathematical Foundations", Academic Press (1975).

2. Mesarovic, M. D., M. Macko, and Y. Takahara,"Theory of Hierarchical Multi-level Systems", Academic Press (1970).

3. Jackson, M. W., "Principles of Program Design", Academic Press (1975).

4. Cerf, V. C., "Multi-Processors, Semaphores, and a Graph Model of Computation", Department of Computer Science, University of California Los Angeles, Report UCLA-ENG-7223, April 1972.

5. Ross, D., "Structured Analysis (SA): A Language for Communicating Ideas", IEEE Transactions on Software Engineering, Volume SE-3, Number 1, January 1977.

6. Lamb, S. S., et al., "SAMM: A Modeling Tool for Requirements and Design Specification", COMSAC 78 Proceedings, IEEE Catalog 78 CHI 338-3 C, November 1978.

7. Hamilton, M. and S. Zeldin, "Higher Order Software - A Methodology for Defining Software", IEEE Transactions on Software Engineering, Volume SE-2, Number 1, pp 9 - 32, March 1976.

8. Dahl, R., E. Dijkstra, C. Hoare, "Structured Programming", Academic Press (1972).

9. Fitzwater, D. R., "A Decomposition of the Complexity of System Development Processes", COMSAC 78 Proceedings, IEEE Catalog No. 78 CHI 338 - 3 C, November 1978.

10. Wymore, A. W., "Systems Engineering Methodology for Interdisciplinary Teams", John Wiley (1976).

11. MIL-STD-490, "Military Standard Specification Practices", 30 October 1968.

12. Bell, T. E., D. C. Bixler, M. E. Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering", IEEE Transactions on Software Engineering, Volume SE-3, Number 1, pp 99 - 60, January 1977.

☆U.S. GOVERNMENT PRINTING OFFICE: 1979-614-023/208